A Hardware-in-the-Loop Simulation Facility for the Attitude Determination and Control System of SUNSAT

by J A A Engelbrecht

Thesis presented in partial fulfilment of the requirements for the degree of Master of Science in Electronic Engineering at the University of Stellenbosch

December 1999

Promoter : Prof. J.J. du Plessis

Declaration:

I, the undersigned, hereby declare that the work contained in this thesis is my own original work and has not previously been submitted at any university, in part or in its entirety, for a degree.

ABSTRACT

A hardware-in-the-loop simulation facility was developed for the attitude determination and control system (ADCS) of the Stellenbosch University Satellite (SUNSAT), a low earth orbit, nadir pointing microsatellite. Software simulations were created to describe the orbital and attitude dynamics of the satellite, and also to describe the space environment, including the geomagnetic field, sun, earth horison and star field. Hardware emulators were designed to emulate the interfaces of the satellite's sensors and actuators with the ADCS tray. The simulation software and the hardware emulators were then combined to systematically close the hardware loop around the engineering model of SUNSAT's ADCS.

A satellite-in-the-loop simulation mode was also included in the hardware-in-the-loop simulation facility so that whole orbit data collected and recorded by the SUNSAT flight model and transmitted to the ground station could be fed to simulations of the on-board ADCS algorithms. The satellite-in-the-loop simulation was used to investigate an unmodelled disturbance torque acting on the satellite body which was revealed by the whole orbit data.

SAMEVATTING

’n Apparatuur-in-die-lus simulasie omgewing is ontwikkel vir die oriëntasie afskatting en beheer stelsel (ADCS) van die Stellenbosch Universiteit Satelliet (SUNSAT), ’n lae aardbaan, aardwysende mikrosatelliet. Sagteware simulasies is geskryf om die wentelbaan en oriëntasie dinamika van die satelliet te beskryf en ook om die ruimte omgewing, insluitend die magneetveld van die aarde, die son, die aardhorison en die sterreveld te beskryf. Hardeware emuleerders is ontwerp om die koppelvlakke van die satelliet se sensore en aktueerders met die ADCS laai na te boots. Daarna is die simulasie sagteware en hardeware emuleerders geïntegreer om die apparatuur lus stelselmatig te sluit rondom die ingenieursmodel van SUNSAT se ADCS.

’n Satelliet-in-die-lus simulasie modus is ook ingesluit by die apparatuur-in-die-lus simulasie omgewing sodat wentelbaan data wat deur die SUNSAT vlugmodel versamel en na die grondstasie versend is aan simulasies van die aanboord ADCS algoritmes gevoer kan word. Die satelliet-in-die-lus simulasie is aangewend om ’n steurdraaimoment op die satelliet liggaam, wat nie voorheen gemodelleer is nie, maar wat deur die wentelbaan data uitgewys is, te ondersoek.

Table of Contents

List of Abbreviations

ACP	Attitude Control Processor
ADAS	Analog-Digital-Analog System
ADCS	Attitude Determination and Control System
CCD	Charge Coupled Device
DC	Direct Current
DCM	Direction Cosine Matrix
EKF	Extended Kalman Filter
FOV	Field of View
FPGA	Field Programmable Gate Array
GPS	Global Positioning System
HIL	Hardware-in-the-Loop
HSSP	Horison / Sun Sensor Piggyback
ICP	Interface Control Processor
IGRF	International Geomagnetic Reference Field
ISA	Industry Standard Architecture
JPL	Jet Propulsion Laboratories
LEO	Low Earth Orbit
LEOSat	Low Earth Orbit Satellite
MMP	Magnetometer Piggyback
NORAD	North American Aerospace Defence Command
OBC	On-Board Computer
OBC1	Primary On-Board Computer
OBC2	Secondary On-Board Computer
OriMag	Orientation Magnetometer
PC	Personal Computer
RPY	Roll, Pitch and Yaw
RWC	Reaction Wheel Controller
SciMag	Scientific Magnetometer
SCP	Star Sensor Control Processor
SGP4	Simplified General Perturbations
SIL	Satellite-in-the-Loop
SSP	Star Sensor Piggyback
SUNSAT	Stellenbosch University Satellite
UART	Universal Asynchronous Receiver / Transmitter
UHF	Ultra High Frequency
UTC	Coordinated Universal Time
VHF	Very High Frequency
WOD	Whole Orbit Data

List of Symbols

a	equatorial radius of the Earth
A	effective magnetic torquer coil area
A_p	total projected area of spacecraft
$\mathbf{A}$	direction cosine matrix
$\hat{\mathbf{A}}$	estimated direction cosine matrix
$\mathbf{B}$	geomagnetic field vector in body coordinates
$\mathbf{B}_c$	calibrated geomagnetic field vector in body coordinates
B_x, B_y, B_z	components of the geomagnetic field vector in body coordinates
B_{cx}, B_{cy}, B_{cz}	components of the geomagnetic field vector in celestial coordinates
B_{ox}, B_{oy}, B_{oz}	components of the geomagnetic field vector in orbit coordinates
B_r, B_θ, B_ϕ	components of the geomagnetic field vector in local tangent coordinates
B_r	radial component of geomagnetic field vector (outward positive) in spherical geocentric inertial coordinates
B_θ	coelevation component of geomagnetic field vector (South positive) in spherical geocentric inertial coordinates
B_ϕ	azimuthal component of geomagnetic field vector (East positive) in spherical geocentric inertial coordinates
$\mathbf{B}_{centre}$	geomagnetic field vector in centre of magnetic torquer pulse
$\mathbf{B}_{meas,k}$	measured geomagnetic field vector in body coordinates
$\mathbf{B}_{o,k}$	modelled geomagnetic field vector in orbit coordinates
c	velocity of light
$\mathbf{c}_p$	vector between centre of mass and centre of pressure
C, S	cosine, sine
d_0	average solar radiation constant
$d\varpi_{xi}, d\varpi_{yi}, d\varpi_{zi}$	inertially referenced angular rate increments
$d\omega'_{xi}, d\omega'_{yi}$	corrected inertially referenced angular rate increments
$d\omega_{ox}, d\omega_{oy}, d\omega_{oz}$	transformation increments to transform the inertially referenced angular rates to orbit referenced angular rates
e	orbit eccentricity
$\mathbf{e}$	unit Euler axis vector
$\mathbf{e}_{k+1}$	innovation error vector
$\mathbf{E}$	euler axis
E_1, E_2, E_3	components of the Euler axis vector
e_x, e_y, e_z	components of the unit Euler axis vector
f	earth flattening factor

$\mathbf{f}(\hat{\mathbf{x}}_{k/k}, \mathbf{u}_k, k)$	non-linear discrete system model
$\mathbf{F}(\mathbf{x}(t_{k+1}), t_{k+1})$	linearised perturbation state matrix
$g_{n,m}$, $h_{n,m}$	Gauss normalised IGRF coefficients
$GM_{\oplus}$	earth's gravitation constant
$\mathbf{G}, \mathbf{b}$	magnetometer calibration gain and offset
$\mathbf{h}$	reaction wheel angular momentum vector in body coordinates
h_x, h_y, h_z	components of the reaction wheel angular momentum in body coordinates
$\mathbf{h}_{\mathrm{lim}}$	reaction wheel angular momentum limit
$\mathbf{h}_{sat}$	reaction wheel saturated angular momentum
$\mathbf{h}(\hat{\mathbf{x}}, k)$	sensor model
$\mathbf{H}_{k+1/k}$	discrete output measurement matrix
i	orbit inclination
I	DC current through the magnetic torquer coil
$\mathbf{I}$	moment of inertia tensor in body coordinates
I_T, I_Z	transverse and Z-axis components of the body moment of inertia in body coordinates
I_{xx}, I_{yy}, I_{zz}	components of the moment of inertia tensor in body coordinates
$\mathbf{K}_{k+1}$	Kalman Filter Gain
$\mathbf{K}_{slew}$	slew torque gain
$\mathbf{M}$	magnetic dipole moment vector generated by the 3-axis magnetic torquer coils in body coordinates
$\mathbf{M}_{\mathrm{constant}}$	constant magnetic dipole moment
M_0	orbit mean anomaly at epoch
$\mathbf{M}_{\mathrm{solar\ panel}}$	modelled solar panel magnetic dipole moment
n	number of turns per magnetic torquer coil
$\hat{n}_{doy}$	estimated aerodynamic disturbance torque
$\mathbf{N}_{add}$	additional linearisation torque in reaction wheel controllers
$\mathbf{N}_{\mathrm{AERO}}$	aerodynamic disturbance torque in body coordinates
$\mathbf{N}_{comp}$	inertia mismatch compensation torque in reaction wheel controllers
$\mathbf{N}_{\mathrm{D}}$	external disturbance torque vector in body coordinates
$\mathbf{N}_{D,\mathrm{unmodelled}}$	unmodelled disturbance torque
$\mathbf{N}_{\mathrm{GG}}$	gravity gradient torque vector in body coordinates
N_{lim}	reaction wheel torque limit
$\mathbf{N}_{\mathrm{M}}$	magnetic torque vector in body coordinates
$\mathbf{N}_{\mathrm{residual\ magnetism}}$	residual magnetism / active magnetic torquers disturbance torque

N_s	scalar slew torque in reaction wheel controller
$\mathbf{N}_{sat}$	reaction wheel saturation torque
$\mathbf{N}_{slew}$	slew torque in reaction wheel controller
$\mathbf{N}_{\mathrm{SOLAR}}$	solar radiation disturbance torque in body coordinates
$\mathbf{N}_{\text{solar panel}}$	modelled solar panel magnetic disturbance torque
$\mathbf{N}_{wheel}$	reaction wheel torque in body coordinates
$P_{n,m}$	Gauss functions
$\mathbf{P}_{k+1/k}$	perturbation covariance matrix
$\mathbf{q}$	quaternion vector
$\hat{\mathbf{q}}$	estimated quaternion vector
$\mathbf{q}_c$	commanded quaternion vector
$\mathbf{q}_e$	quaternion error vector
q_1, q_2, q_3, q_4	Euler symmetric parameters, or quaternions
q_{half}	quaternion halfway mark for slew controller
$\mathbf{Q}$	system noise covariance matrix
$\mathbf{R}$	measurement noise covariance matrix
r	geocentric distance
$\mathrm{R_e}$	oblate earth radius
R_s	geocentric spacecraft position vector length
$\mathrm{s}(t_k)$	process noise
S_x, S_y, S_z	components of the sun vector in body coordinates
$\mathbf{S}_o$	sun vector in orbit coordinates
S_{ox}, S_{oy}, S_{oz}	components of the sun vector in orbit coordinates
$\hat{\mathbf{S}}$	estimated sun vector in body coordinates
$\hat{\mathbf{S}}_{body}$	estimated sun vector in body coordinates
T_s	sampling period
T_x, T_y, T_z	components of the star vector in body coordinates
T_{ox}, T_{oy}, T_{oz}	components of the star vector in orbit coordinates
t	time
t_{accel}	acceleration time for slew controller
t_{coast}	coasting time for slew controller
V	magnitude of spacecraft velocity vector
$\mathbf{V}$	unit spacecraft velocity vector
$\mathbf{v}_{body,k+1/k}$	estimated sensor measurement vector
$\mathbf{v}_{meas,k+1}$	sensor measurement vector

$\mathbf{v}_{\min}$, $\mathbf{v}_{\max}$	horison sensor minimum and maximum field of view vectors in body coordinates
$\mathbf{v}_{orb,k+1}$	modelled measurement vector in orbit coordinates
v_x, v_y, v_z	components of the horison sensor field of view vector in body coordinates
v_{ox}, v_{oy}, v_{oz}	components of the horison sensor field of view vector in orbit coordinates
$\hat{v}_{ox}, \hat{v}_{oy}, \hat{v}_{oz}$	components of the estimated horison sensor measurement vector in orbit coordinates
$\mathbf{v}^{hx}_{meas,k}$	X horison sensor measurement vector
$\mathbf{v}^{hx}_{orb,k}$	X horison sensor modelled vector
$\mathbf{v}^{hy}_{meas,k}$	-Y horison sensor measurement vector
$\mathbf{v}^{hy}_{orb,k}$	-Y horison sensor modelled vector
$\mathbf{v}^{sun}_{meas,k}$	sun sensor measurement vector
$\mathbf{v}^{sun}_{orb,k}$	sun sensor modelled vector
$\mathbf{v}^{star,i}_{meas,k}$	i'th star sensor measurement vector
$\mathbf{v}^{star,i}_{orb,k}$	i'th star sensor modelled vector
$\mathbf{x}(t_k)$	state vector
$\hat{\mathbf{x}}(t_k)$	estimated state vector
x, y, z	axes in rectangular body coordinates
x_C, y_C, z_C	axes in celestial coordinates
x_i, y_i, z_i	axes in rectangular inertial coordinates
x_o, y_o, z_o	axes in rectangular orbital coordinates
$\mathbf{z}$	z-axis unit vector in body coordinates
$\mathbf{z}_o$	nadir pointing unit vector in body coordinates
α	distance angle between the sub-sun point and the horison at the horison sensor boresight azimuth
α	right ascension
α_G	right ascension of the Greenwich meridian
β	distance angle between the sub-satellite point and the sub-sun point
δ	declination
$\delta\mathbf{x}(t_k)$	state perturbation
ε	distance angle between the delayed sub-satellite point and the sub-sub point
ϕ	East longitude from Greenwich

ϕ, λ	azimuth and elevation in spherical body coordinates
ϕ, θ, ψ	Euler angles, or roll, pitch and yaw
$\hat{\phi}, \hat{\theta}, \hat{\psi}$	estimated Euler angles
Φ	rotation angle about the Euler axis
Φ	angle between the sub-sun point azimuth and the horison sensor azimuth
$\Phi_{k+1/k}$	discrete system matrix
γ	distance angle between the sub-satellite point and the horison at the horison sensor boresight azimuth
μ_g, μ_b	variable step sizes for magnetometer calibration gain and bias
ν	orbit true anomaly
θ	coelevation
θ	distance angle between the sub-satellite point and the delayed sub-satellite point
θ_{hx}	X horison sensor measurement
θ_{hy}	-Y horison sensor measurement
θ_{sun}	azimuth angle measured by the sun sensor
ρ_a	atmospheric density
ω	orbit argument of perigee
ω_A	body nutation rate
ω_B^I	body angular rate vector in inertial coordinates
$\hat{\omega}_B^I$	estimated body angular rate vector in inertial coordinates
ω_B^O	body angular rate vector in orbit coordinates
$\hat{\omega}_B^O$	estimated body angular rate vector in orbit coordinates
ω_o	orbit mean motion
$\tilde{\omega}_o(t)$	true orbit angular rate
$\omega_x, \omega_y, \omega_z$	components of the body angular rate vector in inertial coordinates
$\omega_{ox}, \omega_{oy}, \omega_{oz}$	components of the body angular rate vector in orbit coordinates
$\hat{\omega}_{ox}, \hat{\omega}_{oy}, \hat{\omega}_{oz}$	components of the estimated body angular rates in orbit coordinates
$\omega_{xi}, \omega_{yi}, \omega_{zi}$	components of the body angular rate vector in inertial coordinates
$\hat{\omega}_{xi}, \hat{\omega}_{yi}, \hat{\omega}_{zi}$	components of the estimated body angular rates in inertial coordinates
$\hat{\omega}'_{xi}, \hat{\omega}'_{yi}$	corrected estimated inertially referenced angular rates
Ω	orbit right ascension of the ascending node
Ω	sun azimuth in orbit coordinates
Ω'	angle between the orbit plane and the sub-sun point
$\mathbf{\Omega}$	angular rate matrix

Ψ	azimuth of the horison vector in the orbit X_oY_o plane, measured from the east direction

List of Conventions

arctan4	four quadrant arctan

$$\arctan 4\left(\frac{y}{x}\right) = \begin{cases} \arctan\left(\frac{y}{x}\right) & , x > 0 \\ \pi + \arctan\left(\frac{y}{x}\right) & , x < 0, y >= 0 \\ -\pi + \arctan\left(\frac{y}{x}\right) & , x < 0, y < 0 \\ \pi/2 & , x = 0, y >= 0 \\ -\pi/2 & , x = 0, y < 0 \end{cases}$$

List of Figures

List of Tables

1. INTRODUCTION

1.1 Overview

This thesis concerns the hardware-in-the-loop simulation of the attitude determination and control system (ADCS) of a low earth orbit, nadir pointing microsatellite. A hardware-in-the-loop simulation facility was developed for the ADCS of the Stellenbosch University Satellite (SUNSAT). Software simulations were created to describe the orbital and attitude motion of the satellite, as well as its space environment, including the geomagnetic field, sun, earth horison and star field, and hardware emulators were designed to emulate the interfaces of the satellite's sensors and actuators with the ADCS tray. The simulation software and the hardware emulators were then combined to systematically close the hardware loop around the engineering model of SUNSAT's ADCS.

After the launch of the SUNSAT flight model on 23 February 1999, the need also arose for satellite-in-the-loop simulation, which is a special case of hardware-in-the-loop. The simulation software was adapted so that whole orbit data collected and recorded by the flight model and transmitted to the ground station could be fed to simulations of the on-board ADCS algorithms.

A system identification feature was also added to the satellite-in-the-loop simulation to investigate the appearance of an unmodelled disturbance torque which was revealed by the whole orbit data.

The following topics are covered by this thesis:

- Chapter 1 serves as an introduction by describing the SUNSAT microsatellite and giving special attention to the details of its ADCS system. The concepts of hardware-in-the-loop simulation and satellite-in-the-loop simulation are also discussed.
- Chapter 2 defines the coordinate systems used throughout this thesis and outlines several mathematical models used to describe the orbital and attitude motion of the satellite and also the space environment in which it finds itself. The simulation models for the satellite's sensors and actuators are also presented.
- Chapter 3 describes the algorithms which represent the software of the satellite's attitude determination and control system (ADCS), including on-board orbit propagator and space environment models, sensor validation tests, attitude estimators, magnetic torquer controllers, reaction wheel controllers, boom deployment algorithms and an on-board magnetometer calibration algorithm.
- Chapter 4 describes the layout of the hardware-in-the-loop facility, the structure and timing of the hardware-in-the-loop simulation software and also the nature of

the sensor and actuator hardware emulators. It concludes with a discussion about possible ways to augment the hardware-in-the-loop simulation to include the true sensor hardware in the loop.

- Chapter 5 describes satellite-in-the-loop simulation software which was used to analyse whole orbit data. The investigation into the unmodelled disturbance torque which was acting on the satellite body is also presented here.
- Chapter 6 is the conclusion of the thesis. It summarises the contributions made by this thesis and also gives a few recommendations.

1.2 Background - The SUNSAT microsatellite

Since the hardware-in-the-loop simulation facility was developed for the Attitude Determination and Control System (ADCS) of the Stellenbosch University Satellite (SUNSAT), a brief description of this particular satellite will be provided in this paragraph, and a more detailed description of the ADCS system will be presented in the next paragraph.

SUNSAT is an almost cubical microsatellite with a mass of $63.2kg$, external dimensions of $45cm \times 45cm \times 60cm$ and a $2.2m$ long deployable gravity boom with a $4.5kg$ tip mass. The main payload of the satellite is a high resolution pushbroom imager capable of stereo imaging.

The satellite has a modular structure and its cubical body consists of ten trays representing the satellite subsystems, sandwiched by a bottom plate and a top plate. The tray structure facilitates the pre-launch integration since a tray containing a faulty subsystem may easily be removed, repaired and replaced in the satellite body. After the trays have been stacked and secured, four solar panels are mounted to the four side faces of the cubic satellite body. The ten trays, and the top and bottom plate, are stacked in the following order from top to bottom: the top plate, the attitude determination and control system (ADCS), the mass memory, (RAM tray), the secondary on-board computer (OBC2), the primary on-board computer (OBC1), the telecommand system, the telemetry system, the GPS tray, the VHF communications system, the UHF communications system, the power management system, the bottom tray and the bottom plate.

Mounted on the top plate are the deployable gravity boom, four VHF antennas, a small UHF antenna, the patch antenna for the GPS system, three school experiments and all of the satellite's sensors. The sensors include a magnetometer (OriMag), a sun sensor, two horison sensors, a star sensor and five of six cosine law sun cell sensors (the sixth sun cell sensor is located on the bottom plate). The school experiments include two impact sensors (one piezo-electric and the other capacitive) and an experiment for monitoring the effect of radiation on carbon samples. The top plate also houses the VHF phase network which feeds the four VHF antennas, a set of pyrocutters for releasing the boom and both z-axis magnetic torquers. Mounted on the tip mass at the end of the deployable boom are a star sensor and a duplicate magnetometer (SciMag). The tip mass also has a set of laser reflectors which are part of a NASA experiment and may be used to accurately determine the position of SUNSAT by using ground based lasers.

The attitude determination and control system is located directly underneath the top plate since it needs to interface with the sensors which are mounted there. The ADCS has two main processors, the Attitude Control Processor (ACP), a T800 transputer which implements all the control system software, and the Interface Control Processor (ICP), a 80C31 based microcontroller which interfaces directly with all actuators and sensors. The ADCS uses the data from the sensors to determine the attitude of the satellite and controls the attitude with the magnetic torquers, which are located around the side solar panels and in special recesses in the top plate, and with the reaction wheels, which are located in the bottom plate.

The mass memory has a capacity of 64MB and performs two functions: it acts as an interface with the S-band transmitter and the imager, and it serves as a storage space for data from various subsystems.

The secondary on-board computer (OBC2) uses a 80386 processor and a 80387 math coprocessor. It supports the primary on-board computer and can also take over most of the functions of the ACP on the ADCS if it should fail.

The primary on-board computer (OBC1) uses a 80188 processor and performs all of the processing functions of the satellite, excluding the ADCS functions.

The telecommand system controls the switches for activating and deactivating most of the satellite's sensors, actuators, processors, radios and even entire subsystems.

The telemetry system collects and digitises data necessary for monitoring all the subsystems of the satellite. This data includes voltages, currents, temperatures, the power used by the transmitters and the states of important switches. The telemetry data may be stored in the mass memory or transmitted directly by one of the radios. The telemetry system also contains various modems for demodulating low speed data from the ground station and for modulating data to be transmitted to the ground.

Despite its name, the GPS tray actually contains a variety of electronic circuits. The circuit which gives the tray its name, is the Global Positioning System (GPS) receiver which is connected to the patch antenna on the top plate. The GPS receiver was provided by Jet Propulsion Laboratories (JPL) and is part of an experiment for NASA. The GPS tray also contains two more school experiments, and components of the UHF, S-band and L-band communications systems. The school experiments include a microphone for picking up vibrations in the satellite body and an experiment for monitoring the effects of radiation on electronic components.

The VHF communications system consists of transmitters and receivers for communicating at VHF frequencies.

The UHF communication system consists of transmitters and receivers for communicating at UHF frequencies.

The power management system controls the charging of the batteries by the solar panels and also performs load management. In this way it prevents the batteries from becoming overcharged or completely discharged.

The bottom tray contains the high resolution pushbroom imager, a small TV camera, the batteries and the four reaction wheels. The imager takes pictures in the red, near infra-red and green bands and may be rotated with a stepper motor to obtain stereo images.

Mounted on the bottom plate are the UHF phase antenna and the VHF, L-band and S-band antennas.

SUNSAT was launched on 23 February 1999 aboard a Delta II launch vehicle and is currently orbiting the earth in a polar, low earth orbit with an orbital period of approximately 100 minutes. Operators daily communicate with the satellite from the ground station in Stellenbosch, South Africa.

1.3 The Attitude Determination and Control System

1.3.1 SENSORS

Magnetometer

The magnetometer measures the strength and direction of the geomagnetic field vector in three axes. The measured geomagnetic field vector may then be compared to geomagnetic field models to obtain attitude information and may also be used to estimate the torque produced by the magnetic torquers. The advantage of using the magnetometer for attitude determination is that valid measurements are available throughout the satellite orbit; the disadvantage is the inaccuracy of the data due to errors in the geomagnetic field models.

Horison Sensors

Two orthogonal horison sensors obtain orthogonal measurements of the sunlit earth horison. The horison sensors are linear CCD and lens assemblies which look below the local horisontal level with a ±15° field of view. Pitch and roll attitude angles to an accuracy of 0.5 mrad may be measured with these sensors, but the measurements are only valid for valid fields of view and valid horison illumination.

Sun Sensor

A sun sensor measures the sun azimuth angle within a ±60° field of view to an accuracy of 1 mrad. The sensor uses similar CCD technology to the horison sensors and the sensor head consists of a slit aperture perpendicular to the linear CCD. Yaw attitude angles may be obtained from the sun sensor, but its measurements are only valid for a valid field of view and a valid line of sight to the sun.

Star Sensor

The star sensor uses a pixel matrix CCD to take an image of the star field within a 10°×10° field of view. Individual stars are then extracted from this image and constellations are identified, using a star catalogue, based on the magnitudes and separation distances of the stars detected in the image. For this purpose, the star sensor has its own processor, a T800 transputer called the Star Sensor Control Processor (SCP). By comparing the identified constellations in the image with the constellations in the catalogue, the roll and yaw angles may be estimated to an accuracy of 0.5 mrad and the pitch angle may be estimated to an accuracy which depends on the star separation distance.

Sun Cell Sensors

The sun cell sensors are six cosine-law solar cells which are mounted on each facet of SUNSAT's cubic body. Using a satellite orbit propagator and a sun model, full attitude information may be obtained from these sensors to an accuracy of $\pm 5°$. The sun vector with respect to the satellite body may be obtained from the short circuit currents of the solar cells. The temperatures of the cells are also measured to make sensitivity corrections to the current measurements.

1.3.2 ACTUATORS

Gravity Boom

The deployable gravity boom provides passive attitude control by utilising the gravity gradient torque which acts upon the satellite body. After SUNSAT has been released into its orbit by the launch vehicle, it is detumbled and controlled to track a precalculated pitch rate using the magnetic torquers. The boom is then deployed at the correct moment to result in a nadir pointing gravity gradient lock.

Magnetic Torquers

The magnetic torquers are six air core coils wound into recesses around the solar panels and in the top plate. Using these coils, a magnetic dipole moment can be generated in three axes. The interaction of this magnetic dipole moment with the geomagnetic field results in a magnetic torque which is applied to the satellite body. With appropriate control algorithms, this magnetic torque may be utilised for attitude control.

The main control functions of the magnetic torquers are active stabilisation of the satellite through libration damping and Z-spin rate control, and momentum dumping of the reaction wheels. Before boom deployment, the magnetic torquers are used for detumbling and pitch rate tracking.

The magnetic torquers on SUNSAT use a pulse width modulation technique to control the amplitude of the generated magnetic dipole moment.

Reaction Wheels

The reaction wheels provide fast, accurate and continuous attitude control. There are four reaction wheels, one aligned to each of the X, Y and Z body axes and a fourth also aligned to the Z-axis to add redundancy. Two of the wheels are driven by conventional DC motors and the other two are driven by brushless DC motors.

The main control functions of the reaction wheels are to provide accurate pointing and tracking control during imaging and also for performing large angular slew manoeuvres to point the imager at different targets within a short span of time.

Since the reaction wheels have a limited operational life in vacuum, they are only used for pointing and stabilisation during imaging.

1.3.3 PROCESSORS

Attitude Control Processor (ACP)

The Attitude Control Processor (ACP) is a T800 transputer which implements the control software of the ADCS. The ACP communicates with the Interface Control Processor (ICP) via a bidirectional UART and with the on-board computers (OBC's) via its links and link adapters.

Interface Control Processor (ICP)

The Interface Control Processor (ICP) is a 80C31 based microcontroller which interfaces directly with all sensors and actuators. Every second, the ICP receives a data packet containing the control signals for the magnetic torquers and reaction wheels from the ACP and returns a packet containing the sensor data to it via the bidirectional UART.

1.4 Approach to Hardware-in-the-Loop Simulation

1.4.1 HARDWARE-IN-THE-LOOP SIMULATION

The purpose of this thesis was to develop a prototype hardware-in-the-loop simulation facility for the ADCS system of SUNSAT. An overview of the approach taken in implementing this facility will be given in this paragraph.

The foundation of the hardware-in-the-loop simulation is a pure software simulation. This software simulation has two main components which form a software loop. The first component simulates the orbit and attitude motion of the satellite, as well as the space environment, including the geomagnetic field, sun, earth horison and star field. The second component simulates the operation of the ADCS software. The mathematical models used by the simulation were obtained from models used by Steyn, Wertz and Jacobs.

After the software simulation loop was successfully closed, hardware emulators were developed to emulate the sensors and actuators. The purpose of these emulators are to emulate the electrical interfaces of the sensors and actuators with the ADCS tray. The emulators were designed to be integrated into the ISA bus of a personal computer. Existing ISA-based PC cards such as custom ADAS cards, a custom FPGA card and custom transputer cards were used in the design of the emulator hardware. Software drivers were also written to drive the emulators.

Finally, the hardware loop was systematically closed around the ADCS tray using the simulation software and the hardware emulators. The result was a hardware-in-the-loop simulation facility consisting of three personal computers which communicate with each other via their serial ports. The first computer, a 300MHz Pentium II called Simulator, was used to run the software simulation, while the other two, a 60 MHz Pentium called Emulator One and a 486 called Emulator Two, were used to interface with the hardware emulators through their ISA buses.

The next step in hardware-in-the-loop simulation would be to include the true sensor hardware in the loop. Although this is not covered by the thesis, some suggestions are made for bringing the true sensor hardware into the loop, using the C*ontraves Georz* three axis rate table.

1.4.2 SATELLITE-IN-THE-LOOP SIMULATION

This thesis also touches on satellite-in-the-loop simulation which is a special case of hardware-in-the-loop simulation where the satellite flight model itself serves as the hardware in the loop.

Data collected by the flight model is recorded in Whole Orbit Data (WOD) files and transmitted to the ground station. The contents of the WOD file may then be fed to the simulation software. In this way, sensor data can be analysed on the ground to extract attitude information and to verify the correct operation of all on-board ADCS algorithms.

A case study is also presented where the satellite-in-the-loop simulation was used to investigate the appearance of an unmodelled disturbance torque acting on the satellite body.

2. SATELLITE MOTION AND SPACE ENVIRONMENT SIMULATION MODEL

2.1 Introduction

Mathematical models are required to simulate the motion of the satellite in space and also to represent the space environment itself.

The simulation of the satellite motion is divided into simulations of the satellite's orbital motion and its attitude motions. The simulation of the attitude motions are further subdivided into dynamic equations of motion, which relate the time derivative of the satellite's angular momentum to the torques applied to it, and kinematic equations of motion, which are concerned with the satellite's motion irrespective of the forces which bring it about.

The simulation of the satellite's space environment includes mathematical models for the sun orbit, the rotation of the earth, the geomagnetic field, the earth horison and the star field. The outputs of these space environment models are used to simulate sensor data for the magnetometer, sun cell sensors, sun sensor, horison sensors and star sensor and also to determine the influence of actuators such as the magnetic torquers and reaction wheels on the satellite motion.

The satellite motion and space environment simulation models used in this thesis have been constructed from models documented by Steyn, Wertz and Jacobs.

This chapter defines the coordinate systems used throughout this thesis and outlines several mathematical models used to describe the motion of the satellite and also the space environment in which it finds itself. The simulation models for the satellite's sensors and actuators are also included in this chapter.

2.2 SGP4 Orbit Propagator

2.2.1 SATELLITE ORBIT PROPAGATOR

The same on-board SGP4 orbit propagator which the ADCS uses to model the orbital motion of the satellite was used for the simulation software. The orbit propagator is initialised with NORAD parameters which include the inclination, right ascension of the ascending node, argument of perigee, eccentricity, mean motion, mean anomaly and epoch of the orbit, as well as a drag term. The input variable of the propagator is the time since epoch and the most important outputs include the geocentric distance, mean anomaly, true anomaly, right ascension, altitude, latitude, longitude and geodetic latitude of the satellite.

2.2.2 SUN ORBIT PROPAGATOR

The on-board SGP4 orbit propagator used by the ADCS, also models the orbit of the sun. Using the time since epoch as input, the sub-sun latitude and longitude are calculated. This same sun orbit propagator was used for the simulation software.

2.3 Coordinate Systems

Three major coordinate systems are used to define the attitude of a satellite, namely the inertial, orbit and body coordinate systems.

2.3.1 SPACECRAFT-CENTRED COORDINATES

Three basic types of coordinate systems centred on the spacecraft are defined by Wertz: those fixed relative to the body of the spacecraft, those fixed in inertial space and those defined relative to the orbit and not fixed relative to either the spacecraft or inertial space. The relevant definitions and conventions will be repeated here to clarify the rest of the work in this document

Spacecraft-Fixed Coordinates

These coordinates will also be referred to as body coordinates. This coordinate system is used to define the orientation of ADCS hardware and is also the systems in which attitude measurements are made.

Rectangular body coordinates are composed of the three components x, y and z, while spherical body coordinates consist of azimuth ϕ and elevation λ.

The spacecraft-fixed rectangular coordinates for SUNSAT are defined by Steyn as shown in Figure 1. The body z-axis is parallel and opposite to the direction of boom deployment and the body x and y axes are perpendicular to two of the side solar panels.

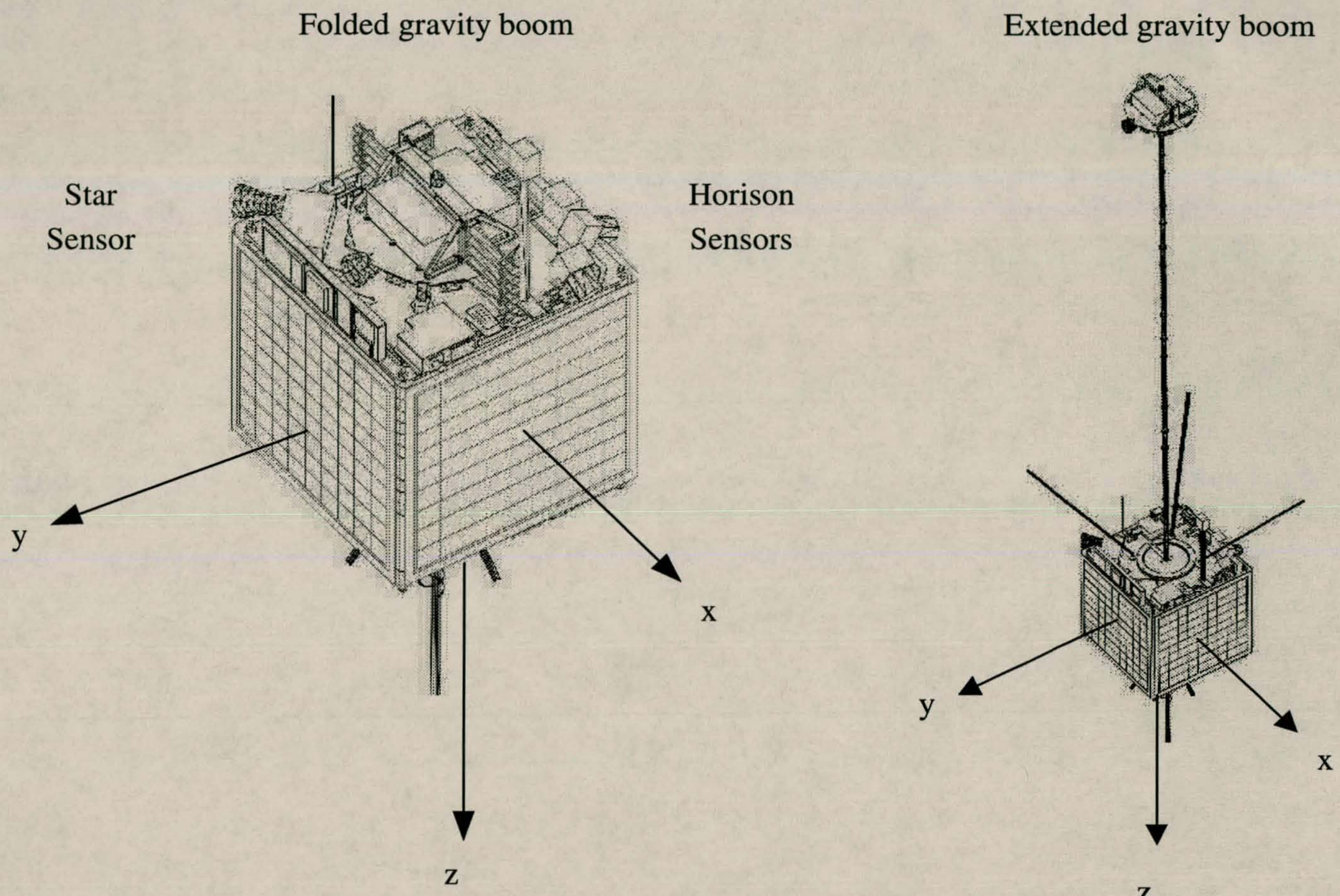


Figure 1: The body coordinates system as defined for SUNSAT.

Orbit-Defined Coordinates

This coordinate system maintains its orientation relative to the Earth as the spacecraft moves in its orbit. The most commonly used orbit-defined coordinate system is *roll*, *pitch* and *yaw* or *RPY* coordinates, also known as *Euler angles*. These are defined as shown in Figure 2. The *yaw* axis or z_o-axis is directed towards the nadir[1], the *pitch* axis or y_o-axis is directed towards the negative orbit normal and the *roll* axis or x_o-axis completes the orthogonal set. In a circular orbit, the roll axis or x_o-axis would point in the direction of the velocity vector. In Figure 2, the page represents the orbital plane and the y_o-axis is directed into the page (as denoted by the X notation).

[1] towards the centre of the earth

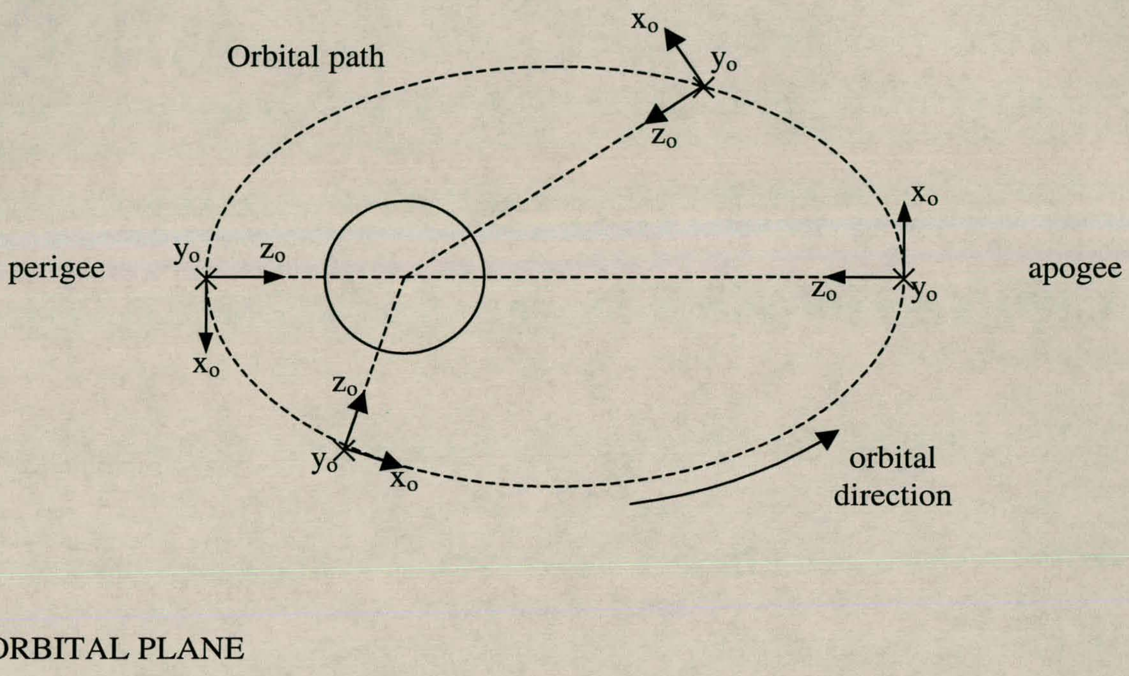


Figure 2: The orbit coordinates system for SUNSAT.

Individual spacecraft ordinarily define RPY systems unique to that spacecraft and may even define them as spacecraft-fixed coordinates rather than orbit-defined coordinates, but SUNSAT follows the standard convention defined above.

The following matrix is used to do an Euler 1-2-3 transformation from orbit to body coordinates:

$$\mathbf{A} = \begin{bmatrix} C\psi C\theta & C\psi S\theta S\phi + S\psi C\phi & -C\psi S\theta C\phi + S\psi S\phi \\ -S\psi C\theta & -S\psi S\theta S\phi + C\psi C\phi & S\psi S\theta C\phi + C\psi S\phi \\ S\theta & -C\theta S\phi & C\theta C\phi \end{bmatrix} \qquad \textbf{(Eq. 2.1)}$$

where,

C = cosine function

S = sine function

and,

ϕ, θ, ψ = roll, pitch and yaw

The transformation matrix **A** is called the *direction cosine matrix* (DCM).

Therefore,

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathbf{A} \begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} \qquad \text{and} \qquad \begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} = \mathbf{A}^{-1} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

transforms the $x_oy_oz_o$ vector in orbit coordinates to the xyz vector in body coordinates, and vice versa.

Although the representation of spacecraft attitude by Euler angles has a clear physical interpretation, it suffers from singularities in the pitch angle θ. This is a serious disadvantage of Euler angle formulations for numerical integration of the equations of motion. For this reason, the direction cosine matrix is parameterised in terms of the *Euler symmetric parameters*, or *quaternions*. The quaternion representation of spacecraft attitude has no singularities and is well suited for numerical integration.

The quaternions are defined by

$$\begin{aligned} q_1 &\equiv e_x \sin\left(\frac{\Phi}{2}\right) \\ q_2 &\equiv e_y \sin\left(\frac{\Phi}{2}\right) \\ q_3 &\equiv e_z \sin\left(\frac{\Phi}{2}\right) \\ q_4 &\equiv \cos\left(\frac{\Phi}{2}\right) \end{aligned} \qquad \textbf{(Eq. 2.2)}$$

where,

$\mathbf{q} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix}$ = the quaternion vector

$\mathbf{e} = \begin{bmatrix} e_x \\ e_y \\ e_z \end{bmatrix}$ = the unit Euler axis vector in orbit coordinates

$\mathbf{\Phi}$ = the rotation angle about the Euler axis

The quaternion components are not independent, but satisfy the following constraint

$$q_1^2 + q_2^2 + q_2^3 + q_4^2 = 1 \qquad \textbf{(Eq. 2.3)}$$

The direction cosine matrix can be expressed in terms of the Euler symmetric parameters as

$$\mathbf{A} = \begin{bmatrix} q_1^2 - q_2^2 - q_3^2 + q_4^2 & 2(q_1q_2 + q_3q_4) & 2(q_1q_3 - q_2q_4) \\ 2(q_1q_2 - q_3q_4) & -q_1^2 + q_2^2 - q_3^2 + q_4^2 & 2(q_2q_3 + q_1q_4) \\ 2(q_1q_3 + q_2q_4) & 2(q_2q_3 - q_1q_4) & -q_1^2 - q_2^2 + q_3^2 + q_4^2 \end{bmatrix} \quad \textbf{(Eq. 2.4)}$$

In reverse, the quaternion components can be calculated from a given direction cosine matrix by

$$\begin{aligned} q_4 &= \frac{1}{2}(1 + A_{11} - A_{22} - A_{33})^{1/2} \\ q_1 &= \frac{1}{4q_4}(A_{23} - A_{32}) \\ q_2 &= \frac{1}{4q_4}(A_{31} - A_{13}) \\ q_3 &= \frac{1}{4q_4}(A_{12} - A_{21}) \end{aligned} \quad \textbf{(Eq. 2.5)}$$

Euler angles can be expressed in terms of quaternion components by

$$\begin{aligned} \phi &= \arctan 4\left(\frac{-A_{32}}{A_{33}}\right) \\ \theta &= \arcsin(A_{31}) \\ \psi &= \arctan 4\left(\frac{-A_{21}}{A_{11}}\right) \end{aligned}$$

Inertial Coordinates

This coordinate system is used as the reference frame for the motion of the spacecraft in inertial space. *Celestial coordinates*, an inertial system defined relative to the rotation axis of the earth, is the most commonly used system. The celestial z_c-axis is parallel to the rotation axis of the earth and taken as positive in the direction of the geometric north pole. The celestial x_c-axis is parallel to the line connecting the centre of the earth and the *vernal equinox*[2]. The y_c-axis completes the orthogonal set.

The spacecraft-centred inertial coordinate system for SUNSAT, as defined by Steyn, differs from celestial coordinates and is shown in Figure 3. These inertial coordinates coincide precisely with the orbit-defined coordinates at perigee. Since the orbital plane of the spacecraft experiences a slow precession, this inertial coordinate system

[2] The vernal equinox is the point where the *ecliptic*, or plane of the earth's orbit around the sun, crosses the equator going from South to North.

is not strictly inertial. However, since this precession is slow enough, it has a negligible effect on the dynamics of SUNSAT.

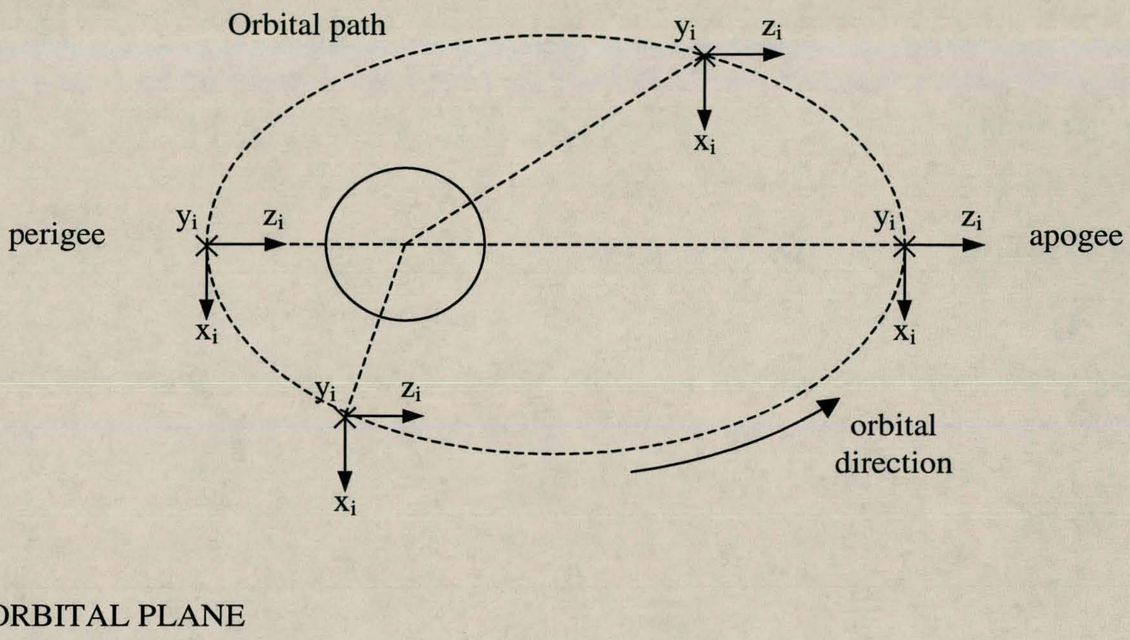


Figure 3: The spacecraft-centred inertial coordinates system for SUNSAT.

The following matrix is used to do an Euler 1-2-3 transformation from inertial coordinates to orbit coordinates.

$$\begin{bmatrix} x_o \\ y_o \\ z_o \end{bmatrix} = \begin{bmatrix} \cos\nu(t) & 0 & \sin\nu(t) \\ 0 & 1 & 0 \\ -\sin\nu(t) & 0 & \cos\nu(t) \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ z_i \end{bmatrix} \qquad \textbf{(Eq. 2.6)}$$

where,

$\nu(t)$	= the true anomaly
t	= time

2.3.2 NONSPACECRAFT-CENTRED COORDINATES

Nonspacecraft-centred coordinate systems are used for obtaining reference vectors such as magnetic field vectors or position vectors to objects seen by the spacecraft. Wertz lists a number of these coordinate systems, i.e. *geocentric* (centred on the earth), *heliocentric* (centred on the sun) and *selenocentric* (centred on the moon). For the purposes of SUNSAT, the geocentric inertial system will be of most importance.

Geocentric Inertial Coordinates

The geocentric inertial coordinate system is equivalent to celestial coordinates, except that the centre of the coordinate system coincides with the centre of the earth. The coordinate system is depicted in Figure 4.

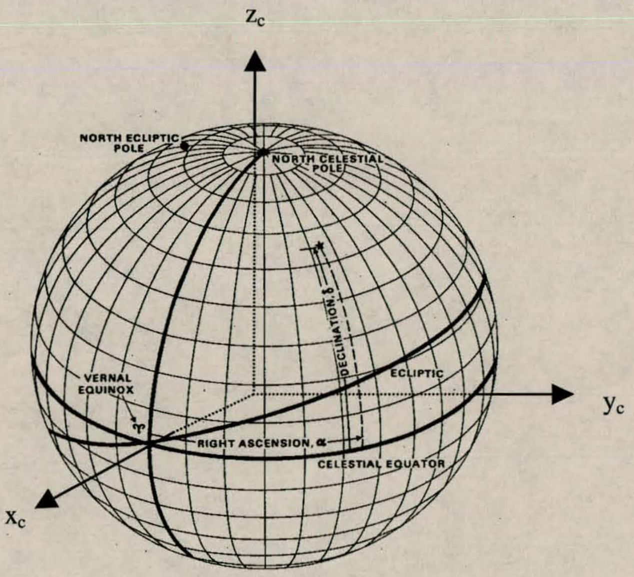


Figure 4: The geocentric inertial coordinates system.

2.4 Equations of motion

The motion of the spacecraft is governed by two sets of equations: the *dynamic equations of motion* and the *kinematic equations of motion*. Dynamics relates the time derivative of the angular momentum of the spacecraft to the applied torque, while kinematics is concerned with spacecraft motion irrespective of the forces which bring about the motion.

2.4.1 EULER DYNAMICS

The dynamics of the spacecraft in inertial space is governed by euler's equations of motion. These differential equations can be expressed in vector form as:

$$\mathbf{I}\dot{\boldsymbol{\omega}}_B^I = \mathbf{N}_{GG} + \mathbf{N}_M + \mathbf{N}_D - \boldsymbol{\omega}_B^I \times (\mathbf{I}\boldsymbol{\omega}_B^I + \mathbf{h}) - \dot{\mathbf{h}} \qquad \textbf{(Eq. 2.7)}$$

where,

$\mathbf{I} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix}$ = moment of inertia tensor in body coordinates

$\boldsymbol{\omega}_B^I = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$ = body angular rate vector in inertial coordinates

$\mathbf{N}_{GG}$ = gravity gradient torque vector in body coordinates

$\mathbf{N}_{M}$ = magnetic torque vector in body coordinates

$\mathbf{N}_{D}$ = external disturbance torque vector in body coordinates

$\mathbf{h} = \begin{bmatrix} h_x \\ h_y \\ h_z \end{bmatrix}$ = reaction wheel angular momentum vector in body coordinates

and,

$$\mathbf{N}_{D} = \mathbf{N}_{AERO} + \mathbf{N}_{SOLAR} \qquad \textbf{(Eq. 2.8)}$$

with,

$\mathbf{N}_{AERO}$ = aerodynamic disturbance torque

$\mathbf{N}_{SOLAR}$ = solar radiation disturbance torque

2.4.2 QUATERNION KINEMATICS

The kinematics of the spacecraft in orbit-defined coordinates is best described by the quaternion formulation of spacecraft attitude since it is well suited for the numerical integration of the equations of motion. These kinematic equations of motion can be expressed in vector form by

$$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{\Omega q} \qquad \textbf{(Eq. 2.9)}$$

with,

$$\mathbf{\Omega} = \begin{bmatrix} 0 & \omega_{oz} & -\omega_{oy} & \omega_{ox} \\ -\omega_{oz} & 0 & \omega_{ox} & \omega_{oy} \\ \omega_{oy} & -\omega_{ox} & 0 & \omega_{oz} \\ -\omega_{ox} & -\omega_{oy} & -\omega_{oz} & 0 \end{bmatrix} \qquad \textbf{(Eq. 2.10)}$$

and,

$$\omega_B^O = \begin{bmatrix} \omega_{ox} \\ \omega_{oy} \\ \omega_{oz} \end{bmatrix} \quad = \text{body angular rate vector in orbit coordinates}$$

The body angular rate vector in orbit coordinates can be obtained from the angular rate vector in inertial coordinates by

$$\omega_B^O = \omega_B^I + \mathbf{A}\begin{bmatrix} 0 \\ \tilde{\omega}_o(t) \\ 0 \end{bmatrix} \qquad \textbf{(Eq. 2.11)}$$

with,

$$\tilde{\omega}_o(t) \approx \omega_o\{1 + 2e\cos(\omega_o t + M_0)\} \quad \text{for small eccentricity, } e \qquad \textbf{(Eq. 2.12)}$$

where,

$\tilde{\omega}_o(t)$	= true orbit angular rate
ω_o	= orbit mean motion
M_0	= orbit mean anomaly at epoch
e	= orbit eccentricity

2.5 External disturbance torques

2.5.1 GRAVITY GRADIENT TORQUE

The gravity gradient torque tends to keep the satellite nadir pointing and is expressed in vector form by

$$\mathbf{N}_{GG} = \frac{3GM_{\oplus}}{R_s^3}\left[I_{zz} - \frac{I_{xx}+I_{yy}}{2}\right](\mathbf{z}_o \cdot \mathbf{z})(\mathbf{z}_o \times \mathbf{z}) \qquad \textbf{(Eq. 2.13)}$$

where,

$GM_{\oplus}$	= earth's gravitation constant
R_s	= geocentric spacecraft position vector length
$\mathbf{z}_o = \begin{bmatrix} A_{13} \\ A_{23} \\ A_{33} \end{bmatrix}$	= nadir pointing unit vector in body coordinates
$\mathbf{z} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$	= z-axis unit vector in body coordinates

2.5.2 AERODYNAMIC TORQUE

The aerodynamic torque vector is expressed as

$$\mathbf{N}_{AERO} = \rho_a V^2 A_p (\mathbf{c}_p \times \mathbf{V}) \qquad \textbf{(Eq. 2.14)}$$

where,

ρ_a	= atmospheric density
V	= magnitude of spacecraft velocity vector
$\mathbf{V}$	= unit spacecraft velocity vector
A	= total projected area of spacecraft
$\mathbf{c}_p$	= vector between centre of mass and centre of pressure

2.5.3 SOLAR RADIATION TORQUE

The solar radiation torque is expressed as

$$\mathbf{N}_{SOLAR} = \frac{d_o}{c} V^2 A_p (\mathbf{c}_p \times \mathbf{V}) \qquad \textbf{(Eq. 2.15)}$$

where,

d_o	= average solar radiation constant
c	= velocity of light

2.6 Sensors

2.6.1 MAGNETOMETER

The magnetometer measures the geomagnetic field vector at the location of the satellite. The axes of the magnetometer are aligned with the satellite body axes and so the measured geomagnetic field vector is in body coordinates.

The following mathematical model is used to simulate the magnetometer measurement :

(1) The sub-satellite latitude, longitude and geocentric distance of the satellite is obtained from a suitable orbit propagator, e.g. the SGP4 propagator.

(2) The geomagnetic field vector in local tangent coordinates is calculated with a tenth order IGRF model, as follows

$$
\begin{aligned}
B_r &= \sum_{n=1}^{k} \left(\frac{a}{r}\right)^{n+2} (n+1) \sum_{m=0}^{n} \left(g_{n,m} \cos m\phi + h_{n,m} \sin m\phi\right) P_{n,m}(\theta) \\
B_\theta &= -\sum_{n=1}^{k} \left(\frac{a}{r}\right)^{n+2} \sum_{m=0}^{n} \left(g_{n,m} \cos m\phi + h_{n,m} \sin m\phi\right) \frac{\partial P_{n,m}(\theta)}{\partial \theta} \\
B_\phi &= \frac{-1}{\sin\theta} \sum_{n=1}^{k} \left(\frac{a}{r}\right)^{n+2} \sum_{m=0}^{n} m\left(-g_{n,m} \cos m\phi + h_{n,m} \sin m\phi\right) P_{n,m}(\theta)
\end{aligned}
\qquad \textbf{(Eq. 2.16)}
$$

where,

B_r	= radial component of field (outward positive)
B_θ	= coelevation component of field (South positive)
B_ϕ	= azimuthal component of field (East positive)
a	= equatorial radius of the Earth
r	= geocentric distance
θ	= coelevation
ϕ	= East longitude from Greenwich
$g_{n,m}$, $h_{n,m}$	= Gauss normalised IGRF coefficients

The Gauss functions $P_{n,m}$ are obtained from the following recursion relations:

$$\begin{aligned} P_{0.0} &= 1 \\ P_{n.n} &= \sin\theta P_{n-1,n-1} \\ P_{n,m} &= \cos\theta P_{n-1,m} - K_{n,m} P_{n-2,m} \end{aligned} \qquad \textbf{(Eq. 2.17)}$$

where,

$$K_{n,m} \equiv \frac{(n-1)^2 - m^2}{(2n-1)(2n-3)} \qquad n > 1$$

$$K_{n,m} \equiv 0 \qquad n = 1 \qquad \textbf{(Eq. 2.18)}$$

The partial derivatives of the Gauss functions are obtained recursively from

$$\frac{\partial P_{0.0}}{\partial\theta} = 0$$

$$\frac{\partial P_{n.n}}{\partial\theta} = (\sin\theta)\frac{\partial P_{n-1.n-1}}{\partial\theta} + (\cos\theta)P_{n-1.n-1} \qquad n \geq 1 \qquad \textbf{(Eq. 2.19)}$$

$$\frac{\partial P_{n,m}}{\partial\theta} = (\cos\theta)\frac{\partial P_{n-1,m}}{\partial\theta} - (\sin\theta)P_{n-1,m} - K_{n,m}\frac{\partial P_{n-2,m}}{\partial\theta}$$

(3) The geomagnetic field vector, which has been obtained in local tangent coordinates, must now be converted to body coordinates to represent the magnetometer measurement. The conversion from local tangent coordinates to body coordinates consists of the following sequence of coordinate transforms:

(a) local tangent coordinates to celestial coordinates:

$$\begin{aligned} B_{cx} &= (B_r\cos\delta + B_\theta\sin\delta)\cos\alpha - B_\phi\sin\alpha \\ B_{cy} &= (B_r\cos\delta + B_\theta\sin\delta)\sin\alpha - B_\phi\cos\alpha \\ B_{cz} &= (B_r\sin\delta - B_\theta\cos\delta) \end{aligned} \qquad \textbf{(Eq. 2.20)}$$

where,

B_{cx}, B_{cy}, B_{cz}	= components of the geomagnetic field vector in celestial coordinates
δ	= declination
α	= right ascension

The declination, δ, and right ascension, α, are obtained from the coelevation, θ, and East longitude from Greenwich, φ, with

$$\begin{aligned} \delta &\equiv 90^\circ - \theta \\ \alpha &= \phi + \alpha_G \end{aligned} \qquad \textbf{(Eq. 2.21)}$$

where,

α_G	= right ascension of the Greenwich meridian

(b) celestial coordinates to orbit coordinates:

The celestial coordinates are converted to orbit coordinates with an Euler angle rotation which consists of four rotations, instead of the standard three. The transformation, which was used by Jacobs, will be called a 3-2-1-2 Euler angle rotation and is represented by

$$\begin{bmatrix} B_{ox} \\ B_{oy} \\ B_{oz} \end{bmatrix} = \begin{bmatrix} C(-i)C(\Omega')C(-\omega') + S(\Omega')S(-\omega') & C(-i)C(\Omega')C(-\omega') - C(\Omega')S(-\omega') & -S(-i)C(-\omega') \\ -S(-i)C(\Omega') & -S(-i)S(\Omega') & -C(-i) \\ C(-i)C(\Omega')S(-\omega') - S(\Omega')C(-\omega') & C(-i)S(\Omega')S(-\omega') + C(\Omega')C(-\omega') & -S(-i)S(-\omega') \end{bmatrix} \begin{bmatrix} B_{cx} \\ B_{cy} \\ B_{cz} \end{bmatrix}$$

(Eq. 2.22)

with,

$$\begin{aligned} \Omega' &= 90^\circ + \Omega \\ \omega' &= \omega + \nu \end{aligned} \qquad \textbf{(Eq. 2.23)}$$

where,

B_{ox}, B_{oy}, B_{oz}	= components of the geomagnetic field vector in orbit coordinates
i	= orbit inclination
Ω	= orbit right ascension of the ascending node
ω	= orbit argument of perigee
ν	= orbit true anomaly

(c) and orbit coordinates to body coordinates:

The geomagnetic field vector is converted from orbit coordinates to body coordinates with the direction cosine matrix.

$$\begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} = \begin{bmatrix} C\psi C\theta & C\psi S\theta S\phi + S\psi C\phi & -C\psi S\theta C\phi + S\psi S\phi \\ -S\psi C\theta & -S\psi S\theta S\phi + C\psi C\phi & S\psi S\theta C\phi + C\psi S\phi \\ S\theta & -C\theta S\phi & C\theta C\phi \end{bmatrix} \begin{bmatrix} B_{ox} \\ B_{oy} \\ B_{oz} \end{bmatrix} \qquad \textbf{(Eq. 2.24)}$$

2.6.2 SUN CELL SENSORS

The sun cell sensors are six sun cells, one on each side of the cube that is the satellite body. Each sun cell sensor is named after the body axis which is normal to it and so the names of the sun cell sensors are -X, X, -Y, Y, -Z and Z. A single sun cell sensor is modelled as a dependent current source which delivers a dc current proportional to its illumination. The illumination that each of the sun cell sensors receives is a function of the projection of the sun vector[3] onto that sun cell sensor's body axis.

The following mathematical model is used to simulate the sun cell sensors:

(1) The sub-satellite and sub-sun latitudes and longitudes are obtained from a suitable orbit propagator and sun position propagator. A delayed sub-satellite latitude and longitude is also obtained.
(2) The following three distance angles are calculated using spherical geometry : the distance angle between the sub-satellite point and the sub-sun point, β, the distance angle between the sub-satellite point and the delayed sub-satellite point, θ, and the distance angle between the delayed sub-satellite point and the sub-sub point, ε.

$$\begin{aligned}
\cos(\beta) &= \sin(Lat)\sin(SunLat) \\
&\quad + \cos(Long)\cos(SunLong)\cos(Long - SunLong), \quad 0 \le \beta \le \pi \\[1em]
\cos(\theta) &= \sin(Lat)\sin(DelayedLat) \\
&\quad + \cos(Long)\cos(DelayedLong)\cos(Long - DelayedLong), \quad 0 \le \theta \le \pi \\[1em]
\cos(\varepsilon) &= \sin(DelayedLat)\sin(SunLat) \\
&\quad + \cos(DelayedLong)\cos(SunLong)\cos(DelayedLong - SunLong), \quad 0 \le \varepsilon \le \pi
\end{aligned}$$

(Eq. 2.25)

(3) The angle, Ω', between the orbit plane and the sub-sun point is calculated from these three distance angles, also using spherical geometry.

$$\cos(\Omega') = \frac{\cos(\varepsilon) - \cos(\theta)\cos(\beta)}{\sin(\theta)\sin(\beta)}, \quad 0 \le \Omega' \le \pi$$ **(Eq. 2.26)**

(4) The sun azimuth, Ω, in orbit coordinates is obtained from the angle between the orbit plane and the sub-sub point using the following rules:

[3] The unit vector in body coordinates which points from the satellite position to the position of the sun.

Normally,

$$\Omega = \pi - \Omega' \qquad \textbf{(Eq. 2.27)}$$

If however, the sub-satellite latitude is very close to the sub-sun latitude, then the following special rules are used:

(a) Calculate the difference between the sub-sun longitude and sub-satellite longitude. The longitude difference must be in the range $(-\pi, \pi)$.

(b) If the satellite is ascending (the sub-satellite latitude is greater than the previous sub-satellite latitude) then

$$\Omega = \begin{cases} \pi - \Omega' & \textit{for longitude difference} > 0 \\ -\pi + \Omega' & \textit{for longitude difference} < 0 \end{cases} \qquad \textbf{(Eq. 2.28)}$$

otherwise, if the satellite is descending (the sub-satellite latitude is lesser than the previous sub-satellite) then

$$\Omega = \begin{cases} \pi - \Omega' & \textit{for longitude difference} < 0 \\ -\pi + \Omega' & \textit{for longitude difference} > 0 \end{cases} \qquad \textbf{(Eq. 2.29)}$$

(5) The sun elevation in orbit coordinates is related to the distance angle between the sub-satellite and sub-sun point by

$$Elevation = -(\frac{\pi}{2} - \beta) \qquad \textbf{(Eq. 2.30)}$$

(6) The sun vector in orbit coordinates can now be obtained from the sun azimuth and elevation.

$$\begin{aligned} S_{ox} &= \sin(\beta)\cos(\Omega) \\ S_{oy} &= \sin(\beta)\sin(\Omega) \\ S_{oz} &= -\cos(\beta) \end{aligned} \qquad \textbf{(Eq. 2.31)}$$

This sun vector is only valid if the sub-satellite point falls within the footprint of the sun. The satellite is considered to be illuminated by the sun when the angle between the sub-satellite point and the sub-sun point is smaller than $110°$.

$$\beta < 110° \qquad \textbf{(Eq. 2.32)}$$

(7) The sun vector is then transformed from orbit coordinates to body coordinates using the direction cosine matrix.

$$\begin{bmatrix} S_x \\ S_y \\ S_z \end{bmatrix} = \begin{bmatrix} C\psi C\theta & C\psi S\theta S\phi + S\psi C\phi & -C\psi S\theta C\phi + S\psi S\phi \\ -S\psi C\theta & -S\psi S\theta S\phi + C\psi C\phi & S\psi S\theta C\phi + C\psi S\phi \\ S\theta & -C\theta S\phi & C\theta C\phi \end{bmatrix} \begin{bmatrix} S_{ox} \\ S_{oy} \\ S_{oz} \end{bmatrix} \qquad \textbf{(Eq. 2.33)}$$

(8) If the sun vector is valid, then the currents measured by the sun cell sensors are given by

$$SSV1-X = \begin{cases} -S_x K_{suncell}, & for \quad S_x < 0 \\ 0, & for \quad S_x \geq 0 \end{cases} \qquad SSV5+X = \begin{cases} 0, & for \quad S_x < 0 \\ S_x K_{suncell}, & for \quad S_x \geq 0 \end{cases}$$

$$SSV2-Y = \begin{cases} -S_y K_{suncell}, & for \quad S_y < 0 \\ 0, & for \quad S_y \geq 0 \end{cases} \qquad SSV4+Y = \begin{cases} 0, & for \quad S_y < 0 \\ S_y K_{suncell}, & for \quad S_y \geq 0 \end{cases}$$

$$SSV3-Z = \begin{cases} -S_z K_{suncell}, & for \quad S_z < 0 \\ 0, & for \quad S_z \geq 0 \end{cases} \qquad SSV6+Z = \begin{cases} 0, & for \quad S_z < 0 \\ S_z K_{suncell}, & for \quad S_z \geq 0 \end{cases}$$

(Eq. 2.34)

2.6.3 SUN SENSOR

The following mathematical model for the fine sun sensor uses variables already calculated by the sun cell sensor model.

(1) The azimuth and elevation of the sun vector in body coordinates is calculated with

$$Azimuth = \arctan 4(S_y, S_x)$$
$$Elevation = \arctan\left(\frac{S_z}{\sqrt{S_x^2 + S_y^2}} \right) \qquad \textbf{(Eq. 2.35)}$$

(2) The azimuth angle measured by the sun sensor is calculated with

$$\theta_{sun} = \pi / 2 + Azimuth \qquad \textbf{(Eq. 2.36)}$$

This measurement is only valid if the sun vector is valid,

$$\beta < 110^\circ \tag{Eq. 2.37}$$

and lies within the field of view of the sun sensor

$$\begin{aligned} -140^\circ &< Azimuth < 40^\circ \\ -60^\circ &< Elevation < 60^\circ \end{aligned} \tag{Eq. 2.38}$$

2.6.4 HORISON SENSORS

The following mathematical model is used for the X and –Y horison sensors:

1. Determine the field of view of the horison sensor in body coordinates and transform it to orbit coordinates. The field of view of the horison sensor is described by two vectors, namely the minimum and maximum field of view vectors which are defined as follows:

X-Horison Sensor:

$$\mathbf{v}_{\min} = \begin{bmatrix} \cos(Minimum \quad Elevation) \\ 0 \\ \sin(Minimum \quad Elevation) \end{bmatrix} \quad \mathbf{v}_{\max} = \begin{bmatrix} \cos(Maximum \quad Elevation) \\ 0 \\ \sin(Maximum \quad Elevation) \end{bmatrix} \tag{Eq. 2.39}$$

-Y Horison Sensor:

$$\mathbf{v}_{\min} = \begin{bmatrix} 0 \\ -\cos(Minimum \quad Elevation) \\ \sin(Minimum \quad Elevation) \end{bmatrix} \quad \mathbf{v}_{\max} = \begin{bmatrix} 0 \\ -\cos(Maximum \quad Elevation) \\ \sin(Maximum \quad Elevation) \end{bmatrix} \tag{Eq. 2.40}$$

The minimum and maximum field of view vectors in body coordinates are transformed to orbit coordinates using the transpose of the direction cosine matrix to represent the field of view of the horison sensor in orbit coordinates:

$$\begin{bmatrix} v_{ox} \\ v_{oy} \\ v_{oz} \end{bmatrix} = \begin{bmatrix} C\psi C\theta & C\psi S\theta S\phi + S\psi C\phi & -C\psi S\theta C\phi + S\psi S\phi \\ -S\psi C\theta & -S\psi S\theta S\phi + C\psi C\phi & S\psi S\theta C\phi + C\psi S\phi \\ S\theta & -C\theta S\phi & C\theta C\phi \end{bmatrix}^T \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \quad \textbf{(Eq. 2.41)}$$

$$\begin{aligned} (\mathbf{v}_o)_{\min} &= \mathbf{A}^T(\mathbf{v}_{\min}) \\ (\mathbf{v}_o)_{\max} &= \mathbf{A}^T(\mathbf{v}_{\max}) \end{aligned} \quad \textbf{(Eq. 2.42)}$$

From the minimum and maximum field of view vectors in orbit coordinates, the minimum and maximum elevation and azimuth of the field of view in orbit coordinates can be obtained

$$\begin{aligned} Min \quad FOV \quad Azimuth \quad (Orbit) &= \arctan 4\left(\frac{(v_{oy})_{\min}}{(v_{ox})_{\min}}\right) \\ Max \quad FOV \quad Azimuth \quad (Orbit) &= \arctan 4\left(\frac{(v_{oy})_{\max}}{(v_{ox})_{\max}}\right) \\ Min \quad FOV \quad Elevation \quad (Orbit) &= \arctan\left(\frac{(v_{oz})_{\min}}{\sqrt{(v_{ox})^2_{\min} + (v_{oy})^2_{\min}}}\right) \\ Max \quad FOV \quad Elevation \quad (Orbit) &= \arctan\left(\frac{(v_{oz})_{\max}}{\sqrt{(v_{ox})^2_{\max} + (v_{oy})^2_{\max}}}\right) \end{aligned} \quad \textbf{(Eq. 2.43)}$$

2. The azimuth of the horison sensor boresight in orbit coordinate is then equal to the average of the minimum and maximum azimuths of the field of view in orbit coordinates.

$$Horison \quad Sensor \quad Azimuth \quad (Orbit) = \frac{Min \quad FOV \quad Azimuth \quad (Orbit) + Max \quad FOV \quad Azimuth \quad (Orbit)}{2}$$

(Eq. 2.44)

3. The equation which computes the elevation of the horison in orbit coordinates, at the azimuth of the horison sensor boresight, requires the azimuth as measured from the East direction, instead of from the X_o axis. Therefore it is necessary to calculate the angle between the X_o axis and the East direction. This angle is obtained from the geomagnetic field vector in local tangent coordinates, $B_{r\theta\phi}$, and the geomagnetic field vector in orbit coordinates, B_o, in the following manner

$$\text{Angle between Orbit } X_o \text{ and East} = \theta - \phi \qquad \textbf{(Eq. 2.45)}$$

with,

$$\theta = \arctan 4\left(\frac{-B_\theta}{B_\phi}\right)$$
$$\phi = \arctan 4\left(\frac{B_{oy}}{B_{ox}}\right) \qquad \textbf{(Eq. 2.46)}$$

4. The elevation of the horison in orbit coordinates, at the azimuth of the horison sensor boresight, will now be computed from the earth oblateness model.

$$Horison \quad Elevation \quad (Orbit) \; =$$
$$\pi/2 - arc\cot\left\{\begin{array}{l}\sqrt{\dfrac{R_s^2 - R_e^2}{a^2}\left[1 + \dfrac{(2-f)fR_e^2\cos^2(Lat)}{(1-f)^2 a^2}\sin^2\Psi\right]} \\ + \dfrac{(2-f)fR_e^2\sin(2Lat)}{2(1-f)^2 a^2}\sin\Psi\end{array}\right\} \qquad \textbf{(Eq. 2.47)}$$

with,

$$R_e = \frac{a(1-f)}{\sqrt{1-(2-f)f\cos^2(Lat)}} \qquad \textbf{(Eq. 2.48)}$$

$$\Psi = Horison \quad Azimuth \quad (Orbit) + Angle \quad between \quad orbit \quad X_o \quad and \quad East \qquad \textbf{(Eq. 2.49)}$$

where,

R_s = the geocentric distance of the satellite
R_e = the oblate earth radius at latitude *Lat*
a = the earth's equatorial radius, 6378.14km
f = the earth flattening factor, 0.00335281
Ψ = the azimuth of the horison vector in the orbit X_oY_o plane, measured from the east direction

5. This elevation is only valid if it falls between the minimum and maximum elevation of the horison sensor field of view

Min FOV Elevation (Orbit)<Horison Elevation (Orbit)<Max FOV Elevation (Orbit)

and the horison is also illuminated. The horison is illuminated if the distance angle, α, between the sub-sun point and the horison at the horison sensor boresight azimuth is smaller than 85°. The angle α is calculated with

$$\cos(\alpha) = \cos(\beta)\cos(\gamma) + \sin(\beta)\sin(\gamma)\cos(\Phi) \qquad \textbf{(Eq. 2.50)}$$

where,

α	= the distance angle between the sub-sun point and the horison at the horison sensor boresight azimuth
β	= the distance angle between the sub-satellite point and the sub-sun point
γ	= the distance angle between the sub-satellite point and the horison at the horison sensor boresight azimuth, (nominal = 25°).
Φ	= the angle between the sub-sun point azimuth and the horison sensor azimuth.

2.6.5 STAR SENSOR

The star sensor uses a pixel matrix CCD to take an image of the star field. The star field is represented by the following mathematical model:

The positions and magnitudes of all stars within a suitable range of magnitudes are recorded in a star catalogue. The position of each star in the catalogue is represented by its right ascension and declination in the geocentric inertial coordinate system. From the right ascension and declination, a unit star vector in geocentric inertial coordinates pointing from the geocentre in the direction of the star may be derived.

$$\begin{aligned} T_{gix} &= \cos(\delta)\cos(\alpha) \\ T_{giy} &= \cos(\delta)\sin(\alpha) \\ T_{giz} &= \sin(\delta) \end{aligned} \qquad \textbf{(Eq. 2.51)}$$

where,

T_{gix}, T_{giy}, T_{giz}	= components of the star vector in geocentric inertial coordinates
δ	= the right ascension of the star
α	= the declination of the star.

The geocentric inertial star vector may be converted to a orbit star vector using the 3-2-1-2 Euler angle rotation used by Jacobs. Although the origins of the geocentric inertial and orbit coordinate systems do not coincide, the translation is neglected since parallax of even the closest stars, due to the orbit motion of the satellite, is negligible.

$$\begin{bmatrix} T_{ox} \\ T_{oy} \\ T_{oz} \end{bmatrix} = \begin{bmatrix} C(-i)C(\Omega')C(-\omega')+S(\Omega')S(-\omega') & C(-i)C(\Omega')C(-\omega')-C(\Omega')S(-\omega') & -S(-i)C(-\omega') \\ -S(-i)C(\Omega') & -S(-i)S(\Omega') & -C(-i) \\ C(-i)C(\Omega')S(-\omega')-S(\Omega')C(-\omega') & C(-i)S(\Omega')S(-\omega')+C(\Omega')C(-\omega') & -S(-i)S(-\omega') \end{bmatrix} \begin{bmatrix} T_{gix} \\ T_{giy} \\ T_{giz} \end{bmatrix}$$

(Eq. 2.52)

with,

$$\Omega' = 90° + \Omega$$
$$\omega' = \omega + \nu$$

(Eq. 2.53)

where,

T_{ox}, T_{oy}, T_{oz}	= components of the star vector in orbit coordinates
i	= orbit inclination
Ω	= orbit right ascension of the ascending node
ω	= orbit argument of perigee
ν	= orbit true anomaly

The orbit star vector may then be converted to a body star vector using the direction cosine matrix.

$$\begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} = \begin{bmatrix} C\psi C\theta & C\psi S\theta S\phi + S\psi C\phi & -C\psi S\theta C\phi + S\psi S\phi \\ -S\psi C\theta & -S\psi S\theta S\phi + C\psi C\phi & S\psi S\theta C\phi + C\psi S\phi \\ S\theta & -C\theta S\phi & C\theta C\phi \end{bmatrix} \begin{bmatrix} T_{ox} \\ T_{oy} \\ T_{oz} \end{bmatrix} \quad \textbf{(Eq. 2.54)}$$

The equations which have been presented up to this point are sufficient for describing the star field. Next, the mathematical model for the star sensor itself is required. Since the software for the star sensor is very complex and too large in scope to be included in this thesis, only a abbreviated description of the star sensor model will be presented here. The complete mathematical model of the star sensor may be found in [Jacobs].

First, it must be determined which stars in the star field fall within the field of view of the star sensor. This is done by mapping the field of view of the star sensor in body coordinates to the corresponding region of right ascension and declination in geocentric inertial coordinates and checking which of the stars in the star catalogue have right ascensions and declinations which fall within this region. The star sensor then compares the field of view stars to its on-board star catalogue to match observed stars with reference stars. Up to three of these matching star pairs are then chosen and their position vectors are returned to the ADCS.

2.7 Actuators

2.7.1 MAGNETIC TORQUERS

The magnetic torque vector generated by the magnetic torquers is expressed as

$$\mathbf{N}_M = \mathbf{M} \times \mathbf{B} \qquad \textbf{(Eq. 2.55)}$$

where

$\mathbf{M}$	= magnetic dipole moment vector generated by the 3-axis magnetic torquer coils in body coordinates
$\mathbf{B}$	= geomagnetic field vector in body coordinates

Magnetic dipole moment vector of 3-axis torquer coils

The magnetic dipole moment vector is easily obtained since its *x*, *y* and *z* components are aligned with the *x*, *y* and *z* axes of the body coordinate system. Since the direction of each component is already known, constructing the vector **M** is simply a matter of calculating the magnitude of each component. The magnitude of the magnetic dipole moment generated by a single magnetic torquer coil is expressed as

$$M = nIA \qquad \textbf{(Eq. 2.56)}$$

where,

n	= number of turns per coil
I	= DC current through the coil
A	= effective coil area

Geomagnetic Field Vector in Centre of Magnetic Torquer Pulse

The pulse width modulated technique used to apply the magnetic dipole moment vector to the magnetic torquers is equivalent to feeding a continuous time signal through a zero order hold device. The value for the geomagnetic field vector in body coordinates, **B**, must be the value of the field in the centre of the sampling period. The geomagnetic field vector in the centre of the sampling period is calculated from the geomagnetic field vector at the beginning of the sampling period by

$$\mathbf{B}(kT_s + 0.5T_s) = \mathbf{A}\mathbf{B}(kT_s)$$

with,

$$\mathbf{A} = \begin{bmatrix} \cos\Phi + E_1^2(1-\cos\Phi) & E_1E_2(1-\cos\Phi) + E_3\sin\Phi & E_1E_3(1-\cos\Phi) - E_2\sin\Phi \\ E_1E_2(1-\cos\Phi) - E_3\sin\Phi & \cos\Phi + E_2^2(1-\cos\Phi) & E_2E_3(1-\cos\Phi) + E_1\sin\Phi \\ E_1E_3(1-\cos\Phi) + E_2\sin\Phi & E_2E_3(1-\cos\Phi) - E_1\sin\Phi & \cos\Phi + E_3^2(1-\cos\Phi) \end{bmatrix}$$

(Eq. 2.57)

$$\Phi = \frac{T_s}{2}\sqrt{\omega_{ox}^2 + \omega_{oy}^2 + \omega_{oz}^2} \qquad \textbf{(Eq. 2.58)}$$

$$\mathbf{E} = \begin{bmatrix} E_1 \\ E_2 \\ E_3 \end{bmatrix} = \frac{1}{\sqrt{\omega_{ox}^2 + \omega_{oy}^2 + \omega_{oz}^2}} \begin{bmatrix} \omega_{ox} \\ \omega_{oy} \\ \omega_{oz} \end{bmatrix} \qquad \textbf{(Eq. 2.59)}$$

where,

$\mathbf{B}(kT_s + 0.5T_s)$	=	the geomagnetic field vector in the centre of the sampling period
$\mathbf{B}(kT_s)$	=	the geomagnetic field vector at the beginning of the sampling period
Φ	=	the euler angle
$\mathbf{E}$	=	the euler axis
T_s	=	the sampling period
ω_{ox}, ω_{oy}, ω_{oz}	=	the angular rates in orbit coordinates at the beginning of the sampling period

2.7.2 REACTION WHEELS

The reaction wheel variables which are of importance are the reaction wheel angular momentum vector, $\mathbf{h}$, and its time derivative, $\dot{\mathbf{h}}$, both in body coordinates. The vector rate of change of reaction wheel angular momentum is obtained from

$$\dot{\mathbf{h}} = \mathbf{N}_{wheel} \qquad \textbf{(Eq. 2.60)}$$

where,

$\mathbf{N}_{wheel}$	=	reaction wheel torque in body coordinates

The reaction wheel angular momentum vector follows from time integration of its derivative

$$\mathbf{h} = \int \dot{\mathbf{h}}\,dt \qquad \textbf{(Eq. 2.61)}$$

3. ADCS SOFTWARE SIMULATION MODEL

3.1 Introduction

The previous chapter described the mathematical models for the motion of the satellite and for its space environment. This chapter describes the algorithms which represent the software of the satellite's attitude determination and control system (ADCS). The ADCS software includes on-board orbit propagator and space environment models, sensor validation tests, attitude estimators, magnetic torquer controllers, reaction wheel controllers, boom deployment algorithms and an on-board magnetometer calibration algorithm.

The ADCS software, which was written in the Parallel Rowley Modula programming language for the T800 transputer, was ported to the Delphi 3 programming language so that it could be simulated on a personal computer. This simulation of the ADCS software can be used to evaluate attitude determination and control strategies and to analyse on the ground all sensor data received from the flight model. More specifically, it was used to extract attitude information from whole orbit data and to investigate a possible unmodelled disturbance torque which seemed to invert the satellite within two days if no magnetic torquer control was applied.

3.2 On-Board Orbit Propagator and Space Environment Models

An on-board SGP4 orbit propagator is used to determine the position of the satellite in its orbit. The orbital position of the satellite is required by the on-board space environment models for the geomagnetic field, sun, horison and star field. On-board models of the geomagnetic field, sun, horison and star field are used to calculate reference vectors in orbit coordinates which are used for sensor validation, attitude estimation and magnetometer calibration. The on-board orbit propagator and space environment models use the same mathematical models as in Chapter 2.

3.3 Sensor Validation

The magnetometer is the only sensor which continuously gives valid measurements. The horison sensors, sun sensor and star sensor all have limited fields of view. Additionally, the horison sensors can only detect a horison which is illuminated by the sun and the sun sensor and sun cell sensors can only detect the sun if the earth does not block the line of sight between the satellite and the sun. The star sensor processor checks the validity of measured stars and returns from zero to three valid matching pairs to the ADCS. The horison sensors, sun sensor and sun cell sensors, however, blindly provide measurements every sampling instant, irrespective of their validity. It is up to the ADCS software to decide whether the measurements of these sensors are valid. The current ADCS software checks the validity of the sun sensor and the horison sensors. The sun cell sensors are not validated, since they are not utilised by the current attitude estimators or control algorithms.

3.3.1 SUN SENSOR

The sun sensor is validated by checking if the sun is in the field of view of the sun sensor and if the line of sight between sun and the satellite is not impeded by the earth.

Field Of View Validation

The sun sensor field of view is valid if the azimuth and elevation of the estimated body referenced sun vector lies within the minimum and maximum body referenced azimuth and elevation of the sun sensor.

The sun sensor validation algorithm obtains the orbit referenced sun vector, $\mathbf{S}_o$, from the on-board sun model. An estimated body referenced sun vector, $\hat{\mathbf{S}}$, is then obtained by transforming the modelled orbit referenced sun vector, $\mathbf{S}_o$, with the estimated direction cosine matrix, $\hat{\mathbf{A}}$, as provided by the attitude estimators.

$$\hat{\mathbf{S}} = \hat{\mathbf{A}}\mathbf{S}_o \qquad \textbf{(Eq. 3.1)}$$

The azimuth and elevation of the estimated body referenced sun vector is then calculated.

$$\hat{A}zimuth = \arctan 4(\hat{S}_y, \hat{S}_x)$$
$$\hat{E}levation = \arctan\left(\frac{\hat{S}_z}{\sqrt{\hat{S}_x^2 + \hat{S}_y^2}}\right) \qquad \textbf{(Eq. 3.2)}$$

The field of view of the sun sensor lies from azimuth -140° to -40° and from elevation -60° to 60°. The sun sensor field of view will therefore be valid for

$$\begin{aligned} -140° &< \hat{Azimuth} < -40° \\ -60° &< \hat{Elevation} < 60° \end{aligned} \qquad \textbf{(Eq. 3.3)}$$

Line of Sight Validation

The sun sensor validation algorithm assumes that there is a valid line of sight between the satellite and the sun if the distance angle between the sub-satellite point and the sub-sun point, β, is smaller than 110°.

$$\beta < 110° \qquad \textbf{(Eq. 3.4)}$$

This distance angle is also obtained from the on-board sun model.

3.3.2 HORISON SENSORS

The X and –Y horison sensors are validated individually using the same principles. A horison sensor is valid if the horison falls within its field of view and that horison is also illuminated by the sun.

Field of View Validation

The horison sensor field of view is validated by projecting the body referenced field of view of the horison sensor into orbit coordinates and then checking if the estimated orbit referenced horison vector falls within this field of view.

The body referenced field of view of a horison sensor is defined by two unit vectors specifying the minimum and maximum body referenced horison elevations which it can measure. Both horison sensors are limited to measuring a minimum elevation of 10° and a maximum elevation of 40°. The minimum and maximum field of view vectors of the X horison sensor lie in the body referenced XZ plane and are represented by

$$MinFOV_{XZ,body} = \begin{bmatrix} \cos(10°) \\ 0 \\ \sin(10°) \end{bmatrix} \quad MaxFOV_{XZ.body} = \begin{bmatrix} \cos(40°) \\ 0 \\ \sin(40°) \end{bmatrix} \qquad \textbf{(Eq. 3.5)}$$

The minimum and maximum field of view vectors of the -Y horison sensor lie in the body referenced -YZ plane and are represented by

$$MinFOV_{-YZ,body} = \begin{bmatrix} 0 \\ -\cos(10°) \\ \sin(10°) \end{bmatrix} \qquad MaxFOV_{-YZ,body} = \begin{bmatrix} 0 \\ -\cos(40°) \\ \sin(40°) \end{bmatrix}$$

(Eq. 3.6)

Except for this difference in the body referenced field of view vectors, the validation procedure is the same for both horison sensors. The minimum and maximum field of view vectors are projected into orbit coordinates using the inverse of the estimated direction cosine matrix.

$$\begin{aligned} Min\hat{F}OV_{orbit} &= \hat{\mathbf{A}}^T MinFOV_{body} \\ Max\hat{F}OV_{orbit} &= \hat{\mathbf{A}}^T MaxFOV_{body} \end{aligned} \qquad \textbf{(Eq. 3.7)}$$

The orbit referenced azimuth and elevation of the minimum and maximum projections are then calculated.

$$\begin{aligned} MinFOV\hat{A}zim_{orbit} &= \arctan 2\left(\frac{MinFOV_{oy}}{MinFOV_{ox}}\right) \\ MinFOV\hat{E}lev_{orbit} &= \arctan\left(\frac{MinFOV_{oz}}{\sqrt{MinFOV_{ox}^2 + MinFOV_{oy}^2}}\right) \end{aligned} \qquad \textbf{(Eq. 3.8)}$$

and,

$$\begin{aligned} MaxFOV\hat{A}zim_{orbit} &= \arctan 2\left(\frac{MaxFOV_{oy}}{MaxFOV_{ox}}\right) \\ MaxFOV\hat{E}lev_{orbit} &= \arctan\left(\frac{MaxFOV_{oz}}{\sqrt{MaxFOV_{ox}^2 + MaxFOV_{oy}^2}}\right) \end{aligned} \qquad \textbf{(Eq. 3.9)}$$

The estimated orbit referenced horison vector will now be obtained by calculating the elevation of the horison at the estimated azimuth of the horison sensor boresight. The estimated orbit referenced azimuth of the horison sensor boresight is then taken as the mean value of the minimum and maximum orbit referenced azimuths.

$$Boresight\hat{A}zim_{orbit} = \frac{MinFOV\hat{A}zim_{orbit} + MaxFOV\hat{A}zim_{orbit}}{2} \qquad \textbf{(Eq. 3.10)}$$

The on-board horison model is then used to calculate the expected orbit referenced elevation, $\hat{E}lev_{orbit}$, at the estimated boresight azimuth. The horison sensor field of view is then validated by checking if the expected orbit referenced elevation, $Hor\hat{E}lev_{orbit}$, falls between the orbit referenced elevations of the minimum and maximum field of view vectors.

Field of view valid for:

$$MinFOV\hat{E}lev_{orbit} < Hor\hat{E}lev_{orbit} < MaxFOV\hat{E}lev_{orbit} \qquad \textbf{(Eq. 3.11)}$$

Horison Illumination Validation

A horison sensor can only detect the horison if it is illuminated by the sun. It is assumed that the horison will be illuminated when the distance angle between the sub-sun point and the point on the horison which is being detected by the horison sensor, α, is smaller than 85°.

$$\cos(\alpha) = \cos(\beta)\cos(\Phi) + \sin(\beta)\sin(\Phi)\cos(\gamma) \qquad \textbf{(Eq. 3.12)}$$

where,

α	= the distance angle between the sub-sun point and the horison at the horison sensor boresight azimuth
β	= the distance angle between the sub-satellite point and the sub-sun point
γ	= the angle between the sun azimuth and the horison sensor azimuth
Φ	= the distance angle between the sub-satellite point and the horison at the horison sensor boresight azimuth, (nominal = 25°).

Horison illumination valid for:

$$\alpha < 110°$$

3.4 Attitude Estimation

Six different algorithms are employed on SUNSAT to estimate the attitude of the satellite from sensor data. The attitude estimators used by the ADCS of SUNSAT are listed in Table 1.

Table 1: Summary of the Attitude Estimators used by the ADCS of SUNSAT.

FIL	Name	Sensors	Estimated Variables	Estimation Error	Note
1	EKF Magnetometer	Magnetometer	Inertial Angular Rates, Quaternions, Disturbance Torque	Pitch/Roll < ±1° Yaw < ±3°	1
2	EKF Horison/ Sun Sensors	Magnetometer, Horison Sensors, Sun Sensor	Inertial Angular Rates, Quaternions, Disturbance Torque	Pitch/Roll < ±0.1° Yaw < ±0.3°	1
3	EKF Star Sensor	Magnetometer Star Sensor	Inertial Angular Rates, Quaternions, Disturbance Torque	Pitch/Roll < ±0.01° Yaw < ±0.03°	1
4	Angular Rate Kalman Filter	Magnetometer	Orbit Angular Rates	Orbit Rates < ±0.06°/s	2
5	Y-Estimator	Magnetometer	Pitch Rate, Pitch	Pitch Rate < ±0.01°/s Pitch < ±1°	3
6	Z-Estimator	Magnetometer	Yaw Rate, Yaw	Yaw Rate < ±0.02°/s Pitch < ±2°	4

[1]Filters 1, 2 and 3 are extended Kalman filters which have been linearised for a nadir pointing satellite experiencing small librations. For this reason, the convergence and accuracy of these filters are only guaranteed when the satellite is in this state.

[2]Filter 4 was designed to estimate angular rates before boom deployment, but may be used after boom deployment in case of emergency provided that the satellite is experiencing large librations. For small librations, problems with convergence may occur.

[3]The Y-Estimator is only valid when the components of the satellite's angular rate along the orbit x- and z-axes are small, for example during the pre-boom deployment Y-Thompson spin.

[4]The Z-Estimator is only valid when the components of the satellite's angular rate along the orbit x- and y-axes are small, for example when the satellite is nadir pointing after boom deployment.

3.4.1 FULL STATE EXTENDED KALMAN FILTER (FILTERS 1, 2, 3)

Introduction

The Magnetometer EKF, Horison/Sun Sensor EKF and Star Sensor EKF are Extended Kalman Filters which are used to estimate the inertial angular rates, the quaternion representation of the attitude and optionally the aerodynamic disturbance torque. The estimated euler angles (roll, pitch and yaw) can be obtained from the estimated quaternions and the estimated orbit angular rates from the estimated inertial rates and orbit propagator variables.

The three filters share the same basic Extended Kalman Filter structure which is shown in Figure 5. The main difference between the three filters is the sensors which each one uses for its innovation. The magnetometer continuously supplies measurements throughout the entire orbit of the satellite. The horison sensors, sun sensor and star sensor supply more accurate measurements, but due to field of view and illumination limitations, these measurements are not continuously or even simultaneously available. For this reason, all three EKF's rely on the magnetometer to supply continuous innovations. The Magnetometer EKF uses only the magnetometer and achieves an estimation error which is smaller than ±1° for pitch and roll and smaller than ±3° for yaw. The Horison/Sun Sensor EKF uses the two horison sensors and the sun sensor whenever their measurements are valid and improves its estimation error to smaller than ±0.1° for pitch and roll and smaller than ±0.3° for yaw. The Star Sensor EKF uses the star sensor whenever its measurements are valid, and improves its accuracy even more to obtain an estimation error which is smaller than ±0.01° for pitch and roll and smaller than ±0.03° for yaw.

A brief overview of the differences in the block diagrams of the Magnetometer EKF, the Horison/Sun Sensor EKF and the Star Sensor EKF, with reference to Figure 5, will now be given.

The first obvious difference is in the construction of the sensor measurement vector $\mathbf{v}_{meas,k+1}$. For the Magnetometer EKF, it is constructed from the magnetometer measurement, for the Horison/Sun Sensor EKF, it is constructed from the horison sensors and sun sensor measurements and for the Star Sensor EKF it is constructed from the star sensor measurement.

The sensor model $\mathbf{h}(\hat{\mathbf{x}},k)$ which is used to calculate the estimated sensor measurement vector $\mathbf{v}_{body,k+1/k}$ also differs according to the sensor used. For the Magnetometer EKF, the sensor model is a magnetometer model, for the Horison/Sun Sensor EKF, the sensor models are horison sensor and sun sensor models and for the star sensor EKF the sensor model is a star sensor model.

The process noise experienced by the satellite is the same for all three EKF's, but the measurement noise differs according to the sensor used. To reflect the difference in measurement noise, the three EKF's use different **Q** and **R** matrices for their Kalman Gain Machines and Ricatti Equation Solvers and are also initialised with different perturbation covariance matrices, $\mathbf{P}_{k/k}$.

The last difference concerns the sampling periods of the EKF's. The magnetometer measurements are sampled at a sampling period of ten seconds, while the horison sensors, sun sensor and star sensor measurements are sampled at a sampling period of one second. This leads to different sampling periods for the different EKF's. The Magnetometer EKF runs with a sampling period of ten seconds, while the Horison/Sun Sensor EKF and the Star Sensor EKF both use a sampling period of one second.

Except for the differences mentioned above, the three EKF's are identical.

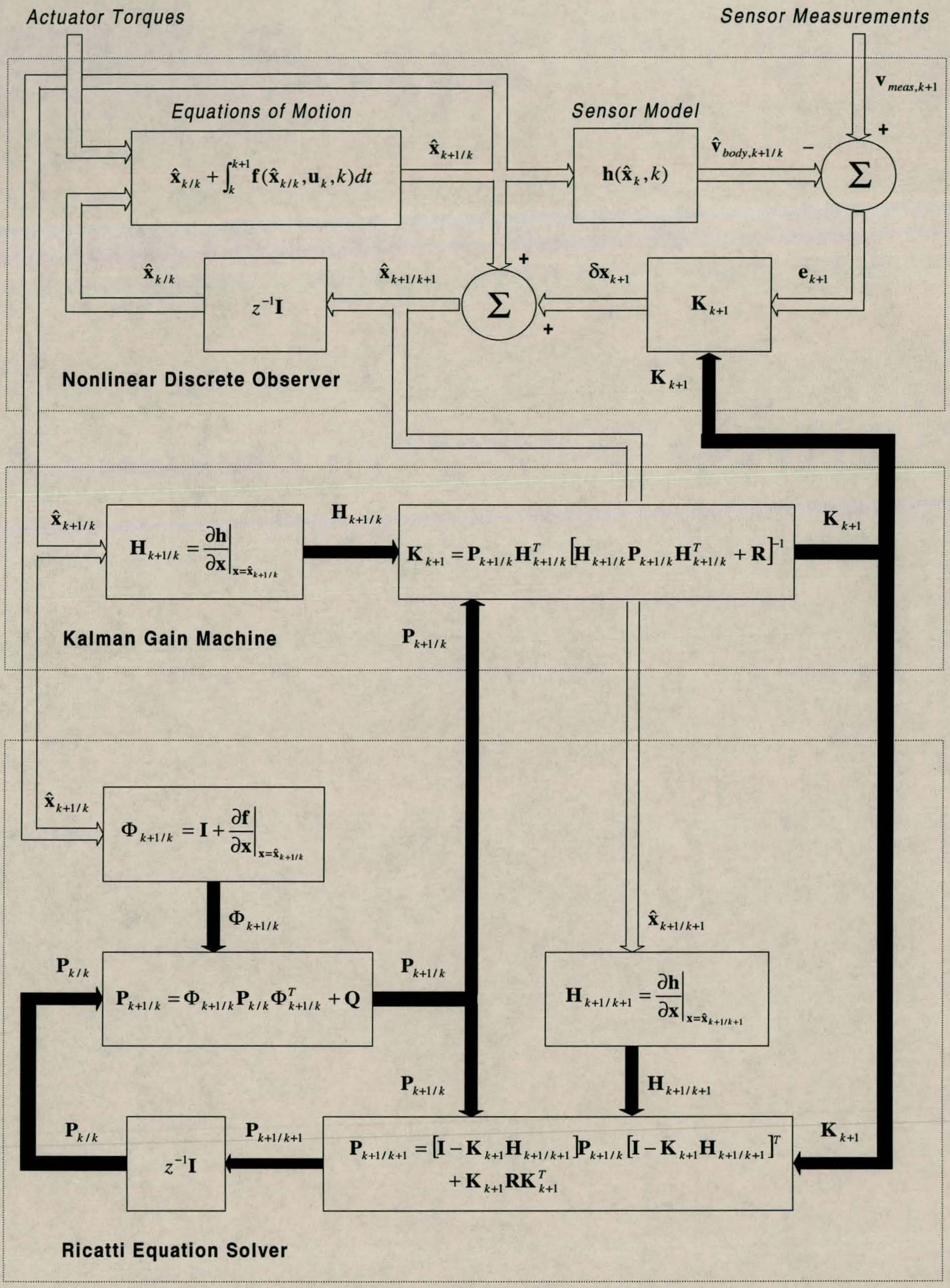

Actuator Torques
Sensor Measurements
Equations of Motion
Sensor Model
$\hat{\mathbf{x}}_{k/k} + \int_k^{k+1} \mathbf{f}(\hat{\mathbf{x}}_{k/k}, \mathbf{u}_k, k)dt$
$\hat{\mathbf{x}}_{k+1/k}$
$\mathbf{h}(\hat{\mathbf{x}}_k, k)$
$\hat{\mathbf{v}}_{body,k+1/k}$
$\mathbf{v}_{meas,k+1}$
$\hat{\mathbf{x}}_{k/k}$
$z^{-1}\mathbf{I}$
$\hat{\mathbf{x}}_{k+1/k+1}$
$\delta\mathbf{x}_{k+1}$
$\mathbf{K}_{k+1}$
$\mathbf{e}_{k+1}$
Nonlinear Discrete Observer
$\mathbf{H}_{k+1/k} = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}_{k+1/k}}$
$\mathbf{H}_{k+1/k}$
$\mathbf{K}_{k+1} = \mathbf{P}_{k+1/k}\mathbf{H}^T_{k+1/k}\left[\mathbf{H}_{k+1/k}\mathbf{P}_{k+1/k}\mathbf{H}^T_{k+1/k} + \mathbf{R}\right]^{-1}$
Kalman Gain Machine
$\mathbf{P}_{k+1/k}$
$\Phi_{k+1/k} = \mathbf{I} + \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}_{k+1/k}}$
$\Phi_{k+1/k}$
$\mathbf{P}_{k/k}$
$\mathbf{P}_{k+1/k} = \Phi_{k+1/k}\mathbf{P}_{k/k}\Phi^T_{k+1/k} + \mathbf{Q}$
$\mathbf{H}_{k+1/k+1} = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}_{k+1/k+1}}$
$\mathbf{H}_{k+1/k+1}$
$\mathbf{P}_{k+1/k+1}$
$\mathbf{P}_{k+1/k+1} = [\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H}_{k+1/k+1}]\mathbf{P}_{k+1/k}[\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H}_{k+1/k+1}]^T + \mathbf{K}_{k+1}\mathbf{R}\mathbf{K}^T_{k+1}$
Ricatti Equation Solver

Figure 5: Block diagram of the Full State Estimation Extended Kalman Filter.

EKF Algorithm

The basic EKF algorithm employed by all three filters will now be presented in detail, but without derivation. A complete derivation can be found in [Steyn].

The continuous full state (8 element) vector to be estimated is:

$$\hat{\mathbf{x}} = \begin{bmatrix} \hat{\omega}_B^I \\ \hat{\mathbf{q}} \\ \hat{n}_{doy} \end{bmatrix} \qquad \textbf{(Eq. 3.13)}$$

Algorithm

1. Propagate the dynamic and kinematic equations of motion (Eq. 2.7) and (Eq. 2.9) by using a numerical integration technique.

$$\hat{\mathbf{x}}_{k+1/k} = \hat{\mathbf{x}}_{k/k} + \int_k^{k+1} \mathbf{f}(\hat{\mathbf{x}}_{k/k}, \mathbf{u}_k, k)dt \qquad \textbf{(Eq. 3.14)}$$

2. Compute the linearised perturbation state matrix $\mathbf{F}(\mathbf{x}(t_{k+1}), t_{k+1})$.

$$\mathbf{F}(\hat{\mathbf{x}}(t_{k+1}), t_{k+1}) = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\mathbf{x}_{k+1/k}} \qquad \textbf{(Eq. 3.15)}$$

3. Obtain the discrete system matrix $\mathbf{\Phi}_{k+1/k}$.

$$\mathbf{\Phi}_{k+1/k} \approx \mathbf{I} + \mathbf{F}(\hat{\mathbf{x}}(t_{k+1}), t_{k+1})T_s + \tfrac{1}{2}\left[\mathbf{F}(\hat{\mathbf{x}}(t_{k+1}), t_{k+1})T_s\right]^2 \qquad \textbf{(Eq. 3.16)}$$

4. Propagate the perturbation covariance matrix, $\mathbf{P}_{k+1/k}$.

$$\mathbf{P}_{k+1/k} = \mathbf{\Phi}_{k+1/k}\mathbf{P}_{k/k}\mathbf{\Phi}_{k+1/k}^T + \mathbf{Q} \qquad \textbf{(Eq. 3.17)}$$

5. Obtain the sensor measurement vector $\mathbf{v}_{meas,k+1}$, and the modelled measurement vector in orbit coordinates, $\mathbf{v}_{orb,k+1}$ for the appropriate sensor.

6. If valid sensor measurements are available, execute steps 7 through 12, otherwise return to step1 and wait for the next sampling instant.

7. Compute the discrete output measurement matrix, $\mathbf{H}_{k+1/k}$.

$$\mathbf{H}_{k+1/k} = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}_{k+1/k}} \qquad \textbf{(Eq. 3.18)}$$

8. Compute the Kalman Filter Gain $\mathbf{K}_{k+1}$.

$$\mathbf{K}_{k+1} = \mathbf{P}_{k+1/k}\mathbf{H}_{k+1/k}^T\left[\mathbf{H}_{k+1/k}\mathbf{P}_{k+1/k}\mathbf{H}_{k+1/k}^T + \mathbf{R}\right]^{-1} \qquad \textbf{(Eq. 3.19)}$$

9. Calculate the innovation error vector, $\mathbf{e}_{k+1}$

$$\mathbf{e}_{k+1} = \mathbf{v}_{meas,k+1} - \hat{\mathbf{A}}(\hat{\mathbf{q}}_{k+1/k})\mathbf{v}_{orb,k+1} \qquad \textbf{(Eq. 3.20)}$$

10. Update the state vector with the innovation.

$$\hat{\mathbf{x}}_{k+1/k+1} = \hat{\mathbf{x}}_{k+1/k} + \mathbf{K}_{k+1}\mathbf{e}_{k+1} \qquad \textbf{(Eq. 3.21)}$$

After the state vector has been updated, the quaternion elements of the state vector are normalised to ensure that the estimated quaternions still have the quaternion property $q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1$.

$$\hat{\mathbf{q}}_{norm,k+1/k+1} = \frac{\hat{\mathbf{q}}_{k+1/k+1}}{\left\|\hat{\mathbf{q}}_{k+1/k+1}\right\|} \qquad \textbf{(Eq. 3.22)}$$

11. Recompute the discrete output measurement matrix, $\mathbf{H}_{k+1/k+1}$, for the updated state vector, $\hat{\mathbf{x}}_{k+1/k+1}$.

$$\mathbf{H}_{k+1/k+1} = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}_{k+1/k+1}} \qquad \textbf{(Eq. 3.23)}$$

12. Update the perturbation covariance matrix.

$$\mathbf{P}_{k+1/k+1} = \left[\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H}_{k+1/k+1}\right]\mathbf{P}_{k+1/k}\left[\mathbf{I} - \mathbf{K}_{k+1}\mathbf{H}_{k+1/k+1}\right]^T + \mathbf{K}_{k+1}\mathbf{R}\mathbf{K}_{k+1}$$

(Eq. 3.24)

One Step Numerical Integration of the Equations of Motion

In step 1 the EKF algorithm uses a numerical integration technique to propagate the dynamic and kinematic equations of motion. An accurate and fast single step method to do the numerical integration at large time steps for axially symmetric satellites, which was first introduced by Hodgart, was used in the EKF algorithm. The single step numerical integration technique will now be presented without derivation.

1. Calculate the body nutation rate

$$\omega_A = \hat{\omega}_{zi}\left(1 - \frac{I_Z}{I_T}\right) \qquad \textbf{(Eq. 3.25)}$$

2. Calculate the inertially referenced angular rate increments.

$$d\varpi_{xi} = \frac{T_s}{I_T}\left[N_{mx} + \hat{n}_{doy}\hat{A}_{12} - \frac{3GM_\oplus}{R_{s,k}^3}(I_T - I_z)\hat{A}_{23}\hat{A}_{33} - \hat{\omega}_{yi,k/k}h_{z,k} + \hat{\omega}_{zi,k/k}h_{y,k}\right]$$
$$d\varpi_{yi} = \frac{T_s}{I_T}\left[N_{my} + \hat{n}_{doy}\hat{A}_{22} + \frac{3GM_\oplus}{R_{s,k}^3}(I_T - I_z)\hat{A}_{13}\hat{A}_{33} + \hat{\omega}_{xi,k/k}h_{z,k} - \hat{\omega}_{zi,k/k}h_{x,k}\right]$$
$$d\varpi_{zi} = \frac{T_s}{I_T}\left[N_{mz} - \hat{\omega}_{xi,k/k}h_{y,k} + \hat{\omega}_{yi,k/k}h_{x,k}\right]$$

(Eq. 3.26)

3. Correct the x- and y-axis inertially referenced angular rate increments by a rotation of $-\omega_A T_s/2$.

$$\begin{aligned}&[d\omega_{xi}, d\omega_{yi}]_{RECTANGULAR} \rightarrow [d\omega, \phi]_{POLAR}\\ &[d\omega, \phi - \omega_A T_s/2]_{POLAR} \rightarrow [d\omega'_{xi}, d\omega'_{yi}]_{RECTANGULAR}\end{aligned}$$ **(Eq. 3.27)**

4. Correct the x- and y-axis inertially referenced angular rates by a rotation of $-\omega_A T_s$.

$$\begin{aligned}&[\hat{\omega}_{xi,k/k}, \hat{\omega}_{yi,k/k}]_{RECTANGULAR} \rightarrow [\omega, \phi]_{POLAR}\\ &[\omega, \phi - \omega_A T_s]_{POLAR} \rightarrow [\hat{\omega}'_{xi,k/k}, \hat{\omega}'_{yi,k/k}]_{RECTANGULAR}\end{aligned}$$ **(Eq. 3.28)**

5. Propagate the inertially referenced angular rates.

$$\begin{aligned}\hat{\omega}_{xi,k+1/k} &= \hat{\omega}'_{xi,k/k} + d\omega'_{xi}\\ \hat{\omega}_{yi,k+1/k} &= \hat{\omega}'_{yi,k/k} + d\omega'_{yi}\\ \hat{\omega}_{zi,k+1/k} &= \hat{\omega}_{zi,k/k} + d\omega_{zi}\end{aligned}$$ **(Eq. 3.29)**

6. Calculate the transformation increments to transform the inertially referenced angular rates to orbit referenced angular rates.

$$\begin{aligned}d\omega_{ox} &= \hat{A}_{12}\tilde{\omega}_{o,k}\\ d\omega_{oy} &= \hat{A}_{22}\tilde{\omega}_{o,k}\\ d\omega_{oz} &= \hat{A}_{32}\tilde{\omega}_{o,k}\end{aligned}$$ **(Eq. 3.30)**

with,

$$\tilde{\omega}_{o,k} = \omega_o\{1 + 2e\cos(M_k)\}$$ **(Eq. 3.31)**

7. Obtain the latest estimate of the Z-axis orbit referenced angular rate.

$$\hat{\omega}_{oz,k+1} = \hat{\omega}_{zi,k+1/k} + d\omega_{oz}$$ **(Eq. 3.32)**

8. Improve the x- and y-axis transformation increments by a rotation of $-0.8\hat{\omega}_{oz,k+1}T_s$.

$$\begin{aligned}&\left[d\omega_{ox}, d\omega_{oy}\right]_{RECTANGULAR} \to \left[d\omega, \phi\right]_{POLAR}\\ &\left[d\omega, \phi - 0.8\hat{\omega}_{oz,k+1}T_s\right]_{POLAR} \to \left[d\omega'_{ox}, d\omega'_{oy}\right]_{RECTANGULAR}\end{aligned} \qquad \textbf{(Eq. 3.33)}$$

9. Obtain the latest estimates of the x- and y-axis orbit referenced angular rates.

$$\begin{aligned}\hat{\omega}_{ox,k+1} &= \hat{\omega}_{xi,k+1/k} + d\omega'_{ox}\\ \hat{\omega}_{oy,k+1} &= \hat{\omega}_{yi,k+1/k} + d\omega'_{oy}\end{aligned} \qquad \textbf{(Eq. 3.34)}$$

10. Use the latest estimates of the orbit referenced angular rates to propagate the estimated quaternions.

$$\hat{\mathbf{q}}_{k+1/k} = \left[\cos\left(\frac{\overline{\omega}_{k+1}T_s}{2}\right)\mathbf{I} + \frac{1}{\overline{\omega}_{k+1}}\sin\left(\frac{\overline{\omega}_{k+1}T_s}{2}\right)\Omega_{k+1}\right]\mathbf{q}_{k/k} \qquad \textbf{(Eq. 3.35)}$$

with,

$$\overline{\omega}_{k+1} = \sqrt{\hat{\omega}^2_{ox,k+1} + \hat{\omega}^2_{oy,k+1} + \hat{\omega}^2_{oz,k+1}} \qquad \textbf{(Eq. 3.36)}$$

$$\Omega_{k+1} = \begin{bmatrix} 0 & \hat{\omega}_{oz,k+1} & -\hat{\omega}_{oy,k+1} & \hat{\omega}_{ox,k+1}\\ -\hat{\omega}_{oz,k+1} & 0 & \hat{\omega}_{ox,k+1} & \hat{\omega}_{oy,k+1}\\ \hat{\omega}_{oy,k+1} & -\hat{\omega}_{ox,k+1} & 0 & \hat{\omega}_{oz,k+1}\\ -\hat{\omega}_{ox,k+1} & -\hat{\omega}_{oy,k+1} & -\hat{\omega}_{oz,k+1} & 0\end{bmatrix} \qquad \textbf{(Eq. 3.37)}$$

Computing the F Matrix

The linearised perturbation state matrix, $\mathbf{F}(\mathbf{x}(t_k),t)$, is an 8x8 matrix which satisfies the equation

$$\delta\dot{\mathbf{x}}(t_k) = \mathbf{F}(\mathbf{x}(t_k),t_k)\delta\mathbf{x}(t_k) + \mathbf{s}(t_k) \qquad \textbf{(Eq. 3.38)}$$

where,

$\delta\mathbf{x}(t_k) = \mathbf{x}(t_k) - \hat{\mathbf{x}}(t_k)$	=	the state perturbation, or the difference between the actual state and the estimated state
$\mathbf{F}(\mathbf{x}(t_k),t_k)$	=	the linearised perturbation state matrix
$\mathbf{s}(t_k)$	=	the process noise in the system (zero mean white noise with covariance matrix **Q**)

The construction of the **F** matrix will now be presented without derivation. A full derivation can be found in [Steyn].

The **F** matrix is given by

$$\mathbf{F}(\hat{\mathbf{x}}(t_k),t_k)=\begin{bmatrix}\frac{\partial\dot{\omega}}{\partial\omega} & \frac{\partial\dot{\omega}}{\partial\mathbf{q}} & \frac{\partial\dot{\omega}}{\partial n_{dyo}}\\ \frac{\partial\dot{\mathbf{q}}}{\partial\omega} & \frac{\partial\dot{\mathbf{q}}}{\partial\mathbf{q}} & \frac{\partial\dot{\mathbf{q}}}{\partial n_{dyo}}\\ \frac{\partial\dot{n}_{dyo}}{\partial\omega} & \frac{\partial\dot{n}_{dyo}}{\partial\mathbf{q}} & \frac{\partial\dot{n}_{dyo}}{\partial n_{dyo}}\end{bmatrix} \qquad \textbf{(Eq. 3.39)}$$

The first row represents the derivatives of $\dot{\omega}_B^I$ with respect to ω_B^I, $\mathbf{q}$ and n_{dyo}.

$$\frac{\partial\dot{\omega}}{\partial\omega}=\begin{bmatrix}0 & \hat{\omega}_{zi}\left(1-\frac{I_z}{I_T}\right)-\frac{h_z}{I_T} & \hat{\omega}_{yi}\left(1-\frac{I_z}{I_T}\right)+\frac{h_y}{I_T}\\ 0 & -\hat{\omega}_{zi}\left(1-\frac{I_z}{I_T}\right)+\frac{h_z}{I_T} & -\hat{\omega}_{xi}\left(1-\frac{I_z}{I_T}\right)-\frac{h_x}{I_T}\\ -\frac{h_y}{I_z} & \frac{h_x}{I_z} & 0\end{bmatrix} \qquad \textbf{(Eq. 3.40)}$$

$$\frac{\partial\dot{\omega}}{\partial\mathbf{q}}=\frac{\partial\mathbf{N}_{GG}}{\partial\mathbf{q}}+\frac{\partial\mathbf{N}_D}{\partial\mathbf{q}} \qquad \textbf{(Eq. 3.41)}$$

$$\frac{\partial\dot{\omega}}{\partial n_{dyo}}=\frac{\partial\mathbf{N}_D}{\partial n_{dyo}}=\begin{bmatrix}\hat{A}_{12}\\ \hat{A}_{22}\\ 0\end{bmatrix} \qquad \textbf{(Eq. 3.42)}$$

with,

$$\frac{\partial\mathbf{N}_{GG}}{\partial\mathbf{q}}=\frac{2GM_{\oplus}(I_T-I_Z)}{R_s^3}\times$$
$$\begin{bmatrix}-\hat{A}_{33}\hat{q}_4+\hat{A}_{23}\hat{q}_1 & -\hat{A}_{33}\hat{q}_3+\hat{A}_{23}\hat{q}_2 & -\hat{A}_{33}\hat{q}_2-\hat{A}_{23}\hat{q}_3 & -\hat{A}_{33}\hat{q}_1-\hat{A}_{23}\hat{q}_4\\ -\hat{A}_{33}\hat{q}_3-\hat{A}_{13}\hat{q}_1 & -\hat{A}_{33}\hat{q}_4-\hat{A}_{13}\hat{q}_2 & \hat{A}_{33}\hat{q}_1+\hat{A}_{13}\hat{q}_3 & -\hat{A}_{33}\hat{q}_2+\hat{A}_{13}\hat{q}_4\\ 0 & 0 & 0 & 0\end{bmatrix}$$

(Eq. 3.43)

and,

$$\frac{\partial\mathbf{N}_D}{\partial\mathbf{q}}=2\begin{bmatrix}\hat{q}_2\hat{n}_{dyo} & \hat{q}_1\hat{n}_{dyo} & \hat{q}_4\hat{n}_{dyo} & \hat{q}_3\hat{n}_{dyo}\\ -\hat{q}_1\hat{n}_{dyo} & \hat{q}_2\hat{n}_{dyo} & -\hat{q}_3\hat{n}_{dyo} & \hat{q}_4\hat{n}_{dyo}\\ 0 & 0 & 0 & 0\end{bmatrix} \qquad \textbf{(Eq. 3.44)}$$

The second row represents the derivatives of $\dot{\mathbf{q}}$ with respect to ω_B^I, $\mathbf{q}$ and n_{dyo}.

$$\frac{\partial \dot{\mathbf{q}}}{\partial \omega} = \hat{\beta} = \frac{1}{2}\begin{bmatrix} \hat{q}_4 & -\hat{q}_3 & \hat{q}_2 \\ \hat{q}_3 & \hat{q}_4 & -\hat{q}_1 \\ -\hat{q}_2 & \hat{q}_1 & \hat{q}_4 \\ -\hat{q}_1 & -\hat{q}_2 & -\hat{q}_3 \end{bmatrix} \qquad \textbf{(Eq. 3.45)}$$

$$\frac{\partial \dot{\mathbf{q}}}{\partial \mathbf{q}} = \frac{1}{2}\hat{\Omega} = \frac{1}{2}\begin{bmatrix} 0 & \hat{\omega}_{oz} & -\hat{\omega}_{oy} & \hat{\omega}_{ox} \\ -\hat{\omega}_{oz} & 0 & \hat{\omega}_{ox} & \hat{\omega}_{oy} \\ \hat{\omega}_{oy} & -\hat{\omega}_{ox} & 0 & \hat{\omega}_{oz} \\ -\hat{\omega}_{ox} & -\hat{\omega}_{oy} & -\hat{\omega}_{oz} & 0 \end{bmatrix} \qquad \textbf{(Eq. 3.46)}$$

$$\frac{\partial \dot{\mathbf{q}}}{\partial n_{dyo}} = \mathbf{0}_{4\times1} \qquad \textbf{(Eq. 3.47)}$$

And the third row represents the derivatives of $\dot{n}_{dyo}$ with respect to ω_B^I, $\mathbf{q}$ and n_{dyo}.

$$\frac{\partial \dot{n}_{dyo}}{\partial \omega} = \mathbf{0}_{1\times3} \qquad \frac{\partial \dot{n}_{dyo}}{\partial \mathbf{q}} = \mathbf{0}_{1\times4} \qquad \frac{\partial \dot{n}_{dyo}}{\partial n_{dyo}} = 0 \qquad \textbf{(Eq. 3.48)}$$

Since a discrete version of the EKF is used, the **F** matrix must be converted to the discrete system matrix, $\mathbf{\Phi}_k$, so that

$$\delta\mathbf{x}_{k+1} = \exp\left(\mathbf{F}(\hat{\mathbf{x}}(t_k), t_k)T_s\right)\delta\mathbf{x}_k = \mathbf{\Phi}_k\delta\mathbf{x}_k \qquad \textbf{(Eq. 3.49)}$$

therefore,

$$\mathbf{\Phi}_k = \exp\left(\mathbf{F}(\hat{\mathbf{x}}(t_k), t_k)T_s\right) \qquad \textbf{(Eq. 3.50)}$$

which can be approximated with a second order Taylor series expansion,

$$\mathbf{\Phi}_k \approx \mathbf{I} + \mathbf{F}(\hat{\mathbf{x}}(t_k), t_k)T_s + \tfrac{1}{2}\left[\mathbf{F}(\hat{\mathbf{x}}(t_k), t_k)T_s\right]^2 \qquad \textbf{(Eq. 3.51)}$$

Computing the H Matrix

The discrete output measurement matrix $\mathbf{H}_k$ is a 3x8 matrix which relates the innovation error vector, $\mathbf{e}_k$, to the state perturbation $\delta\mathbf{x}_k$ in the equation

$$\mathbf{e}_k = \mathbf{H}_k(\hat{\mathbf{q}}_k)\delta\mathbf{x}_k + \mathbf{m}_k \qquad \textbf{(Eq. 3.52)}$$

which is derived from the innovation error vector calculation in the following manner

$$\begin{aligned}
\mathbf{e}_k &= \mathbf{v}_{meas,k} - \mathbf{A}(\hat{\mathbf{q}}_k)\mathbf{v}_{orb,k} \\
&= \left[\sum_{i=1}^{4} \frac{\delta \mathbf{A}(\hat{\mathbf{q}}_k)}{\delta q_{i,k}} \delta q_{i,k}\right] \mathbf{v}_{orb,k} + \mathbf{m}_k \\
&= \left[\sum_{i=1}^{4} \frac{\delta \mathbf{A}(\hat{\mathbf{q}}_k)}{\delta q_{i,k}} \mathbf{v}_{orb,k}\right] \delta \mathbf{q}_k + \mathbf{m}_k \\
&= [\mathbf{h}_1 \quad \mathbf{h}_2 \quad \mathbf{h}_3 \quad \mathbf{h}_4] \delta \mathbf{q}_k + \mathbf{m}_k \\
&= [\mathbf{0}_{3\times3} \quad \mathbf{h}_1 \quad \mathbf{h}_2 \quad \mathbf{h}_3 \quad \mathbf{h}_4 \quad \mathbf{0}_{3\times1}] \delta \mathbf{x}_k + \mathbf{m}_k \\
&= \mathbf{H}_k(\hat{\mathbf{q}}_k) \delta \mathbf{x}_k + \mathbf{m}_k
\end{aligned} \qquad \textbf{(Eq. 3.53)}$$

From this equation it can be seen that the **H** matrix must be constructed as follows

$$\mathbf{H}_k = [\mathbf{0}_{3\times3} \quad \mathbf{h}_1 \quad \mathbf{h}_2 \quad \mathbf{h}_3 \quad \mathbf{h}_4 \quad \mathbf{0}_{3\times1}] \qquad \textbf{(Eq. 3.54)}$$

with

$$\mathbf{h}_i = \frac{\delta \mathbf{A}(\hat{\mathbf{q}}_k)}{\delta q_{i,k}} \mathbf{v}_{orb,k} \quad , \quad i = 1,2,3,4 \qquad \textbf{(Eq. 3.55)}$$

which results in

$$\begin{aligned}
\mathbf{h}_1 &= 2\begin{bmatrix} \hat{q}_{1,k} & \hat{q}_{2,k} & \hat{q}_{3,k} \\ \hat{q}_{2,k} & -\hat{q}_{1,k} & \hat{q}_{4,k} \\ \hat{q}_{3,k} & -\hat{q}_{4,k} & -\hat{q}_{1,k} \end{bmatrix} \mathbf{v}_{orb,k} \\
\mathbf{h}_2 &= 2\begin{bmatrix} -\hat{q}_{2,k} & \hat{q}_{1,k} & -\hat{q}_{4,k} \\ \hat{q}_{1,k} & \hat{q}_{2,k} & \hat{q}_{3,k} \\ \hat{q}_{4,k} & \hat{q}_{3,k} & -\hat{q}_{2,k} \end{bmatrix} \mathbf{v}_{orb,k} \\
\mathbf{h}_3 &= 2\begin{bmatrix} -\hat{q}_{3,k} & \hat{q}_{4,k} & \hat{q}_{1,k} \\ -\hat{q}_{4,k} & -\hat{q}_{3,k} & \hat{q}_{2,k} \\ \hat{q}_{1,k} & \hat{q}_{2,k} & \hat{q}_{3,k} \end{bmatrix} \mathbf{v}_{orb,k} \\
\mathbf{h}_4 &= 2\begin{bmatrix} \hat{q}_{4,k} & \hat{q}_{3,k} & -\hat{q}_{2,k} \\ -\hat{q}_{3,k} & \hat{q}_{4,k} & \hat{q}_{1,k} \\ \hat{q}_{2,k} & -\hat{q}_{1,k} & \hat{q}_{4,k} \end{bmatrix} \mathbf{v}_{orb,k}
\end{aligned} \qquad \textbf{(Eq. 3.56)}$$

Magnetometer Innovation (Filter 1)

All three EKF's use the magnetometer measurements for innovation. The Magnetometer EKF relies solely on the magnetometer for innovation, while the Horison/Sun Sensor EKF and the Star Sensor EKF relies on the magnetometer for innovation when measurements from the horison sensors, sun sensor and star sensor are not valid. The magnetometer measures the geomagnetic field vector in body coordinates, $\mathbf{B}_{meas,k}$, with a sampling period of ten seconds. At the same sampling instant that the measurement is taken, the modelled geomagnetic field vector in orbit coordinates, $\mathbf{B}_{o,k}$, is also calculated with the on-board IGRF model. The sensor measurement vector, $\mathbf{v}_{meas,k}$, and the modelled measurement vector, $\mathbf{v}_{orb,k}$, which are used to calculate the innovation error vector, are then obtained by normalising the measured and modelled geomagnetic field vectors.

$$\mathbf{v}_{meas,k} = \frac{\mathbf{B}_{meas,k}}{\left\|\mathbf{B}_{meas,k}\right\|} \tag{Eq. 3.57}$$

and,

$$\mathbf{v}_{orb,k} = \frac{\mathbf{B}_{o,k}}{\left\|\mathbf{B}_{o,k}\right\|} \tag{Eq. 3.58}$$

Horison / Sun Sensors Innovation (Filter 2)

The Horison / Sun Sensor EKF uses the horison sensors and sun sensor for its innovation whenever they give valid measurements. The horison sensors measure the elevation of the horison in body coordinates at the azimuths of the x and -y body axes. The sun sensor measures the azimuth of the sun in body coordinates. All three sensors take measurements with a sampling period of one second. Since these three sensors are not necessarily simultaneously valid, innovations are calculated and applied to the EKF individually for each sensor. This means that if any one sensor is valid, it will apply an innovation to the EKF, even if both the other sensors are invalid. If all three sensors are valid, three different innovations will be applied to the EKF in one sampling instant.

X-Horison Sensor

The sensor measurement vector, $\mathbf{v}^{hx}_{meas,k}$, and the modelled measurement vector, $\mathbf{v}^{hx}_{orb,k}$, for the X-horison sensor are obtained from the X-horison sensor measurement, θ_{hx}, and a model of the oblate earth horison as follows.

The sensor measurement vector is obtained with

$$\mathbf{v}^{hx}_{meas,k} = \begin{bmatrix} \cos(\theta_{hx}) \\ 0 \\ \sin(\theta_{hx}) \end{bmatrix} \tag{Eq. 3.59}$$

The modelled measurement vector is obtained with the following steps.

1. The azimuth in orbit coordinates of the X-horison sensor boresight is estimated by transforming the sensor measurement vector to orbit coordinates with the inverse of the estimated DCM and calculating its azimuth angle.

$$\begin{bmatrix} \hat{v}_{ox,k} \\ \hat{v}_{oy,k} \\ \hat{v}_{oz,k} \end{bmatrix} = \mathbf{A}^T(\hat{\mathbf{q}}_k)\mathbf{v}^{hx}_{meas,k} \qquad \textbf{(Eq. 3.60)}$$

$$\hat{Azim}_k = \arctan\left(\frac{\hat{v}_{oy,k}}{\hat{v}_{ox,k}}\right) \qquad \textbf{(Eq. 3.61)}$$

2. Calculate the angle between the X_o axis and the East direction.

$$\text{Angle between Orbit } X_o \text{ and East} = \theta - \phi \qquad \textbf{(Eq. 3.62)}$$

with,

$$\begin{aligned} \theta &= \arctan 4\left(\frac{-B_\theta}{B_\phi}\right) \\ \phi &= \arctan 4\left(\frac{B_{oy}}{B_{ox}}\right) \end{aligned} \qquad \textbf{(Eq. 3.63)}$$

3. Obtain the horison elevation, *Elev*, in orbit coordinates from the oblate earth horison model and the estimated azimuth.

$$Elev = \pi/2 - arc\cot\left\{ \begin{aligned} &\sqrt{\frac{R_s^2 - R_e^2}{a^2}\left[1 + \frac{(2-f)fR_e^2\cos^2(Lat)}{(1-f)^2a^2}\sin^2\Psi\right]} \\ &+ \frac{(2-f)fR_e^2\sin(2Lat)}{2(1-f)^2a^2}\sin\Psi \end{aligned} \right\}$$

$$\textbf{(Eq. 3.64)}$$

with,

$$R_e = \frac{a(1-f)}{\sqrt{1-(2-f)f\cos^2(Lat)}} \qquad \textbf{(Eq. 3.65)}$$

$$\Psi = \hat{Azim}_k + \text{Angle between Orbit } X_o \text{ and East} \qquad \textbf{(Eq. 3.66)}$$

where,

R_s	=	the geocentric distance of the satellite
R_e	=	the oblate earth radius at latitude *Lat*
a	=	the earth's equatorial radius, 6378.14km
f	=	the earth flattening factor, 0.00335281
Ψ	=	the azimuth of the horison vector in the orbit X_oY_o plane, measured from the east direction

4. Construct the modelled measurement vector in orbit coordinates, $\mathbf{v}_{orb,k}^{hx}$, from the estimated azimuth, $\hat{Azim}_k$, and elevation, $Elev$, in orbit coordinates.

$$\mathbf{v}_{orb,k}^{hx} = \begin{bmatrix} \cos\left(\hat{Azim}_k\right)\cos(Elev) \\ \sin\left(\hat{Azim}_k\right)\cos(Elev) \\ \sin(Elev) \end{bmatrix} \qquad \textbf{(Eq. 3.67)}$$

Y Horison Sensor

The sensor measurement vector, $\mathbf{v}_{meas,k}^{hy}$, and the modelled measurement vector, $\mathbf{v}_{orb,k}^{hy}$, for the Y horison sensor are obtained from the Y horison sensor measurement, θ_{hy}, and a model of the oblate earth horison. The procedure differs only slightly from that used by the X horison sensor.

The sensor measurement vector is obtained with

$$\mathbf{v}_{meas,k}^{hy} = \begin{bmatrix} 0 \\ -\cos\left(\theta_{hy}\right) \\ \sin\left(\theta_{hy}\right) \end{bmatrix} \qquad \textbf{(Eq. 3.68)}$$

The modelled measurement vector, $\mathbf{v}_{orb,k}^{hy}$, is obtained with exactly the same procedure as was used for the X horison sensor, except that $\mathbf{v}_{meas,k}^{hy}$ is used instead of $\mathbf{v}_{meas,k}^{hx}$.

Sun Sensor

The sensor measurement vector, $\mathbf{v}_{meas,k}^{sun}$, and the modelled measurement vector, $\mathbf{v}_{orb,k}^{sun}$, for the sun sensor are obtained from the sun sensor measurement, θ_{sun}, and the on-board models for the satellite and sun orbits.

The modelled measurement vector, $\mathbf{v}_{orb,k}^{sun}$, is obtained first using the sun vector model presented in Chapter 2.

$$\mathbf{v}_{orb,k}^{sun} = \begin{bmatrix} v_{ox,k}^{sun} \\ v_{oy,k}^{sun} \\ v_{oz,k}^{sun} \end{bmatrix} = \begin{bmatrix} S_{ox,k} \\ S_{oy,k} \\ S_{oz,k} \end{bmatrix} \qquad \textbf{(Eq. 3.69)}$$

The sensor measurement vector, $\mathbf{v}_{meas,k}^{sun}$, is then obtained from the sun sensor measurement, θ_{sun}, and the estimated z-body axis component of the sun vector. The following steps must be followed.

1. Calculate the measured azimuth of the sun vector from the sun sensor measurement.

$$Azim = \theta_{sun} - \frac{\pi}{2} \qquad \textbf{(Eq. 3.70)}$$

2. Obtain the estimated z-body axis component of the sun vector from the modelled measurement vector and the estimated DCM.

$$\hat{v}_{z,k}^{sun} = \hat{A}_{31} v_{ox,k}^{sun} + \hat{A}_{32} v_{oy,k}^{sun} + \hat{A}_{33} v_{oz,k}^{sun} \qquad \textbf{(Eq. 3.71)}$$

3. Construct the sensor measurement vector, $\mathbf{v}_{meas,k}^{sun}$.

$$\mathbf{v}_{meas,k}^{sun} = \begin{bmatrix} \sqrt{1 - \hat{v}_{z,k}^{sun}} \cos(Azim) \\ \sqrt{1 - \hat{v}_{z,k}^{sun}} \sin(Azim) \\ \hat{v}_{z,k}^{sun} \end{bmatrix} \qquad \textbf{(Eq. 3.72)}$$

Star Sensor Innovation (Filter 3)

When the star sensor is active, it supplies from zero to three matching pairs of modelled and measured star vectors to the ADCS every second. An innovation is performed on the Star Sensor EKF for every matching pair. This means that up to three different innovations may be performed in a single sampling instant. If no matching pairs were found, no innovation is done and the EKF runs open loop.

For the *i*'th matching pair, the measured innovation vector, $\mathbf{v}_{meas,k}^{star,i}$, equals the measured star vector and the modelled innovation vector, $\mathbf{v}_{orb,k}^{star,i}$, equals the modelled star vector.

3.4.2 ANGULAR RATE KALMAN FILTER (FILTER 4)

Introduction

The angular rate kalman filter is a discrete Kalman filter, with a time variant measurement model, which estimates the angular rates in orbit coordinates from the rate of change of the geomagnetic field vector as measured by the magnetometer. The filter was designed to be used after SUNSAT was separated from the launcher to determine the initial tumbling angular rate and for detumbling and rate tracking control before the gravity gradient boom was deployed. Since the filter does take the state of the boom into account, it can also be used after boom deployment.

System model

The state vector to be estimated is the angular rate vector in orbit coordinates, $\omega_B^O(k)$.

Assuming the reaction wheels will not be operated, the euler dynamic equation of motion can be simplified to

$$\dot{\omega}_B^I = \mathbf{I}^{-1}\left(\mathbf{N}_{GG} + \mathbf{N}_D + \mathbf{N}_M - \omega_B^I \times \mathbf{I}\omega_B^I\right) \quad \textbf{(Eq. 3.73)}$$

Using (Eq. 2.11) and assuming a near circular orbit, the simplified euler dynamic equation can be written in terms of angular rates in orbit coordinates.

$$\dot{\omega}_B^O = \mathbf{I}^{-1}\left(\mathbf{N}_{GG} + \mathbf{N}_D + \mathbf{N}_M - \omega_B^O \times \mathbf{I}\omega_B^O\right) + \dot{\mathbf{A}}\omega_o \quad \textbf{(Eq. 3.74)}$$

The gravity gradient torque, $\mathbf{N}_{GG}$, the external disturbance torque, $\mathbf{N}_D$, and the $\dot{\mathbf{A}}\omega_o$ term can be lumped together and modelled as process noise.

$$\dot{\omega}_B^O = \mathbf{I}^{-1}\left(\mathbf{N}_M - \omega_B^O \times \mathbf{I}\omega_B^O\right) + \mathbf{s}(t) \quad \textbf{(Eq. 3.75)}$$

with,

$$\mathbf{s}(t) = \mathbf{I}^{-1}\left(\mathbf{N}_{GG} + \mathbf{N}_D\right) + \dot{\mathbf{A}}\omega_o \quad \textbf{(Eq. 3.76)}$$

where $\mathbf{s}(t)$ can be modelled as continuous process noise with zero mean and a covariance matrix **Q**.

Since the angular rate Kalman filter is a discrete Kalman filter which operates with a sampling period of ten seconds (the period at which the magnetometer is sampled), the discrete system model is required.

$$\hat{\omega}_B^O(k+1) = \hat{\omega}_B^O(k) + \mathbf{I}^{-1}\left(\mathbf{N}_M - \hat{\omega}_B^O(k) \times \mathbf{I}\hat{\omega}_B^O(k)\right)T_s + \mathbf{s}(k) \quad \textbf{(Eq. 3.77)}$$

Innovation

The Kalman filter innovation is done with an innovation error vector obtained from the geomagnetic field measurements from the current and previous sampling instants.

The measured innovation vector, $\mathbf{v}_{meas}(k)$ is obtained by normalising the measured geomagnetic field vector.

$$\mathbf{v}_{meas}(k) = \frac{\mathbf{B}_{meas}(k)}{\left\|\mathbf{B}_{meas}(k)\right\|} \qquad \textbf{(Eq. 3.78)}$$

The modelled innovation vector, $\mathbf{v}_{body}(k)$, is obtained by calculating the normalised modelled geomagnetic field vector in body coordinates given the normalised measured geomagnetic field vector at the previous sampling instant and the estimated angular rates in orbit coordinates.

$$\mathbf{v}_{body}(k) = \mathbf{A}\left(\hat{\omega}_B^O(k)\right)\mathbf{v}_{meas}(k-1) \qquad \textbf{(Eq. 3.79)}$$

The transformation $\mathbf{A}\left(\hat{\omega}_B^O(k)\right)$ is obtained by using the small-angle approximation of the direction cosine matrix.

$$\mathbf{A} \approx \begin{bmatrix} 1 & \psi & -\theta \\ -\psi & 1 & \phi \\ \theta & -\phi & 1 \end{bmatrix} \qquad \textbf{(Eq. 3.80)}$$

where,

ϕ, θ, ψ are small roll, pitch and yaw rotation angles

Assuming constant angular rates, the small rotation angles can be approximated by

$$\begin{aligned} \phi(k) &\approx \hat{\omega}_{ox}(k)T_s \\ \theta(k) &\approx \hat{\omega}_{oy}(k)T_s \\ \theta(k) &\approx \hat{\omega}_{oz}(k)T_s \end{aligned} \qquad \textbf{(Eq. 3.81)}$$

Resulting in,

$$\mathbf{A}\left(\hat{\omega}_B^O(k)\right) = \begin{bmatrix} 1 & \hat{\omega}_{oz}(k)T_s & -\hat{\omega}_{oy}(k)T_s \\ -\hat{\omega}_{oz}(k)T_s & 1 & \hat{\omega}_{ox}(k)T_s \\ \hat{\omega}_{oy}(k)T_s & -\hat{\omega}_{ox}(k)T_s & 1 \end{bmatrix} \qquad \textbf{(Eq. 3.82)}$$

Measurement Model

The measurement model for the Kalman filter is defined as

$$\begin{aligned}\delta\mathbf{v}(k) &= \mathbf{v}_{meas}(k) - \mathbf{v}_{meas}(k-1) \\ &= \mathbf{H}(k)\omega_B^O(k) + \mathbf{m}(k)\end{aligned} \qquad \textbf{(Eq. 3.83)}$$

where,

$\delta\mathbf{v}(k)$	=	the output vector,
$\mathbf{v}_{meas}(k)$	=	the measured innovation vector at the current sampling instant,
$\mathbf{v}_{meas}(k-1)$	=	the measured innovation vector at the previous sampling instant,
$\mathbf{H}(k)$	=	the discrete output measurement matrix,
$\omega_B^O(k)$	=	the state vector,
$\mathbf{m}(k)$	=	the measurement noise (zero mean with covariance matrix **R**)

The discrete output measurement matrix, $\mathbf{H}(k)$, is derived from this definition as follows.

$$\mathbf{v}_{meas}(k) = \mathbf{A}\left(\hat{\omega}_B^O(k)\right)\mathbf{v}_{meas}(k-1) + \mathbf{m}(k) \qquad \textbf{(Eq. 3.84)}$$

or,

$$\mathbf{v}_{body}(k) - \mathbf{v}_{meas}(k-1) = \left[\mathbf{A}\left(\hat{\omega}_B^O(k)\right) - \mathbf{I}\right]\mathbf{v}_{meas}(k-1) + \mathbf{m}(k)$$

$$\delta\mathbf{v}(k) = \begin{bmatrix} 0 & \hat{\omega}_{oz}(k)T_s & -\hat{\omega}_{oy}(k)T_s \\ -\hat{\omega}_{oz}(k)T_s & 0 & \hat{\omega}_{ox}(k)T_s \\ \hat{\omega}_{oy}(k)T_s & -\hat{\omega}_{ox}(k)T_s & 0 \end{bmatrix}\begin{bmatrix} v_{x,meas}(k-1) \\ v_{y,meas}(k-1) \\ v_{z,meas}(k-1) \end{bmatrix} + \mathbf{m}(k)$$

(Eq. 3.85)

which can be rewritten to bring the state vector out of the matrix,

$$\delta\mathbf{v}(k) = \begin{bmatrix} 0 & -v_{z,meas}(k-1)T_s & v_{y,meas}(k-1)T_s \\ v_{z,meas}(k-1)T_s & 0 & -v_{x,meas}(k-1)T_s \\ -v_{y,meas}(k-1)T_s & v_{x,meas}(k-1)T_s & 0 \end{bmatrix}\begin{bmatrix} \hat{\omega}_{ox}(k) \\ \hat{\omega}_{oy}(k) \\ \hat{\omega}_{oz}(k) \end{bmatrix} + \mathbf{m}(k)$$

or,

$$\delta\mathbf{v}(k) = \begin{bmatrix} 0 & -v_{z,meas}(k-1)T_s & v_{y,meas}(k-1)T_s \\ v_{z,meas}(k-1)T_s & 0 & -v_{x,meas}(k-1)T_s \\ -v_{y,meas}(k-1)T_s & v_{x,meas}(k-1)T_s & 0 \end{bmatrix}\omega_B^O(k) + \mathbf{m}(k)$$

(Eq. 3.86)

which leads to,

$$\mathbf{H}(k)=\begin{bmatrix} 0 & -v_{z,meas}(k-1)T_s & v_{y,meas}(k-1)T_s \\ v_{z,meas}(k-1)T_s & 0 & -v_{x,meas}(k-1)T_s \\ -v_{y,meas}(k-1)T_s & v_{x,meas}(k-1)T_s & 0 \end{bmatrix} \quad \textbf{(Eq. 3.87)}$$

Kalman Filter Algorithm

The Kalman filter algorithm is executed every ten seconds.

1. Propagate the state vector using the system model.

$$\hat{\omega}_B^O(k+1/k)=\hat{\omega}_B^O(k/k)+\mathbf{I}^{-1}\left(\mathbf{N}_M-\hat{\omega}_B^O(k/k)\times\mathbf{I}\hat{\omega}_B^O(k/k)\right)T_s \quad \textbf{(Eq. 3.88)}$$

2. Propagate the perturbation covariance matrix.

$$\mathbf{P}_{k+1/k}=\mathbf{P}_{k/k}+\mathbf{Q} \quad \textbf{(Eq. 3.89)}$$

3. Update the discrete output measurement matrix.

$$\mathbf{H}_{k+1}=\begin{bmatrix} 0 & -v_{z,meas}(k)T_s & v_{y,meas}(k)T_s \\ v_{z,meas}(k)T_s & 0 & -v_{x,meas}(k)T_s \\ -v_{y,meas}(k)T_s & v_{x,meas}(k)T_s & 0 \end{bmatrix} \quad \textbf{(Eq. 3.90)}$$

4. Compute the Kalman filter gain.

$$\mathbf{K}_{k+1}=\mathbf{P}_{k+1/k}\mathbf{H}_{k+1}^T\left[\mathbf{H}_{k+1}\mathbf{P}_{k+1/k}\mathbf{H}_{k+1}^T+\mathbf{R}\right]^{-1} \quad \textbf{(Eq. 3.91)}$$

5. Obtain the measured innovation vector, $\mathbf{v}_{meas}(k+1)$, from the current magnetometer measurement.

$$\mathbf{v}_{meas}(k+1)=\frac{\mathbf{B}_{meas}(k+1)}{\left\|\mathbf{B}_{meas}(k+1)\right\|} \quad \textbf{(Eq. 3.92)}$$

6. Obtain the modelled innovation vector, $\mathbf{v}_{body}(k+1)$.

$$\mathbf{v}_{body}(k+1)=\mathbf{v}_{meas}(k)+\mathbf{H}_{k+1}\hat{\omega}_B^O(k+1/k) \quad \textbf{(Eq. 3.93)}$$

7. Calculate the innovation error vector, $\mathbf{e}_{k+1}$.

$$\mathbf{e}_{k+1}=\mathbf{v}_{meas}(k+1)-\mathbf{v}_{body}(k+1) \quad \textbf{(Eq. 3.94)}$$

8. Update the state vector with the innovation.

$$\hat{\omega}_B^O(k+1/k+1)=\hat{\omega}_B^O(k+1/k)+\mathbf{K}_{k+1}\mathbf{e}_{k+1} \quad \textbf{(Eq. 3.95)}$$

9. Update the perturbation covariance matrix.

$$\mathbf{P}_{k+1/k+1}=\left[\mathbf{I}-\mathbf{K}_{k+1}\mathbf{H}_{k+1}\right]\mathbf{P}_{k+1/k} \quad \textbf{(Eq. 3.96)}$$

Yaw Angle Estimator

A rough yaw angle estimator was also added to the angular rate Kalman filter. This added feature utilises the geomagnetic field in orbit coordinates as supplied by the on-board IGRF model to estimate the yaw angle.

1. Propagate the yaw angle given the z-axis orbit angular rate estimated by the Kalman filter.

$$\hat{\psi}(k+1) = \hat{\psi}(k) + 1.1\hat{\omega}_{oz}(k)T_s \quad \textbf{(Eq. 3.97)}$$

2. Obtain the projection of the measured geomagnetic field vector in the xy plane of the body coordinates system.

$$\mathbf{v}_{meas}(k+1) = \frac{1}{\left\|\mathbf{B}_{meas}(k+1)\right\|}\begin{bmatrix} B_{x,meas}(k+1) \\ B_{y,meas}(k+1) \end{bmatrix} \quad \textbf{(Eq. 3.98)}$$

3. Obtain the projection of the IGRF geomagnetic field vector in the xy plane of orbit coordinates system.

$$\mathbf{v}_{orb}(k+1) = \frac{1}{\left\|\mathbf{B}_{o}(k+1)\right\|}\begin{bmatrix} B_{ox}(k+1) \\ B_{oy}(k+1) \end{bmatrix} \quad \textbf{(Eq. 3.99)}$$

4. Calculate the measured yaw angle from the xy plane projections of the measured and modelled geomagnetic field vectors.

$$\psi_{meas} = \arctan 4\left(\frac{v_{x,meas}v_{y,orb} - v_{y,meas}v_{x,orb}}{v_{x,meas}v_{x,orb} + v_{y,meas}v_{y,orb}}\right) \quad \textbf{(Eq. 3.100)}$$

5. Calculate the yaw angle innovation.

$$\psi_{error} = \psi_{meas} - \hat{\psi} \quad \textbf{(Eq. 3.101)}$$

6. Update the estimated yaw angle with the yaw angle innovation.

$$\hat{\psi}(k+1/k+1) = \hat{\psi}(k+1/k) + 0.2\psi_{error} \quad \textbf{(Eq. 3.102)}$$

3.4.3 Y- AND Z- ESTIMATORS (FILTERS 5, 6)

The Y-Estimator and the Z-Estimator are both second order state filters and are used to estimate the angle and angular rate of the satellite about the y and z orbit axes respectively. The Y-Estimator was designed to estimate the pitch and pitch rate during the Y-Thompson spin phase of the pre-boom deployment sequence. The Z-Estimator was designed to estimate the yaw and yaw rate when the satellite is nadir pointing after boom deployment. Since both the Y- and Z-Estimator take the state of the boom into account, they may be used both before and after boom deployment if necessary. Both estimators make strong assumptions and only give valid estimates under specific circumstances. The Y-Estimator assumes that the x and z components of the satellite angular rate are very small and the Z-Estimator assumes that the x and y components are very small. Both filters use the magnetometer measurements to make their estimations.

Y-Estimator (Filter 5)

1. Obtain the projection of the measured geomagnetic field vector in the xz plane of the body coordinates system.

$$\mathbf{v}_{meas}(k) = \frac{1}{\left\|\mathbf{B}_{meas}(k)\right\|}\begin{bmatrix} B_{x,meas}(k) \\ B_{z,meas}(k) \end{bmatrix} \qquad \textbf{(Eq. 3.103)}$$

2. Obtain the projection of the IGRF geomagnetic field vector in the xz plane of orbit coordinates system.

$$\mathbf{v}_{orb}(k) = \frac{1}{\left\|\mathbf{B}_{o}(k)\right\|}\begin{bmatrix} B_{ox}(k) \\ B_{oz}(k) \end{bmatrix} \qquad \textbf{(Eq. 3.104)}$$

3. Calculate the measured pitch angle from the xz plane projections of the measured and modelled geomagnetic field vectors.

$$\theta_{meas} = \arctan 4\left(\frac{-v_{x,meas}v_{z,orb} + v_{z,meas}v_{x,orb}}{v_{x,meas}v_{x,orb} + v_{z,meas}v_{z,orb}}\right) \qquad \textbf{(Eq. 3.105)}$$

4. Calculate the pitch angle innovation.

$$\theta_{error} = \theta_{meas} - \hat{\theta}(k)$$

5. Propagate the estimated pitch rate.

$$\hat{\omega}_{oy}(k+1) = \hat{\omega}_{oy}(k) + \frac{N_{my}T_s}{I_T} + 0.0018 \times \theta_{error} \qquad \textbf{(Eq. 3.106)}$$

6. Propagate the estimated pitch.

$$\hat{\theta}(k+1) = \hat{\theta}(k) + \frac{\left(\hat{\omega}_{oy}(k+1) + \hat{\omega}_{oy}(k)\right)}{2}T_s + 0.15966 \times \theta_{error} \qquad \textbf{(Eq. 3.107)}$$

Z-Estimator (Filter 6)

1. Propagate the estimated yaw rate.

$$\hat{\omega}_{oz}(k+1/k) = \hat{\omega}_{oz}(k/k) + \frac{N_{mz}T_s}{I_z} \qquad \textbf{(Eq. 3.108)}$$

2. Propagate the estimated yaw.

$$\hat{\psi}(k+1/k) = \hat{\psi}(k/k) + \frac{\left(\hat{\omega}_{oz}(k+1/k) + \hat{\omega}_{oz}(k/k)\right)}{2}T_s \qquad \textbf{(Eq. 3.109)}$$

3. Obtain the projection of the measured geomagnetic field vector in the xy plane of

the body coordinates system.

$$\mathbf{v}_{meas}(k+1) = \frac{1}{\|\mathbf{B}_{meas}(k+1)\|}\begin{bmatrix} B_{x,meas}(k+1) \\ B_{y,meas}(k+1) \end{bmatrix} \qquad \textbf{(Eq. 3.110)}$$

4. Obtain the projection of the IGRF geomagnetic field vector in the xy plane of orbit coordinates system.

$$\mathbf{v}_{orb}(k+1) = \frac{1}{\|\mathbf{B}_{o}(k+1)\|}\begin{bmatrix} B_{ox}(k+1) \\ B_{oy}(k+1) \end{bmatrix} \qquad \textbf{(Eq. 3.111)}$$

5. Calculate the measured yaw angle from the xy plane projections of the measured and modelled geomagnetic field vectors.

$$\psi_{meas} = \arctan 4\left(\frac{v_{x,meas}v_{y,orb} - v_{y,meas}v_{x,orb}}{v_{x,meas}v_{x,orb} + v_{y,meas}v_{y,orb}} \right) \qquad \textbf{(Eq. 3.112)}$$

6. Calculate the yaw angle innovation.

$$\psi_{error} = \psi_{meas} - \hat{\psi}(k+1/k) \qquad \textbf{(Eq. 3.113)}$$

7. Update the estimated yaw rate with the yaw angle innovation.

$$\hat{\omega}_{oz}(k+1/k+1) = \hat{\omega}_{oz}(k+1/k) + 0.00005 \times \psi_{error} \qquad \textbf{(Eq. 3.114)}$$

8. Update the estimated yaw angle with the yaw angle innovation.

$$\hat{\psi}(k+1/k+1) = \hat{\psi}(k+1/k) + 0.06 \times \psi_{error} \qquad \textbf{(Eq. 3.115)}$$

3.4.4 LOW PASS FILTER Y-RATE ESTIMATION (HIDDEN FILTER)

The pre-boom deployment y-spin tracking controller contains a hidden low pass filter Y-rate estimator which the controller uses if it finds that the angular rate Kalman filter is not active. This low pass filter rate estimator is a simple filter which estimates only the angular rate of the satellite about the y orbit axis.

Low Pass Filter Y-Rate Estimator

$$\hat{\omega}_{oy}(k+1) = (1-LPF)\hat{\omega}_{oy}(k) + 0.1 \times LPF \times \arcsin\left(\frac{B_{z,meas}(k+1)B_{x,meas}(k) - B_{x,meas}(k+1)B_{z,meas}(k)}{\sqrt{B^2_{x,meas}(k) + B^2_{z,meas}(k)}\sqrt{B^2_{x,meas}(k+1) + B^2_{z,meas}(k+1)}} \right) \qquad \textbf{(Eq. 3.116)}$$

with

$LPF = 0.1$

3.5 Magnetic Torquer Control

3.5.1 LIBRATION DAMPING AND Z-SPIN CONTROL (CONTROL 1, 2, 3)

After boom deployment the magnetic torquers are used to damp the pitch and roll librations induced by external disturbance torques and to track a reference yaw rate. There are three libration damping and z-spin control magnetic torquer controllers: nominal (Controller 1), high gain (Controller 2) and low gain (Controller 3). All three controllers use the same control law and differ only in the gains they use.

Libration Damping and Z-Spin Control Law

$$\mathbf{M} = \frac{\mathbf{e} \times \mathbf{B}_{centre}}{\left\| \mathbf{B}_{centre} \right\|} \qquad \textbf{(Eq. 3.117)}$$

with,

$$\mathbf{e} = K_p \begin{bmatrix} 0.1 \\ 0.1 \\ 0.05 \end{bmatrix} \left(\frac{\omega_B^O}{\omega_o} - \omega_{B,ref}^O \right) \qquad \textbf{(Eq. 3.118)}$$

The only difference between the three libration damping and z-spin control controllers is the value of K_p. The nominal controller uses $K_p = 1$, the high gain controller uses $K_p = 2$ and the low gain controller uses $K_p = 0.4$.

Estimating the Geomagnetic Field Vector in the Centre of the Sampling Period

The control law for libration damping and z-spin control uses an estimate of the geomagnetic field vector in the centre of the sampling period, $\mathbf{B}_{centre}$. This estimate is obtained from the geomagnetic field vector as measured at the beginning of the sampling period and the estimated orbit angular rates.

$$\mathbf{B}_{centre}(kT_s + 0.5T_s) = \mathbf{A}\mathbf{B}_{meas}(kT_s)$$

with,

$$\mathbf{A} = \begin{bmatrix} \cos\Phi + E_1^2(1-\cos\Phi) & E_1E_2(1-\cos\Phi) + E_3\sin\Phi & E_1E_3(1-\cos\Phi) - E_2\sin\Phi \\ E_1E_2(1-\cos\Phi) - E_3\sin\Phi & \cos\Phi + E_2^2(1-\cos\Phi) & E_2E_3(1-\cos\Phi) + E_1\sin\Phi \\ E_1E_3(1-\cos\Phi) + E_2\sin\Phi & E_2E_3(1-\cos\Phi) - E_1\sin\Phi & \cos\Phi + E_3^2(1-\cos\Phi) \end{bmatrix}$$

(Eq. 3.119)

$$\Phi = \frac{T_s}{2}\sqrt{\hat{\omega}_{ox}^2 + \hat{\omega}_{oy}^2 + \hat{\omega}_{oz}^2} \qquad \textbf{(Eq. 3.120)}$$

$$\mathbf{E} = \begin{bmatrix} E_1 \\ E_2 \\ E_3 \end{bmatrix} = \frac{1}{\sqrt{\hat{\omega}_{ox}^2 + \hat{\omega}_{oy}^2 + \hat{\omega}_{oz}^2}} \begin{bmatrix} \hat{\omega}_{ox} \\ \hat{\omega}_{oy} \\ \hat{\omega}_{oz} \end{bmatrix} \qquad \textbf{(Eq. 3.121)}$$

where,

$\mathbf{B}_{centre}(kT_s + 0.5T_s)$	= the geomagnetic field vector in the centre of the sampling period
$\mathbf{B}_{meas}(kT_s)$	= the geomagnetic field vector as measured at the beginning of the sampling period
Φ	= the euler angle
$\mathbf{E}$	= the euler axis
T_s	= the sampling period
$\hat{\omega}_{ox}, \hat{\omega}_{oy}, \hat{\omega}_{oz}$	= the estimated angular rates in orbit coordinates at the beginning of the sampling period

3.5.2 REACTION WHEEL MOMENTUM DUMPING (CONTROL 4)

The reaction wheel momentum dumping control law uses a magnetic torquer control algorithm in conjunction with a reaction wheel control algorithm. The momentum dumping algorithm will be discussed in more detail after the reaction wheel control laws have been introduced. For now, only the magnetic torquer control law will be presented without discussion.

Reaction Wheel Momentum Dumping Law (Magnetic Torquer Part)

$$\mathbf{M} = K_m \frac{(\mathbf{h} \times \mathbf{B})}{\|\mathbf{B}\|} \qquad \textbf{(Eq. 3.122)}$$

with,

$$K_m = 8$$

$$\mathbf{h} = \begin{bmatrix} h_x \\ h_y \\ h_z - h_{z,ref}\,\omega_o \end{bmatrix} \qquad \textbf{(Eq. 3.123)}$$

3.5.3 Y-THOMPSON DETUMBLING (CONTROL 5)

The Y-Thompson detumbling controller uses the y axis magnetic torquer to remove the x and z components of the angular rate vector and to align the angular rate vector with the orbit normal, or y axis. The Y-Thompson controller was designed to be used before boom deployment.

Y-Thompson Detumbling Control Law

$$M_y = K_d \left(\frac{\beta(k) - \beta(k-1)}{\omega_o T_s} \right) \qquad \textbf{(Eq. 3.124)}$$

with,

$$K_d = 0.05$$

and,

$$\beta(k) = \arccos \left(\frac{B_{y,meas}(k)}{\left\| \mathbf{B}_{meas}(k) \right\|} \right) \qquad \textbf{(Eq. 3.125)}$$

3.5.4 PRE-BOOM DEPLOYMENT Y-SPIN CONTROL (CONTROL 6)

This magnetic torquer controller is used after Y-Thompson detumbling to track a reference y-axis angular rate. To ensure gravity gradient capture, the angular rate vector must be sufficiently close to the orbit normal and the y-spin rate must be approximately

$$\omega_{yi} = -\frac{I_{yf}}{I_{yi}} \omega_o \qquad \textbf{(Eq. 3.126)}$$

where,

ω_{yi}	= the y-spin rate just before boom deployment
ω_o	= the orbit mean motion
I_{yf}	= the y-axis moment of inertia before boom deployment
I_{yf}	= the y-axis moment of inertia after boom deployment

before the boom is deployed. The pre-boom deployment y-spin tracking controller uses the Y-Thompson Detumbling control law in conjunction with the following Y-Spin Tracking control law to prepare the satellite for boom deployment.

Y-Spin Tracking Control Law

$$M_x = \begin{cases} K_s \left(\hat{\omega}_{oy} / \omega_o - \omega_{oy,ref} \right) & , B_{z,meas} > 0 \\ -K_s \left(\hat{\omega}_{oy} / \omega_o - \omega_{oy,ref} \right) & , B_{z,meas} \leq 0 \end{cases} \qquad \textbf{(Eq. 3.127)}$$

with,

$$K_s = 0.03$$

$$\omega_{oy,ref} = 1 - \frac{I_{yf}}{I_{yi}}$$

where,

$\omega_{oy,ref}$	= the reference y-spin rate to be tracked

3.5.5 MAGNETIC TORQUE ESTIMATION

An estimate of the magnetic torque is needed by the attitude estimators. This estimate is obtained using either the estimated geomagnetic field in the centre of the sampling period if controllers 1,2 or 3 are used, or simply the measured geomagnetic field vector if the controllers 4, 5 or 6 are used.

$$\hat{\mathbf{N}}_M = \mathbf{M} \times \mathbf{B} \qquad \textbf{(Eq. 3.128)}$$

with,

$$\mathbf{B} = \begin{cases} \mathbf{B}_{centre} & \text{, if controllers 1, 2 or 3 are active} \\ \mathbf{B}_{meas} & \text{, if controllers 4, 5 or 6 are active} \end{cases}$$

3.6 Reaction Wheel Control

3.6.1 COMMANDED QUATERNION

The attitude command which must be tracked by the reaction wheel control algorithms are given as Euler angles, or a commanded roll, pitch and yaw. The reaction wheel control laws, however, use quaternions and need the attitude command in terms of a commanded quaternion. The following calculations are used to convert commanded euler angles to a commanded quaternion.

$$q_{4,c} = \tfrac{1}{2}\sqrt{1+\cos\Psi_c\cos\theta_c - \sin\Psi_c\sin\theta_c\sin\phi_c + \cos\Psi_c\cos\phi_c + \cos\theta_c\cos\phi_c}$$

(Eq. 3.129)

If $q_{4,c} > \tfrac{1}{2}$, then

$$q_{1,c} = \frac{1}{4q_{4,c}}\left(\sin\Psi_c\sin\theta_c\cos\phi_c + \cos\Psi_c\sin\phi_c + \cos\theta_c\sin\phi_c\right)$$

$$q_{2,c} = \frac{1}{4q_{4,c}}\left(\sin\theta_c + \cos\Psi_c\sin\theta_c\cos\phi_c - \sin\Psi_c\sin\phi_c\right)$$

$$q_{3,c} = \frac{1}{4q_{4,c}}\left(\cos\Psi_c\sin\theta_c\sin\phi_c + \sin\Psi_c\cos\phi_c + \sin\Psi_c\cos\theta_c\right)$$

(Eq. 3.130)

else, if $q_{4,c} \le \tfrac{1}{2}$, then

$$q_{3,c} = \tfrac{1}{2}\sqrt{1-\cos\Psi_c\cos\theta_c + \sin\Psi_c\sin\theta_c\sin\phi_c - \cos\Psi_c\cos\phi_c + \cos\theta_c\cos\phi_c}$$

(Eq. 3.131)

and,

$$q_{1,c} = \frac{1}{4q_{3,c}}\left(\sin\theta_c - \cos\Psi_c\sin\theta_c\cos\phi_c + \sin\Psi_c\sin\phi_c\right)$$

$$q_{2,c} = \frac{1}{4q_{3,c}}\left(\sin\Psi_c\sin\theta_c\cos\phi_c + \cos\Psi_c\sin\phi_c - \cos\theta_c\sin\phi_c\right)$$

$$q_{4,c} = \frac{1}{4q_{3,c}}\left(\cos\Psi_c\sin\theta_c\sin\phi_c + \sin\Psi_c\cos\phi_c + \sin\Psi_c\cos\theta_c\right)$$

(Eq. 3.132)

3.6.2 ERROR QUATERNION

The reaction wheel control laws all use the concept of an error quaternion. The error quaternion is the difference between the current quaternion and the commanded quaternion. The error quaternions is calculated using the definition of quaternion division.

$$\begin{bmatrix} q_{1,e} \\ q_{2,e} \\ q_{3,e} \\ q_{4,e} \end{bmatrix} = \begin{bmatrix} q_{4,c} & q_{3,c} & -q_{2,c} & -q_{1,c} \\ -q_{3,c} & q_{4,c} & q_{1,c} & -q_{2,c} \\ q_{2,c} & -q_{1,c} & q_{4,c} & -q_{3,c} \\ q_{1,c} & q_{2,c} & q_{3,c} & q_{4,c} \end{bmatrix} \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} \qquad \textbf{(Eq. 3.133)}$$

where,

$\mathbf{q}$	= the current quaternion
$\mathbf{q}_c$	= the commanded quaternion
$\mathbf{q}_e$	= the error quaternion

3.6.3 POINTING CONTROLLER

The reaction wheel pointing controller is provides accurate pointing and tracking control and implements the following control law:

Pointing Controller Control Law

$$\mathbf{N}_{wheel} = \mathbf{K}\mathbf{q}_{e,vec} + \mathbf{D}\boldsymbol{\omega}_B^O + \mathbf{N}_{add} \qquad \textbf{(Eq. 3.134)}$$

with,

$$\mathbf{K} = diag\begin{bmatrix}0.05 & 0.05 & 0.0025\end{bmatrix}$$

$$\mathbf{D} = diag\begin{bmatrix}2 & 2 & 0.1\end{bmatrix}$$

$$\mathbf{q}_{e,vec} = \begin{bmatrix} q_{1,e} \\ q_{2,e} \\ q_{3,e} \end{bmatrix}$$

The additional torque, $\mathbf{N}_{add}$, cancels the gravity gradient torque and the gyroscopic torque and is calculated with

$$\mathbf{N}_{add} = \hat{\mathbf{N}}_{GG} - \boldsymbol{\omega}_B^I \times \left(\mathbf{I}\boldsymbol{\omega}_B^I + \mathbf{h}\right) - \mathbf{I}\boldsymbol{\omega}_D \qquad \textbf{(Eq. 3.135)}$$

The gravity gradient torque is estimated as follows

$$\hat{\mathbf{N}}_{GG} = \frac{3GM_{\oplus}}{R_s^3}[I_z - I_T](\hat{\mathbf{z}}_o \cdot \mathbf{z})(\hat{\mathbf{z}}_o \times \mathbf{z}) \qquad \textbf{(Eq. 3.136)}$$

where,

$GM_{\oplus}$	= earth's gravitation constant
R_s	= geocentric spacecraft position vector length
$\hat{\mathbf{z}}_o = \begin{bmatrix} \hat{A}_{13} \\ \hat{A}_{23} \\ \hat{A}_{33} \end{bmatrix}$	= estimated nadir pointing unit vector in body coordinates
$\mathbf{z} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$	= z-axis unit vector in body coordinates

and,

$$\omega_D = \omega_B^O - \begin{bmatrix} \hat{A}_{12} \\ \hat{A}_{22} \\ \hat{A}_{32} \end{bmatrix} \tilde{\omega}_o \qquad \textbf{(Eq. 3.137)}$$

with,

$$\tilde{\omega}_o = \omega_o \{1 + 2e\cos(M)\} \qquad \textbf{(Eq. 3.138)}$$

3.6.4 SLEW CONTROLLER

The slew controller enables the satellite to perform near minimum time eigenaxis manoeuvres in order to point its payload at different targets in a short span of time.

Eigenaxis Manoeuvre

An eigenaxis rotation results in the shortest angular path between two attitudes. During an eigenaxis manoeuvre. the rotation axis, **E**, remains fixed in the orbit reference frame and only the rotation angle, Φ, changes.

When performing an eigenaxis rotation with the reaction wheels, the limits on the wheel speed and acceleration must be taken into account. If the wheel speed limit is not encountered, an eigenaxis rotation can be performed by accelerating about the eigenaxis at near limiting acceleration until the halfway mark is reached and then decelerating at near limiting deceleration for a time equal to the acceleration time. Such an eigenaxis rotation without wheel speed limiting is shown in Figure 6.

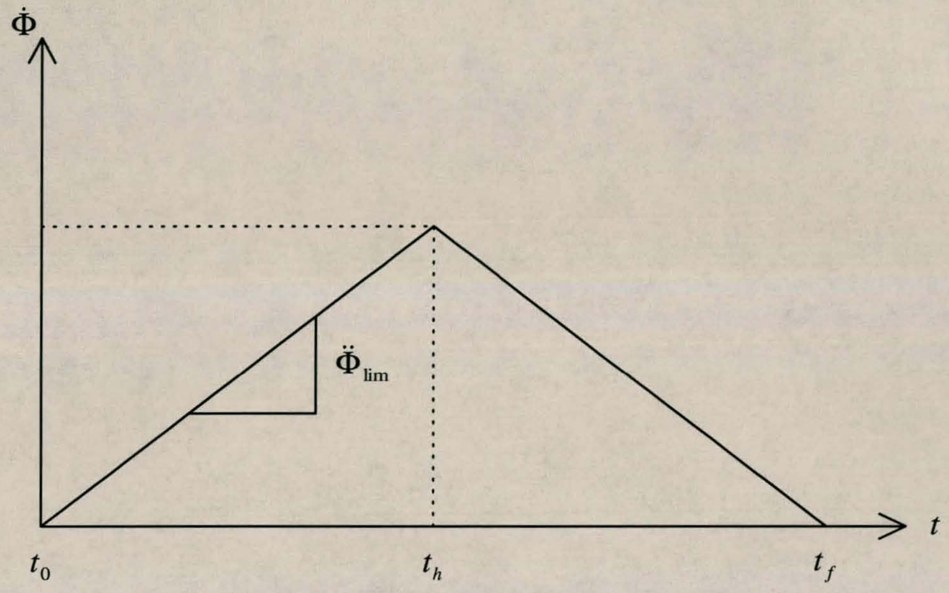


Figure 6: An eigenaxis rotation without wheel speed limiting.

If the wheel speed limit is encountered during the acceleration phase, acceleration is stopped and the satellite is allowed to coast for equal periods of time before and after the halfway mark before deceleration commences. Such an eigenaxis manoeuvre with wheel speed limiting is shown in Figure 7.

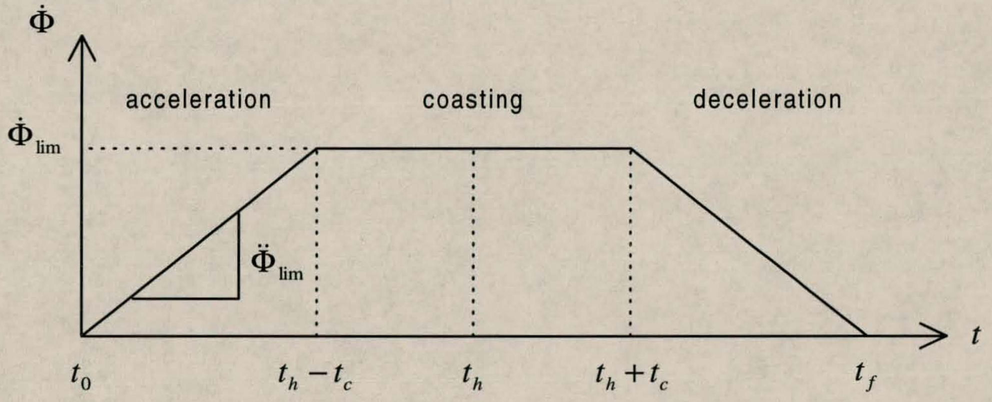


Figure 7: An eigenaxis rotation with wheel speed limiting.

Large Angular Slew Manoeuvre Control Law

The slew controller will calculate the appropriate reaction wheel torque to produce the eigenaxis trajectories as described above. This reaction wheel torque is obtained as the sum of a slew torque, $\mathbf{N}_{slew}$, which produces the eigenaxis trajectories, an additional torque, $\mathbf{N}_{add}$, which cancels the gravity gradient torque and the gyroscopic torque, and a compensation torque, $\mathbf{N}_{comp}$, which compensates for inertia mismatches.

$$\mathbf{N}_{wheel} = \mathbf{N}_{slew} + \mathbf{N}_{add} + \mathbf{N}_{comp} \qquad \textbf{(Eq. 3.139)}$$

Slew Torque Algorithm

The slew torque is obtained from the following control law.

$$\mathbf{N}_{slew} = \mathbf{K}_{slew} N_s \qquad \textbf{(Eq. 3.140)}$$

where,

$\mathbf{N}_{slew}$	= the slew torque vector
$\mathbf{K}_{slew}$	= the slew torque gain
N_s	= the scalar slew torque

The following algorithm is used to calculate the slew torque gain, $\mathbf{K}_{slew}$, and the scalar slew torque, N_s, which produces the appropriate slew torque, $\mathbf{N}_{slew}$, to give the eigenaxis trajectories.

Initialisation

1. When the slew controller is activated, it obtains the eigenaxis, **E**, and the rotation angle, Φ, for the eigenaxis manoeuvre from the initial error quaternion.

$$\Phi = \arccos(q_{4,e}) \qquad \textbf{(Eq. 3.141)}$$

and,

$$\mathbf{E} = \begin{bmatrix} E_1 \\ E_2 \\ E_3 \end{bmatrix} = \frac{1}{\sin\Phi} \begin{bmatrix} q_{1,e} I_T \\ q_{2,e} I_T \\ q_{3,e} I_z \end{bmatrix} \qquad \textbf{(Eq. 3.142)}$$

2. The initial slew torque gain, $\mathbf{K}_{slew}$, is obtained by dividing the eigenaxis by its largest component.

$$\mathbf{K}_{slew} = \frac{\mathbf{E}}{\max(E_1, E_2, E_3)} \qquad \textbf{(Eq. 3.143)}$$

4. The halfway mark of the slew manoeuvre is defined as the halfway mark of the error quaternion component corresponding to the largest component of the eigenaxis. The halfway mark is calculated as

$$q_{half} = \begin{cases} \dfrac{|E_1 \sin(\Phi/2)|}{I_T} & ,\max(E_1, E_3, E_3) = E_1 \\ \dfrac{|E_2 \sin(\Phi/2)|}{I_T} & ,\max(E_1, E_3, E_3) = E_2 \\ \dfrac{|E_3 \sin(\Phi/2)|}{I_z} & ,\max(E_1, E_3, E_3) = E_3 \end{cases} \qquad \textbf{(Eq. 3.144)}$$

and the error quaternion component to be tested, is

$$q_{i,e} \quad ,i = qtest \qquad \textbf{(Eq. 3.145)}$$

with,

$$qtest = \begin{cases} 1 & ,\max(E_1, E_3, E_3) = E_1 \\ 2 & ,\max(E_1, E_3, E_3) = E_2 \\ 3 & ,\max(E_1, E_3, E_3) = E_3 \end{cases} \qquad \textbf{(Eq. 3.146)}$$

Algorithm

1. Check whether the halfway mark has been reached. If it has not been reached yet, the controller is in the acceleration phase. If the halfway mark has been reached, then the controller is in its deceleration phase.

Acceleration phase: $q_{i,e} > q_{half} \quad ,i = qtest$

Deceleration phase: $q_{i,e} \leq q_{half} \quad ,i = qtest$ **(Eq. 3.147)**

2. During the acceleration phase:

(a) Increment a counter which records the duration of the acceleration phase.

$$t_{accel} = t_{accel} + T_s \qquad \textbf{(Eq. 3.148)}$$

(b) Check whether any of the reaction wheels have reached their limiting speed. If so, enter the coasting phase.

(c) If the coasting phase has not been reached, set the slew torque to the limit of the accelerating reaction wheel torque.

$$N_s = N_{\lim} \qquad \textbf{(Eq. 3.149)}$$

(d) During the coasting phase:

(i) Increment a counter which records the duration of the coasting phase until the halfway mark is reached.

$$t_{coast} = t_{coast} + T_s \qquad \textbf{(Eq. 3.150)}$$

(ii) Set the slew torque to zero.

$$N_s = 0 \qquad \textbf{(Eq. 3.151)}$$

During the deceleration phase:

(a) Decrement the counter which recorded the duration of the acceleration phase.

$$t_{accel} = t_{accel} - T_s \qquad \textbf{(Eq. 3.152)}$$

If the counter reaches zero, the duration of the deceleration phase equals the duration of the acceleration phase and the slew controller is finished. The slew controller will then activate the pointing controller and deactivate itself.
(b) During the coasting phase:
(i) Decrement the counter which indicates the duration of the coasting phase during the acceleration phase.

$$t_{coast} = t_{coast} - T_s \qquad \textbf{(Eq. 3.153)}$$

If the counter reaches zero the controller has coasted for equal periods of time before and after the halfway mark and the controller must exit the coasting phase.
(ii) Set the slew torque to zero.

$$N_s = 0 \qquad \textbf{(Eq. 3.154)}$$

(c) If the coasting phase is over, set the slew torque to the limit of the decelerating reaction wheel torque.

$$N_s = -N_{\lim} \qquad \textbf{(Eq. 3.155)}$$

Additional Torque

The additional torque, $\mathbf{N}_{add}$, which compensates for the gravity gradient torque and the gyroscopic torque is calculated in the same way as for the pointing controller.

Compensation Torque

A compensation feedback torque, $\mathbf{N}_{comp}$, is used to compensate for inertia mismatches.

$$\mathbf{N}_{comp} = \mathbf{C}\left[\omega_B^O - \omega_{B,ref}^O\right] \qquad \textbf{(Eq. 3.156)}$$

where $\omega_{B,ref}^O$ is calculated recursively every sampling instant with,

$$\omega_{B,ref}^O(k+1) = \omega_{B,ref}^O(k) - \mathbf{I}^{-1}\mathbf{N}_{slew} \qquad \textbf{(Eq. 3.157)}$$

and with,

$$\mathbf{C} = \begin{bmatrix} 5 \\ 5 \\ 1 \end{bmatrix}$$

3.6.5 REACTION WHEEL CONTROLLER SELECTION

When an attitude command is given, the ADCS chooses either the pointing controller or the slew controller based on the size of the Euler rotation angle, Φ. For rotation angles of smaller than 5°, the pointing controller is used, and for greater than 5°, the slew controller is used.

3.6.6 STOPPING THE REACTION WHEELS

The following feedback law is used to stop the reaction wheels.

$$N_{i,wheel} = \begin{cases} -0.001 & ,h_i > 0.001 \\ 0 & ,-0.001 \le h_i \le 0.001 \\ 0.001 & ,h_i < -0.001 \end{cases} \quad ,i = x, y, z \qquad \textbf{(Eq. 3.158)}$$

In other words, as long as the angular momentums of the reaction wheels are nonzero, a small torque is applied to it in the opposite direction of the momentum.

3.6.7 REACTION WHEEL ANGULAR MOMENTUM AND MOMENTUM RATE ESTIMATION

Simple Euler integration is used to keep track of the angular momentum, $\mathbf{h}$, and the angular momentum rate, $\dot{\mathbf{h}}$, of the reaction wheels. A limiting function is used to account for the limit on the reaction wheel torque and a limited integrator is used to reflect the wheel speed limit.

Reaction wheel torque limiting:

$$\mathbf{N}_{wheel,\lim} = \begin{cases} \mathbf{N}_{sat} & ,\mathbf{N}_{wheel} > \mathbf{N}_{sat} \\ \mathbf{N}_{wheel} & ,-\mathbf{N}_{sat} \le \mathbf{N}_{wheel} \le \mathbf{N}_{sat} \\ -\mathbf{N}_{sat} & ,\mathbf{N}_{wheel} < -\mathbf{N}_{sat} \end{cases} \qquad \textbf{(Eq. 3.159)}$$

Reaction wheel angular momentum rate:

$$\dot{\mathbf{h}}(k) = \mathbf{N}_{wheel,\lim}(k) \qquad \textbf{(Eq. 3.160)}$$

Euler integration of the reaction wheel angular momentum:

$$\mathbf{h}(k+1) = \mathbf{h}(k) + \dot{\mathbf{h}}(k)T_s \qquad \textbf{(Eq. 3.161)}$$

Reaction wheel speed limiting:

$$\mathbf{h}_{\lim}(k+1) = \begin{cases} \mathbf{h}(k) & ,\mathbf{h}(k+1) > \mathbf{h}_{sat} \\ \mathbf{h}(k+1) & ,-\mathbf{h}_{sat} \le \mathbf{h}(k+1) \le \mathbf{h}_{sat} \\ \mathbf{h}(k) & ,\mathbf{h}(k+1) < -\mathbf{h}_{sat} \end{cases} \qquad \textbf{(Eq. 3.162)}$$

When wheel speed limiting occurs, the wheel torque and, by implication, the angular momentum rate, are set to zero.

$$\mathbf{N}_{wheel,\lim}(k) = 0$$

$$\dot{\mathbf{h}}(k) = 0 \qquad \textbf{(Eq. 3.163)}$$

3.7 Momentum Dumping

After reaction wheel manoeuvres, the reaction wheels may retain residual momentum due to external disturbance torques such as gravity gradient torque, aerodynamic torque and solar radiation torque. This residual momentum may build up over successive reaction wheel manoeuvres and ultimately limits the control effort available to the reaction wheels.

To remove the residual momentum on the reaction wheels without transferring the momentum to the satellite body, a magnetic torquer momentum dumping algorithm is used. After a reaction wheel manoeuvre, the reaction wheel pointing controller is activated and instructed to keep the satellite nadir pointing. The magnetic torquer momentum dumping algorithm is then activated. The cross product law implemented by the magnetic torquer algorithm produces an artificial magnetic disturbance torque which causes the reaction wheel pointing controller to compensates for it by controlling the reaction wheel momentum to zero. When the residual reaction wheel momentum reaches zero, the pointing controller and magnetic torquer momentum dumping algorithm are deactivated and the libration damping and z-spin control algorithm is reactivated. In this way the residual momentum on the reaction wheels is removed while the satellite body remains nadir pointing.

3.8 Boom Deployment

3.8.1 NADIR DETECTION

After the satellite has been released into its orbit by the launch vehicle, the magnetic torquers are used to detumble it and to control it to track a precalculated y-spin rate before the boom is deployed. To ensure a nadir pointing gravity gradient lock, the boom must be deployed when the y-spinning satellite is nadir pointing. Two methods of nadir detection are used by the on-board ADCS software: the first checks when the estimated pitch produced by the attitude estimators falls within an envelope of 5° (since the estimated roll is not checked, this method assumes that the satellite is in a y-spin)

$$\left|\hat{\phi}\right| < 5^\circ \qquad \textbf{(Eq. 3.164)}$$

and the second checks when the horison sensor measurements fall within envelopes of 10° each

$$\begin{aligned} \left|\theta_{hx}\right| &< 10^\circ \\ \left|\theta_{hy}\right| &< 10^\circ \end{aligned} \qquad \textbf{(Eq. 3.165)}$$

When these conditions are true, the satellite is nadir pointing.

3.8.2 EFFECT ON ATTITUDE ESTIMATION

When the boom is deployed, the transverse moment of inertia of the satellite changes from $I_T = I_z = 1.875$ to $I_T = I_{TT} = 31.607$. Due to the law of conservation of angular momentum, the inertially referenced angular rates aligned to the x and y axes change by a factor which is the inverse of the ratio of moment of inertia change.

$$\begin{aligned} \omega_{xi}(k+1) &= \omega_{xi}(k) \times \frac{I_Z}{I_{TT}} \\ \omega_{yi}(k+1) &= \omega_{yi}(k) \times \frac{I_Z}{I_{TT}} \end{aligned} \qquad \textbf{(Eq. 3.166)}$$

Since the attitude estimators need to know about this sudden change, the transverse moment of inertia variable, I_T, and the estimated inertially referenced angular rates, $\hat{\omega}_{xi}$ and $\hat{\omega}_{yi}$, are modified as above when the boom is deployed.

3.9 Magnetometer Calibration

The first order calibration model for the magnetometer is

$$\mathbf{B}_c(k) = \mathbf{G}\mathbf{B}_m(k) + \mathbf{b} \qquad \textbf{(Eq. 3.167)}$$

with,

$$\mathbf{G} = diag\lfloor g_x \quad g_y \quad g_z \rfloor$$
$$\mathbf{b} = diag\lfloor b_x \quad b_y \quad b_z \rfloor$$

The calibration gain, **G,** and the calibration bias, **b,** are obtained from a Least Mean Square algorithm which minimises the difference between the magnitude of the orbit referenced IGRF geomagnetic field vector, $\mathbf{B}_o(k)$, and the magnitude of the calibrated measured geomagnetic field vector, $\mathbf{B}_c(k)$.

$$e(k) = \|\mathbf{B}_o(k)\| - \|\mathbf{B}_c(k)\| \qquad \textbf{(Eq. 3.168)}$$

The LMS algorithm is executed as follows:

$$\begin{aligned} g_i(k+1) &= g_i(k) + 2\mu_g(k) f\{e(k)\}\|\mathbf{B}_o(k)\|^2 B_{i,c}(k) B_{i,m}(k) \\ b_i(k+1) &= b_i(k) + 2\mu_b(k) f\{e(k)\}\|\mathbf{B}_o(k)\|^2 B_{i,c}(k) \end{aligned} \qquad , i = x, y, z$$

(Eq. 3.169)

The error, $e(k)$, is modified by a non-linear function, $f\{e(k)\}$, where

$$f\{e(k)\} = \frac{e(k)}{1 + a|e(k)|} \quad , a = 0.1 \qquad \textbf{(Eq. 3.170)}$$

to improve the robustness of the LMS algorithm against large initial residuals and measurement outliers (noise spikes).

A variable step size is used for both the gain step, $\mu_g(k)$, and the bias step, $\mu_b(k)$, using the functions

$$\begin{aligned} \mu_g(k) &= \mu_{g,\max}\lfloor 1 - \exp(-e^2(k)) \rfloor & , \mu_{g,\max} &= 10^{-8} \\ \mu_b(k) &= \mu_{b,\max}\lfloor 1 - \exp(-e^2(k)) \rfloor & , \mu_{b,\max} &= 3 \times 10^{-6} \end{aligned}$$

(Eq. 3.171)

4. HARDWARE-IN-THE-LOOP SIMULATION

4.1 Introduction

A hardware-in-the-loop simulation facility was developed to serve as a tool for designing, analysing, testing and debugging ADCS hardware and software. The facility consists of two main components: a software simulation component and a hardware emulation component.

The software simulation simulates the orbital motion and the attitude motion of the satellite, taking into account the external disturbance torques and the torques provided by the magnetic torquer and reaction wheel actuators, and also simulates the sensor data for the magnetometer, sun cell sensors, horison sensors, sun sensor and star sensor.

The hardware emulation feeds the simulated sensor data to the ADCS tray via sensor emulators which emulate the electrical interfaces of the true sensors. These sensor emulators include a magnetometer emulator, a sun cell sensor emulator, a horison / sun sensor emulator and a star sensor emulator. The actuator signals which are returned by the ADCS are also measured at their electrical interfaces and fed back to the software simulation, closing the loop. The actuator emulators include magnetic torquer emulators and reaction wheel emulators.

The hardware emulation also includes an on-board computer emulator which emulates the communications between the ADCS and the OBC which are needed to activate different ADCS algorithms and also monitor the internal variables of the ADCS.

This chapter describes the layout of the hardware-in-the-loop facility, the structure and timing of the hardware-in-the-loop simulation software and the also the nature of the sensor and actuator hardware emulators. At the end of the chapter there is also a discussion about possible ways to augment the hardware-in-the-loop simulation to include the true sensor hardware in the loop.

4.2 Pure Software Simulation

Before any hardware is added to the simulation loop, the simulation facility must be able to perform pure software simulations.

A pure software simulation loop was created using the satellite motion and space environment simulation models of chapter 2 and the ADCS software simulation of chapter 3. This software simulation was written in the Delphi 3 programming language and is based on an earlier simulation program which was written by Steyn in Turbo Pascal 5.5 for the purpose of designing the ADCS algorithms.

The structure of the pure software simulation is shown in Figure 8. The pure software simulation executes the following sequence of procedures: The simulation loop starts by incrementing the simulation time in seconds. The dynamic and kinematic equations of motion are then numerically integrated to determine the attitude and angular rates of the satellite body. Next the satellite orbit and sun orbit are propagated using the SGP4 model. The satellite orbit propagator provides the geocentric distance, the mean anomaly and the true anomaly of the satellite, the sub-satellite latitude, longitude and right ascension and also its altitude. The sun orbit propagator provides the latitude and longitude of the sub-sun point. The outputs of the satellite and sun orbit propagators then serve as inputs for the geomagnetic field, sun and horison models. The geocentric distance of the satellite and its sub-satellite latitude, longitude and right ascension are fed to the IGRF model to calculate the orbit referenced geomagnetic field vector. The sub-satellite and sub-sun latitudes and longitudes are used by the sun model to obtain the orbit referenced sun vector. The geomagnetic field vector and sun vector are then transformed from orbit coordinates to body coordinates using the satellite's attitude. The horison model uses the attitude of the satellite as well as its geocentric distance and sub-satellite latitude to calculate the orbit referenced elevations of the earth horison as seen by the X and –Y horison sensors. The outputs of the geomagnetic field, sun and horison models are then fed to the sensor models to generate simulated sensor measurements. The magnetometer measurement is obtained from the body referenced geomagnetic field vector, the sun cell sensor measurements and sun sensor measurement are obtained from the body referenced sun vector and the horison sensor measurements are obtained from the orbit referenced X and –Y horison elevations.

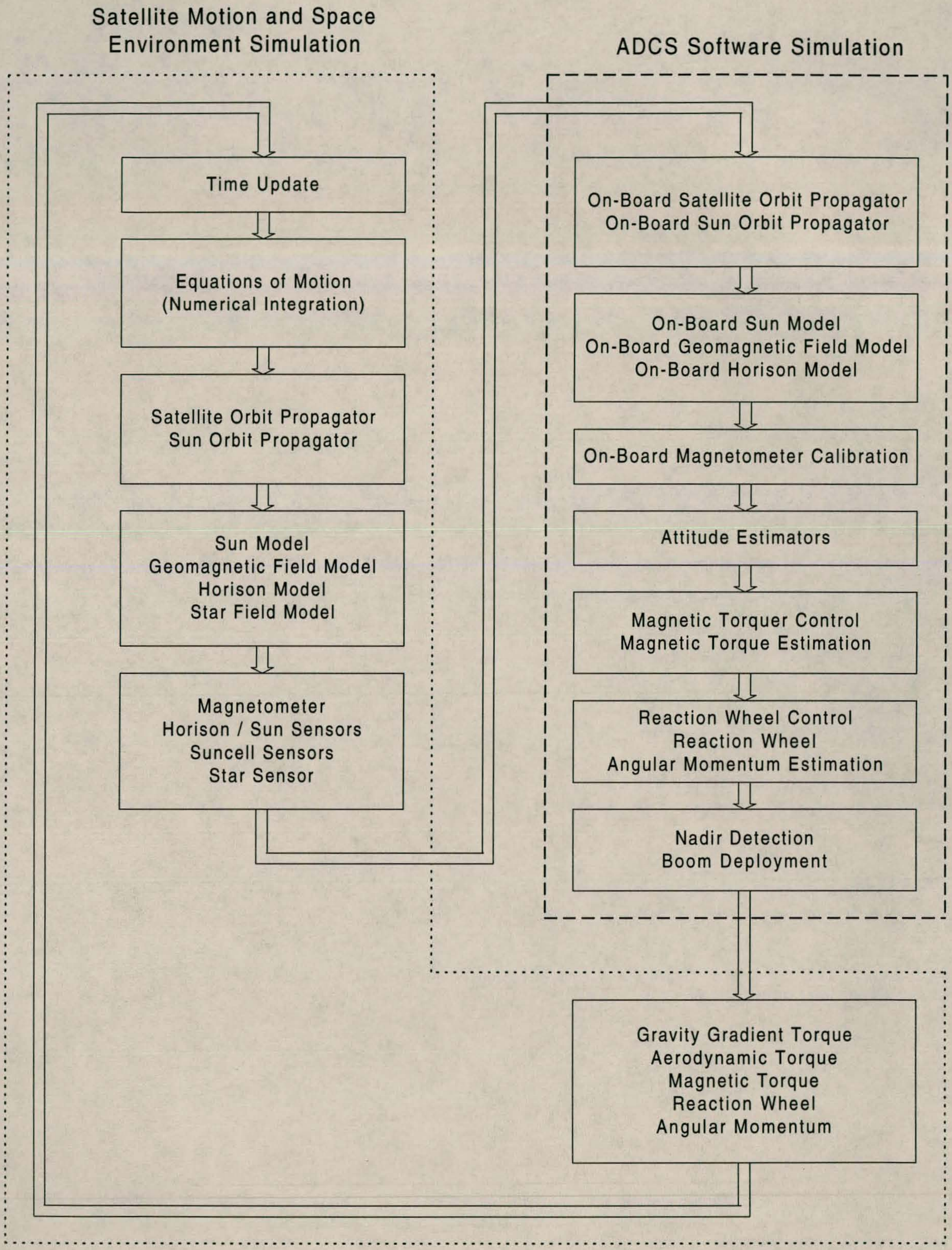


Figure 8: The structure of the simulation software.

The simulated sensor measurements from the satellite motion and space environment simulation become inputs for the simulation of the ADCS software. The ADCS software simulation starts by executing its own set of on-board space environment models, which includes on-board satellite and sun orbit propagators and also on-board geomagnetic field, sun and horison models. At this stage the sensor measurements are also validated. Depending on which ADCS algorithms have been activated by the user, the on-board magnetometer calibration, attitude estimators, magnetic torquer controllers and reaction wheel controllers may be executed. The outputs of the attitude estimators and magnetic torquer and reaction wheel control algorithms are also used to estimated the magnetic torque generated by the torquers and the angular momentum and momentum rate of the reaction wheels. If the gravity gradient boom has not been deployed yet, the nadir detection and boom deployment algorithms are also executed.

The magnetic dipole moment generated by the torquers and the reaction wheel angular momentum and momentum rate calculated by the ADCS control algorithms are then fed back to satellite motion simulation. The magnetic dipole moment and the body referenced geomagnetic field vector are used to calculate the magnetic torque applied to the satellite body. The external disturbance torques, namely the gravity gradient torque and aerodynamic torque are also calculated. The gravity gradient torque, aerodynamic torque, magnetic torque and reaction wheel angular momentum and momentum rate are then ready to be used by the dynamic equations of motion when the simulation loop repeats.

4.3 Emulated Sensor/Actuator HIL Simulation

4.3.1 SETUP

The hardware-in-the-loop simulation facility consists of three personal computers which are connected to each other via their serial ports. The first computer, a 300MHz Pentium II called Simulator, was used to run the software simulation. The other two personal computers, a 60 MHz Pentium called Emulator One and a 486 called Emulator Two, were used to emulate the sensors and actuators as well as the primary on-board computer (OBC1). Emulator One was used to emulate the magnetometer, the horison sensors, the sun sensor, the star sensor, the magnetic torquers and the Y and Z_1 reaction wheels. Emulator Two was used to emulate OBC1, the sun cell sensors and the X and Z_2 reaction wheels.

The magnetometer and magnetic torquers were emulated with two custom ADAS cards, the horison sensors and sun sensor were emulated with a single custom FPGA card, the star sensor and on-board computer were emulated with two separate custom transputer cards, the sun cell sensors were emulated with two custom ADAS cards, the Y and Z_1 reaction wheels were emulated with the Com2 serial port of Emulator One and the X and Z_2 reaction wheels were emulated with the Com2 serial port of Emulator Two.

The setup of the emulated sensor/actuator hardware-in-the-loop simulation facility is shown in Figure 9.

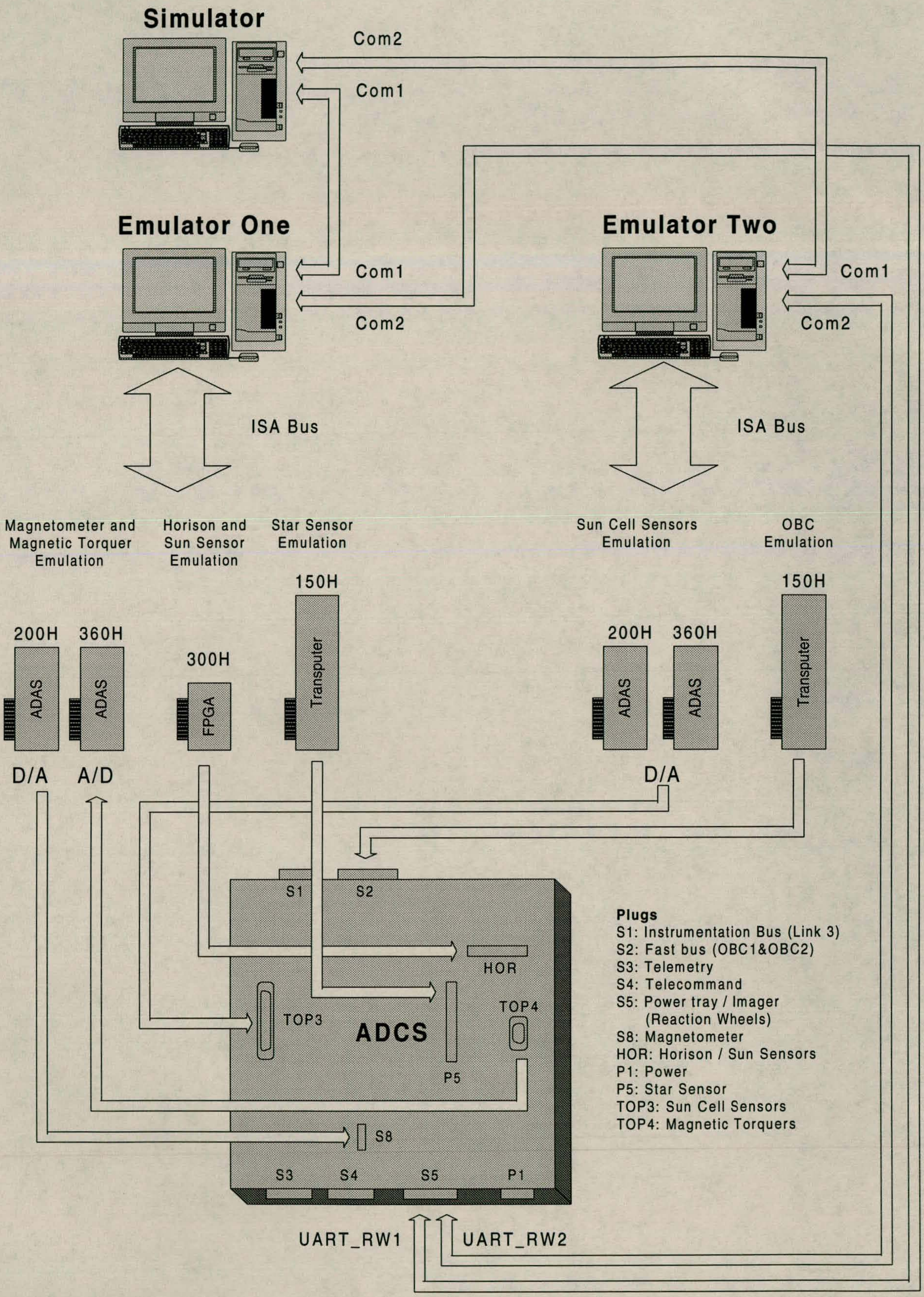


Figure 9: The setup of the emulated sensor/actuator hardware-in-the-loop simulation facility.

4.3.2 ON-BOARD COMPUTER EMULATION

Original Interface with the On-Board Computer

The two on-board computers, OBC1 and OBC2, may communicate with the attitude control processor (ACP), which is a T800 transputer, through its standard transputer communication links. Three of the ACP's communication links, namely Link 0, Link 1 and Link 3, have been reserved for communication with the on-board computer. Link 0 communicates with OBC1, Link 1 with OBC2 and Link 3 with the transputer card on a personal computer. Link 2 is used for communication with the star sensor control processor (SCP). OBC1 and OBC2 interface with Link 0 and Link 1 of the ACP through the FASTBUS plug and the transputer card on a personal computer interfaces with Link 3 through the INSTRUMENTATION BUS plug. Both of these plugs are shown in Figure 10.

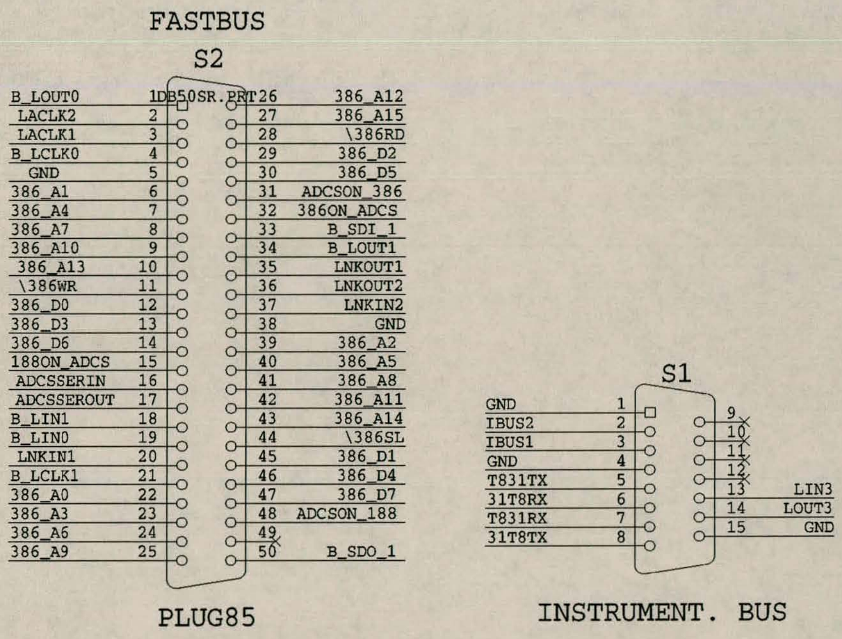


Figure 10: The FASTBUS and INSTRUMENTATION BUS plugs.

The signals transmitted and received by Link 0 and Link 1 are converted to differential signals by differential line drivers (DS26C31) and receivers (DS26C32). The output signal of Link 0, LOUT0, becomes the differential output signal pair at output pins B_LIN0 and B_LIN1, and the input signal of Link 0, LIN0, is obtained from the differential input signal pair at input pins B_LOUT0 and B_LOUT1. In the same way, the output signal of Link 1, LOUT1, becomes LINKIN1 and LINKIN2, and the input signal of Link1, LIN1, is obtained from LNKOUT1 and LNKOUT2.

The signals transmitted and received by Link 3 are not converted to differential signals. The output and input signals of Link 3 are on pins LOUT3 and LIN3.

Communications using Link 0, Link 1 and Link 3 are identical. The first of these three to receive a byte of 55H after the ACP has been rebooted will become the chosen channel of communication until the ACP is rebooted again. The communications protocol which has been adopted for communications between the OBC and the ACP on these three communications links can be found in Appendix A.

The rest of the pins on the FASTBUS and INSTRUMENTATION BUS plugs are not important for the preliminary on-board computer emulations and will not be discussed here.

On-Board Computer Emulation

A custom ISA based transputer card was used to emulate the link adapter which OBC1 uses to communicate with the ACP. The transputer card is mapped to the ISA bus as shown in Table 2.

Table 2: The ISA address mapping of the custom transputer card.

Register	*ISA bus address*
Input Data	150H
Output Data	151H
Input Status	152H
Output Status	153H
Reset (write) / Error (read)	160H
Analyse (write)	161H
Auxiliary	164H

Using the transputer card, bytes may be written to and read from the link adapter (which may be connected to the communications link of a remote transputer) via the ISA bus of a personal computer. To read a byte from the link adapter, the input data register is polled until it indicates that a byte has been received from the remote transputer, and then the byte may be read from the input data register. Likewise, to write a byte to the link adapter, the output data register is polled until it indicates that the remote transputer is ready to receive another byte, and then the byte may be written to the output data register.

The reset and analyse registers are used to initialise the transputer card. The auxiliary register is not used for this application.

The device drivers for transputer card can also be found in Appendix A.

4.3.3 MAGNETOMETER EMULATION

Original Magnetometer Piggyback

The OriMag triaxial fluxgate magnetometer piggyback was designed and manufactured at Hermanus Magneto Observatory.

The magnetometer piggyback interfaces with the ADCS tray via plug S8 as shown in Figure 11.

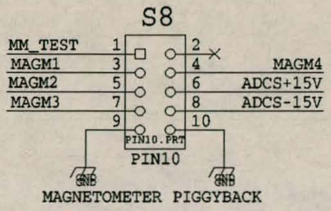


Figure 11: Magnetometer piggyback plug.

Output pins MAGM1, MAGM2 and MAGM3 carry analogue output voltages proportional to the magnetic flux measured along the X, Y and Z axes of the magnetometer. For the preliminary emulation, the following specifications are of importance:

- measuring range: ±60 000 nT (nominal)
- output voltage swing: ±5 V
- output scale factor: 12 nT/mV (nominal)

Output pin MAGM4 carries an analogue output voltage proportional to the measured temperature of the magnetometer housing.

Input pin MM_TEST activates the Built-in Test (BIT) function of the magnetometer. When MM_Test is driven high, the function is activated; when it is driven low the function is inactive.

Magnetometer Piggyback Emulation

Two 12 bit ADAS cards were used to emulate the magnetometer piggyback. Output pins MAGM1 and MAGM2 were emulated by the two D/A channels supplied by the first ADAS card, and output pin MAGM3 was emulated by one of the two D/A channels supplied by the second ADAS card. Output pin MAGM4 and input pin MM_Test were not emulated since neither the temperature nor the built-in test are of importance for the preliminary HIL simulations.

The channels of the two ADAS cards are mapped to the ISA bus of the emulation PC as shown in Table 3. A single channel is updated by writing the low and high bytes of a 12 bit digital value to the corresponding addresses and executing a dummy read from the lower byte address.

Table 3: Magnetometer Piggyback Emulator ISA Bus Addressing.

Channel	ISA bus address
MAGM1 low byte	154H
High byte	155H
MAGM2 low byte	156H
High byte	157H
MAGM3 low byte	204H
High byte	205H

Since there are no pins on the MMP interface which would indicate to the emulator that the ADCS has finished reading the magnetometer values, another way must be found to determine when MAGM1 to MAGM3 should be updated with fresh simulated magnetometer data. One way is to use the interrupt generated by the HSSP emulator on IRQ3, since the ADCS always reads the MMP directly before reading the MMP. This solution is not completely satisfactory, because some simulations will require the HSSP to be off, while the MMP is always on. Another possible solution is to tap the /BUSY pin of the magnetometer A/D converter on the ADCS tray and use its rising flank on the conversion of MAGM3 to indicate that the all three channels may be updated.

4.3.4 HORISON / SUN SENSOR EMULATION

Original Horison / Sun Sensor Piggyback

The two horison sensors and the single fine sun sensor are essentially clones of each other. All three sensors contain linear CCD's and identical circuits which determine the transition from light to dark on their respective CCD's. In the case of the two horison sensors images of the earth horison is focused on their CCD's, while the fine sun sensor uses a slit to allow a thin strip of sunlight to fall on its CCD. Each CCD contains 2048 pixels and the transition from light to dark occurs at one of these pixels. Therefore, the transitions from light to dark can be represented by three 11 bit values, one for each sensor. These 11 bit values are named A10..0 for the X-horison sensor, B10..0 for the Y-horison sensor and S10..0 for the fine sun sensor.

The horison /sun sensor piggyback interfaces with the ADCS tray through plug HOR as shown in Figure 12.

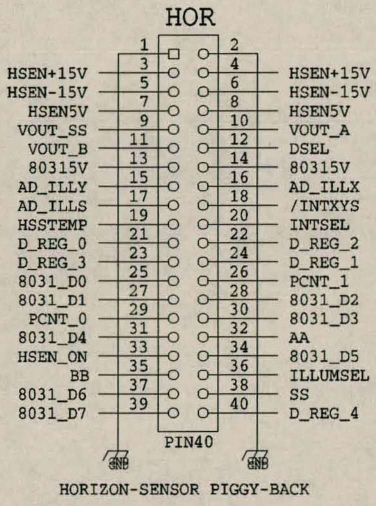


Figure 12: Plug HOR.

Pins 8031_D0..7 are on the ICP data bus and act as an I/O port for data transfer between the horison / sun sensor piggyback and the ICP.

The horison / sun sensor hardware contains five 8 bit registers, namely D_REG_0 to D_REG_4, which are used to store the values of A10..0, B10..0 and S10..0. D_REG_0 to D_REG_2 are used by the horison sensors, while D_REG_3 to D_REG_4 are used by the fine sun sensor. Three other bits, AA, BB and SS, which are used to indicate if the CCD's saturated, are also stored in registers D_REG_1 and D_REG_4. The registers are organised as in Figure 13.

D_REG_0

A7	A6	A5	A4	A3	A2	A1	A0

D_REG_1

BB	B10	B9	B8	AA	A10	A9	A8

D_REG_2

B7	B6	B5	B4	B3	B2	B1	B0

D_REG_3

S7	S6	S5	S4	S3	S2	S1	S0

D_REG_4

-	-	-	-	SS	S10	S9	S8

Symbol	**Name and significance**
A10..0	X-horison sensor data: 11 bit value representing X horison transition
B10..0	Y-horison sensor data: 11 bit value representing Y horison transition
S10..0	fine sun sensor data: 11 bit value representing sun transition
AA	X-horison sensor saturation bit
BB	Y-horison sensor saturation bit
SS	fine sun sensor saturation bit

Figure 13: Organisation of registers D_REG_0 to D_REG_4.

Registers D_REG_0 to D_REG_4 can be read through the 8 bit I/O port of the horison / sun sensor piggyback. The three 11 bit values A10..0, B10..0 and S10..0 are also converted to analogue voltages and output on pins VOUT_A, VOUT_B and VOUT_SS.

The illumination time of the CCD's of all three sensors are mutually set by programming one 8 bit register ILLUMSEL via port 8031_D. A hardwired illumination time can also be selected. When the input pin DSEL is driven low the hardwired time is selected and when it is driven high the programmed time is selected.

The value of any pixel on each of the three CCD's can be obtained individually. This feature of the horison / sun sensor piggyback is useful for debugging purposes to determine if a specific pixel has become faulty. To obtain the value of a pixel, the pixel number, an 11 bit value, is written to registers PCNT_0 and PCNT_1 via port 8031_D. PCNT_0 contains the lower 8 bits and PCNT_1 the higher 3 bits. Three sample and hold circuits will then sample the analogue output voltages of all three

CCD's at the specific pixel number and output them on pins AD_ILLX for the X-horison sensor, AD_ILLY for the Y-horison sensor and AD_ILLS for the fine sun sensor. An active low interrupt will also be generated on pin /INTXYS to signal that the horison / sun sensor has obtained the pixel values. Interrupt /INTXYS is multiplexed with interrupt /ADCSINTR which comes from the RAM tray. Output pin INTSEL is used to indicate which interrupt is selected. When INTSEL is low, the ICP responds to interrupt /ADCSINTR and when INTSEL is high, the ICP responds to interrupt /INTXYS. INTSEL is set with bit 8 in register PCNT_1.

Pins D_REG_0 to D_REG_4 are active low control signals which are used to read the contents of registers D_REG_0 to D_REG_4 from the horison / sun sensor piggyback. Normally all five control signals are driven high, keeping I/O port 8031_D in a high impedance state. When one of the signals is driven low, its corresponding register is output on I/O port 8031_D. Pins D_REG_0 to D_REG_4 may only be driven low one at a time, otherwise bus contention will occur.

Horison / Sun Sensor Emulator

An FPGA-based ISA card was used to emulate the horison / sun sensor piggyback. The circuit diagram of the emulator, as well as the VHDL code for the FPGA, is presented in Appendix C.

Interface with the ADCS

Only the following pins of the original horison / sun sensor are emulated by the FPGA:

- 8 bit I/O port 8031_D
- input pins D_REG_0 to D_REG_4
- output pins INTSEL and /INTXYS

Port 8031_D and read signals D_REG_0 to D_REG_4 and internal registers D_REG_0 to D_REG_4 operate in exactly the same manner as described in the previous section. The interrupt selection pin INTSEL and the interrupt pin /INTXYS are permanently driven low and high respectively, to ensure that a pixel value interrupt is not generated.

The following pins are not emulated by the FPGA:

- Output pins AA, BB and SS are not used by the current flight software, but only made available to the telemetry system.
- VOUT_A, VOUT_B and VOUT_SS are also telemetry signals and carry analogue voltages which cannot be emulated by FPGA.
- During the preliminary stages of HIL simulation, emulating the illumination time of the CCD's is not necessary. Therefore internal register ILLUMSEL and input

pins DSEL and ILLUMSEL were not implemented in the first version of the HSSP emulator.

- The pixel value feature was also not necessary for preliminary simulations and therefore internal registers PCNT_0 and PCNT_1, input pins PCNT_0 and PCNT_1 and analogue output pins AD_ILLX, AD_ILLY and AD_ILLS were also not implemented.

Interface with the emulation PC

The HSSP emulator interfaces with the following pins of the ISA bus:

- the lower ten bits SA9..0 of the address bus
- the lower eight bits D7..0 of the data bus
- the address enable signal AEN
- the bus clock BCLK
- the I/O write signal /IOWC
- the no wait state signal /NOWS
- the hardware reset signal RESDRV
- the input-only interrupt lines IRQ3 to IRQ5 and IRQ7

The HSSP emulator is mapped to the ISA bus address space as shown in Table 4.

Table 4: Horison / Sun Sensor Piggyback Emulator ISA Bus Address Mapping

Register	ISA Bus Address
D_REG_0	300H
D_REG_1	301H
D_REG_2	302H
D_REG_3	303H
D_REG_4	304H
Acknowledge IRQ3	305H

Operation

The operation of the HSSP emulator will be explained with reference to the timing diagram in Figure 14. The PC loads internal registers D_REG_0 to D_REG_4 with simulated horison and sun sensor data by executing ISA bus write cycles to addresses 300H to 304H. It then waits for the ADCS to read the registers. After all five registers have been read by the ADCS, at the falling edge of read signal D_REG_4, the emulator generates an interrupt on IRQ3 of the ISA bus. This indicates to the PC that it must reload registers D_REG_0 to D_REG_4 with fresh simulated sensor data. When the PC services IRQ3, it acknowledges the interrupt by executing a dummy write to address 305H on the ISA bus.

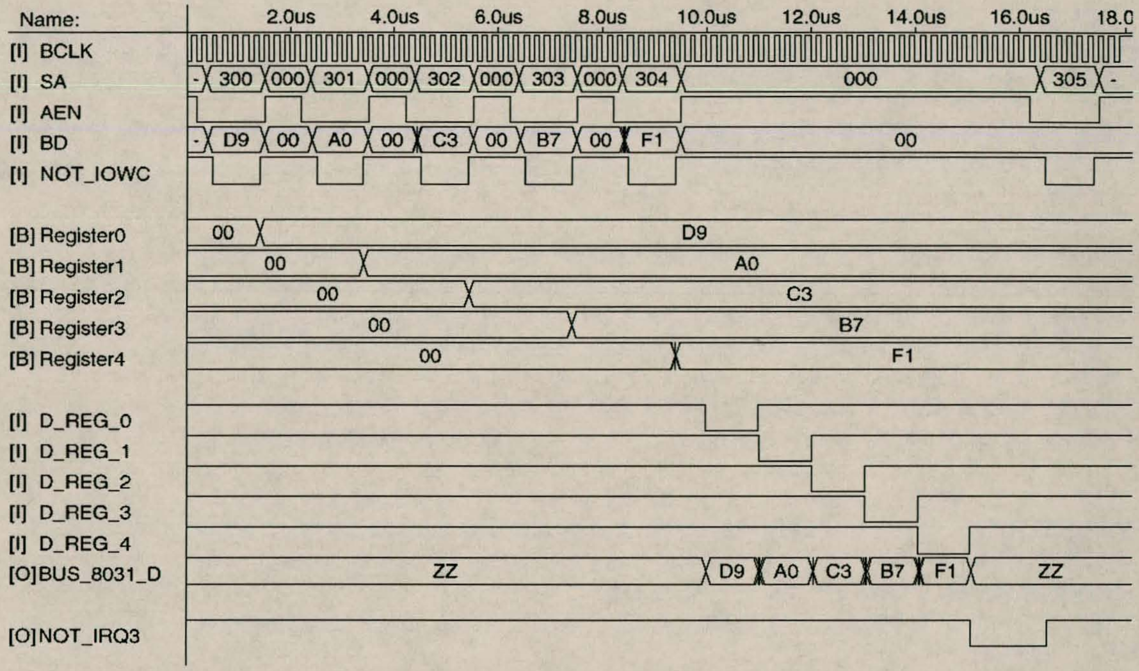


Figure 14: The timing diagram of the HSSP emulator operation.

4.3.5 STAR SENSOR EMULATION

Original Star Sensor Link Interface

The ADCS ACP communicates with the star sensor SCP via link 2. Communication follows the following sequence: the ACP initiates communication by transmitting the synchronisation code 1234567890, followed by a data packet to link 2. The data packet contains the true anomaly plus the argument of perigee, the inclination, the right ascension of ascending node and the roll, pitch and yaw angles as estimated by the ADCS. The SCP the responds by transmitting the synchronisation code 1234567890 and a data packet containing observed vectors in body coordinates and reference vectors in orbit coordinates.

The star sensor interfaces with the ADCS tray through plug STSEN as shown in Figure 15.

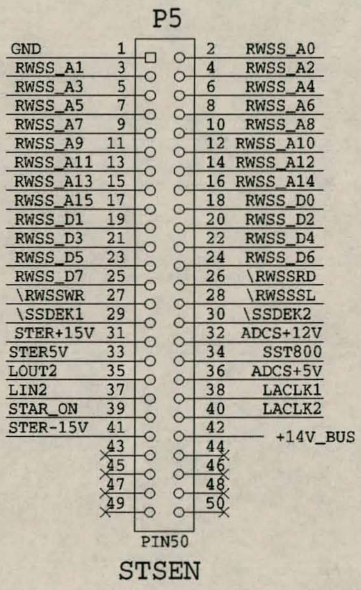


Figure 15: The STSEN plug.

The input and output signals of Link 2 of the ACP are found on pins LIN2 and LOUT2. Unlike the communications on Link 0 and Link 1 with OBC1 and OBC2, the communications with the star sensor on Link 2 are not differential signals.

The rest of the pins on the STSEN plug are not important for preliminary star sensor emulation and will not be discussed.

Star Sensor Link Emulation

A custom ISA based transputer card can be used to emulate the transputer communications link which the SCP uses to communicate the ACP. The operation of the transputer card has already been discussed under OBC emulation. To emulate the star sensor communications, the transputer card simply imitates the communications protocol as described above.

The software which may be used to drive the star sensor emulator can be found in Appendix D.

Star Sensor SRAM Emulation

The true star sensor can also be included in the HIL simulation by emulating its SRAM with an ISA-based FPGA card. The SRAM contains both the star catalogue and the image taken by the star sensor matrix CCD.

4.3.6 COARSE SUN SENSORS EMULATION

The coarse sun sensors, or solar cell sensors, consist of six solar cells, one on each face of the cube that is SUNSAT's main body. Each solar cell also has its own thermistor, which measures its temperature. The solar cell sensors interface with the ADCS tray through plugs SONSEL and POWER TRAY / IMAGER. Both plugs are shown in Figure 16.

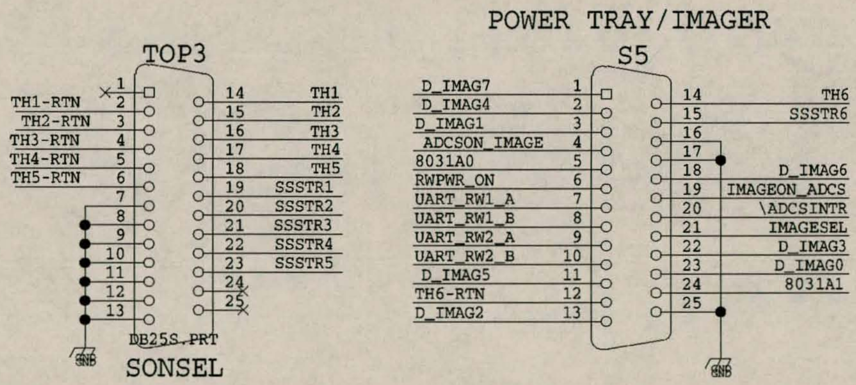


Figure 16: Plugs SONSEL and POWER TRAY / IMAGER.

Pins SSSTR1..6 carry the output voltages of the solar cells, while pins TH1..6 and pins TH1..6-RTN are connected to the thermistors. Pins SSSTR1..5, TH1..5 and TH1..5-RTN are located on plug SONSEL, while pins SSSTR6, TH6 and TH6-RTN are located on plug POWER TRAY / IMAGER.

The outputs of the solar cell sensors are not utilised by the current ADCS flight software, but are only transmitted upon request to the telemetry system for use on the ground. For this reason, the emulation of these sensors is not essential to the HIL simulation of the ADCS and was not implemented.

If however a coarse sun sensor emulation is desired in the future, for example if the interface between the ADCS tray and the telemetry tray needs to be tested, it would be a simple exercise to emulate the output voltages of the solar cells with three ADAS cards, each card possessing two D/A channels. Emulating the thermistors, which are temperature controlled resistors, would be more difficult but could probably be accomplished with BJT's, which are current controlled resistors or with FET's which are basically voltage controlled resistors.

4.3.7 MAGNETIC TORQUER EMULATION

Original magnetic torquers

There are six magnetic torquer coils on SUNSAT. Coils X1 and X2 generate magnetic dipole moments aligned to the x body axis, coils Y1 and Y2 generate dipole moments aligned the y body axis, and coils Z1 and Z2 generate dipole moments aligned to the z-body axis Each coil has an input resistance of about 50 ohms and an inductance which is negligible at its operating frequencies. The current in each magnetic torquer is supplied by a MOSFET driver circuit which in turn is controlled by the interface control processor (ICP). Since the MOSFET drivers circuits are identical, their operation will be discussed with reference to the MOSFET driver for coil X1 as shown in Figure 17.

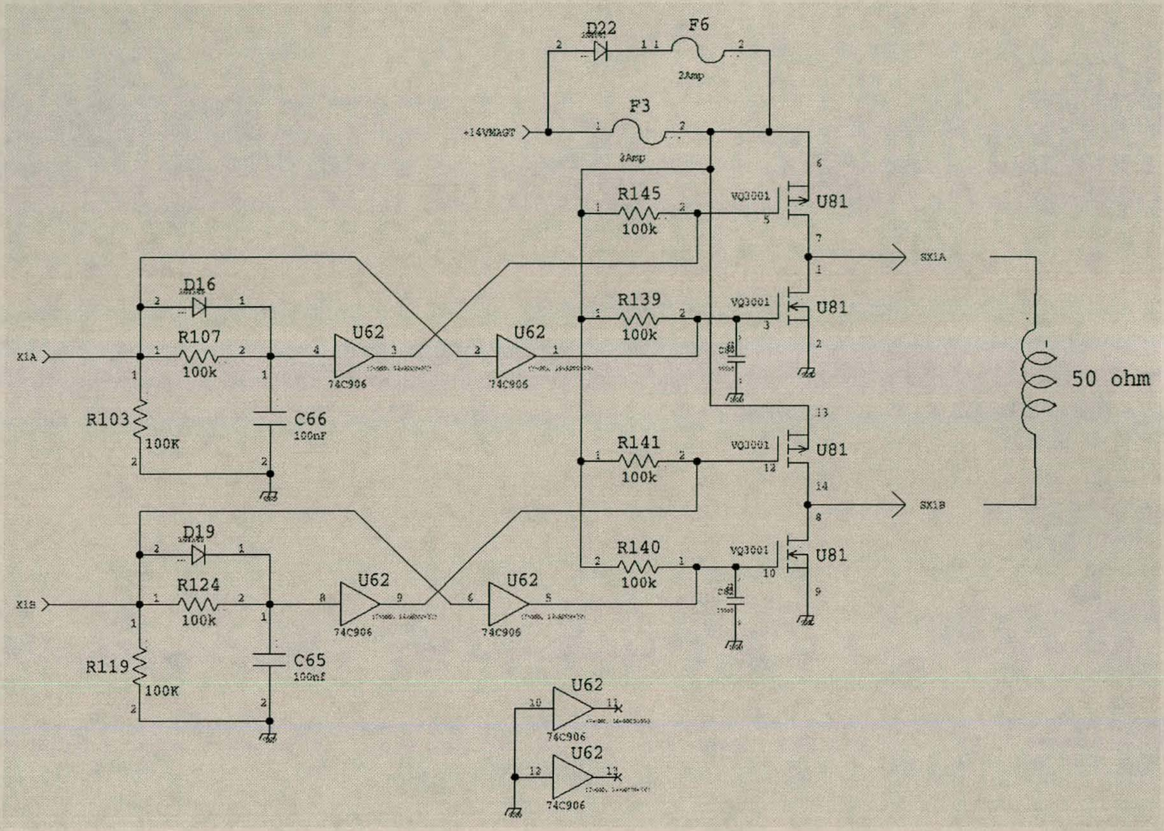


Figure 17: The MOSFET drivers for the X1 magnetic torquer coil.

The current in the coil is controlled by the voltages at points SX1A and SX1B. Both SX1A and SX1B may have one of two discrete voltages, namely ground and 14 volts, depending on how the MOSFET's are switched. If SX1A and SX1B carry the same voltage, either ground or 14 volts, no current flows in the coil and no magnetic dipole moment is generated. If SX1A is at 14 volts and SX1B is at ground, the current flows in the positive direction and a positive magnetic dipole moment is generated. If SX1A is at ground and SX1B is at 14 volts, the current flows in a negative direction and a negative dipole moment is generated. In other words, the magnetic dipole moment generated by coil X1 is a function of the differential voltage between points SX1A and SX1B.

The six magnetic torquer coils interface with the ADCS at the MAGNETORQ plug which is shown in Figure 18.

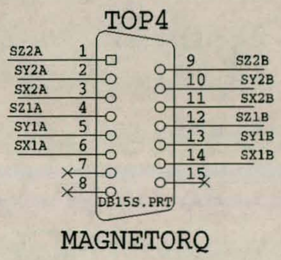


Figure 18: The MAGNETORQ plug.

Coil X1 is connected to SX1A and SX1B, X2 is connected to SX2A and SX2B, Y1 is connected to SY1A and SY1B, Y2 is connected to SY2A and SY2B, Z1 is connected to SZ1A and SZ1B, and Z2 is connected to SZ2A and SZ2B.

Magnetic torquer emulation

The magnetic torquers can be emulated with six 50 ohm dummy loads, a set of op amp difference amplifiers and two custom ISA based ADAS cards. The magnetic torquer coils are replaced with the dummy loads, the differential voltages applied to the dummy loads are measured with six op amp difference amplifiers (one for each coil) and the outputs of the difference amplifiers are acquired by the personal computer via six of the eight A/D channels of the two ADAS cards. As an example, the emulator for the X1 magnetic torquer is shown in Figure 19.

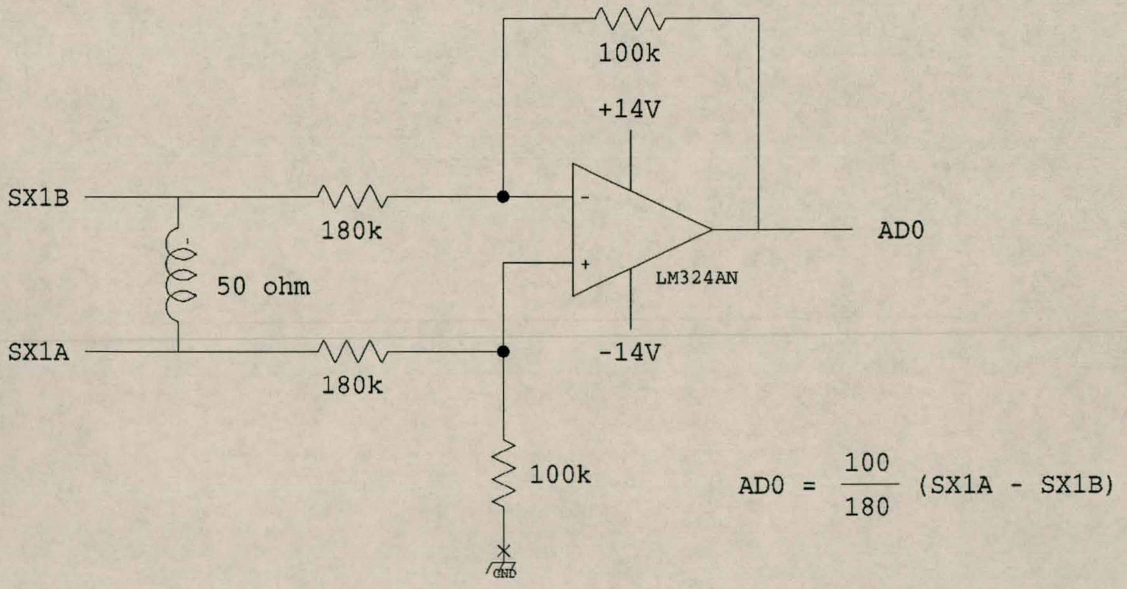


Figure 19: The magnetic torquer emulator for coil X1.

The difference amplifier applies a gain of 100/180 to the difference voltage between SX1A and SX1B. This gain was chosen because the difference voltage may be as high as 14 volts, while the A/D inputs of the ADAS cards can measure only up to 10 volts. Since the difference voltage may be positive or negative, the operational amplifier was also given a split supply of ±14 volts to enable the output voltage AD0 to reflect this sign.

4.3.8 REACTION WHEEL EMULATION

Original reaction wheel interface

The reaction wheels are controlled by two 80C51 microcontrollers. One controls the X and Z1 wheels and the other controls the Y and Z2 wheels. These reaction wheel controllers control the reaction wheels to track speed references supplied by the ICP.

The ICP communicates with the reaction wheel controllers through two 485 UARTS using a simple protocol. The ICP transmits a data packet containing the speed references for the reaction wheels and then waits for data packets from the reaction wheel controllers which contain the measured reaction wheel speeds. The packet from the brush reaction wheels also contains an angular counter measurement and the packet from the brushless reaction wheels contains the current reference measurement.

The ADCS interfaces with the reaction wheels through the POWER TRAY / IMAGER plug which is shown in Figure 20.

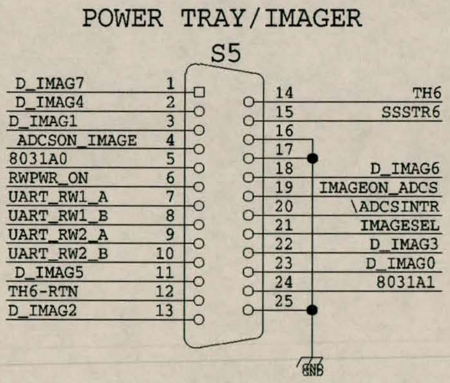


Figure 20: The POWER TRAY / IMAGER plug.

The UART connection with RWC1 is represented by pins UART_RW1A and UART_RW1B and the UART connection with RWC2 is represented by UART_RW2A and UART_RW2B.

Reaction wheel UART emulation

The 485 UART interfaces of the reaction wheel controllers can be emulated with the 232 UART interfaces of the personal computers called Emulator One and Emulator Two and RS485 Interface Transceivers.

4.3.9 SIMULATOR SOFTWARE

Simulator uses the mathematical models for the satellite orbit motion and attitude motion, as well as the space environment models for the geomagnetic field, sun, earth horison and star field to generate simulated sensor data and to determine the effect of the satellite's magnetic torquer and reaction wheel control actions on the motion of the satellite. Simulator transmits simulated sensor data to and receives magnetic torquer and reaction wheel measurements from Emulators one and Two through its two serial port connections at COM1 and COM2.

The structure of the software code which is executed on Simulator is shown in Figure 21. First, communication is established with Emulators One and Two by transmitting 55H to both and waiting for both Emulators to acknowledge with 55H. After communication has been successfully established, Simulator enters its simulation loop. During the simulation loop, the satellite motion and space environment simulation is executed to generate simulated sensor data. The simulated sensor data is then transmitted to Emulators One and Two. Any commands to the ADCS generated by user input is also transmitted to Emulator One, which contains the OBC emulator. Simulator then waits until it receives the ADCS data packet from Emulator One which it, in turn, received from the ADCS via the OBC emulator. This indicates that the Emulators have finished their hardware emulation steps. The magnetic torquer and reaction wheel measurements are then extracted from the ADCS data packet and fed to the satellite motion simulation, closing the simulation loop.

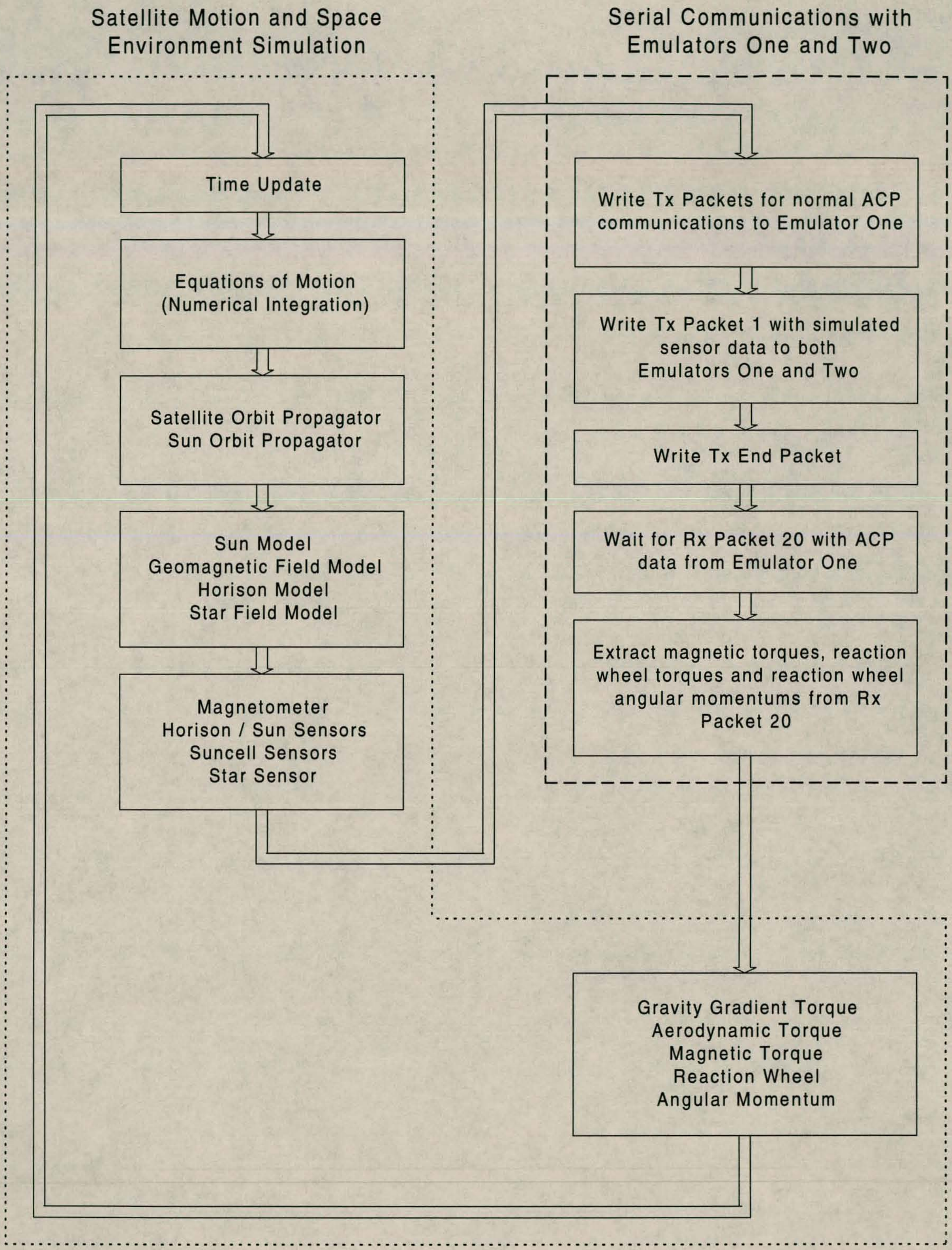

Satellite Motion and Space Environment Simulation
Serial Communications with Emulators One and Two
Time Update
Equations of Motion (Numerical Integration)
Satellite Orbit Propagator
Sun Orbit Propagator
Sun Model
Geomagnetic Field Model
Horison Model
Star Field Model
Magnetometer
Horison / Sun Sensors
Suncell Sensors
Star Sensor
Write Tx Packets for normal ACP communications to Emulator One
Write Tx Packet 1 with simulated sensor data to both Emulators One and Two
Write Tx End Packet
Wait for Rx Packet 20 with ACP data from Emulator One
Extract magnetic torques, reaction wheel torques and reaction wheel angular momentums from Rx Packet 20
Gravity Gradient Torque
Aerodynamic Torque
Magnetic Torque
Reaction Wheel
Angular Momentum

Figure 21: The structure of the software for the Simulator PC.

4.3.10 EMULATOR ONE SOFTWARE

Emulator One contains the OBC emulator, magnetometer emulator, the horison / sun sensor emulator and the magnetic torquer emulators. It receives simulated sensor data and commands for the ADCS from Simulator through its serial port connection. The commands are transmitted to the ADCS through the OBC emulator and the simulated sensor data is written to the sensor emulators. Emulator One also receives a data packet from the ADCS through its OBC emulator and measures the magnetic torquer control actions, which it then transmits back to Simulator.

The structure of the software code which is executed on Emulator One is shown in Figure 22. First, the serial ports are configured and the hardware emulators are initialised. Emulator One then waits for Simulator to initiate communications with it on the COM1 serial port connection. Before acknowledging communications to Simulator, Emulator One uses the OBC emulator to link boot the ADCS and to establish communications with the ADCS by transmitting 55H and waiting for the ADCS to acknowledge with 55H. After the ADCS has been booted and communications with it has been established successfully, Emulator One transmits an acknowledge to Simulator and enters its emulation loop.

During the emulation loop, Emulator One waits for the packets containing the ADCS commands and simulated sensor data from Simulator. When these packets have been received, the simulated sensor data is extracted and written to the sensor emulators, and the ADCS commands are transmitted to the ACP through the OBC emulator. Emulator One then waits for the ADCS to transmit the data packet. When the ADCS data packet has been received, Emulator One retransmits it to Simulator and the emulation loop repeats.

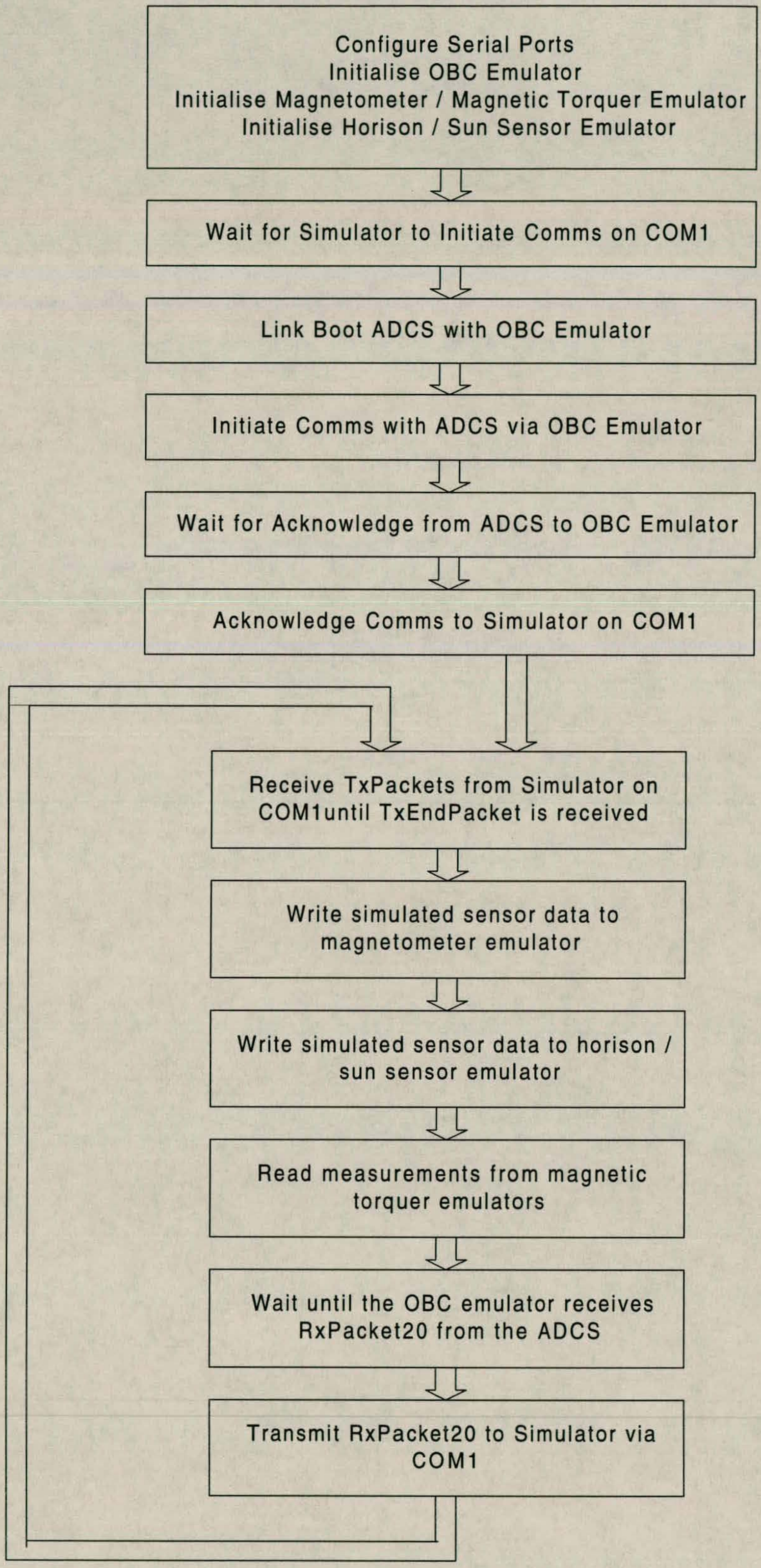

Configure Serial Ports
Initialise OBC Emulator
Initialise Magnetometer / Magnetic Torquer Emulator
Initialise Horison / Sun Sensor Emulator
Wait for Simulator to Initiate Comms on COM1
Link Boot ADCS with OBC Emulator
Initiate Comms with ADCS via OBC Emulator
Wait for Acknowledge from ADCS to OBC Emulator
Acknowledge Comms to Simulator on COM1
Receive TxPackets from Simulator on COM1until TxEndPacket is received
Write simulated sensor data to magnetometer emulator
Write simulated sensor data to horison / sun sensor emulator
Read measurements from magnetic torquer emulators
Wait until the OBC emulator receives RxPacket20 from the ADCS
Transmit RxPacket20 to Simulator via COM1

Figure 22: The structure of the software for Emulator One.

4.3.11 EMULATOR TWO SOFTWARE

Emulator Two contains the star sensor emulator and the sun cell sensor emulator. It receives simulated sensor data from Simulator through its serial port connection and then writes it to the sensor emulators.

The structure of the software code which executes on Emulator Two is shown in Figure 23. First, the serial ports are configured and the sensor emulators are initialised. Then Emulator Two waits for Simulator to initiate communications on the serial port connection. When Simulator initiates communications, Emulator Two simply acknowledges it and enters its emulation loop.

During its emulation loop, Emulator Two executes a simple sequence of operations. It waits for the packet of simulated sensor data from Simulator, extracts the data and writes it to the star sensor and sun cell sensor emulators. Then the loop repeats.

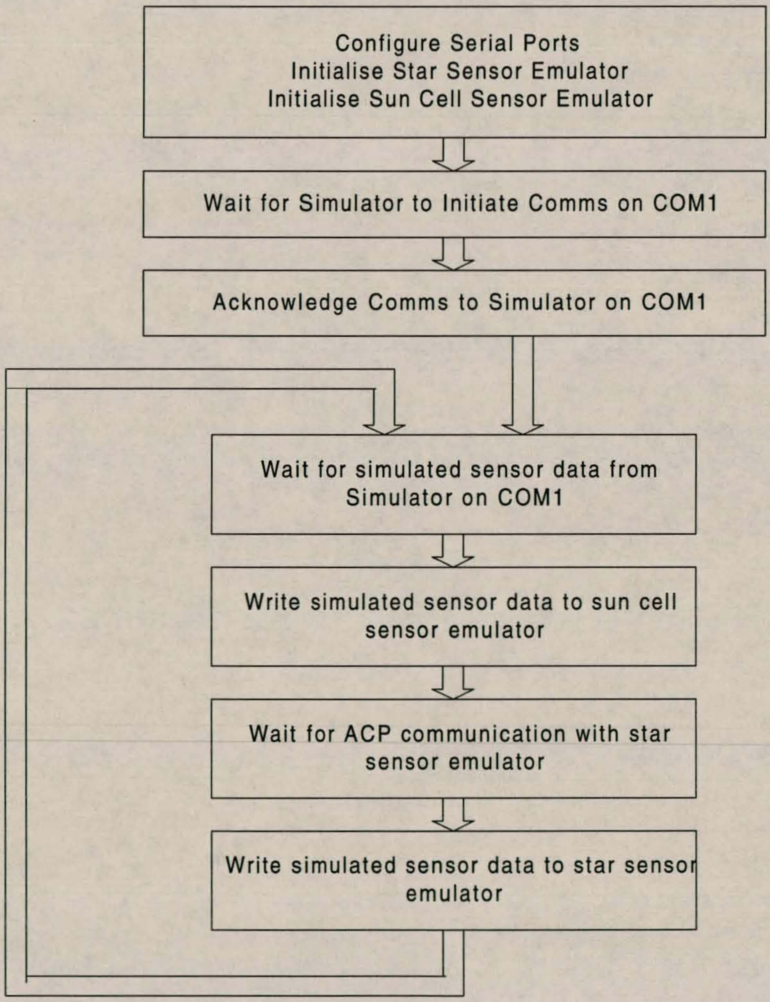

Configure Serial Ports
Initialise Star Sensor Emulator
Initialise Sun Cell Sensor Emulator
Wait for Simulator to Initiate Comms on COM1
Acknowledge Comms to Simulator on COM1
Wait for simulated sensor data from Simulator on COM1
Write simulated sensor data to sun cell sensor emulator
Wait for ACP communication with star sensor emulator
Write simulated sensor data to star sensor emulator

Figure 23: The structure of the software for Emulator Two.

4.4 True Sensor HIL Simulation

Up to this point, the hardware-in-the-loop simulation facility has left the actual sensor hardware out of the loop. The sensors have been replaced by emulators and only the ADCS tray has been included in the loop. The final paragraph in this chapter will describe additions that could be made to the hardware-in-the-loop simulation facility to bring the true sensor hardware into the loop.

4.4.1 CONTRAVES RATE TABLE

The key to bringing the sensors into the loop is the *Contraves Georz* three axis rate table, a rotation platform which is able to track angular rates and positions accurately. Sensors could be mounted on this platform and oriented accurately or rotated at an accurate angular rate and the space environment itself, including the geomagnetic field, sun, earth horison and star field, could then be emulated to provide inputs for the sensor hardware.

An ISA-based General Purpose Interface Bus (GPIB) card which can be used to interface with the *Contraves* rate table already exists and could easily be integrated with Emulator Two. The angular rates and attitude calculated by Simulator could then be used to orient the mounted sensors in such a way that they receive the correct input from the environment emulators.

5. SATELLITE-IN-THE-LOOP SIMULATION

5.1 Introduction

Satellite-in-the-loop simulation is a special case of hardware-in-the-loop simulation where the satellite flight model itself serves as the hardware in the loop.

Data from the flight model is recorded in a whole orbit data (WOD) file by the on-board computer and transmitted to the ground station during a satellite pass. This data may include sensor data from the magnetometer, horison sensors, sun sensor and star sensor, actuator data such as magnetic dipole moments generated by the magnetic torquers and angular momentums and momentum rates of the reaction wheels, data from the on-board orbit propagators and on-board space environment models (geomagnetic field, sun and horison models) and also data from all on-board sensor validation, attitude estimation and actuator control algorithms.

The sensor data in the WOD file is used to replace the sensor simulations presented in chapter 2 to serve as inputs for the ADCS software simulation presented in chapter 3. The sensor validation, attitude estimation, magnetic torquer control and reaction wheel control algorithms of the ADCS software simulation may then operate on the whole orbit data. In this way, attitude information may be extracted from the satellite's sensors using the software simulation. If the outputs of the on-board sensor validation, attitude estimation or actuator control algorithms are included in the WOD file, then they may be compared to the outputs of their counterparts in the ADCS software simulation to verify that the on-board algorithms are functioning correctly. If the WOD file includes data from the on-board orbit propagators or geomagnetic field, sun and horison models, then the environment simulation may also be executed in parallel with the WOD file to verify the correct operation of the on-board orbit and environment models.

The loop is closed when attitude information obtained from and on-board software errors revealed by the analysis of the WOD file influences decisions concerning the operation of the satellite.

This chapter describes how the software simulation presented at the beginning of chapter 3 was modified to analyse whole orbit data and also how this satellite-in-the-loop simulation software was used to investigate an unmodelled disturbance torque which was acting on the satellite body.

5.2 Analysis of Whole Orbit Data

The simulation software presented at the beginning of chapter 3 was modified so that whole orbit data recorded by the SUNSAT flight model could be analysed on the ground using the simulation of the ADCS software. The modification was to replace the simulated sensor data generated by the sensor simulations of chapter 2 with the true sensor data as recorded in the WOD file. The structure of the resulting whole orbit data analysis software is shown in Figure 24.

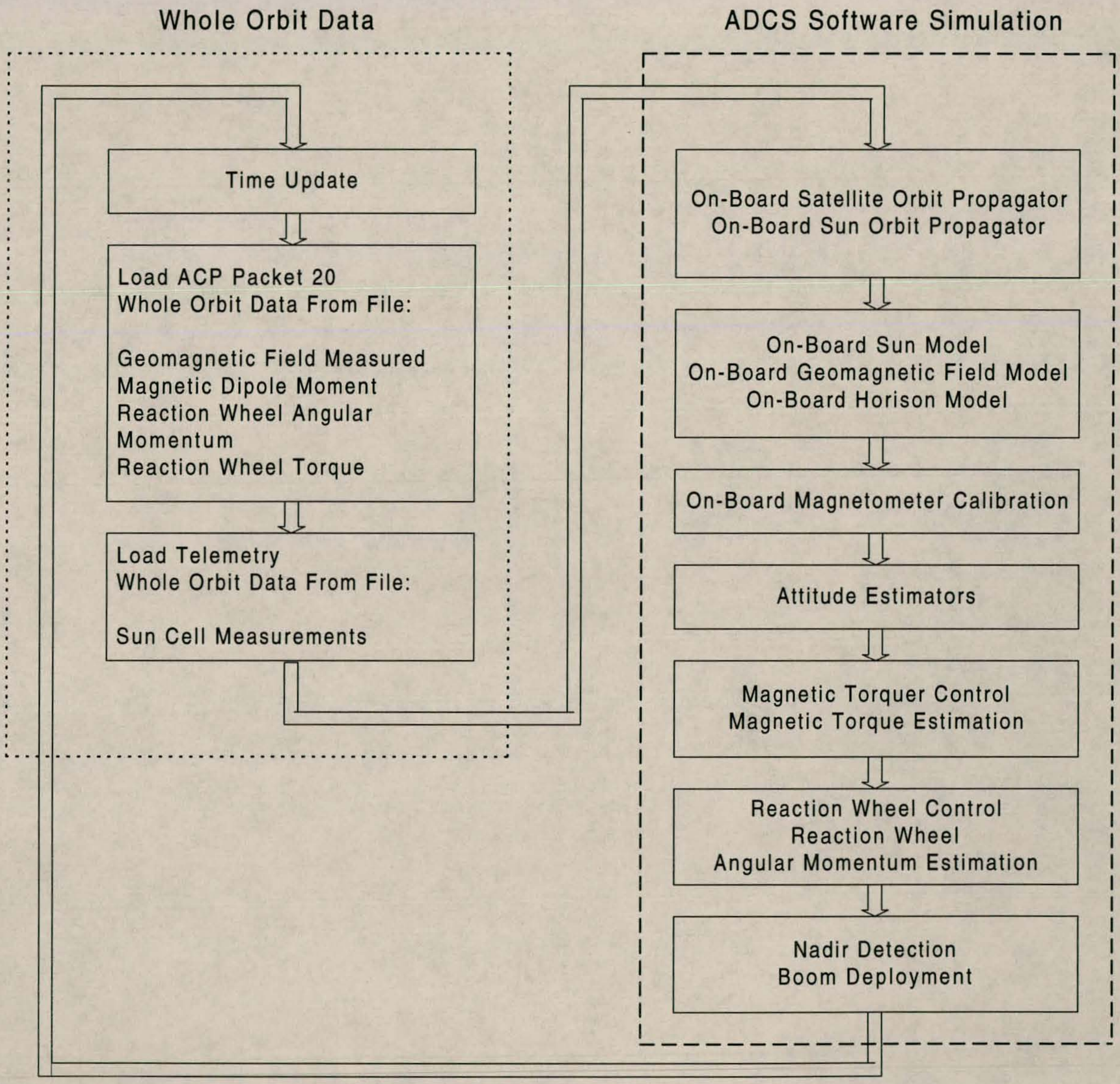


Figure 24: The structure of the whole orbit data analysis software.

The UTC time is calculated first and then used to index and read the appropriate entry of whole orbit data from file. There are two possible sources of whole orbit data, the first being the software telemetry the OBC receives from the ACP and the second being the hardware telemetry the OBC receives from the telemetry tray. The software telemetry variables used by the WOD Analysis Mode include the geomagnetic field vector as measured by the magnetometer, the magnetic dipole moment generated by the magnetic torquers, the torque applied to the reaction wheels and the resulting reaction wheel angular momentum. The only hardware telemetry variables used by the WOD Analysis Mode are the sun cell sensor measurements which are not sampled by the software telemetry. The whole orbit data variables are fed to the PC simulation of the ADCS software which follows the same sequence of routines as the original Software Simulation Mode. After the ADCS software algorithms have been applied to the WOD data, time is incremented and the loop is repeated.

In this way, sensor data can be analysed on the ground to extract attitude information and the correct operation of all on-board ADCS algorithms can also be verified. Beside these two major applications of the software, the WOD Analysis Mode was also used for a third purpose: to investigate the appearance of an unmodelled disturbance torque.

5.3 Investigation of an Unmodelled Disturbance Torque

An investigation into an unmodelled disturbance torque acting on the satellite body was prompted by observations that SUNSAT was turning upside down in an unusually short period of time (approximately two days) when the satellite was left in an uncontrolled state.

A software simulation of uncontrolled satellite motion initialised with an initial roll and pitch of 20° each and an initial Z-spin rate of 5 revolutions per orbit shows that the amplitude of roll and pitch librations should not increase significantly and the Z-spin rate should decrease only slightly over a period of 30 orbits (just over two days). The simulated roll, pitch and yaw angles are shown in Figure 25 and the simulated orbit angular rates are shown in Figure 26.

An analysis of 14 orbits of whole orbit data recorded from 1999 August 2, 11:28:31 to 1999 August 3, 12:29:01 UTC confirmed these simulation results. The estimated roll, pitch and yaw angles from the whole orbit data analysis are shown in Figure 27 and the estimated orbit angular rates are shown in Figure 28.

However, an analysis of whole orbit data recorded from 1999 August 4, 11:44:39 to 1999 August 5, 9:28:08 UTC showed the amplitude of SUNSAT's roll and pitch librations increasing and its Z-spin rate decreasing to zero in only 24 hours. This indicated the presence of a disturbance torque which was not modelled by the software simulation and which had not been present when the first set of whole orbit data was recorded. The estimated euler angles exhibiting the unmodelled disturbance torque is shown in Figure 29 and the estimated orbit angular rates are shown in Figure 30.

The following four possible sources of the unmodelled disturbance torque were identified: unmodelled aerodynamic effects, residual magnetism of the satellite body, magnetic dipole moments generated by current loops in the solar panels, or magnetic torquers which mistakenly remained active after the ADCS had deactivated them.

A system identification feature, using the Magnetometer EKF, was added to the simulation software to determine which one of these sources was the culprit.

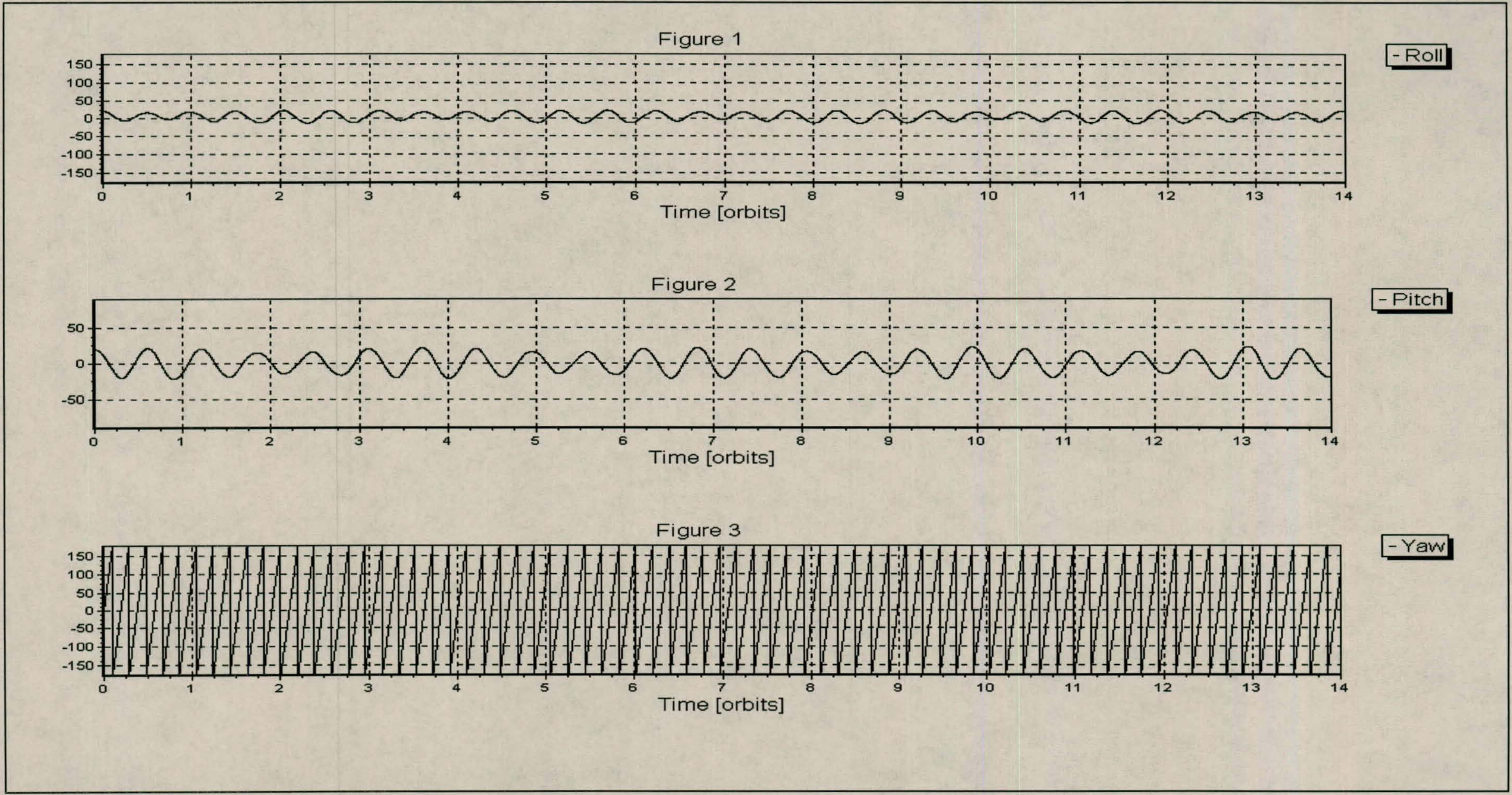


Figure 25: The euler angles predicted by the simulation.

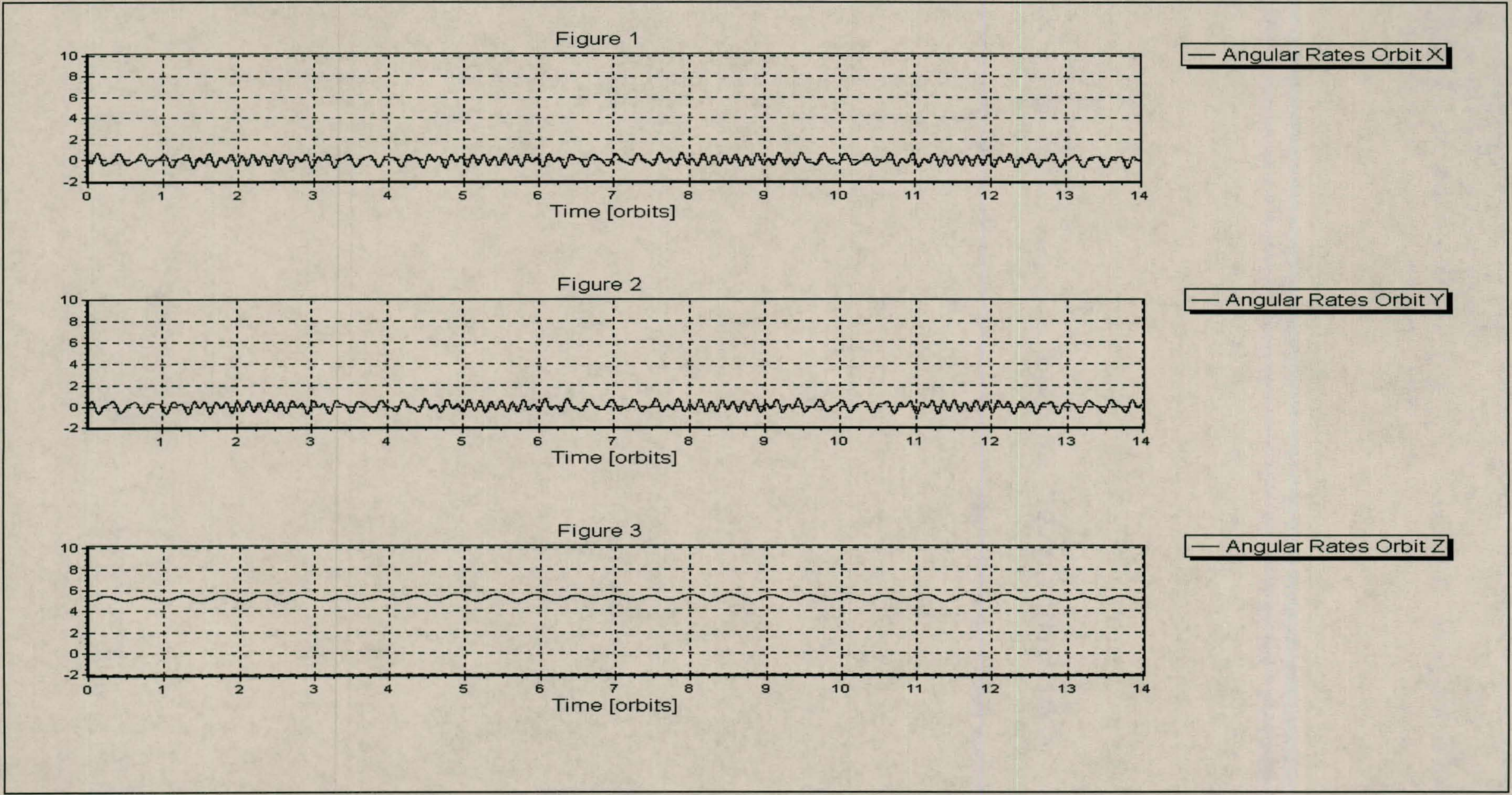


Figure 26: The orbit angular rates predicted by the simulation.

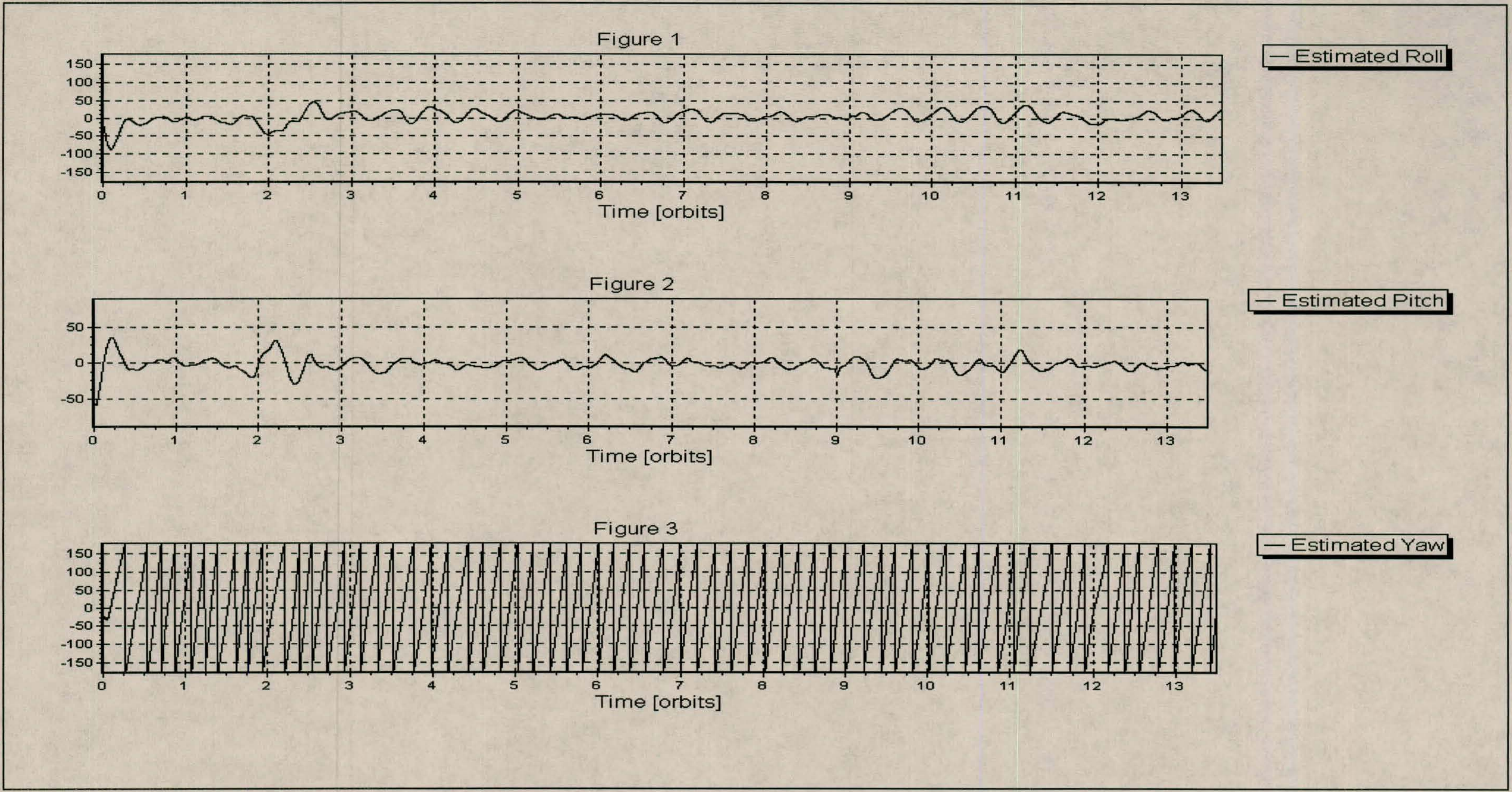


Figure 27: The estimated euler angles from a whole orbit data analysis which exhibits very little unmodelled disturbance torque.

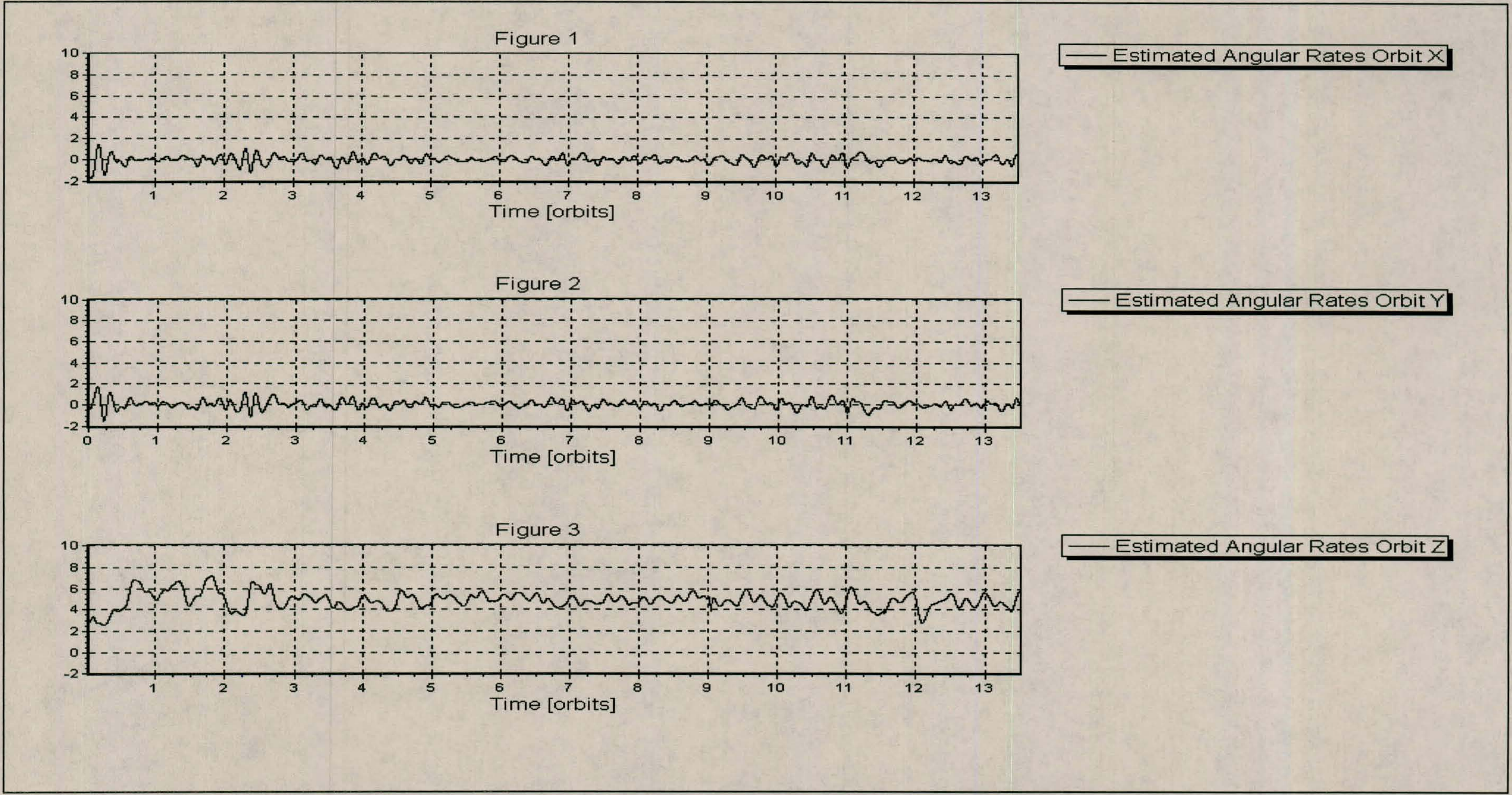


Figure 28: The estimated orbit angular rates of the same whole orbit data analysis which exhibits very little unmodelled disturbance torque.

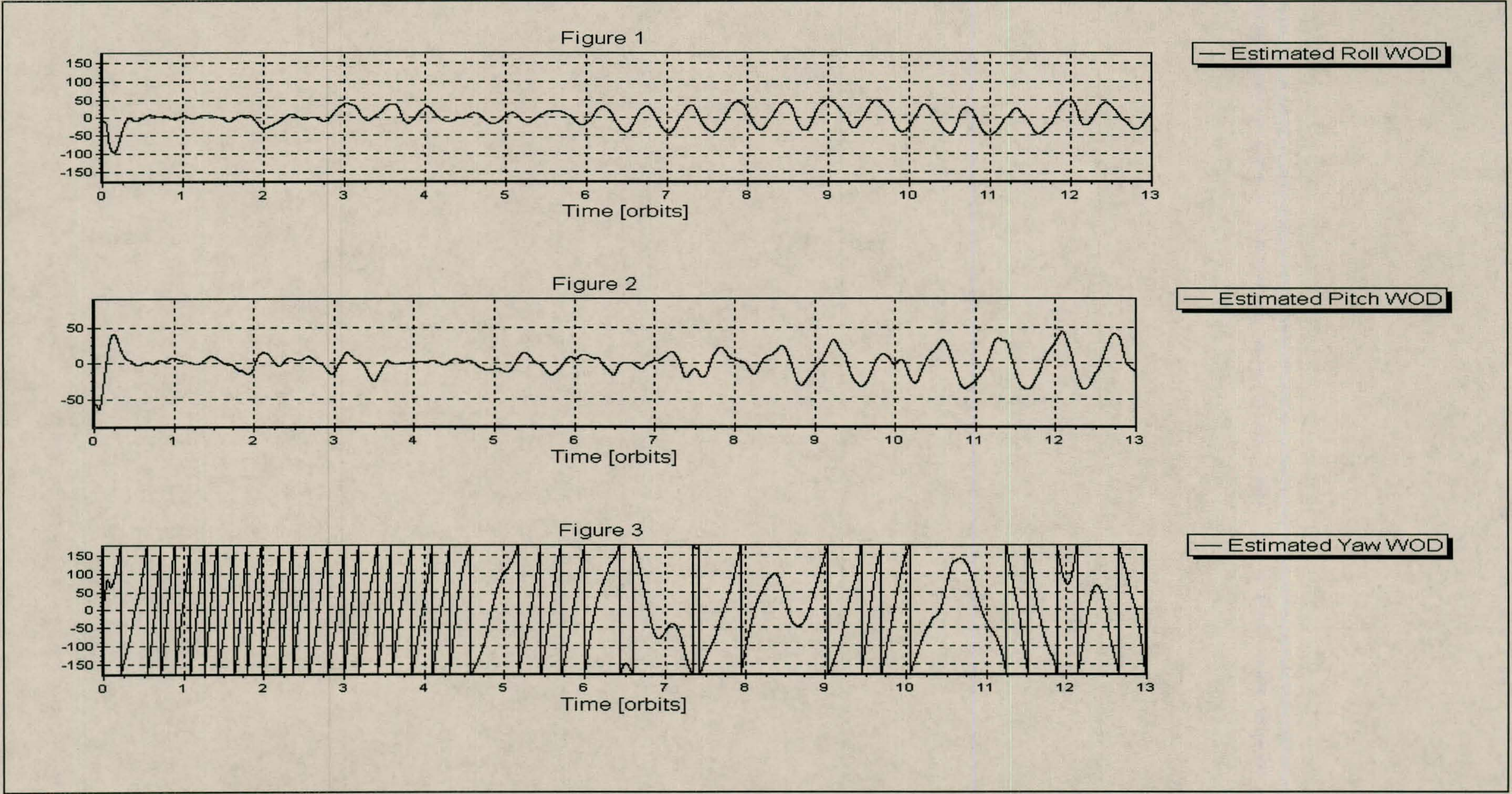


Figure 29: The estimated euler angles from a whole orbit data analysis which exhibits a significant unmodelled disturbance torque.

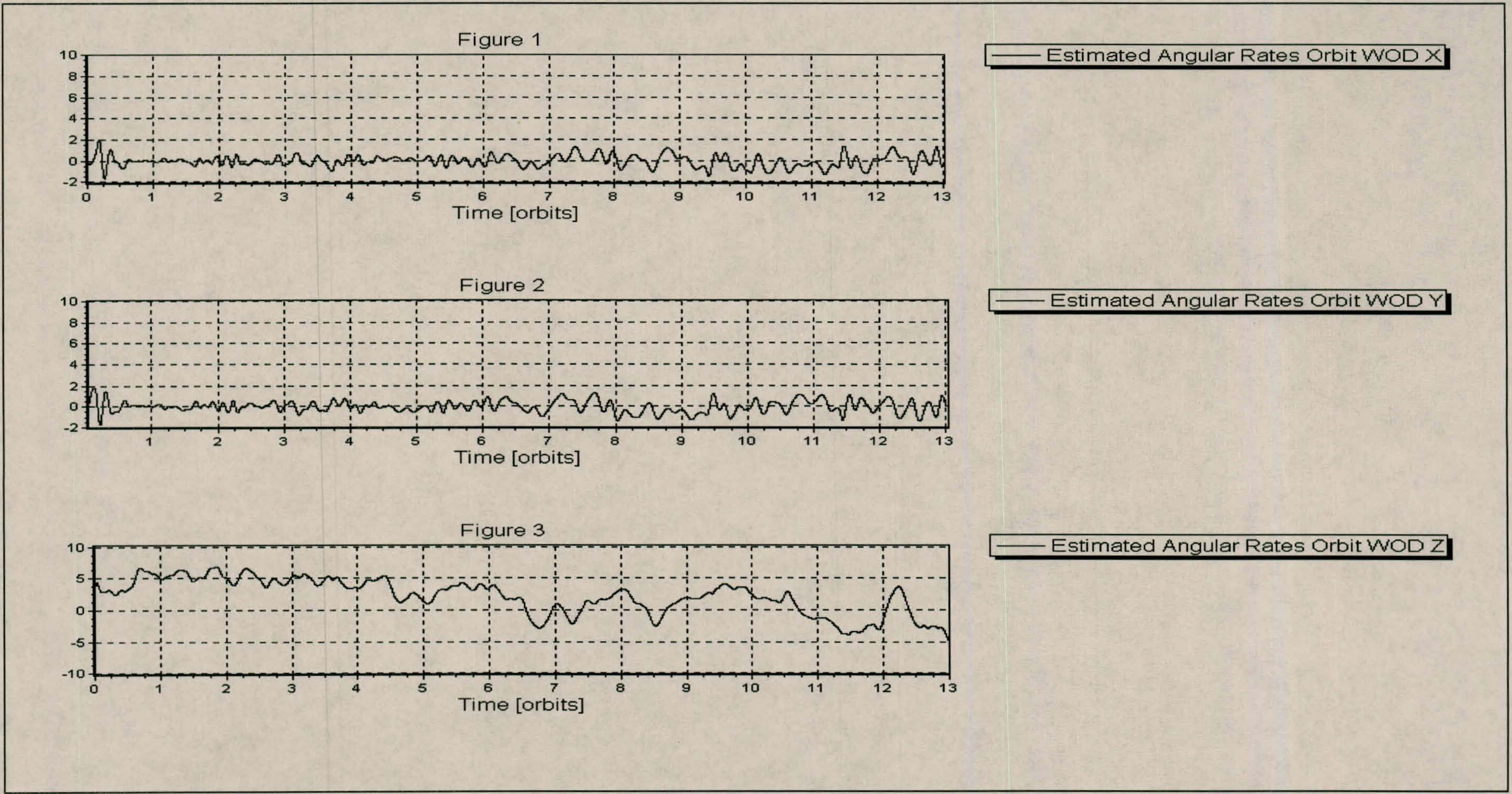


Figure 30: The estimated orbit angular rates from the whole orbit data analysis which exhibits a significant unmodelled disturbance torque.

5.3.1 SYSTEM IDENTIFICATION MODE

To identify the source of the disturbance torque, each one of the four possible unmodelled disturbance torques was added in turn to the equations of motion propagated by the Magnetometer EKF (Filter 1). The addition of the correct model for the unmodelled disturbance torque to the equations of motion would decrease the error energy in the innovation error vector of the Magnetometer EKF since the filter would be predicting the satellite states more accurately and less measurement corrections would be necessary. The unmodelled disturbance torque which produced the minimum error energy in the innovation error vector of the EKF would therefore be the most likely candidate. The structure of the System Identification Mode software which uses this principle is shown in Figure 31.

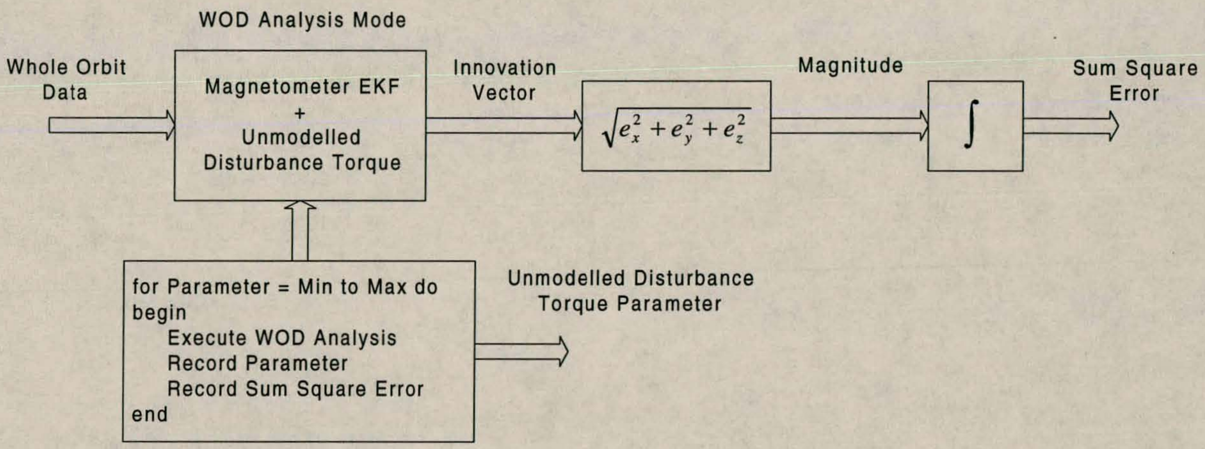


Figure 31: The structure of the system identification software.

The structure of the unmodelled disturbance torque which is added to the Magnetometer EKF can be deduced from its suspected source, but since the magnitude of the unmodelled disturbance torque is unknown, the system identification mode must identify the magnitude parameter which results in the minimum innovation vector error energy and provides the best fit for the whole orbit data. The System Identification Mode executes the following sequence:

1. The magnitude parameter is initialised with its minimum value.
2. A WOD Analysis is done by feeding the WOD file exhibiting the effects of the unmodelled disturbance torque to the Magnetometer EKF with the suspected disturbance torque source model added to its equations of motion.
3. The sum square error of the innovation vector error series resulting from this WOD Analysis is calculated and recorded along with the magnitude parameter value.
4. Both the WOD Analysis simulation and the WOD file are reset.

5. The magnitude parameter is incremented and the loop is repeated until the parameter reaches its maximum value.

After the System Identification Mode simulation has terminated, the innovation error vector sum square error is plotted as a function of the magnitude parameter and the parameter value which produces the minimum innovation vector energy is determined visually.

Some might argue that a gradient-based minimisation algorithm would be a more efficient use of processing power than the "brute force" parameter stepping and visual inspection used by this algorithm. However, gradient-based system identification algorithms easily converge to local minima especially when the system to be identified is non-linear. Since the equations of motion used by the Magnetometer EKF, as well as the unmodelled disturbance torque, are both described by non-linear equations, a gradient-based algorithm would definitely experience difficulties. The parameter stepping method, on the other hand, evaluates the entire parameter space and reveals all local minima. Although parameter stepping is much more processing intensive, with the amount of processing time required increasing exponentially with the dimension of the parameter vector, the magnitude parameter of the unmodelled disturbance torque is one dimensional and would not demand an unreasonable amount of processing time. For these reasons, it was decided to use parameter stepping instead of a gradient-based minimisation algorithm.

5.3.2 UNMODELLED AERODYNAMIC DISTURBANCE TORQUE

Simulations show that if the magnitude of the aerodynamic torque were much larger than it is currently modelled, it would be possible for the satellite to overturn within 14 orbits.

The mathematical model for the aerodynamic disturbance torque, $\mathbf{N}_{AERO}$ has already been presented. The unmodelled disturbance torque was postulated to be the modelled aerodynamic torque multiplied by the scalar magnitude parameter.

$$\mathbf{N}_{D,\text{unmodelled}} = K\mathbf{N}_{AERO} \qquad \textbf{(Eq. 5.1)}$$

where,

	$\mathbf{N}_{D,\text{unmodelled}}$	is the unmodelled disturbance torque,
	K	is the magnitude parameter,
and	$\mathbf{N}_{AERO}$	is the modelled aerodynamic disturbance torque.

This postulated aerodynamic torque model was added to the equations of motion of the Magnetometer EKF and the system identification mode software was applied to the whole orbit data exhibiting the unmodelled disturbance torque. The resulting magnitude parameter vs. innovation vector energy plot is shown in Figure 32.

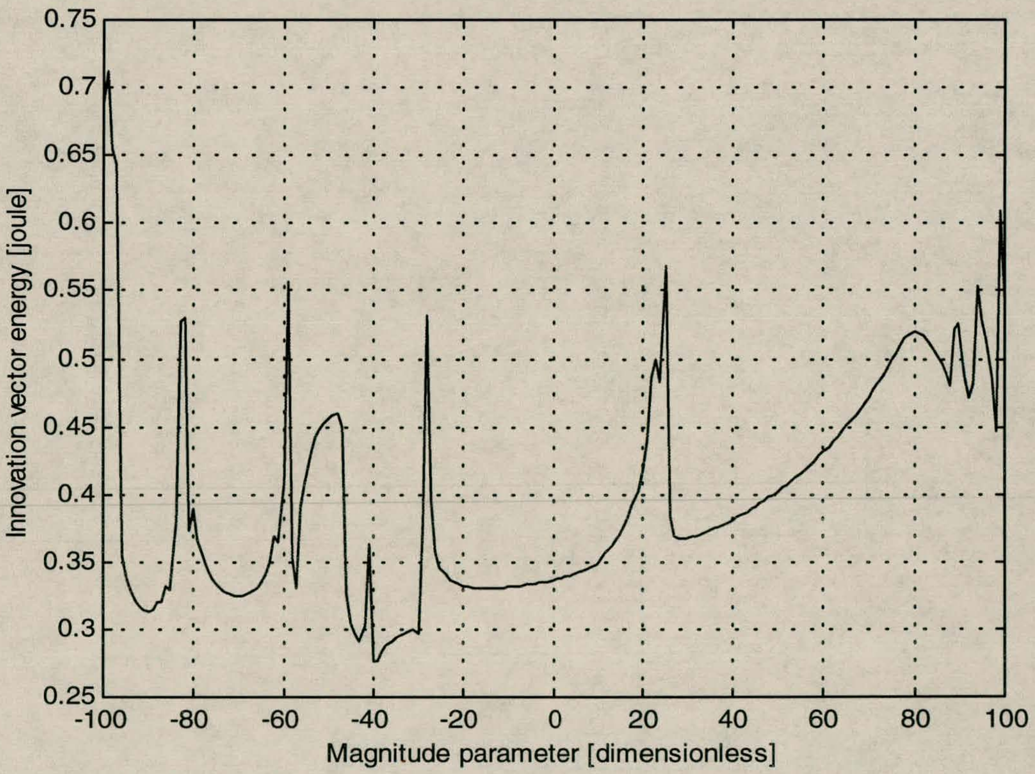


Figure 32: The system identification results for a postulated unmodelled aerodynamic torque.

Figure 32 shows that if the unmodelled disturbance is a unmodelled aerodynamic torque then the minimum innovation vector energy for this set of whole orbit data is 0.2768 joule and occurs at a magnitude parameter value of –40. This result suggests that if an aerodynamic torque is the culprit, it would be forty times larger and in the opposite direction from what the mathematical model predicts.

5.3.3 SOLAR PANEL MAGNETIC DIPOLE MOMENTS

SUNSAT is equipped with four solar panels, one on each side face of its cubed body. The solar cells of each solar panel are grouped in vertical strings which are connected in series. Whenever a solar panel is illuminated, the currents generated by its solar cells flow in these strings. If the currents flowing in the solar cell strings form current loops, the solar panel would generate an unwanted magnetic dipole moment which would in turn cause an unmodelled magnetic disturbance torque. This solar panel magnetic dipole moment would vary according to the direction and magnitude of the sun vector illuminating the solar panels.

The unmodelled disturbance torque could be postulated to be this solar panel magnetic disturbance torque. The solar panel magnetic disturbance torque is obtained by taking the cross product between the measured geomagnetic field and the solar panel magnetic dipole moment.

$$\mathbf{N}_{D,\text{unmodelled}} = \mathbf{N}_{\text{solar panel}} \qquad \textbf{(Eq. 5.2)}$$

with,

$$\mathbf{N}_{\text{solar panel}} = \mathbf{M}_{\text{solar panel}} \times \mathbf{B}_{\text{measured}} \qquad \textbf{(Eq. 5.3)}$$

where,

	$\mathbf{N}_{\text{solar panel}}$	is the modelled solar panel magnetic disturbance torque,
	$\mathbf{M}_{\text{solar panel}}$	is the modelled solar panel magnetic dipole moment,
and	$\mathbf{B}_{\text{measured}}$	is the measured geomagnetic field.

The solar panel magnetic dipole moment is modelled as having the same direction as the estimated sun vector in body coordinates, but with the z-component zeroed since there are no solar panels on the top and bottom faces of the cubed satellite body.

$$\mathbf{M}_{\text{solar panel}} = K\hat{\mathbf{S}}_{body} \qquad \textbf{(Eq. 5.4)}$$

with,

$$\hat{\mathbf{S}}_{body} = \hat{\mathbf{A}}\mathbf{S}_o$$

where,

K	is the magnitude parameter,
$\hat{\mathbf{S}}_{body}$	is the estimated sun vector in body coordinates,

	$\hat{\mathbf{A}}$	is the direction cosine matrix as estimated by the magnetometer EKF,
and	$\mathbf{S}_o$	is the sun vector in body coordinates as modelled by the on-board sun model.

Furthermore, the solar panel magnetic dipole moment is only present when the solar panels are illuminated by the sun, which means that there has to be a valid line-of-sight between the satellite and the sun.

This postulated solar panel magnetic torque model was added to the equations of motion of the Magnetometer EKF and the system identification was repeated. The resulting magnitude parameter vs. innovation vector energy plot is shown in Figure 33.

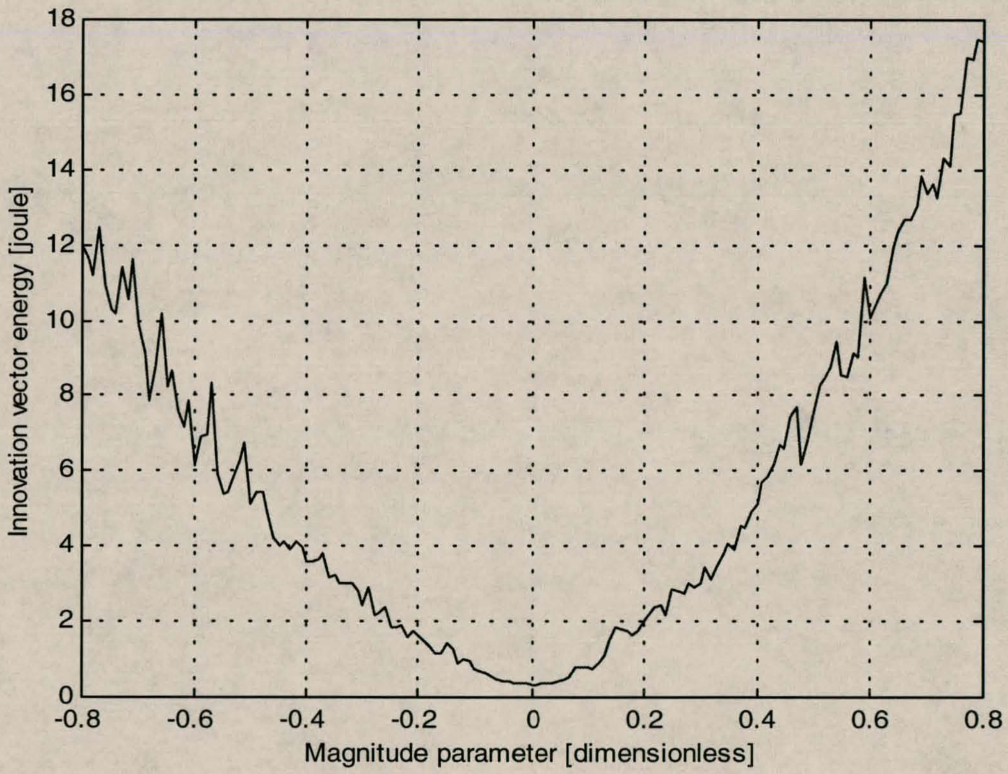


Figure 33: The system identification results for a postulated solar panel magnetic disturbance torque.

The results show that if the unmodelled disturbance torque was a solar panel magnetic torque then a magnitude parameter of zero would give the minimum innovation vector energy of 0.3365 joule. The fact that a solar panel magnetic dipole moment of zero best fits the whole orbit data suggests that the source of the unmodelled disturbance torque is not the solar panels.

5.3.4 RESIDUAL MAGNETISM / ACTIVE MAGNETIC TORQUERS

A constant magnetic dipole moment could also be the source of the unmodelled disturbance torque. This dipole moment could either be the result of residual magnetisation of the satellite body or it could be generated by magnetic torquers which remain active due to hardware or software errors. In both these cases the unmodelled disturbance torque is modelled by taking the cross product between a constant magnetic dipole moment and the measured geomagnetic field.

$$\mathbf{N}_{\text{residual magnetism}} = \mathbf{M}_{\text{constant}} \times \mathbf{B}_{\text{measured}} \qquad \textbf{(Eq. 5.5)}$$

where,

	$\mathbf{N}_{\text{residual magnetism}}$	is the residual magnetism / active magnetic torquers disturbance torque
and	$\mathbf{M}_{\text{constant}}$	is the constant magnetic dipole moment.

To minimise the amount of processing time required for the parameter stepping system identification algorithm, the magnitude parameter is kept one dimensional by postulating the direction of the magnetic dipole moment and only varying its magnitude. The three directions which were postulated were the x, y and z body axes respectively.

$$\mathbf{M}_{\text{constant}} = K\mathbf{u} \qquad \text{, where } \mathbf{u} = \mathbf{i}, \mathbf{j}, \mathbf{k}. \qquad \textbf{(Eq. 5.6)}$$

where,

	K	is the magnitude of the magnetic dipole moment
and	$\mathbf{u}$	is a unit vector with the postulated direction of the magnetic dipole moment.
and	$\mathbf{i}, \mathbf{j}, \mathbf{k}$	are unit vectors in the directions of the x, y and z body axes.

The results of the system identifications done for postulated magnetic dipole moments in directions of the x, y and z body axes are shown in Figure 34.

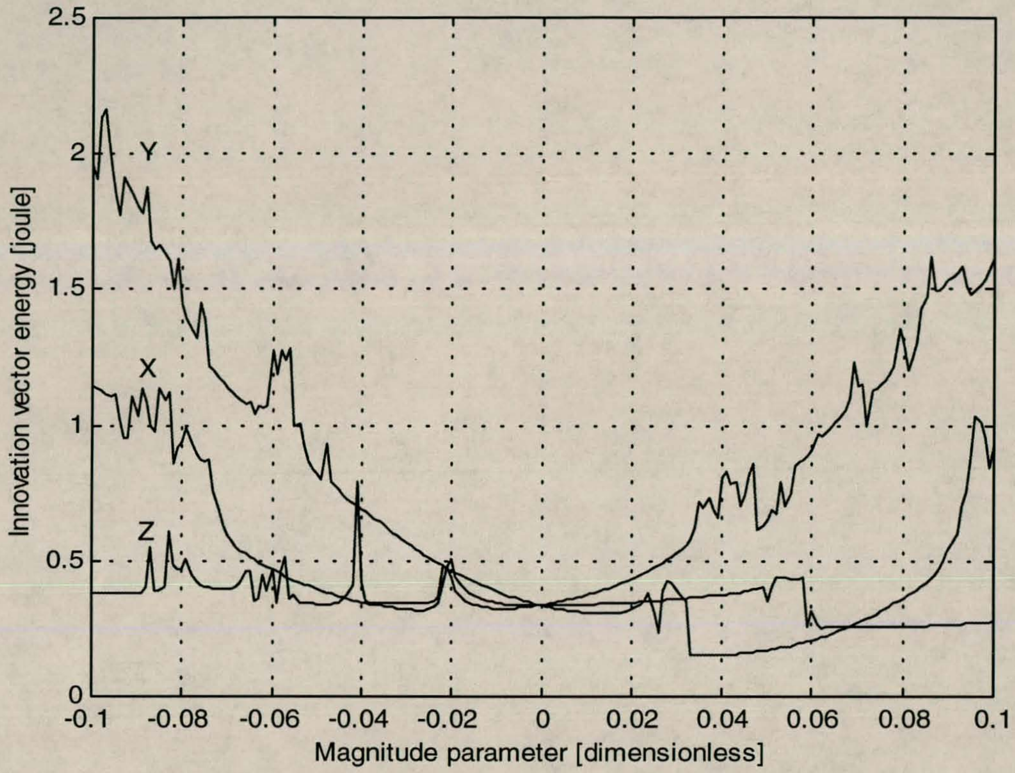


Figure 34: The system identification results for a postulated residual magnetism / active magnetic torquers disturbance torque.

The minimum innovation vector energies and their corresponding magnitude parameters for the constant magnetic dipole moments postulated in the directions of the x, y and z body axes were determined visually from Figure 34 and are summarised in Table 5.

Table 5: The minimum innovation vector energies and corresponding magnitude parameters for postulated constant magnetic dipole moments in the directions of the X, Y and Z body axes.

Postulated magnetic dipole moment direction	*Minimum innovation vector energy*	*Magnitude parameter at minimum energy*
X	0.3173	-0.026
Y	**0.1512**	**0.037**
Z	0.2515	0.062

The results indicated that a constant magnetic dipole moment in the direction of the y body axis with a magnitude of 0.037 results in the smallest minimum innovation vector energy and is the best fit for the whole orbit data.

5.3.5 CONCLUSIONS OF THE INVESTIGATION

The minimum innovation vector energies for the system identifications done with all three postulated unmodelled disturbance torques, namely the aerodynamic torque, the solar panel magnetic torque and the residual magnetism / active magnetic torquers disturbance torque are summarised in Table 6.

Table 6: A summary of the system identification results for all postulated sources of the unmodelled disturbance torque.

Postulated unmodelled disturbance torque	*Minimum innovation vector energy*
Aerodynamic torque	0.2768
Solar panel magnetic torque	0.3365
Residual magnetism / Active magnetic torquers	**0.1512**

The system identification results show that the smallest minimum innovation vector energy is obtained when the source of the unmodelled disturbance torque is postulated to be a constant magnetic dipole moment directed along the y body axis and possibly caused by either residual magnetism or magnetic torquers which mistakenly remain active.

Since the magnitude of the postulated aerodynamic torque which fits the whole orbit data is forty times greater and in the opposite direction from the aerodynamic torque predicted by theory, it is an unlikely candidate for the unmodelled disturbance torque.

The system identification also determined that a postulated solar panel magnetic torque with a magnitude of zero gave the best fit for the whole orbit data, so it is clearly also not the source of the unmodelled disturbance torque.

After this process of elimination only the residual magnetism and the mistakenly active magnetic torquers, which share the same mathematical model, are left. This conclusion is also supported by the global system identification results shown in Table 6 which indicate that a constant magnetic dipole moment aligned to the y body axis direction and with a magnitude of 0.037 provides the best fit for the whole orbit data of all postulated disturbance torques.

At this point it is interesting to note that the unmodelled disturbance torque exhibited by the whole orbit data of recorded from 1999 August 4, 11:44:39 UTC to 1999 August 5, 9:28:08 UTC, is conspicuously absent from the data recorded from 1999 August 2, 11:28:31 UTC to 1999 August 3, 12:29:01, which is only 24 hours earlier. This means that the magnitude and / or direction of the constant magnetic dipole moment changed from the one set of whole orbit data to the next. It is unlikely that a magnetic dipole moment caused by residual magnetisation of the satellite body would have changed so rapidly. A magnetic dipole moment caused by mistakenly active magnetic torquers, however, would certainly have changed if a magnetic torquer control algorithm had been activated somewhere between the two recordings of whole orbit data. An inspection of the session logs of communications with SUNSAT revealed that magnetic torquer controller 7 was indeed active from 1999 August 3, 21:55:59 (timestamp 933717359) to 1999 August 4, 11:41:09 (timestamp 933766869). The SUNSAT engineer heading the ADCS team also found an error in the ACP software code which would cause the magnetic torquers to retain their last commanded magnetic dipole moment when they are commanded to deactivate. Mistakenly active magnetic torquers are therefore the most likely source of the unmodelled disturbance torque.

It is the conclusion of this investigation that the most likely source of the unmodelled disturbance torque perceived in the whole orbit data is a magnetic dipole moment generated by the magnetic torquers themselves which mistakenly remain active due to a code error in the ACP software.

5.3.6 FURTHER INVESTIGATIONS

The conclusion reached by this investigation was based on two sets of whole orbit data. Subsequent tests which have been performed on the flight model to confirm this theory have been somewhat inconclusive, but do indicate that the magnetic torquers may not be the only culprit. There have also been speculations that the permanent magnets in the reaction wheels and in the stepper motor of the imager may be the source of the constant magnetic dipole moment. Further investigations are therefore recommended to confirm the source of the unmodelled torque.

6. CONCLUSION

6.1 Summary of Contributions

6.1.1 SIMULATION SOFTWARE

Simulation software was created to simulate the satellite motion and space environment, as well as the on-board ADCS software, using mathematical models used by Steyn, Wertz and Jacobs. This software simulation was written in the Delphi 3 programming language and is based on an earlier simulation program which was written by Steyn in Turbo Pascal 5.5.

The simulation software can be used to analyse and design new ADCS algorithms, including attitude estimators, magnetic torquer controllers, reaction wheel controllers and on-board sensor calibration algorithms. The simulation is also useful for mission planning and evaluation, since the sequence of control actions required by a certain mission scenario may be simulated and the results evaluated to modify and refine the mission sequence.

The software simulation also serves as the basis for the hardware-in-the-loop simulation facility. The simulated sensor data generated by the software is fed to sensor emulators connected to the ADCS tray and the magnetic torquer and reaction wheel control actions measured by the actuator emulators are fed back to the software to determine their effect on the satellite motion.

The satellite-in-the-loop simulation is also based on the software simulation. The sensor data and magnetic torquer and reaction wheel control actions recorded by the flight model in a whole orbit data file are fed to the software simulation of the on-board ADCS software. The data generated by the on-board orbit propagators and space environment models may also be compared to their counterparts in the software simulation.

6.1.2 SENSOR AND ACTUATOR HARDWARE EMULATION

Hardware emulators were developed to emulate the interfaces of the sensors and actuators with the ADCS tray. The sensor emulators include emulators for the magnetometer, horison / sun sensor piggyback, star sensor and sun cell sensors. The actuator emulators include emulators for the magnetic torquers and the reaction wheels.

The magnetometer emulator consists of two custom ADAS cards, the horison / sun sensor piggyback emulator is a single custom FPGA card, the star sensor emulator is a single custom transputer cards and the sun cell sensor emulator consists of a single custom ADAS card and an operational amplifier circuit. The magnetic torquers can be emulated with the same two custom ADAS cards used by the magnetometer emulator combined with a set of op amp difference amplifiers. The serial interfaces with the reaction wheel driving circuits can be emulated with two IBM PC serial ports combined with RS485 Interface Transceivers.

The interface between the ADCS and the on-board computer was also emulated using a custom transputer card. This OBC emulator already existed and was used by Steyn to operate the ADCS in its simulation mode.

6.1.3 HARDWARE-IN-THE-LOOP SIMULATION FACILITY

A hardware-in-the-loop simulation facility was constructed from the software simulation and the hardware emulators. The facility consists of three personal computers which are connected to each other via their serial ports. The first computer, a 300MHz Pentium II called Simulator, was used to run the software simulation. The other two personal computers, a 60 MHz Pentium called Emulator One and a 486 called Emulator Two, were used to emulate the sensors and actuators as well as the primary on-board computer (OBC1). Emulator One contains the sensor emulators for the magnetometer and the horison / sun sensor piggyback, the actuator emulators for the magnetic torquers and the Y and Z_1 reaction wheels and also the OBC emulator. Emulator Two contains the sensor emulators for the star sensor and the sun cell sensors, and the actuator emulators for the X and Z_2 reaction wheels.

6.1.4 SATELLITE-IN-THE-LOOP SIMULATION SOFTWARE

The software simulation was modified to create a satellite-in-the-loop simulation which could be used to analyse whole orbit data recorded by the flight model. This satellite-in-the-loop simulation software was used to analyse sensor data on the ground to extract attitude information and to verify the correct operation of all on-board ADCS algorithms. The software was also used to investigate the appearance of an unmodelled disturbance torque acting on the satellite body.

6.1.5 INVESTIGATION OF AN UNMODELLED DISTURBANCE TORQUE

The effects of an unmodelled disturbance torque acting on the satellite body were observed in the sensor data of the flight model. This prompted an investigation to find the source of the torque. The following four possible sources were identified: unmodelled aerodynamic effects, residual magnetism of the satellite body, magnetic dipole moments generated by current loops in the solar panels, or magnetic torquers which mistakenly remained active after the ADCS had deactivated them.

A system identification feature was added to the satellite-in-the-loop simulation software to determine which one of these sources was the culprit. The mathematical model of the postulated source would be added to the system model of the Magnetometer EKF and this modified EKF would be applied to the data exhibiting the unmodelled torque. Based on the assumption that the addition of the correct postulated source would result in the minimum energy in the innovation vector of the EKF, the magnitude of the torque would be varied until the minimum innovation vector energy was found. The minimum innovation vector energy of each of the four postulated sources would then be compared to find which one is the most likely source of the unmodelled disturbance torque.

The results of the investigation showed that magnetic torquers, which mistakenly remained active after the ADCS deactivated them, were the most likely source of the unmodelled disturbance torque.

6.2 Recommendations

6.2.1 BRINGING SENSOR HARDWARE INTO THE LOOP

This thesis was only concerned with emulating the interfaces of the sensors and actuators with the ADCS tray. The next step would be to bring the sensors themselves into the loop. This would be very useful for testing the sensor hardware and would also increase the accuracy of the simulation by incorporating the true sensor noise added to measurements by each sensor.

This expansion of the hardware-in-the-loop simulation facility would require the development of space environment emulators to imitate the geomagnetic field, sun, earth horison and star field. For example, the geomagnetic field could be emulated by a magnetic field vector, the sun by a point light source, the earth horison by an illuminated curved object and the star field by a projected image.

The sensors could then be mounted on the *Contraves Georz* three axis rate table and accurately pointed in the correct direction relative to the environment emulators. An ISA-based General Purpose Interface Bus (GPIB) card which can be used by an IBM

PC to interface with the *Contraves* rate table already exists and could easily be integrated into the hardware-in-the-loop simulation facility.

6.2.2 FURTHER INVESTIGATIONS INTO THE UNMODELLED TORQUE

Based on two sets of whole orbit data, the conclusion was reached that the most likely source of the unmodelled disturbance torque is a constant magnetic dipole moment generated by magnetic torquers which mistakenly remain active after the ADCS has deactivated them. Subsequent tests which have been performed on the flight model to confirm this theory have been somewhat inconclusive, but do indicate that the magnetic torquers may not be the only culprit. There have also been speculations that the permanent magnets in the reaction wheels and in the stepper motor of the imager may be the source of the constant magnetic dipole moment.

It is therefore recommended that more sets of whole orbit data be analysed and that further tests be done on the flight model to confirm the source of the unmodelled torque.

7. REFERENCES

7.1 Public Domain

De Villiers R. [1994]
The Block Diagram Simulation of Satellite Systems, Masters Thesis, University of Stellenbosch, November 1994.

Jacobs M. J. [1995]
A Low Cost, High Precision Star Sensor, Masters Thesis, University of Stellenbosch, December 1995.

Joubert M. L. [1999]
A Reaction Wheel Design for the SUNSAT Microsatellite, Masters Thesis, University of Stellenbosch, March 1999.

Steyn W.H. [1990]
Attitude Control Algorithms and Simulation Programs for Low Earth Orbit Spacecraft, Masters Thesis, University of Surrey, September 1990.

Steyn W. H. [1995]
A Multi-mode Attitude Determination and Control System for Small Satellites, PhD Thesis, University of Stellenbosch, December 1995.

Wertz J.R. [1988]
Spacecraft Attitude Determination and Control, D. Reidel Publishing Company, Dordrecht Holland, Reprint 1988.

7.2 Internal Documents

Farr X.C. [1998]
ICP Software Documentation for SUNSAT1, SUNSAT internal document, University of Stellenbosch, September 1998.

Joubert M.L. [1997]
ADCS Flight Model Board, ESL/SED Work Package, University of Stellenbosch, December 1997.

Joubert M.L. [1997]
ADCS FM Board Update, ESL/SED Work Package, University of Stellenbosch, December 1997.

Joubert M. L. [1997]
Magnetometer - "Orimag", ESL/SED Work Package, University of Stellenbosch, December 1997.

Joubert M. L. [1997]
Horison and Sun Sensors, ESL/SED Work Package, University of Stellenbosch, December 1997.

Joubert M. L. [1997]
Horison and Sun Sensors FM Update, ESL/SED Work Package, University of Stellenbosch, December 1997.

Joubert M. L. [1997]
Brush Reaction Wheel Flight Model, ESL/SED Work Package, University of Stellenbosch, December 1997.

Joubert M. L. [1997]
Brushless Reaction Wheel Flight Model, ESL/SED Work Package, University of Stellenbosch, December 1997.

Joubert M. L. [1998]
A Programmer's Guide to the ADCS, SUNSAT Work Package, University of Stellenbosch, February 1998.

Oosthuizen P. J. [1997]
ADCS Processor Communication, SUNSAT internal document, University of Stellenbosch, December 1997

Oosthuizen P. J. [1997]
Top Plate Star Camera, ESL/SED Work Package, University of Stellenbosch, December 1997.

SED Technical Manual [1992]
Analogue-to-Digital, Digital-to-Analogue and Digital I/O Card for the PC, SED Technical Manual, University of Stellenbosch, July 1992.

APPENDIX A. OBC EMULATOR

A.1 OBC Communications Protocol

ADCS Processor Communication

Message Structure:

	Length,	Type,	Data,	End ($FF)
Byte count:	1 ,	1 ,	n ,	1

OBC → ADCS

Initialisation:

- OBC first sends byte 55H (after ADCS T800 processor reset) via Link0 (OBC1), Link1 (OBC2) or Link3 (External via top plate)
- ADCS T800 replies with byte 55H on corresponding link
- All communication will then continue only on this link (until ADCS T800 reset)

Type:

1	-	Simulation sensor data	(37,1,36-byte data,$FF)
		Data:	
		BXR,BYR,BZR	(MagM field components)
		HTETA1,HTETA2,STETA	(HorX,Y + SunS angles)
		HXM,HYM,HZM	(RW angular momentum)
20	-	Select an Estimator (0-6)	(2,20,FILTER,$FF)
21	-	MT controller select (0-6)	(2,21,MAGCON,$FF)
22	-	RW on/off select (T/F)	(2,22,RWON,$FF)
23	-	Parameter selection from ADCS (0-10)	(2,23,SELEC,$FF)
24	-	Boom deployed switch (T/F)	(2,24,BOOM,$FF)
25	-	Aerodynamic disturbance estimation (T/F)	(2,25,AERO,$FF)
26	-	Current time in seconds (4-byte word)	(5,26,TIME,$FF)
27	-	RW_Z momentum reference (4-byte real)	(5,27,HZ_REF,$FF)
28	-	Satellite X,Y,Z body rate reference (real)	(13,28,WX/WY/WZ_REF,$FF)
29	-	Satellite P,R,Y attitude reference (real)	(13,29,PPC,RRC,YYC,$FF)
30	-	MT magnetic moment constants (real)	(13,30,MQX,MQY,MQZ,$FF)
31	-	RW MOI constants (real)	(13,31,RWX,RWY,RWZ,$FF)
32	-	Satellite MOI constants (real)	(9,32,IT,IZ,$FF)
33	-	SunS calibration constants (real)	(9,33,SBIAS,SGAIN,$FF)
34	-	MagM calibration constants (real)	(33,34,MBIAS[1..4],MGAIN[1..4],$FF)
35	-	HorS calibration constants (real)	(33,35,CHX[1..4],CHY[1..4],$FF)
36	-	Satellite Norad orbit parameters	(37,36,Parameters,$FF)
		Parameters:	
		ECCEN, INCL0, RAAN0, AP0, MA0, MM0, BSTAR (real), EPOCH (longreal).	

37	-	Magnetometer calibration switch (T/F)	(2,37,MM_CAL,$FF)
38	-	OBC active switch (T/F)	(2,38,OBC_ON,$FF)
39	-	Simulation switch (T/F)	(2,39,SIMULATE,$FF)
40	-	ICP active switch (T/F)	(2,40,ICP_ON,$FF)
41	-	StarS active switch (T/F)	(2,41,SS_ON,$FF)

FILTER:
1 - EKF Magnetometer (T = 10 sec)
2 - EKF Horison/Sun Sensors (T = 1 sec)
3 - EKF Star Sensor (T = 1 sec)
4 - Angular Rate Kalman Filter (T = 10 sec)
5 - Y-Estimator (T = 10 sec)
6 - Z-Estimator (T = 10 sec)

ADCS $\rightarrow$ OBC

Type:

20 - ADCS data to OBC (101,20,Data,$FF)

Data:

23 x Single type reals (4-byte each):

EWX,EWY,EWZ (rad/s^2)	X,Y,Z estimated angular rates
EPP,ERR,EYY (deg)	Pitch, roll and yaw est.attitude
EDY0 (Nm)	Aerodynamic est.disturbance
LAT,LON,RADIUS (deg, km)	Satellite position
NMX,NMY,NMZ (Nm)	MT torques
NWX,NWY,NWZ (Nm)	RW torques
HXR,HYR,HZR (Nms)	RW reference angular momentum
TIME (seconds)	Current ADCS time

SELEC = 0:	BXM,BYM,BZM (μT)	MagM direct measurements
= 1:	BX0,BY0,BZ0 (μT)	IGRF model values
= 2:	HXM,HYM,HZM (Nms)	RW measured angularmomentum
= 3:	MTX,MTY,MTZ (seconds)	MT pulse commands
= 4:	HTETA1,HTETA2,STETA (deg)	HorS_X/Y + SunS measurements
= 5:	SLAT,SLON,BETA (deg)	Sun position + angle to sat.orbit
= 6:	SAZIM,SELEV,OMEGA (deg)	Sun angles w.r.t. satellite
= 7:	ALPHA1,ALPHA2,X0EAST (deg)	HorS to sun + orbit/east angles
= 8:	MGAIN[1..3]	MagM gain calibration values
= 9:	MBIAS[1..3] (μT)	MagM offset calibration values
= 10:	TPP,TRR,TYY (deg)	Triad pitch, roll and yaw attitude

8 x Byte variables:

NADIR (0/1)	Byte to indicate nadir pointing attitude.
SUNOK (0/1)	Byte to indicate valid sun sensor data.
HOR1 (0/1)	Byte to indicate valid X-horizon data.
HOR2 (0/1)	Byte to indicate valid Y-horizon data.

A.2 OBC Emulator Software

A.2.1 TRANSPUTER CARD DRIVERS

```
unit T800Lib;

interface

uses
  Crt;

procedure ResetT800LinkAdaptor;

function WriteByteToLinkAdaptor(TxByte : Byte) : Boolean;
function WriteSingleToLinkAdaptor(TxSingle : Single) : Boolean;
function WriteLongIntToLinkAdaptor(TxLongInt : LongInt) : Boolean;
function WriteDoubleToLinkAdaptor(TxDouble : Double) : Boolean;

function ReadByteFromLinkAdaptor(var RxByte : Byte) : Boolean;
function ReadSingleFromLinkAdaptor(var RxSingle : Single) : Boolean;
function  ReadLongIntFromLinkAdaptor(var  RxLongInt  :  LongInt)  :
Boolean;
function ReadDoubleFromLinkAdaptor(var RxDouble : Double) : Boolean;

function BootAdcsThroughLinkAdaptor : Boolean;

implementation

const
  {register addresses}
  rd_byte = $150;   {Input Data Reg}
  wr_byte = $151;   {Output Data Reg}
  rd_stat = $152;   {Input Status Reg}
  wr_stat = $153;   {Output Status Reg}
  res_reg = $160;   {Reset(write)/Error(read) Reg}
  anl_reg = $161;   {Analyse (write)}
  aux = $164;       {Auxiliary Reg}

type
  TSingleOrBytesAccessBuffer = record
    case tag : Integer of
      0 : (SingleField : Single);
      1 : (ByteField : array [0..3] of byte);
  end;

  TDoubleOrBytesAccessBuffer = record
    case tag : Integer of
      0 : (DoubleField : double);
      1 : (ByteField : array [0..7] of byte);
  end;

  TLongIntOrBytesAccessBuffer = record
    case AccessAsBytes : Integer of
      0 : (LongIntField : LongInt);
      1 : (ByteField : array [0..3] of byte);
  end;

procedure ResetT800LinkAdaptor;
begin
```

```
  {reset link adaptor}
  port[anl_reg] := 0;
  port[res_reg] := 0;
  Delay(50);
  port[res_reg] := 1;
  Delay(50);                {minimum allowable time = 1.6us}
  port[res_reg] := 0;
end;

function WriteByteToLinkAdaptor(TxByte : Byte) : Boolean;
var
  count : word;
begin
  WriteByteToLinkAdaptor := False;
  count := 0;
  repeat
    if (port[wr_stat] and 1) = 1 then
      begin
        port[wr_byte] := TxByte;
        exit;
      end
    else
      begin
        Inc(count);
        Delay(1)
      end;
  until (count = 1000);
  WriteByteToLinkAdaptor := True;
  WriteLn('LinkAdaptor Write Byte TimeOut');
end;

function WriteSingleToLinkAdaptor(TxSingle : Single) : Boolean;
var
  TxTimeOut : Boolean;
  SingleOrBytesAccessBuffer : TSingleOrBytesAccessBuffer;
  ByteFieldIndex : Byte;
begin
  WriteSingleToLinkAdaptor := False;
  SingleOrBytesAccessBuffer.SingleField := TxSingle;
  ByteFieldIndex := 0;
  repeat
    TxTimeOut := WriteByteToLinkAdaptor(
      SingleOrBytesAccessBuffer.ByteField[ByteFieldIndex]);
    Inc(ByteFieldIndex);
  until (ByteFieldIndex = 4) or TxTimeOut;
  if TxTimeOut then WriteSingleToLinkAdaptor := True;
end;
```

```
function WriteLongIntToLinkAdaptor(TxLongInt : LongInt) : Boolean;
var
  TxTimeOut : Boolean;
  LongIntOrBytesAccessBuffer : TLongIntOrBytesAccessBuffer;
  ByteFieldIndex : Byte;
begin
  WriteLongIntToLinkAdaptor := False;
  LongIntOrBytesAccessBuffer.LongIntField := TxLongInt;
  ByteFieldIndex := 0;
  repeat
    TxTimeOut := WriteByteToLinkAdaptor(
      LongIntOrBytesAccessBuffer.ByteField[ByteFieldIndex]);
    Inc(ByteFieldIndex);
  until (ByteFieldIndex = 4) or TxTimeOut;
  if TxTimeOut then WriteLongIntToLinkAdaptor := True;
end;

function WriteDoubleToLinkAdaptor(TxDouble : Double) : Boolean;
var
  TxTimeOut : Boolean;
  DoubleOrBytesAccessBuffer : TDoubleOrBytesAccessBuffer;
  ByteFieldIndex : Byte;
begin
  WriteDoubleToLinkAdaptor := False;
  DoubleOrBytesAccessBuffer.DoubleField := TxDouble;
  ByteFieldIndex := 0;
  repeat
    TxTimeOut := WriteByteToLinkAdaptor(
      DoubleOrBytesAccessBuffer.ByteField[ByteFieldIndex]);
    Inc(ByteFieldIndex);
  until (ByteFieldIndex = 8) or TxTimeOut;
  if TxTimeOut then WriteDoubleToLinkAdaptor := True;
end;

function ReadByteFromLinkAdaptor(var RxByte : Byte) : Boolean;
var
  count : word;
begin
  ReadByteFromLinkAdaptor := False;
  count := 0;
  repeat
    if (port[rd_stat] and 1) = 1 then
      begin
        RxByte := port[rd_byte];
        exit;
      end
    else
      begin
        {Inc(count);}
        Delay(1);
      end
  until (count = 1000);
  ReadByteFromLinkAdaptor := True;
  WriteLn('Link Adaptor Read Byte TimeOut');
end;
```

```
function ReadSingleFromLinkAdaptor(var RxSingle : Single) : Boolean;
var
  RxTimeOut : Boolean;
  SingleOrBytesAccessBuffer : TSingleOrBytesAccessBuffer;
  ByteFieldIndex : Byte;
begin
  ReadSingleFromLinkAdaptor := False;
  ByteFieldIndex := 0;
  repeat
    RxTimeOut := ReadByteFromLinkAdaptor(
      SingleOrBytesAccessBuffer.ByteField[ByteFieldIndex]);
    Inc(ByteFieldIndex);
  until (ByteFieldIndex = 4) or RxTimeOut;
  If RxTimeOut then
    ReadSingleFromLinkAdaptor := True
  else
    RxSingle := SingleOrBytesAccessBuffer.SingleField;
end;

function  ReadLongIntFromLinkAdaptor(var  RxLongInt  :  LongInt)  :
Boolean;
var
  RxTimeOut : Boolean;
  LongIntOrBytesAccessBuffer : TLongIntOrBytesAccessBuffer;
  ByteFieldIndex : Byte;
begin
  ReadLongIntFromLinkAdaptor := False;
  ByteFieldIndex := 0;
  repeat
    RxTimeOut := ReadByteFromLinkAdaptor(
      LongIntOrBytesAccessBuffer.ByteField[ByteFieldIndex]);
    Inc(ByteFieldIndex);
  until (ByteFieldIndex = 4) or RxTimeOut;
  If RxTimeOut then
    ReadLongIntFromLinkAdaptor := True
  else
    RxLongInt := LongIntOrBytesAccessBuffer.LongIntField;
end;

function ReadDoubleFromLinkAdaptor(var RxDouble : Double) : Boolean;
var
  RxTimeOut : Boolean;
  DoubleOrBytesAccessBuffer : TDoubleOrBytesAccessBuffer;
  ByteFieldIndex : Byte;
begin
  ReadDoubleFromLinkAdaptor := False;
  ByteFieldIndex := 0;
  repeat
    RxTimeOut := ReadByteFromLinkAdaptor(
      DoubleOrBytesAccessBuffer.ByteField[ByteFieldIndex]);
    Inc(ByteFieldIndex);
  until (ByteFieldIndex = 4) or RxTimeOut;
  If RxTimeOut then
    ReadDoubleFromLinkAdaptor := True
  else
    RxDouble := DoubleOrBytesAccessBuffer.DoubleField;
end;
```

```
function BootAdcsThroughLinkAdaptor : Boolean;
var
  T800CodeFileHandle : File of Byte;
  TxByte : Byte;
begin
  BootAdcsThroughLinkAdaptor := False;
  ResetT800LinkAdaptor;
  Delay(5000);
  assign(T800CodeFileHandle,'Japie.app');
  reset(T800CodeFileHandle);
  repeat
    read(T800CodeFileHandle,TxByte);
    if WriteByteToLinkAdaptor(TxByte) then
      begin
        BootADCSThroughLinkAdaptor := True;
        exit;
      end;
  until Eof(T800CodeFileHandle);
  close(T800CodeFileHandle);
  Delay(5000);
end;

end.
```

APPENDIX B. MAGNETOMETER EMULATOR

B.1 Magnetometer Emulator Software

B.1.1 MAGNETOMETER EMULATION SOFTWARE

```
Unit MagMeter;

interface

uses
  ADASLib, VecTypes;

procedure InitMagnetometerEmulator;
procedure WriteMagnetometer(
                        GeomagneticFieldMeasured : TSingleVector);

procedure ReadMagneticTorquers(var X1, Y1, Z1, X2, Y2, Z2 : double);

implementation

var
  ADASCard0, ADASCard1 : TADASCard;

procedure InitMagnetometerEmulator;
begin
  ADASCard0.SetBaseAddress(b360H);
  ADASCard1.SetBaseAddress(b200H);

  ADASCard0.WriteAnalogVoltage(DA0, 0.0);
  ADASCard0.WriteAnalogVoltage(DA1, 0.0);
  ADASCard1.WriteAnalogVoltage(DA0, 0.0);
  ADASCard1.WriteAnalogVoltage(DA1, 0.0);
end;

procedure WriteMagnetometer(GeomagneticFieldMeasured :
TSingleVector);
var
  MagnetometerVoltages : TSingleVector;
begin

  MagnetometerVoltages[1] := GeomagneticFieldMeasured[1]/60*5;
  MagnetometerVoltages[2] := GeomagneticFieldMeasured[2]/60*5;
  MagnetometerVoltages[3] := GeomagneticFieldMeasured[3]/60*5;

  if MagnetometerVoltages[1] >=5 then MagnetometerVoltages[1] := 5;
  if MagnetometerVoltages[1] <=-5 then MagnetometerVoltages[1] := -5;
  if MagnetometerVoltages[2] >=5 then MagnetometerVoltages[2] := 5;
  if MagnetometerVoltages[2] <=-5 then MagnetometerVoltages[2] := -5;
  if MagnetometerVoltages[3] >=5 then MagnetometerVoltages[3] := 5;
  if MagnetometerVoltages[3] <=-5 then MagnetometerVoltages[3] := -5;

  ADASCard0.WriteAnalogVoltage(DA0, MagnetometerVoltages[1]);
  ADASCard0.WriteAnalogVoltage(DA1, MagnetometerVoltages[2]);
  ADASCard1.WriteAnalogVoltage(DA0, MagnetometerVoltages[3]);
end;
```

```
procedure ReadMagneticTorquers(var X1, Y1, Z1, X2, Y2, Z2 : double);
begin
  ADASCard0.AnalogToDigital;
  ADASCard1.AnalogToDigital;

  X1 := ADASCard0.ReadAnalogVoltage(AD0);
  Y1 := ADASCard0.ReadAnalogVoltage(AD1);
  Z1 := ADASCard0.ReadAnalogVoltage(AD2);

  X2 := ADASCard1.ReadAnalogVoltage(AD0);
  Y2 := ADASCard1.ReadAnalogVoltage(AD1);
  Z2 := ADASCard1.ReadAnalogVoltage(AD2);
end;

end.
```

B.1.2 ADAS CARD DRIVERS

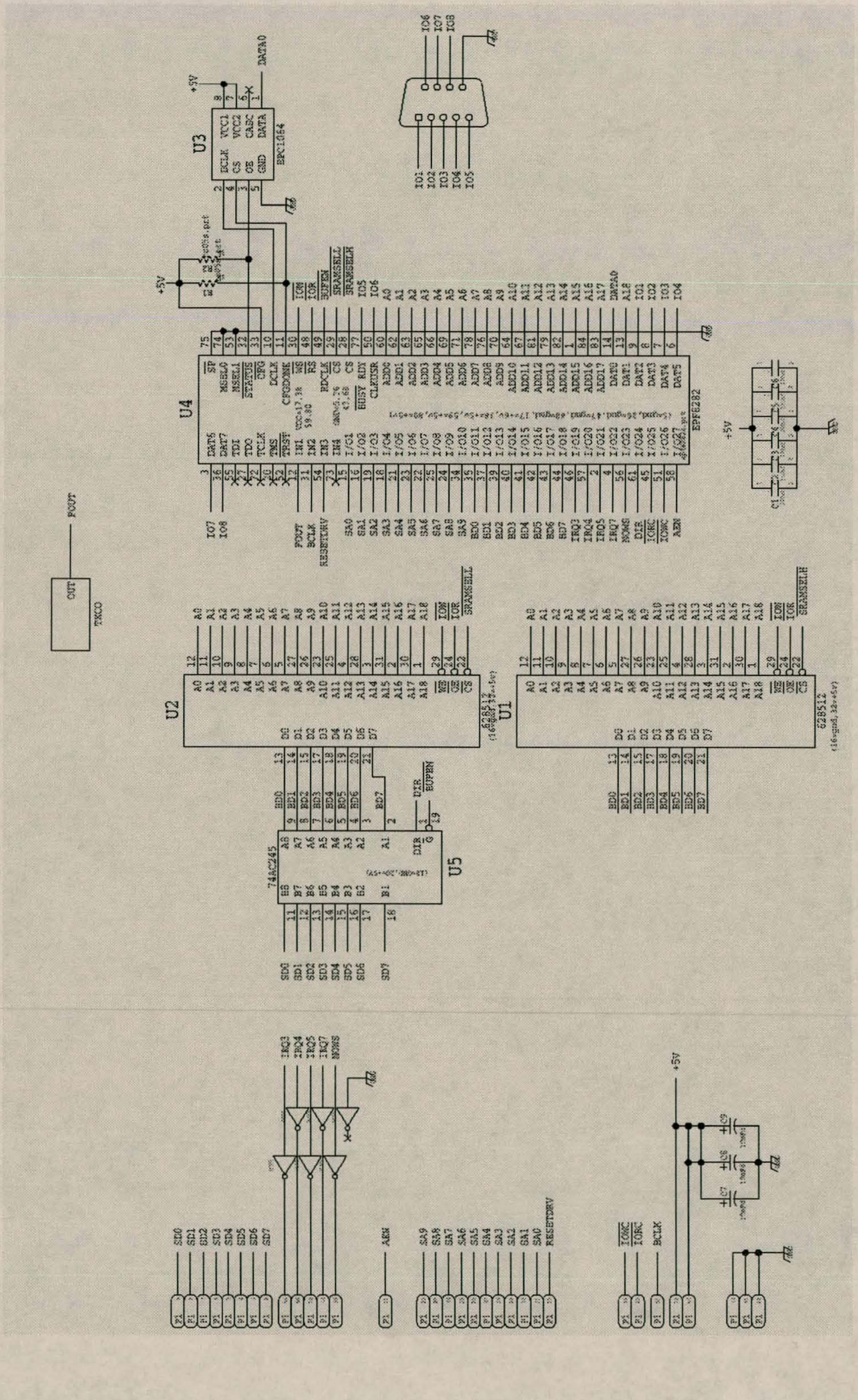
```
unit AdasLib;

interface

type
  TBaseAddressName = (b150H, b200H, b360H);
  TAToDChannel = (AD0, AD1, AD2, AD3);
  TDToAChannel = (DA0, DA1);

type
  TADASCard = object
    BaseAddressName : TBaseAddressName;
    BaseAddress : Word;
    AToDData : array [0..3] of Integer;
    procedure SetBaseAddress(NewBaseAddressName : TBaseAddressName);

    procedure AnalogToDigital;
    procedure DigitalToAnalog(DToAChannel : TDToAChannel;
                              DToAUnits : Integer);

    function ReadAnalogVoltage(AToDChannel : TAToDChannel) : double;
    procedure WriteAnalogVoltage(DToAChannel : TDToAChannel;
                                 AnalogVoltage : double);
  end;

implementation

var
  ADDATA : array [0..3] of Integer;

procedure TADASCard.SetBaseAddress(
                              NewBaseAddressName : TBaseAddressName);
begin
  BaseAddressName := NewBaseAddressName;
  case BaseAddressName of
    b150H : BaseAddress := $150;
    b200H : BaseAddress := $200;
    b360H : BaseAddress := $360;
  end;
end;
```

```
procedure AnalogToDigitalBaseAddress150H;
begin
  inline(
    $BA/$50/$01/      {mov dx,ADC}
    $EC/              {in al,dx}                  {Dummy Read}
    $EE/              {out dx,al}                 {Start conversion}
    $B9/$00/$01/      {mov cx,100H}
    $BA/$52/$01/      {mov dx,ADCHi}
                    {@1:}
    $EE/              {out dx,al}       {Clock interrupt line in latch}
    $EC/              {in al,dx}                  {Poll ADC interrupt}
    $24/$80/          {and al,80H}
    $E0/$FA/          {loopnz @1}
    $BF/ADDATA/       {mov di,offset ADDATA}
    $B9/$04/$00/      {mov cx,4}
                    {@2:}
    $BA/$50/$01/      {mov dx,ADC}
    $EC/              {in al,dx}                  {Read low byte}
    $88/$C3/          {mov bl,al}
    $BA/$52/$01/      {mov dx,ADCHi}
    $EC/              {in al,dx}                  {Read high byte}
    $24/$0F/          {and al,0FH}
    $88/$C7/          {mov bh,al}
    $24/$08/          {and al,08H}
    $74/$04/          {jz @3}
    $81/$EB/$00/$10/ {sub bx,1000H}
                    {@3:}
    $89/$1D/          {mov [di],bx}           {Store channel A/D data}
    $47/              {inc di}
    $47/              {inc di}
    $E2/$E4);         {loop @2}
end;  {AtoD}

procedure AnalogToDigitalBaseAddress200H;
begin
  inline(
    $BA/$00/$02/      {mov dx,ADC}
    $EC/              {in al,dx}                  {Dummy Read}
    $EE/              {out dx,al}                 {Start conversion}
    $B9/$00/$01/      {mov cx,100H}
    $BA/$02/$02/      {mov dx,ADCHi}
                    {@1:}
    $EE/              {out dx,al}       {Clock interrupt line in latch}
    $EC/              {in al,dx}                  {Poll ADC interrupt}
    $24/$80/          {and al,80H}
    $E0/$FA/          {loopnz @1}
    $BF/ADDATA/       {mov di,offset ADDATA}
    $B9/$04/$00/      {mov cx,4}
                    {@2:}
    $BA/$00/$02/      {mov dx,ADC}
    $EC/              {in al,dx}                  {Read low byte}
    $88/$C3/          {mov bl,al}
    $BA/$02/$02/      {mov dx,ADCHi}
    $EC/              {in al,dx}                  {Read high byte}
    $24/$0F/          {and al,0FH}
    $88/$C7/          {mov bh,al}
    $24/$08/          {and al,08H}
    $74/$04/          {jz @3}
    $81/$EB/$00/$10/ {sub bx,1000H}
                    {@3:}
    $89/$1D/          {mov [di],bx}           {Store channel A/D data}
    $47/               {inc di}
    $47/               {inc di}
    $E2/$E4);          {loop @2}
end;  {AtoD}

procedure AnalogToDigitalBaseAddress360H;
begin
  inline(
    $BA/$60/$03/       {mov dx,ADC}
    $EC/               {in al,dx}                 {Dummy Read}
    $EE/               {out dx,al}                {Start conversion}
    $B9/$00/$01/       {mov cx,100H}
    $BA/$62/$03/       {mov dx,ADCHi}
                    {@1:}
    $EE/               {out dx,al}                {Clock interrupt line
in latch}
    $EC/               {in al,dx}                 {Poll ADC interrupt}
    $24/$80/           {and al,80H}
    $E0/$FA/           {loopnz @1}
    $BF/ADDATA/        {mov di,offset ADDATA}
    $B9/$04/$00/       {mov cx,4}
                    {@2:}
    $BA/$60/$03/       {mov dx,ADC}
    $EC/               {in al,dx}                 {Read low byte}
    $88/$C3/           {mov bl,al}
    $BA/$62/$03/       {mov dx,ADCHi}
    $EC/               {in al,dx}                 {Read high byte}
    $24/$0F/           {and al,0FH}
    $88/$C7/           {mov bh,al}
    $24/$08/           {and al,08H}
    $74/$04/           {jz @3}
    $81/$EB/$00/$10/ {sub bx,1000H}
                    {@3:}
    $89/$1D/            {mov [di],bx}                {Store channel A/D
data}
    $47/               {inc di}
    $47/               {inc di}
    $E2/$E4);          {loop @2}
end;  {AtoD}

procedure TADASCard.AnalogToDigital;
var
  AToDChannelIndex : Integer;
begin
  case BaseAddressName of
    b150H : AnalogToDigitalBaseAddress150H;
    b200H : AnalogToDigitalBaseAddress200H;
    b360H : AnalogToDigitalBaseAddress360H;
  end;
  AToDData[0] := ADDATA[0];
  AToDData[1] := ADDATA[1];
  AToDData[2] := ADDATA[2];
  AToDData[3] := ADDATA[3];
end;
```

```
procedure TADASCard.DigitalToAnalog(
                  DToAChannel : TDToAChannel; DToAUnits : Integer);
var
  DummyRead : byte;
begin
  if DToAUnits > 2047 then DToAUnits := 2047;
  if DToAUnits < -2048 then DToAUnits := -2048;
  DToAUnits := DToAUnits + 2048;
  case DToAChannel of
    DA0 :
      begin
        port[BaseAddress + $4] := lo(DToAUnits);
        port[BaseAddress + $5] := hi(DToAUnits);
        DummyRead := port[BaseAddress + $4];
      end;
    DA1 :
      begin
        port[BaseAddress + $6] := lo(DToAUnits);
        port[BaseAddress + $7] := hi(DToAUnits);
        DummyRead := port[BaseAddress + $6];
      end;
  end;
end;

function  TADASCard.ReadAnalogVoltage(AToDChannel  :  TAToDChannel)  :
double;
var
  AToDChannelIndex : Integer;
begin
  AToDChannelIndex := ord(AToDChannel);
  ReadAnalogVoltage := AToDData[AToDChannelIndex]/2048*10;
end;

procedure TADASCard.WriteAnalogVoltage(DToAChannel : TDToAChannel;
                                       AnalogVoltage : double);
var
  DToAUnits : Integer;
begin
  DToAUnits := round(AnalogVoltage/10*2048);
  DigitalToAnalog(DToAChannel, DToAUnits);
end;

end.
```

APPENDIX C. HORISON / SUN SENSOR EMULATOR

C.1 Circuit Diagram for the ISA-Based FPGA card



C.2 Pin assignment for the FPGA

```
D_REG_4 :           INPUT_PIN = 66;
D_REG_3 :           INPUT_PIN = 65;
D_REG_2 :           INPUT_PIN = 63;
D_REG_1 :           INPUT_PIN = 62;
D_REG_0 :           INPUT_PIN = 60;
BUS_8031_D7 :       BIDIR_PIN = 6;
BUS_8031_D6 :       BIDIR_PIN = 8;
BUS_8031_D5 :       BIDIR_PIN = 50;
BUS_8031_D4 :       BIDIR_PIN = 77;
BUS_8031_D3 :       BIDIR_PIN = 4;
BUS_8031_D2 :       BIDIR_PIN = 7;
BUS_8031_D1 :       BIDIR_PIN = 9;
BUS_8031_D0 :       BIDIR_PIN = 13;
DIR :               OUTPUT_PIN = 61;
NOT_IOWC :          INPUT_PIN = 51;
NOT_IORC :          INPUT_PIN = 45;
NOT_DWS :           OUTPUT_PIN = 56;
NOT_BUFEN :         OUTPUT_PIN = 49;
RESETDRV :          INPUT_PIN = 54;
NOT_IRQ7 :          OUTPUT_PIN = 36;
NOT_IRQ5 :          OUTPUT_PIN = 57;
NOT_IRQ4 :          OUTPUT_PIN = 46;
NOT_IRQ3 :          OUTPUT_PIN = 2;
BD7 :               BIDIR_PIN = 44;
BD6 :               BIDIR_PIN = 43;
BD5 :               BIDIR_PIN = 42;
BD4 :               BIDIR_PIN = 41;
BD3 :               BIDIR_PIN = 40;
BD2 :               BIDIR_PIN = 39;
BD1 :               BIDIR_PIN = 37;
BD0 :               BIDIR_PIN = 35;
SA9 :               INPUT_PIN = 34;
SA8 :               INPUT_PIN = 24;
SA7 :               INPUT_PIN = 25;
SA6 :               INPUT_PIN = 22;
SA5 :               INPUT_PIN = 23;
SA4 :               INPUT_PIN = 21;
SA3 :               INPUT_PIN = 18;
SA2 :               INPUT_PIN = 19;
SA1 :               INPUT_PIN = 16;
SA0 :               INPUT_PIN = 15;
BCLK :              INPUT_PIN = 31;
AEN :               INPUT_PIN = 58;
DEVICE = EPF8636ALC84-3;
```

C.3 VHDL Code for the FPGA

```
library IEEE;
   use IEEE.std_logic_1164.all;
   use IEEE.std_logic_arith.all;

ENTITY Emulator IS
      PORT(
            -- Pins connected to ISA bus
            RESETDRV : IN STD_LOGIC;
                              -- Resets ISA_InputState to No_Input
                              -- and Register0..4 to null registers
            SA : IN  STD_LOGIC_VECTOR(9 downto 0);
                              -- ISA address bus SA0..9
            BD : INOUT STD_LOGIC_VECTOR(7 downto 0);
                              -- ISA data bus BD0..7
            AEN : IN    STD_LOGIC;
                              -- ISA address enable
            BCLK : IN   STD_LOGIC;
                              -- ISA bus clock
            NOT_IORC : IN STD_LOGIC;
                              -- ISA I/O read control signal
            NOT_IOWC : IN STD_LOGIC;
                              -- ISA I/O write control signal
            NOT_DWS : OUT STD_LOGIC;
                              -- ISA NO wait state control signal
            NOT_IRQ3 : OUT STD_LOGIC;
                              -- ISA COM2 Interrupt request
            NOT_IRQ4 : OUT STD_LOGIC;
                              -- ISA COM1 Interrupt request
            NOT_IRQ5 : OUT STD_LOGIC;
                              -- ISA LPT2 Interrupt request
            NOT_IRQ7 : OUT STD_LOGIC;
                              -- ISA LPT1 Interrupt request

            -- Pins connected to ADCS HOR plug
            D_REG_0, D_REG_1, D_REG_2,
            D_REG_3, D_REG_4                 : IN  STD_LOGIC;
                              -- ICP requests Horison/Sunsensor
                              -- Piggyback to drive Register0..4
                              -- onto the ICP databus
            BUS_8031_D : INOUT STD_LOGIC_VECTOR(7 downto 0);
                              -- ICP databus 8031_D0..7
            NOT_INTXYS : OUT  STD_LOGIC;
                              -- Interrupt generated by Horison/Sun-
                              -- sensor Piggyback when pixel profiles
                              -- are ready
            INTSEL : OUT STD_LOGIC;
                                  -- Multiplexes interrupts /ADCSINTR
                                  -- from the RAM tray and /INTXYS from
                                  -- the Horison/Sunsensor Piggyback
                                  -- LO - /ADCSINTR enabled
                                  -- HI - /INTXYS enabled

            -- Pins connected to buffer
            NOT_BUFEN : OUT   STD_LOGIC;
                                  -- Buffer enable
            DIR : OUT STD_LOGIC
                                  -- Buffer direction
            );
END Emulator;

ARCHITECTURE a OF Emulator IS
      TYPE TypeBUS_8031_D_OutputState IS (
                  Output_D_REG_0, Output_D_REG_1, Output_D_REG_2,
                  Output_D_REG_3, Output_D_REG_4, Output_Tristate);
      TYPE TypeISA_InputState IS (
                  Input_D_REG_0, Input_D_REG_1, Input_D_REG_2,
                  Input_D_REG_3, Input_D_REG_4, No_Input,
                  ICP_Int_Ack);
      SIGNAL B_8031_D_OutputState : TypeBUS_8031_D_OutputState;
      SIGNAL Register0, Register1,
             Register2, Register3,
             Register4              : STD_LOGIC_VECTOR(7 downto 0);
      SIGNAL ISA_InputState : TypeISA_InputState;

BEGIN

      INTSEL <= '0';              -- /INTXYS is disabled and held HI
      NOT_INTXYS <= '1';

--    NOT_IRQ3 <= '1';            -- IRQ's may be used later to signal
      NOT_IRQ4 <= '1';            -- to ISA bus
      NOT_IRQ5 <= '1';
      NOT_IRQ7 <= '1';
      NOT_DWS <= '0';             -- DWS not used

      BD <= "ZZZZZZZZ";
      DIR <= NOT NOT_IORC;

      D_REG_Select:
      PROCESS (D_REG_0, D_REG_1, D_REG_2, D_REG_3, D_REG_4)
      BEGIN
            IF D_REG_0 = '0' THEN
                  B_8031_D_OutputState <= Output_D_REG_0;
            ELSIF D_REG_1 = '0' THEN
                  B_8031_D_OutputState <= Output_D_REG_1;
            ELSIF D_REG_2 = '0' THEN
                  B_8031_D_OutputState <= Output_D_REG_2;
            ELSIF D_REG_3 = '0' THEN
                  B_8031_D_OutputState <= Output_D_REG_3;
            ELSIF D_REG_4 = '0' THEN
                  B_8031_D_OutputState <= Output_D_REG_4;
            ELSE
                B_8031_D_OutputState <= Output_Tristate;
            END IF;
      END PROCESS D_REG_Select;
```

```
BUS_8031_D_Output:
PROCESS (B_8031_D_OutputState)
BEGIN
      CASE B_8031_D_OutputState IS
            WHEN Output_D_REG_0 =>
                BUS_8031_D <= Register0;
            WHEN Output_D_REG_1 =>
                BUS_8031_D <= Register1;
            WHEN Output_D_REG_2 =>
                BUS_8031_D <= Register2;
            WHEN Output_D_REG_3 =>
                BUS_8031_D <= Register3;
            WHEN Output_D_REG_4 =>
                BUS_8031_D <= Register4;
            WHEN OTHERS =>
                BUS_8031_D <= "ZZZZZZZZ";
      END CASE;
END PROCESS BUS_8031_D_Output;

ISA_Address_Decoding:
PROCESS (RESETDRV, BCLK, SA, AEN)
BEGIN
  IF RESETDRV = '1' then
      ISA_InputState <= No_Input;
  ELSE
      IF BCLK'EVENT AND BCLK = '1' THEN
            IF (SA(9 downto 4) = "110000")
            AND (AEN = '0') THEN
                  CASE SA(3 downto 0) IS
                        WHEN "0000" =>
                              ISA_InputState <= Input_D_REG_0;
                              NOT_BUFEN <= '0';
                        WHEN "0001" =>
                              ISA_InputState <= Input_D_REG_1;
                              NOT_BUFEN <= '0';
                        WHEN "0010" =>
                              ISA_InputState <= Input_D_REG_2;
                              NOT_BUFEN <= '0';
                        WHEN "0011" =>
                              ISA_InputState <= Input_D_REG_3;
                              NOT_BUFEN <= '0';
                        WHEN "0100" =>
                              ISA_InputState <= Input_D_REG_4;
                              NOT_BUFEN <= '0';
                        WHEN "0101" =>
                              ISA_InputState <= ICP_Int_Ack;
                              NOT_BUFEN <= '1';
                        WHEN OTHERS =>
                              ISA_InputState <= No_Input;
                              NOT_BUFEN <= '1';
                  END CASE;
            ELSE
                  ISA_InputState <= No_Input;
                  NOT_BUFEN <= '1';
            END IF;
      END IF;
  END IF;
END PROCESS ISA_Address_Decoding;
```

```
      ISA_Write_Cycle:
      PROCESS (RESETDRV, NOT_IOWC)
      BEGIN
            IF RESETDRV = '1' THEN
                  Register0 <= "00000000";
                  Register1 <= "00000000";
                  Register2 <= "00000000";
                  Register3 <= "00000000";
                  Register4 <= "00000000";
            ELSE
                  IF NOT_IOWC'EVENT AND NOT_IOWC = '1' THEN
                        CASE ISA_InputState IS
                              WHEN Input_D_REG_0 =>
                              Register0 <= BD;
                              WHEN Input_D_REG_1 =>
                                  Register1 <= BD;
                              WHEN Input_D_REG_2 =>
                                  Register2 <= BD;
                              WHEN Input_D_REG_3 =>
                                  Register3 <= BD;
                              WHEN Input_D_REG_4 =>
                                  Register4 <= BD;
                              WHEN OTHERS =>
                        END CASE;
                  END IF;
            END IF;
      END PROCESS ISA_Write_Cycle;

      --NOT_IRQ3 <= '1';

      ICP_Acknowledge_Interrupt:
      PROCESS (ISA_InputState, D_REG_4, NOT_IOWC)
      BEGIN
            IF (ISA_InputState = ICP_Int_Ack) AND (NOT_IOWC = '0')
            THEN
                NOT_IRQ3 <= '1';
            ELSE
                IF D_REG_4'EVENT AND D_REG_4 = '1' THEN
                   NOT_IRQ3 <= '0';
                END IF;
            END IF;
      END PROCESS ICP_Acknowledge_Interrupt;

END a;
```

C.4 Horison / Sun Sensor Emulator Software

C.4.1 HORISON / SUN SENSOR EMULATION SOFTWARE

```
Unit HorSun;

interface

uses
  VecTypes, CubicRts;

procedure InitHorisonSunSensorEmulator;
procedure WriteHorisonSunSensorEmulator(
                         XHorisonSensorMeasurement : Single;
                         MinusYHorisonSensorMeasurement : Single;
                         SunSensorMeasurement : Single);

implementation

{$R-}

const
  D_REG_0 = $300;
  D_REG_1 = $301;
  D_REG_2 = $302;
  D_REG_3 = $303;
  D_REG_4 = $304;

  DefaultXHorisonSensorCalibrationConstants : TSingleVector4
    = (-3e-9, 8.84e-6, 0.0202, -27.19);
  DefaultMinusYHorisonSensorCalibrationConstants : TSingleVector4
    = (-2.71e-9, 7.24e-6, 0.02185, -27.03);
  DefaultSunSensorGain = 741.44;
  DefaultSunSensorOffset = 1021;
```

```
var
  XHorisonSensorCalibrationConstants : TSingleVector4;
  MinusYHorisonSensorCalibrationConstants : TSingleVector4;
  SunSensorGain, SunSensorOffset : Single;

  XHorisonSensorMeasurementDecalibrated : Single;
  MinusYHorisonSensorMeasurementDecalibrated : Single;
  SunSensorMeasurementDecalibrated : Single;

  A10_8, A7_0 : Byte;
  B10_8, B7_0 : Byte;
  S10_8, S7_0 : Byte;

  D_REG_0Data,
  D_REG_1Data,
  D_REG_2Data,
  D_REG_3Data,
  D_REG_4Data : Byte;

procedure WriteHorisonSunSensorEmulatorRegister(
                                    Address : Word; Data : Byte);
begin
  Port[Address] := Data;
end;

procedure SingleTo11Bit(var Hi3Bits, Low8Bits : Byte;
                        SingleData : Single);
var
  WordData : Word;
begin
  WordData := round(SingleData);

  Low8Bits := Lo(WordData);
  Hi3Bits := Hi(WordData);

  Hi3Bits := Hi3Bits and $07;
end;

function GetXHorisonSensorMeasurementDecalibrated(
                        XHorisonSensorMeasurement : Single) : Single;
var
  a3, a2, a1, a0 : double;
  z2 : double;
begin
  a3 := XHorisonSensorCalibrationConstants[1];
  a2 := XHorisonSensorCalibrationConstants[2];
  a1 := XHorisonSensorCalibrationConstants[3];
  a0 := XHorisonSensorCalibrationConstants[4];

  z2 := GetCubicRoot(a3, a2, a1, a0 - XHorisonSensorMeasurement);

  GetXHorisonSensorMeasurementDecalibrated := z2;
end;
```

```
function GetMinusYHorisonSensorMeasurementDecalibrated(
                  MinusYHorisonSensorMeasurement : Single) : Single;
var
  a3, a2, a1, a0 : double;
  z2 : double;
begin
  a3 := MinusYHorisonSensorCalibrationConstants[1];
  a2 := MinusYHorisonSensorCalibrationConstants[2];
  a1 := MinusYHorisonSensorCalibrationConstants[3];
  a0 := MinusYHorisonSensorCalibrationConstants[4];

  z2 := GetCubicRoot(
                  a3, a2, a1, a0 - MinusYHorisonSensorMeasurement);

  GetMinusYHorisonSensorMeasurementDecalibrated := z2;
end;

function Tan(x : double) : double;
begin
  Tan := sin(x)/cos(x);
end;

function GetSunSensorMeasurementDecalibrated(
                        SunSensorMeasurement : Single) : Single;
var
  dummy : double;
begin
  dummy := SunSensorGain*Tan(SunSensorMeasurement*pi/180)
         - SunSensorOffset;
  GetSunSensorMeasurementDecalibrated := dummy;
end;

procedure InitHorisonSunSensorEmulator;
begin
  WriteHorisonSunSensorEmulatorRegister(D_REG_0, $00);
  WriteHorisonSunSensorEmulatorRegister(D_REG_1, $00);
  WriteHorisonSunSensorEmulatorRegister(D_REG_2, $00);
  WriteHorisonSunSensorEmulatorRegister(D_REG_3, $00);
  WriteHorisonSunSensorEmulatorRegister(D_REG_4, $00);

  XHorisonSensorCalibrationConstants
      := DefaultXHorisonSensorCalibrationConstants;
  MinusYHorisonSensorCalibrationConstants
      := DefaultMinusYHorisonSensorCalibrationConstants;
  SunSensorGain := DefaultSunSensorGain;
  SunSensorOffset := DefaultSunSensorOffset;
end;

procedure SetHorisonSensorCalibrationConstants(
      SetXHorisonSensorCalibrationConstants : TSingleVector4;
      SetMinusYHorisonSensorCalibrationConstants : TSingleVector4);
begin
  XHorisonSensorCalibrationConstants
    := SetXHorisonSensorCalibrationConstants;

  MinusYHorisonSensorCalibrationConstants
    := SetMinusYHorisonSensorCalibrationConstants;
end;
```

```
procedure SetSunSensorCalibrationConstants(
                                    SetSunSensorGain : Single;
                                    SetSunSensorOffset : Single);
begin
  SunSensorGain := SetSunSensorGain;
  SunSensorOffset := SetSunSensorOffset;
end;

procedure WriteHorisonSunSensorEmulator(
                        XHorisonSensorMeasurement : Single;
                        MinusYHorisonSensorMeasurement : Single;
                        SunSensorMeasurement : Single);
begin

  XHorisonSensorMeasurementDecalibrated
      := GetXHorisonSensorMeasurementDecalibrated(
                                         XHorisonSensorMeasurement);

  MinusYHorisonSensorMeasurementDecalibrated
    := GetMinusYHorisonSensorMeasurementDecalibrated(
                                   MinusYHorisonSensorMeasurement);

  SunSensorMeasurementDecalibrated
    := GetSunSensorMeasurementDecalibrated(SunSensorMeasurement);

  SingleTo11Bit(A10_8, A7_0, XHorisonSensorMeasurementDecalibrated);
  SingleTo11Bit(B10_8, B7_0,
                MinusYHorisonSensorMeasurementDecalibrated);
  SingleTo11Bit(S10_8, S7_0, SunSensorMeasurementDecalibrated);

  D_REG_0Data := A7_0;
  D_REG_1Data := B10_8*16 + A10_8;
  D_REG_2Data := B7_0;
  D_REG_3Data := S7_0;
  D_REG_4Data := S10_8;

  WriteHorisonSunSensorEmulatorRegister(D_REG_0, D_REG_0Data);
  WriteHorisonSunSensorEmulatorRegister(D_REG_1, D_REG_1Data);
  WriteHorisonSunSensorEmulatorRegister(D_REG_2, D_REG_2Data);
  WriteHorisonSunSensorEmulatorRegister(D_REG_3, D_REG_3Data);
  WriteHorisonSunSensorEmulatorRegister(D_REG_4, D_REG_4Data);

end;

end.
```

APPENDIX D. STAR SENSOR EMULATOR

D.1 Star Sensor Emulator Software

D.1.1 STAR SENSOR EMULATOR DRIVERS

```
Unit StarEmul;

interface

uses
  T800Lib;

type
  TSingleVectorArray = array [1..3,1..3] of Single;

procedure WriteToStarSensorEmulator(NumberOfStars : Word;
            ObservedVectorInBodyCoordinates : TSingleVectorArray;
            ReferenceVectorInOrbitCoordinates : TSingleVectorArray;
            var ArgumentOfPerigeePlusTrueAnomaly : Single;
            var Inclination : Single;
            var RAAN : Single;
            var EstimatedRoll : Single;
            var EstimatedPitch : Single;
            var EstimatedYaw : Single);

implementation

procedure WriteToStarSensorEmulator(NumberOfStars : Word;
            ObservedVectorInBodyCoordinates : TSingleVectorArray;
            ReferenceVectorInOrbitCoordinates : TSingleVectorArray;
            var ArgumentOfPerigeePlusTrueAnomaly : Single;
            var Inclination : Single;
            var RAAN : Single;
            var EstimatedRoll : Single;
            var EstimatedPitch : Single;
            var EstimatedYaw : Single);
var
  SyncCode : LongInt;
  RxError : Boolean;
  RxBuffer : array [1..6] of Single;
  TxError : Boolean;
  m, n : integer;
begin
  repeat
    RxError := ReadLongIntFromLinkAdaptor(SyncCode);
    If RxError then writeln('Link Adaptor Rx TimeOut at SyncCode');
  until (SyncCode = 1234567890) or RxError;
  if not RxError then
    begin
      for n := 1 to 6 do
        begin
          RxError := ReadSingleFromLinkAdaptor(RxBuffer[n]);
          if RxError then
            begin
              Writeln('Link Adaptor Rx TimeOut at RxBuffer[',n,']');
              exit;
          end;
        end;
      if not RxError then
      begin
        ArgumentOfPerigeePlusTrueAnomaly := RxBuffer[1];
        Inclination := RxBuffer[2];
        RAAN := RxBuffer[3];
        EstimatedRoll := RxBuffer[4];
        EstimatedPitch := RxBuffer[5];
        EstimatedYaw := RxBuffer[6];
      end;
    end;
  TxError := WriteLongIntToLinkAdaptor(1234567890);
  if TxError then Writeln('Link Adaptor Tx TimeOut at SyncCode');
  TxError := WriteByteToLinkAdaptor(Hi(NumberOfStars));
  TxError := WriteByteToLinkAdaptor(Lo(NumberOfStars));
  for m := 1 to NumberOfStars do
    begin
      for n := 1 to 3 do
      begin
        TxError := WriteSingleToLinkAdaptor(
            ObservedVectorInBodyCoordinates[m,n]);
        if TxError then
          begin
            Writeln('Link Adaptor Tx TimeOut at Observed Vector[',
                     m,',  ',n,']');
            exit;
          end;
      end;
    end;
  for m := 1 to NumberOfStars do
    begin
      for n := 1 to 3 do
      begin
        TxError := WriteSingleToLinkAdaptor(
            ReferenceVectorInOrbitCoordinates[m,n]);
        if TxError then
          begin
            Writeln('Link Adaptor Tx TimeOut at Reference Vector [',
                     m,',  ',n,']');
            exit;
          end;
      end;
    end;
end;

end.
```

D.1.2 TRANSPUTER CARD DRIVERS

The transputer card for the star sensor emulator uses the same drivers which are used for the transputer card of the OBC emulator.