Aspect of a hardware-in-the-loop Integrated Test System

Lungile L. Grungxu

Thesis presented in partial fulfillment of the requirements for the degree of Masters of Science in Engineering Science at the University of Stellenbosch

Supervisor: Prof. J. J. du Plessis

Co-Supervisor: Dr. E. M. Hugo

March 2003

Stellenbosch University http://scholar.sun.ac.za

Declaration

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

Stellenbosch University http://scholar.sun.ac.za

ABSTRACT

A multiprocessor hardware-in-the-loop operating system was developed for the Integrated Test System (ITS) and is aimed at implementing the ITS as a space emulation vehicle. The thesis contains a study of satellite orbits, Kepler elements, geomagnetic fields and communication protocol between the processors.

The system structure consists of an orbit generator, a core-operating system and is presented with a study of the satellite sensors. In implementing the orbit propagator, there was a need to pay special attention to the Halving algorithm, the Newton Raphson method and the True Solution. These algorithms were used to calculate the true anomaly angle as a function of eccentric anomaly. The communications protocol was tested and all the errors, with their solutions, have been discussed.

A concept of a geomagnetic field emulator has also been included in the hardware-in-the-loop operating system. The evaluation of those aspects of the system and the conclusion are presented together with recommendations.

Stellenbosch University http://scholar.sun.ac.za

OPSOMMING

'n multiprosesseerder Hardeware in die lus bedryfstelsel is ontwikkel vir 'n Geintegreerde Toets Stelsel (ITS) en poog om die ITS te implementeer as 'n ruimte emulasie stelsel. Die tesis behels die studie van sateliet wentelbane, Kepler wentelbaan elemente, geomagnetiese velde en kommunikasie protokolle tussen die prosesseerders.

Die stelsel struktuur betaal uit 'n wentelbaan propageerder, 'n kern bedryfstelsel en 'n studie van satelliet instrumentasie. As 'n deel van die implementering van die wentelbaan propageerder is die halveer algoritme, Newton-Raphson algoritme en die ware oplossing as numeriese oplossings ondersoek. Die kommunikasie protokol is getoets en foute ondersoek en word bespreek.

'n konsep vir 'n Geomagnetiese veld emulasie word die hardeware in die lus stelsel ingesluit. Die stelsel word ge-ewalueer en die gevolgtrekkings en aanbevelings gemaak.

Stellenbosch University http://scholar.sun.ac.za

CONTENTS

DECLARATION I

ABSTRACT II

OPSOMMING III

CONTENTS IV

LIST OF ABREVIATIONS VII

LIST OF SYMBOLS VIII

LIST OF FIGURES XI

1 INTRODUCTION TO THE HARDWARE-IN-THE-LOOP OPERATING SYSTEM 1

1.1 HARDWARE-IN-THE-LOOP OPERATING SYSTEM (SOFTWARE) 2

1.2 LITERATURE SUMMARY 5

1.2.1 Satellite orbits and Kepler elements 5

1.2.2 Communication protocols 17

1.2.3 Geomagnetic fields 19

2 THE SYSTEM STRUCTURE 22

2.1 THE ORBIT GENERATOR 23

2.2 THE CORE-OPERATING SYSTEM 24

2.3 THE COMMUNICATIONS PROTOCOL 24

2.4 SENSORS 25

2.4.1 Star sensors 26

2.4.2 Magnetic field sensor (Magnetometers) 26

2.4.3 Sun sensor 27

3 IMPLEMENTATION OF THE SYSTEM 28

3.1 THE ORBIT GENERATOR/PROPAGATOR 28

3.1.1 True solution 31

3.1.2 Halving algorithm 33

Stellenbosch University http://scholar.sun.ac.za

3.1.3 Newtonian method [3] *34*

3.1.4 Analysis based on the algorithms and with the True Solution as a reference ... *36*

3.1.5 Comparing the approximate solution with the True Solution *36*

3.1.6 Finding the error measure between the approximation and the True Solution. *37*

3.1.7 Halving algorithm and the True Solution *39*

3.1.8 Finding the error measure between the halving algorithm and the true solution *40*

3.1.9 Comparing the Newton Raphson method with the True Solution *40*

3.1.10 Finding the error measure between the Newton Raphson method and the True Solution *41*

3.1.11 Conclusion based on the error measure graphs *42*

3.1.12 Software Design *43*

3.2 THE CORE-OPERATING SYSTEM 44

3.3 THE COMMUNICATIONS PROTOCOL 45

3.3.1 The Receive thread *46*

3.3.2 The Send thread *48*

3.3.3 The major problems encountered while testing the protocol *49*

3.3.4 Solution to the protocol problems *50*

3.4 MAGNETIC FIELD EMULATOR 50

3.4.2 Software design for magnetic field emulator *50*

3.4.3 Implementation of magnetic field emulator *51*

4. EVALUATION OF THE SYSTEM, CONCLUSION AND RECOMMENDATIONS ... 55

REFERENCES 58

APPENDIX A: LISTING OF JAVA PROGRAMS 60

A.1 NEWTON RAPHSON METHOD 60

A.2 TRUE SOLUTION 61

A.3 HALVING ALGORITHM 62

Stellenbosch University http://scholar.sun.ac.za

A.4 ADCS CLASS 63

A.5 OBC1 SIMULATION CLASS 66

A.6 SIMULATION CLASS 68

A.7 SEND THREAD FOR ORIENTATION 74

A.8 RECEIVE THREAD FOR ORIENTATION CLASS 79

A.9 DISPLAYAREA CLASS 89

A.10 KEPLERORBIT CLASS 91

A.11 ORBIT CLASS 95

A.12 ORIENTATION CLASS 97

A.13 SATELLITEORBIT CLASS 99

A.14 SATORBIT CLASS 105

A.15 SEND THREAD FOR THE PROPAGATOR 109

A.16 RECEIVE THREAD FOR THE PROPAGATOR 113

Stellenbosch University http://scholar.sun.ac.za

LIST OF ABREVIATIONS

ITS	Integrated Test System
HIL	Hardware-in-the-loop
OBC1	Primary On-Board Computer
ADCS	Attitude Determination and Control System
RxFlag	Receive Flag
TxFlag	Transmit Flag
TxAck	Transmit Acknowledge flag
lb	lower bound
ub	upper bound
mid	middle term
IGRF	International Geomagnetic Reference Field

Stellenbosch University http://scholar.sun.ac.za

LIST OF SYMBOLS

a	semi-major axis
b	semi-minor axis
r	radial distance between the mass centres of bodies
r_a	apoapsis radius
r_p	periapsis radius
ν	true anomaly angle
e	eccentricity
t	time instant
t_0	time instant at epoch
Δt_1	delta t_1 (change in first time instant)
Δt_2	delta t_2 (change in second time instant)
T	orbital period
μ	Gravitational parameter of the earth, with a value of $3.986 \times km^3/sec^2$
Ω	longitude of ascending node
ω	argument of perigee
P	period
E	eccentric anomaly
M	mean anomaly
M_0	mean anomaly at epoch
i	orbital inclination
θ	longitude
β	latitude

Stellenbosch University http://scholar.sun.ac.za

ϕ	angle between the z-axis and the earth's radial distance to the satellite
R	radius of the earth
n	number of revolutions
$\dot{\Omega}_{Moon}$	rate of change of right ascension of ascending node due to the moon
$\dot{\Omega}_{Sun}$	rate of change of right ascension of ascending node due to the sun
$\dot{\omega}_{Moon}$	rate of change of argument of perigee due to the moon
$\dot{\omega}_{Sun}$	rate of change of argument of perigee due to the sun
$d\Omega_{Moon}$	rate of change of right ascension of ascending node due to the moon
$d\omega_{Sun}$	rate of change of right ascension of ascending node due to the sun
J_n	dimensionless geopotential coefficients, $n = 1, 2, 3, \ldots nth$ term
$\dot{\Omega}_{J_2}$	rate of change of right ascension due to J_2
$d\Omega_{J_2}$	rate of change of right ascension due to J_2
$\dot{\omega}_{J_2}$	rate of change of argument of perigee due to J_2
$d\omega_{J_2}$	rate of change of argument of perigee due to J_2
dt	time increment (time change)
ϕ	angle in the direction of East longitude
B_r	magnetic field dipole aligned in the direction of geocentric distance, r
B_θ	magnetic field dipole in the direction of co-elevation angle, θ
B_ϕ	magnetic field dipole in the direction of East longitude east of the Greenwich

Stellenbosch University http://scholar.sun.ac.za

V	scalar potential function
P_n^m	Legendre function of *m*-order and *n*-degree
g_n^m, h_n^m	Gaussian coefficients of *m*-order and *n*-degree
∇	a gradient vector

Stellenbosch University http://scholar.sun.ac.za

LIST OF FIGURES

FIGURE 1: REAL-TIME SYSTEMS 2

FIGURE 1.1: HARDWARE-IN-THE-LOOP OPERATING SYSTEM 4

FIGURE 1.2.1: AREA/TIME RELATIONSHIP. KEPLER'S SECOND LAW STATES THAT THE VELOCITY OF AN OBJECT CHANGES WITH ORBITAL RADIUS [8]. 8

FIGURE 1.2.2: TRUE ANOMALY ANGLE DECREASES AS THE SATELLITE APPROACHES THE APOGEE POINT. 9

FIGURE 1.2.3: ORBITAL ELEMENTS. SIX INDEPENDENT QUANTITIES ARE REQUIRED TO COMPLETELY DESCRIBE AN ORBIT. 10

FIGURE 1.2.4: DEFINITION OF TRUE ANOMALY, ν, AND ECCENTRIC ANOMALY, E. THE OUTER FIGURE IS A CIRCLE WITH RADIUS EQUAL TO THE SEMI-MAJOR AXIS [1]. 12

FIGURE 1.2.5: DEFINING AN ORIENTATION OF AN ORBIT IN SPACE. SEE ALSO FIGURE 1.2.3 13

FIGURE 1.2.6: THE RADIAL DISTANCE, CO-ELEVATION AND THE LONGITUDE EAST OF GREENWICH. 21

FIGURE 2: HARDWARE-IN-THE-LOOP OPERATING SYSTEM. 22

FIGURE 2.1: TRUE ANOMALY ANGLE DECREASES AS THE SATELLITE APPROACHES THE APOGEE POINT. 23

FIGURE 3: RADIAL LINES CLOSELY PACKED TO EACH OTHER IN THE SECOND AND THIRD QUADRANT 29

FIGURE 3.1: TRUE ANOMALY VERSUS MEAN ANOMALY ANGLE. 30

FIGURE 3.2: THE TRUE SOLUTION METHOD. 31

FIGURE 3.3: GRAPH OF TRUE ANOMALY VERSUS MEAN ANOMALY USING TRUE SOLUTION 32

FIGURE 3.4: THE HALVING ALGORITHM THAT ITERATES UNTIL THE VALUE OF F(E) = 0, WHERE E = MEDIAN 33

FIGURE 3.5: TRUE ANOMALY VERSUS MEAN ANOMALY ANGLE FOR THE HALVING ALGORITHM. 34

FIGURE 3.6: TRUE ANOMALY VERSUS MEAN ANOMALY ANGLE FOR THE NEWTON RAPHSON METHOD 35

FIGURE 3.7: COMPARING THE APPROXIMATE SOLUTION WITH THE TRUE SOLUTION. 36

FIGURE 3.8: FINDING THE ERROR MEASURE BETWEEN THE APPROXIMATION AND THE TRUE SOLUTION 37

Stellenbosch University http://scholar.sun.ac.za

FIGURE 3.9: COMPARING THE HALVING ALGORITHM WITH THE TRUE SOLUTION. 39

FIGURE 3.10: FINDING THE ERROR MEASURE BETWEEN THE HALVING ALGORITHM AND THE 40

TRUE SOLUTION GRAPHS. 40

FIGURE 3.11: COMPARING THE NEWTON RAPHSON METHOD WITH THE TRUE SOLUTION...... 41

FIGURE 3.12: FINDING THE ERROR MEASURE BETWEEN THE NEWTON RAPHSON METHOD AND THE TRUE SOLUTION GRAPHS. 42

FIGURE 3.13: FLOWCHART FOR THE ORBIT GENERATOR 43

FIGURE 3.14: FLOWCHART FOR THE CORE-OPERATING SYSTEM. 45

FIGURE 3.15: A FLOWCHART FOR THE RECEIVE THREAD 47

FIGURE 3.16: A FLOWCHART FOR THE SEND THREAD. 49

FIGURE 3.17: EARTH'S MAGNETIC FIELD EMULATOR 51

FIGURE 3.18: A JAVA CODE FOR DETERMINING B_r WHICH IS THE MAGNETIC FIELD DIPOLE ALIGNED IN THE DIRECTION OF GEOCENTRIC DISTANCE. 52

FIGURE 3.19: A JAVA CODE FOR COMPUTING B_θ WHICH IS THE MAGNETIC FIELD DIPOLE ALIGNED IN THE DIRECTION OF THE CO-ELEVATION ANGLE. 53

FIGURE 3.20: A JAVA CODE FOR CALCULATING B_ϕ WHICH IS THE MAGNETIC FIELD DIPOLE AGLINED IN THE DIRECTION OF LONGITUDE, ϕ, EAST OF GREENWICH. 54

Stellenbosch University http://scholar.sun.ac.za

1 INTRODUCTION TO THE HARDWARE-IN-THE-LOOP OPERATING SYSTEM

This thesis involves the development of a hardware-in-the-loop (HIL) operating system for the Integrated Test System (ITS). The Integrated Test System is a computer system that is used to test satellite components. It acts as a simulation machine, as the conditions in space can be emulated in order to test whether the component is going to behave properly or in an unpredictable manner. The component where there are errors is rectified and again tested in the integrated test system. Therefore, the system acts as a simulation environment in which components or software products for the satellite can be tested to see how they will behave when put in the real space environment. The ITS was built by Stellenbosch University in order to test the components for the SUNSAT satellite. To implement the ITS as a vehicle for space environment emulation, an operating system shell to support the hardware-in-the-loop space simulation role of the ITS therefore needs to be developed.

The following topics are covered in this thesis:

- Chapter 1 presents an introduction to the hardware-in-the-loop operating system and covers the theory of Kepler's orbital elements.

- Chapter 2 contains a discussion of the system structure, including the orbit propagator, which is also called an orbit generator, the core-operating system, which is a controlling interface between the orbit propagator and the on-board computer 1, the communication protocol and the sensors.

- Chapter 3 discusses the implementation of the system, which includes the orbit generator, the core-operating system, the communication protocol and the magnetic field emulator.

- Chapter 4 presents an evaluation of the system, a conclusion and recommendations.

Stellenbosch University http://scholar.sun.ac.za

1.1 Hardware-in-the-loop operating system (Software)

In short, a real-time system is a computing (operating) system that must react within precise time constraints to events in the environment [5]. Time constraint means that the system must react at the time at which the events occur in the environment. Furthermore, the correct behaviour of the system also depends on the logical value of the computation. It means that the results that are measured from the environment must be within the tolerable limits.

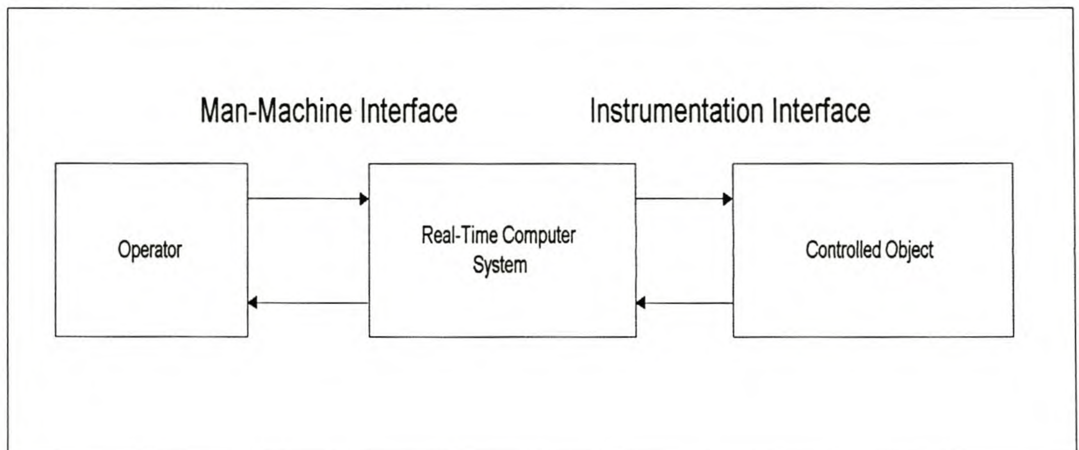


Figure 1: Real-time systems

Many real-time systems are characterised by the fact that severe consequences will result if the timing and logical correctness properties of the system are not satisfied [7]. Figure 1 shows a sketch of a real-time system [6]. It consists of an operator sub-system, a real-time computer system, and a controlled sub-system or object. A real-time computer system is part of a larger system called a real-time system.

Real-time systems can be decomposed into a set of sub-systems called **clusters** [6]. For example, as depicted in Figure 1, the controlled object (controlled cluster), the real-time computer system (the computational cluster) and the human operator (the operator cluster) [6] make up a real-time system. The interface between the human operator and the real-time computer system is called the **man-machine interface**, and the interface between the

Stellenbosch University http://scholar.sun.ac.za

controlled object and the real-time computer system is called the **instrumentation interface** [6]. The man-machine interface consists of input devices (e.g. keyboards) and output devices (e.g. display) that interface with the human operator [6]. The instrumentation interface consists of the sensors and actuators that transform the physical signals (e.g. voltages, currents) in the controlled object into a digital form, and vice versa [6].

The hardware-in-the-loop operating system (HIL) is a real-time system that consists of a distributed operating system, which consists of the orbit propagator with a graphical user interface, the core-operating system and the hardware that will interface with the system. Therefore, the man-machine interface for the HIL will be the graphical user interface, the real-time computer system will be the orbit propagator and the ITS, and the instrumentation interface will be the hardware connected to the ITS. The Integrated Test System is a system that has been designed to simulate the space environment for a satellite. A hardware will be designed and integrated into such a system. This system will then simulate the conditions for the hardware and the hardware response will be monitored and analysed very thoroughly. In fact, it will be as if the system is a satellite in orbit, and the environmental conditions that it will simulate are exactly the same as those that will be encountered by the satellite in space. However this system will need to have an operating system shell in order to implement it as a vehicle for a space emulation environment. Hence, a hardware-in-the-loop operating system needs to be developed.

Stellenbosch University http://scholar.sun.ac.za

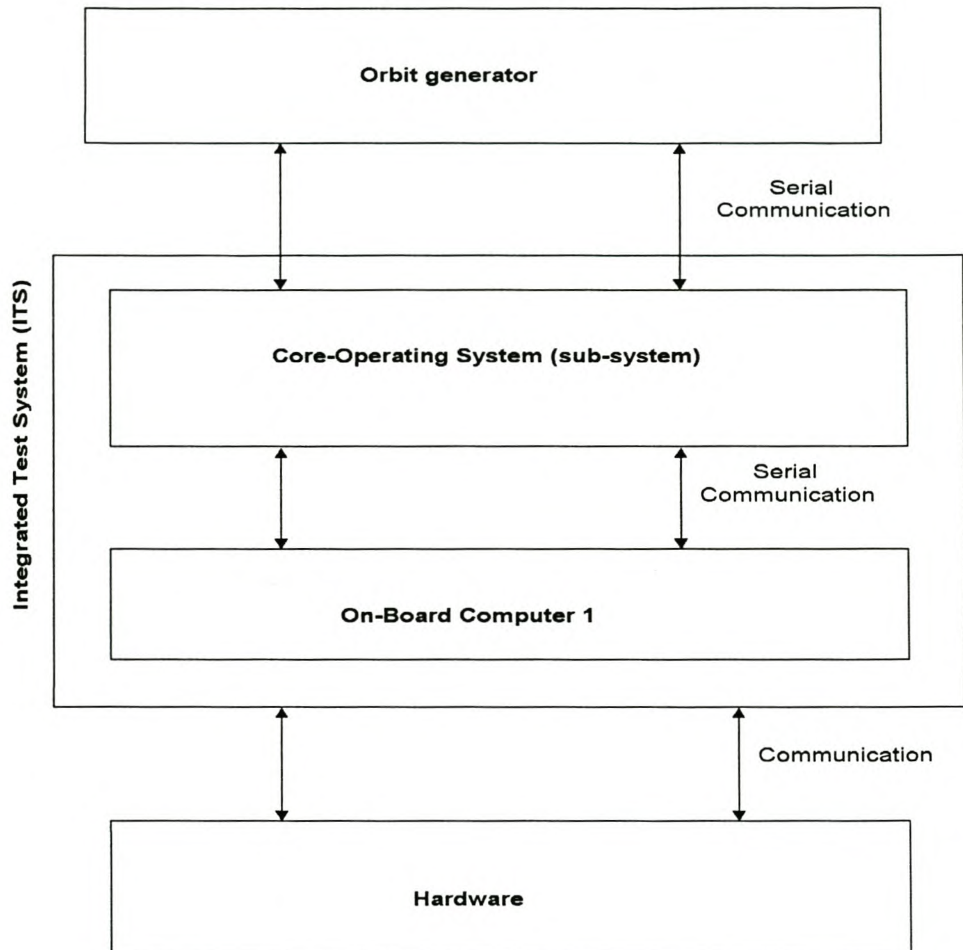


Figure 1.1: Hardware-in-the-loop operating system

This project involved the development of such an operating system. As shown in Figure 1.1 above, the system will be a distributed operating system consisting of two parts: the orbit propagator, which is labelled as orbit generator, and the core-operating system, which will be part of the ITS.

The orbit propagator will be able to communicate with the core-operating system, which also is connected to OBC1. The hardware is connected to the ITS, as shown in Figure 1.1, and the orbit propagator will simulate positioning parameters based on Kepler elements and pass them to the ITS. The ITS will be controlled by the core-operating system and will be

Stellenbosch University http://scholar.sun.ac.za

asked to perform certain services, and it will communicate the message to the hardware. The hardware will perform the service requested by the ITS and send back a response. The response will be captured by the ITS and communicated to the core-operating system. The ITS can then issue another command, depending on the kind of response the hardware is sending. This process will continue and will become a software command and hardware response command loop, forming what can be referred to as the hardware-in-the-loop operating system.

1.2 Literature summary

An analysis and thorough understanding of Kepler elements are required for this thesis. These elements will be used to propagate the positioning and attitude of the satellite. However, it is also necessary to provide a detailed discussion of the satellite's orbits, as they form the foundation for the study of Kepler elements.

1.2.1 Satellite orbits and Kepler elements

Kepler's First Law states that captive satellites, those with closed orbital paths, will travel around the earth in elliptical or circular paths, with the center of the earth located at one of the foci [9], as depicted in Figure 1.2 below.

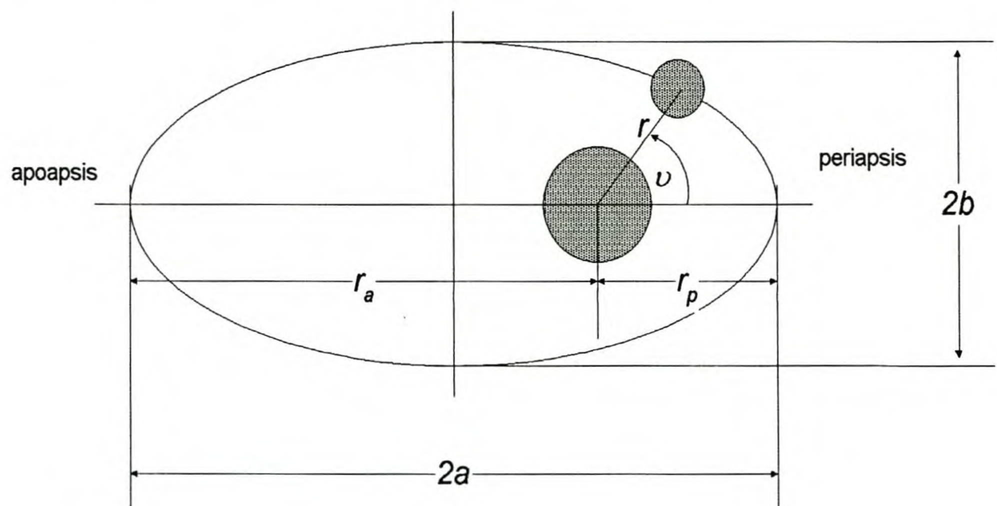


Figure 2.2: The ellipse, of which the geometry is described by the orbital elements

Stellenbosch University http://scholar.sun.ac.za

a = semi-major axis

b = semi-minor axis

r = radial distance between the mass centres of the bodies

r_a = apoapsis radius (maximum distance between bodies)

r_p = periapsis radius (minimum distance between bodies)

ν = true anomaly (measured in the same direction as the movement in the orbit)

It is necessary to discuss the orbital parameters and the geometry of the ellipses, because they reveal some very important relationships between these orbital parameters. As can be seen from Figure 1.2 above, it is clear that:

$$r_a + r_p = 2a \tag{1.2.1}$$

$$a = \frac{r_a + r_p}{2} \tag{1.2.2}$$

The eccentricity, ***e***, describes the shape of an ellipse in terms of how fat it is with respect to the semi-major axis, ***a***, and the semi-minor axis, ***b***.

$$e = \sqrt{1 - \frac{b^2}{a^2}} \tag{1.2.3}$$

Together, the orbital parameters ***a*** and ***e*** define the size and shape of an ellipse [8]. As can be seen in Figure 1.2, as ***b* increases**, the **shape of the orbit becomes like the shape of a circle**. **Equation 1.2.3** shows that, as ***b*** approaches the value of ***a***, the value of ***e*** approaches **0**. This implies that, when ***e*=0**, the orbit is a circle. The other observation from **equation 1.2.3** is that, when ***b*** approaches **0**, the value of ***e*** approaches **1** and the orbit becomes more elliptical in shape. Figure 1.2 shows that, if ***a*** is kept constant with ***b*** approaching zero, the shape of the orbit becomes more elliptical. The general equation for a conic section, of which an ellipse is just one type, may be written in a form that reveals many useful relationships:

Stellenbosch University http://scholar.sun.ac.za

$$r = \frac{a(1-e^2)}{1+e(\cos\upsilon)} \qquad (1.2.4)$$

where r represents the radial distance (or orbital radius) between the mass centers of the bodies. The true anomaly, ν, is the angle measured from the major axis line (in the direction pointing towards the periapsis) to the radial line between the two bodies. This is measured in the same direction as the motion of the orbiting body.

To summarise the whole discussion, a closer look at the conic section equation reveals that the eccentricity can tell us immediately what type of orbit we are in. If we only consider positive eccentricities, the first limiting case is when ***e*=0**, which indicates a conic section and, thus, a **circular orbit**. The next limiting case occurs when ***e*=1**. For this value, the conic equation gives a value of infinity for the radius at the point where the true anomaly approaches **180°**. This corresponds to a parabolic conic section. Values of eccentricity greater than 1 indicate a hyperbolic conic section. Parabolic and hyperbolic orbits represent "open" or non-repeating orbits. The final case is when the eccentricity is greater than zero, but still less than one, that is $0 < e < 1$. For this range of values of ***e***, the orbit is elliptical.

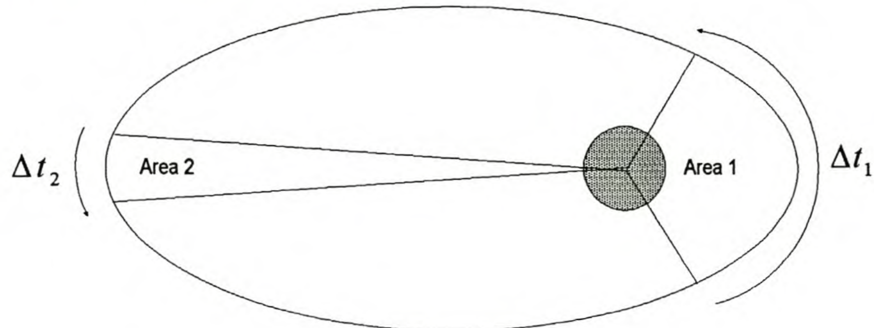
For fixed values of the eccentricity and semi-major axis, the orbital radius is a function of the true anomaly, which is a measure of where along the conic section the orbiting body is [8]. Substituting different values for the true anomaly, one would find that the minimum radius occurs when ν = **0°**, and the maximum radius occurs when ν = **180°** [8]. These distances to the periapsis and apoapsis points in the orbit shown in Figure 1.2 prove that they are associated with the minimum and maximum distances between the bodies in an elliptical orbit [8]. Values of eccentricity with this range therefore correspond to an elliptical conic section and would represent an elliptical orbit [8]. At periapsis (ν = 0°) and apoapsis (ν = 180°), the conic section equation (equation 1.2.4) simplifies to reveal two useful relationships [8]:

$$r_a = a(1+e) \text{ and } r_p = a(1-e) \qquad (1.2.5)$$

Stellenbosch University http://scholar.sun.ac.za

It is important to realise that the orbital radius increases continuously from periapsis to apoapsis and decreases continuously from apoapsis to periapsis when moving in an elliptical orbit [8].

Kepler's second law reveals that a line drawn between the two bodies will sweep out the same amount of area during the same time period anywhere along the orbital path [8]. This characteristic is illustrated in figure 1.2.1 below [8].



$\Delta t_2 = \Delta t_1$

Area 1 = Area 2

Figure 1.2.1: Area/time relationship. Kepler's second law states that the velocity of an object changes with orbital radius [8].

The most important point revealed by this law is that the speed of an object in an orbit changes with changing distance between the bodies [8]. For the areas shown in Figure 1.2.1 to be the same, the orbiting body must slow down when it is farther away, so that the line between the bodies sweeps out the same area in the same amount of time as when it is travelling closer. Hence, a satellite in orbit is expected to move faster when it approaches the perigee, and to move slowly when it approaches the apogee point, as shown in Figure 1.2.2.

Stellenbosch University http://scholar.sun.ac.za

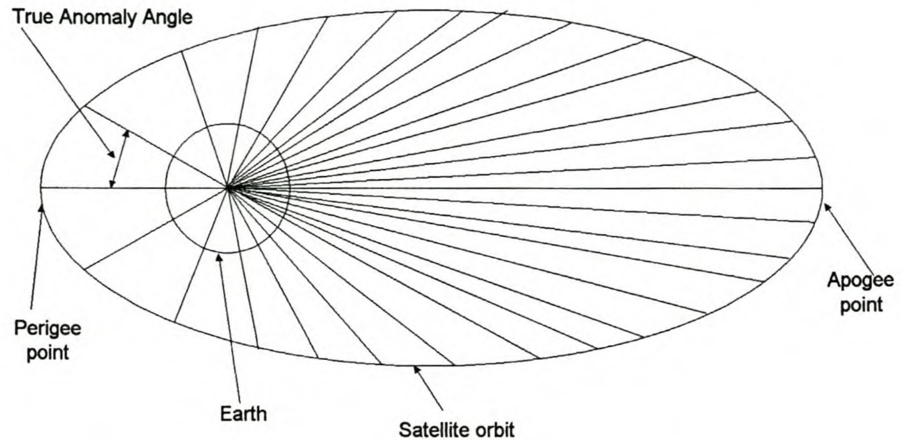


Figure 1.2.2: True anomaly angle steps per second decreases as the satellite approaches the apogee point.

In other words, an object travels slowly when it is in apogee and moves faster when it is in perigee [8]. This is illustrated by the fact that the radial distances appear to be closely packed to each other at the region of the apogee point, and are far apart at the region of the perigee point.

Kepler's third law reveals that a relationship exists between the semi-major axis and the orbital period [8]. Stated mathematically [8]:

$$T^2 = \left(\frac{4\pi^2}{\mu}\right)a^3 \tag{1.2.6}$$

where ***T*** represents an orbital period, or the time it takes to travel through one complete orbit with semi-major axis ***a*** [8]. The term μ represents a gravitational parameter that has a specific value for each body around which an orbit may be described [8]. The gravitational parameter for the earth is $\mu = 3.986 \times 10^5 km^3 / \sec^2$ [8].

Equation 1.2.6 can be simplified to make the semi-major axis a subject of the formula, as follows:

Stellenbosch University http://scholar.sun.ac.za

$$a = \left[\left(\frac{T}{2\pi} \right)^2 \mu \right]^{\frac{1}{3}} \tag{1.2.7}$$

$$\cong 331.24915 T^{\frac{2}{3}} km \tag{1.2.8}$$

where the orbital period is in minutes [4].

Orbital elements: To completely describe the shape and orientation of an orbit around the earth, as well as the position of the satellite in the orbit, six quantities must be specified, as shown in Figure 1.2.3 [8] below.

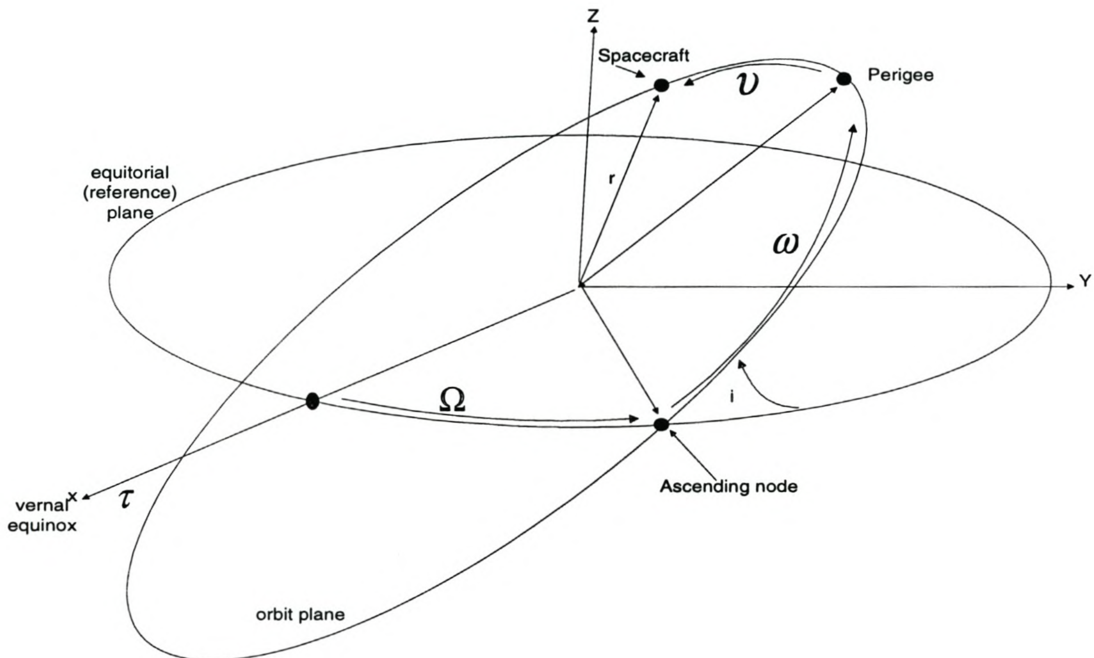


Figure 1.2.3: Orbital elements. Six independent quantities are required to completely describe an orbit.

Stellenbosch University http://scholar.sun.ac.za

While doing that, it is necessary to bear in mind that, once set, the plane in which an orbit lies remains inertially fixed in space [8]. As stated earlier, the eccentricity, ***e***, and semi-major axis, ***a***, define the shape and size of the orbit respectively.

The orientation of the orbital plane is described by two angles: the **inclination angle, *i***, and the **longitude of the ascending node, Ω** [8]**.** The inclination describes the angle that the orbital plane makes with a reference plane (chosen to the plane of the equator for earth-orbiting satellites) [8]. The **longitude of the ascending node** is an angle measured from the line, which is pointing towards the **vernal equinox** on the **reference plane**, toward the **line of nodes**, as shown in Figure 1.2.4 [8]. More clearly put, it describes the rotation of the orbital plane from the vernal equinox reference line to the line of nodes. This **longitude of the ascending node** is always measured where the orbital motion is from south to north [8]. The vernal equinox is a direction that is inertially fixed infinitely far away in space. The **line of nodes** is the line that is formed by the intersection of the reference plane with the orbital plane [8].

The **argument of perigee,** ω, describes the orientation of the elliptically-shaped orbit within the orbital plane. It is measured as the angle from the line of nodes to the radius of perigee of the orbit [8]. Finally, a parameter must be used to describe the actual position of the orbiting body at any particular time or location within the orbit. This parameter is called the **true anomaly angle,** ν. This angle is measured from the perigee point to the satellite.

Calculation of true anomaly: The true anomaly angle is difficult to calculate, necessitating the introduction of some geometry, as shown in Figure 1.2.4 below. Firstly, there is a parameter called mean anomaly, ***M***, which is **360.(*Δt/P*)deg**, where ***P*** is the orbital period and ***Δt*** is the time since perigee passage of the satellite [1]. Therefore, ***M*** = ν for a satellite in circular orbit [1]. The **eccentric anomaly, *E*,** is an intermediate angle that relates the true anomaly, ν, and the mean anomaly, ***M*** [1], to one another. ***E*** is the angle measured at the centre of the orbit between perigee and the projection of the satellite onto a circular orbit with the same semi-major axis, as shown in Figure 1.2.4 [1]. The mean and eccentric anomalies are related through Kepler's equation [1]:

Stellenbosch University http://scholar.sun.ac.za

$$M = E - e\sin E \tag{1.2.9}$$

where ***e*** is the eccentricity.

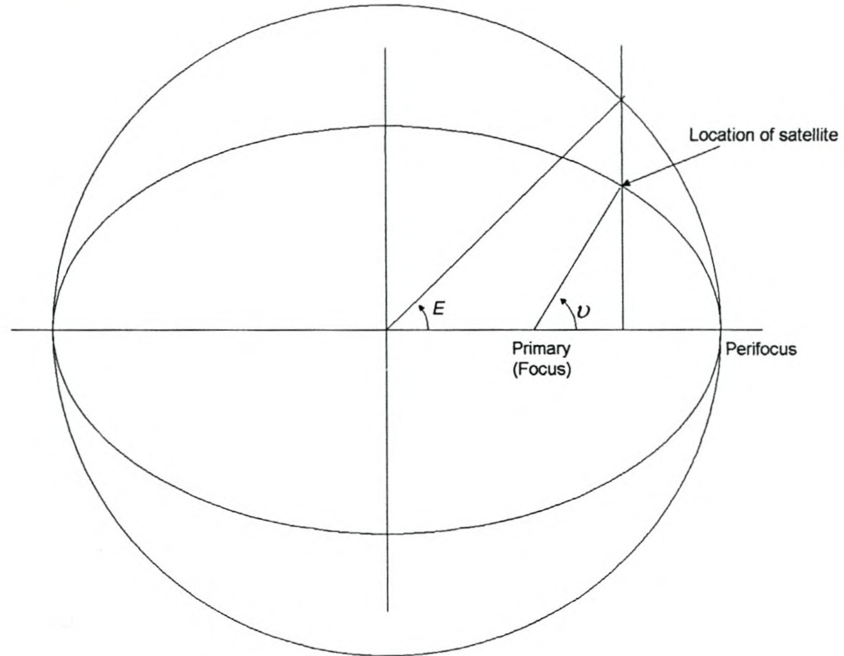


Figure 1.2.4: Definition of true anomaly, ν, and eccentric anomaly, E. The outer figure is a circle with radius equal to the semi-major axis [1].

If t_0 is the epoch time of the elements, then the mean anomaly at time t is found from the mean anomaly at epoch by

$$M = M_0 + n(t - t_0) \tag{1.2.9}$$

where $n \equiv 2\pi / period$ is the mean motion, and $\boldsymbol{M_0}$ is the mean anomaly at epoch [1].

E is then related to ν by Gauss' Equation:

$$\tan\left(\frac{\upsilon}{2}\right) = \left(\frac{1+e}{1-e}\right)^{1/2} \tan\left(\frac{E}{2}\right) \tag{1.2.10}$$

$\left(\frac{E}{2}\right)$ and $\left(\frac{\upsilon}{2}\right)$ are always used, because these quantities are always in the same quadrant [1]. For small eccentricities, ν may be expressed directly as a function of ***M*** by expanding in a power series in ***e*** to yield:

Stellenbosch University http://scholar.sun.ac.za

$$\upsilon \approx M + 2e\sin 2M + \frac{5}{4}e^2 \sin 2M + \vartheta(e^3) \tag{1.2.11}$$

Once ν is found, the longitude, latitude and height can be determined. As presented earlier in equation 1.2.4, the radial distance ***r*** is given by:

$$r(t) = \frac{a(1-e^2)}{1+e(\cos\upsilon(t))} \tag{1.2.12}$$

This means that it now is possible to find the position of a spacecraft in the orbit plane. Using spherical triangles, as shown in Figure 1.2.5 below [1], it is possible to give the x-component [1] of a satellite position as

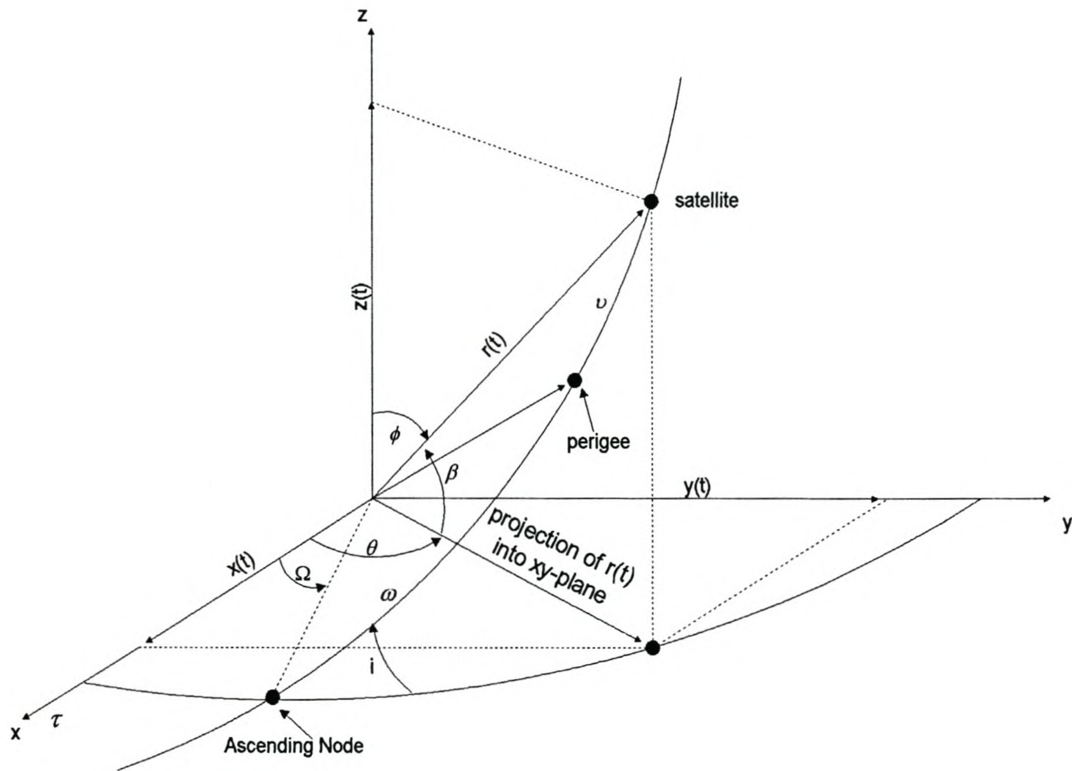


Figure 1.2.5: Defining an orientation of an orbit in space. See also Figure 1.2.3

$$x = r[\cos(w+\upsilon).\cos(\Omega) - \sin(w+\upsilon).\sin(\Omega).\cos(i)]$$

Stellenbosch University http://scholar.sun.ac.za

$$\therefore\ x(t) = r(t).[\cos(w + \upsilon(t)).\cos(\Omega) - \sin(w + \upsilon(t)).\sin(\Omega).\cos(i)] \qquad (1.2.13)$$

and the y-component [1] is given by

$$y = r[\cos(w + \upsilon).\sin(\Omega) + \sin(w + \upsilon).\cos(\Omega).\cos(i)]$$

$$\therefore\ y(t) = r(t).[\cos(w + \upsilon(t)).\sin(\Omega) + \sin(w + \upsilon(t)).\cos(\Omega).\cos(i)] \qquad (1.2.14)$$

The z-component is therefore given by the following equation [1]:

$$z = r[\sin(w + \upsilon).\sin(i)]$$

$$\Rightarrow\ z(t) = r(t)[\sin(w + \upsilon(t)).\sin(i)] \qquad (1.2.15)$$

Now, since the position of the satellite has been found in *catersian coordinates*, it has to be converted into *spherical coordinates* in order to obtain *longitude* (***θ(t)***), *latitude* (***β(t)***) and the *height*. The measurements will be assumed to be relative to the earth surface for the reasons of simplicity. Hence, the *longitude* is given in terms of ***x*** and ***y*** as a function of time by

$$\begin{aligned} longitude &= \theta(t) \\ &= \arctan\left[\frac{y(t)}{x(t)}\right] \end{aligned} \qquad (1.2.16)$$

and the *latitude* in terms of ***z*** and ***r*** as a function of time by

$$\begin{aligned} latitude &= 90^\circ - \phi(t) \\ &= 90^\circ - \arccos\left[\frac{z(t)}{r(t)}\right] \\ &= \beta(t) \\ &= \arcsin\left(\frac{z(t)}{r(t)}\right) \end{aligned} \qquad (1.2.17)$$

Stellenbosch University http://scholar.sun.ac.za

The height of a satellite above the surface of the earth will be given by subtracting the radius ***R*** of the earth from the radial distance ***r(t)***. Hence, the *height* is given by

$$height = r(t) - R \qquad (1.2.18)$$

as a function of time. However, the **orbit undergoes some perturbations** due to the variation of the gravitational forces of the moon and sun. These variations affect only the right ascension of **the ascending node, Ω, the argument of perigee,** ω, **and the mean anomaly,** ***M***. Therefore, for nearly circular orbits, the rates of change due to the sun and moon are:

Right ascension of the ascending node:

$$\dot{\Omega}_{Moon} = -0.00338(\cos i)/n$$

$$\Rightarrow \quad d\Omega_{Moon} = -0.00338(\cos i)/n \qquad (1.2.19)$$

and

$$\dot{\Omega}_{Sun} = -0.00154(\cos i)/n$$

$$\Rightarrow \quad d\Omega_{Sun} = -0.00338(\cos i)/n \qquad (1.2.20)$$

Argument of perigee:

$$\dot{\omega}_{Moon} = 0.00169\left(4 - 5\sin^2 i\right)/n$$

$$\Rightarrow \quad d\omega_{Moon} = 0.00169\left(4 - 5\sin^2 i\right)/n \qquad (1.2.21)$$

and

$$\dot{\omega}_{Sun} = 0.00077\left(4 - 5\sin^2 i\right)/n$$

$$\Rightarrow \quad d\omega_{Sun} = 0.00077\left(4 - 5\sin^2 i\right)/n \qquad (1.2.22)$$

n = number of revolutions

i = orbital inclination

Stellenbosch University http://scholar.sun.ac.za

dw and $d\Omega$ are in deg/*day*

The above equations were developed on the assumption that the earth has a spherically-symmetric mass distribution [4]. But, the fact is that the **earth has a bulge at the equator**, being slightly pear shaped and **flattened at the poles** [4]. The potential generated by the non-spherical earth causes periodic variations in all of the orbital elements. However, the dominant effects are secular variations in right ascension of the ascending node and the argument of perigee because of the earth's oblateness. The rates of change of **Ω** and ω are

$$\dot{\Omega}_{J2} = -1.5nJ_2.(R_E/a)^2.(\cos i).(1-e^2)^{-2}$$

$$\Rightarrow d\Omega_{J_2} = -1.5nJ_2.(R_E/a)^2.(\cos i).(1-e^2)^{-2}$$

$$\cong -2.06474\times 10^{14} a^{-7/2}(\cos i).(1-e^2)^{-2} \qquad (1.2.23)$$

and

$$\dot{\omega}_{J_2} = 0.75nJ_2.(R_E/a)^2.(4-5\sin^2 i).(1-e^2)^{-2}$$

$$\Rightarrow d\omega_{J_2} = 0.75nJ_2.(R_E/a)^2.(4-5\sin^2 i).(1-e^2)^{-2}$$

$$\cong 1.0323710^{14} nJ_2.a^{-7/2}.(4-5\sin^2 i).(1-e^2)^{-2} \qquad (1.2.23)$$

where ***n*** is mean motion in deg/*day*, R_E is the earth's equitorial radius, ***a*** is the semi-major axis in km, e is eccentricity, i is inclination, and $\dot{\Omega}$ and $\dot{\omega}$ are in deg/*day*. J_2 ... J_n are the zonal coefficients or geo-potential coefficients.

For satellites in GEO and below, J_2 perturbations dominate, but for the satellites above GEO, the sun and moon perturbations dominate. Orbit designers choose the orbital inclination so that the rate of change of perigee is zero.

$$J_2 = 0.00108263 \qquad (1.2.24)$$

$$J_3 = -0.00000254 \qquad (1.2.25)$$

Stellenbosch University http://scholar.sun.ac.za

It is now possible to add equations for the angle of the ascending node and the argument of perigee after taking their rates of change due to perturbations into consideration.

$$n = \frac{24 \times 60}{period}$$

$$\Omega = \Omega + d\Omega_{Moon}\, dt \quad (1.2.26)$$

$$\omega = \omega + d\omega_{Moon}\, dt \quad (1.2.27)$$

and

$$\Omega = \Omega + d\Omega_{Sun}\, dt \quad (1.2.28)$$

$$\omega = \omega + d\omega_{Sun}\, dt \quad (1.2.29)$$

Therefore:

$$\Omega = \Omega + (d\Omega_{Moon} + d\Omega_{Sun}).dt \quad (1.2.30)$$

$$\omega = \omega + (d\omega_{Moon} + d\omega_{Sun}).dt \quad (1.2.31)$$

For this thesis, we will make the assumption that the earth is spherical, and the rates of change that are used are given by equations 1.2.19 to 1.2.22. The results from these equations for the rates of change will be substituted into equations 1.2.30 and 1.2.31. This will give the angle of the ascending node and the argument of perigee. With these parameters, it is possible to tell exactly the position of a satellite at a specific instant in time.

1.2.2 Communication protocols

To summarise, it is proper to indicate that the thesis will cover the communications protocol as well as magnetic fields theory. It therefore is important to briefly discuss data communications, because the thesis covers communications protocol. Firstly, two computers are interconnected if they are capable of exchanging information [10]. The connection has to be via copper wires, lasers, microwaves, or earth satellites [10]. This exchange of information between two or more computers is known as data communication or computer communication. Any interconnection of two or more computers is called a computer network [10]. Furthermore, communication networks of computers are in existence today to

Stellenbosch University http://scholar.sun.ac.za

facilitate the communications between various computers, and also to facilitate the sharing of resources [11].

There are different types of design architectures of communication networks [12], but these will not be discussed, as only the serial communication of two computers is of interest for the purposes of this thesis. The major part of these network architectures is a set of communication protocols [12]. A communication protocol can be defined informally as a set of rules or conventions used to control the transfer of data in a computer communications system.

Protocols contain precise specifications of data and control message formats (e.g. headers, trailers and maximum lengths) [12]. By defining procedures to control message flows (e.g. error handling and speed matching), protocols establish what can be viewed as logical communication paths between communicating entities [12]. These logical paths may or may not correspond to direct physical connections [12]. Protocol formats may be interpreted as defining protocol syntax and control procedures as defining protocol semantics and timing [12].

Protocols are essential for providing basic network functionality [12]. Data Link Control protocols are concerned with the communication of data between different machines [12]. Communication is considered to be accomplished if the intended receiver acquires access to the same serial stream of bits as that of the sender within some acceptable time delay [12]. Thus, "communication" here is used to mean the synchronisation of data [12]. Communication between "processes" within the same machine is easily accomplished with shared memory and an operating system that coordinates the communicating processes [12]. The communications protocol has also been explained in section 2.3, and the magnetic fields in section 2.4.3. Since the thesis concerns operating systems, a brief discussion of these are required. An operating system is a system that is used to control all the resources that are available to it, as well as the subsystems that work with such a system. Deitel [9] defines an operating system as the programs that make the hardware usable, that is the software that controls the hardware. In a computer system, the hardware provides the raw

Stellenbosch University http://scholar.sun.ac.za

computing power, while the operating system makes this conveniently available to the users, who manage the hardware to achieve good performance [9].

Furthermore, operating systems are primarily resource managers, and the resources they manage are the computer hardware, such as processors, storage, input/output devices, communication devices, and data [9]. Operating systems also perform many functions, such as implementing the user interface, sharing hardware among users, allowing users to share data among themselves, preventing users from interfering with one another, scheduling resources among users, facilitating input/output, recovering from errors, accounting for resource usage, facilitating parallel operations, organising data for secure and rapid access and also handling network communications [9].

1.2.3 Geomagnetic fields

To conclude the literature study, it is important to briefly discuss geomagnetic fields, since they form an important part of the thesis. Geomagnetic fields are fundamental to the study of satellite control. The measurement of their precise direction and magnitude help in determining the attitude of a satellite in a specified position. They are modelled by means of the potential function and its gradient vectors [1]. If V is a potential function, then the earth's magnetic fields, $\boldsymbol{B}$, can be represented as the gradient of a scalar potential function [1], V, as follows:

$$B = -\nabla V \qquad 1.2.32$$

V can be represented as a series of spherical harmonics [1],

$$V(r,\theta,\phi) = a \sum_{n=1}^{k} \left(\frac{a}{r}\right)^{n+1} \sum_{m=0}^{n} \left(g_n^m \cos(m\phi) + h_n^m \sin(m\phi)\right) P_n^m(\theta) \qquad 1.2.33$$

Where $\boldsymbol{a}$ is the equatorial radius of the earth (6371.2 km adopted for the International Geomagnetic Reference Field, IGRF), g_n^m and h_n^m are Gaussian coefficients [1], r, θ, and ϕ are the geocentric distance, the co-elevation, and the longitude East of Greenwich that

Stellenbosch University http://scholar.sun.ac.za

define any point in space [1]. The Gaussian coefficients are determined empirically by a least squares fit to measurements of the field [1]. A set of these coefficients constitutes a model of the fields [1]. With these coefficients and a definition of the associated Legendre functions, P_n^m, it is possible to calculate the magnetic field at any point in space via equations 1.2.32 and 1.2.33 [1]. It is convenient to obtain a dipole model by expanding the field model to first degree (*n=1*) and all orders (*m=0,1*) [1]. Equation 1.2.33 then becomes

$$V(r,\theta,\phi) = \frac{a^3}{r^2}\left[g_1^0 P_1^0(\theta) + (g_1^1\cos(\phi) + h_1^1\sin(\theta))P_1^1(\theta)\right]$$
$$= \frac{1}{r^2}\left(g_1^0 a^3\cos(\theta) + g_1^1 a^3\cos(\phi)\sin(\theta) + h_1^1 a^3\sin(\phi)\sin(\theta)\right)$$

The $\cos(\theta)$ term is the potential due to a dipole of strength $g_1^0 a^3$ aligned with the polar axis [1]. Similarly, the $\sin(\theta)$ terms are dipoles aligned with the x and y axes [1]. Relying on the principle of linear superposition, these three terms are the Cartesian components of the dipole component of the earth's magnetic field [1].

The dipole field in local tangent coordinates is given by:

$$B_r = 2\left(\frac{a}{r}\right)^3\left[g_1^0\cos\theta + \left(g_1^1\cos\theta + h_1^1\sin\phi\right)\sin\theta\right] \qquad 1.2.34$$

$$B_\theta = \left(\frac{a}{r}\right)^3\left[g_1^0\sin\theta - \left(g_1^1\sin\theta + h_1^1\sin\phi\right)\cos\theta\right] \qquad 1.2.35$$

$$B_\phi = \left(\frac{a}{r}\right)^3\left[g_1^1\sin\phi - h_1^1\cos\phi\right] \qquad 1.2.36$$

where B_r is the dipole field aligned with the geocentric distance, r, B_θ is the dipole field aligned with the co-elevation, and B_ϕ is aligned with the longitude, ϕ, East of Greenwich

Stellenbosch University http://scholar.sun.ac.za

[1]. The orientation, geocentric distance and the longitude East of the Greenwich is shown in Figure 1.2.6 below.

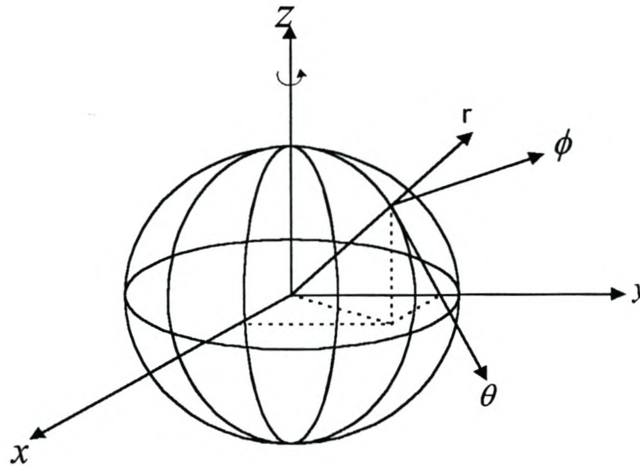


Figure 1.2.6: The radial distance, co-elevation and the longitude East of Greenwich.

The geocentric distance, *r*, is actually the radial distance from the centre of earth to the satellite. The co-elevation is the latitude and the longitude East of Greenwich. This implies that the spacecraft will experience a torque due to the magnetic dipoles and will have to be stabilised. The aim of this thesis is to find out how much of the dipole fields are experienced by the vehicle in each position. It has been assumed for the reasons of simplicity that the motion of the satellite is measured with relative to the earth surface.

Stellenbosch University http://scholar.sun.ac.za

2 THE SYSTEM STRUCTURE

The system will consist of two structures, as shown in Figure 2 below. The **orbit generator** will simulate the positioning of a spacecraft. The second **subsystem** (Core-Operating System) will be used to generate the attitude of a spacecraft. The positioning parameters that will be generated are the height, longitude and the latitude of a satellite relative to the earth. The attitude parameters are the yaw, the pitch and the roll.

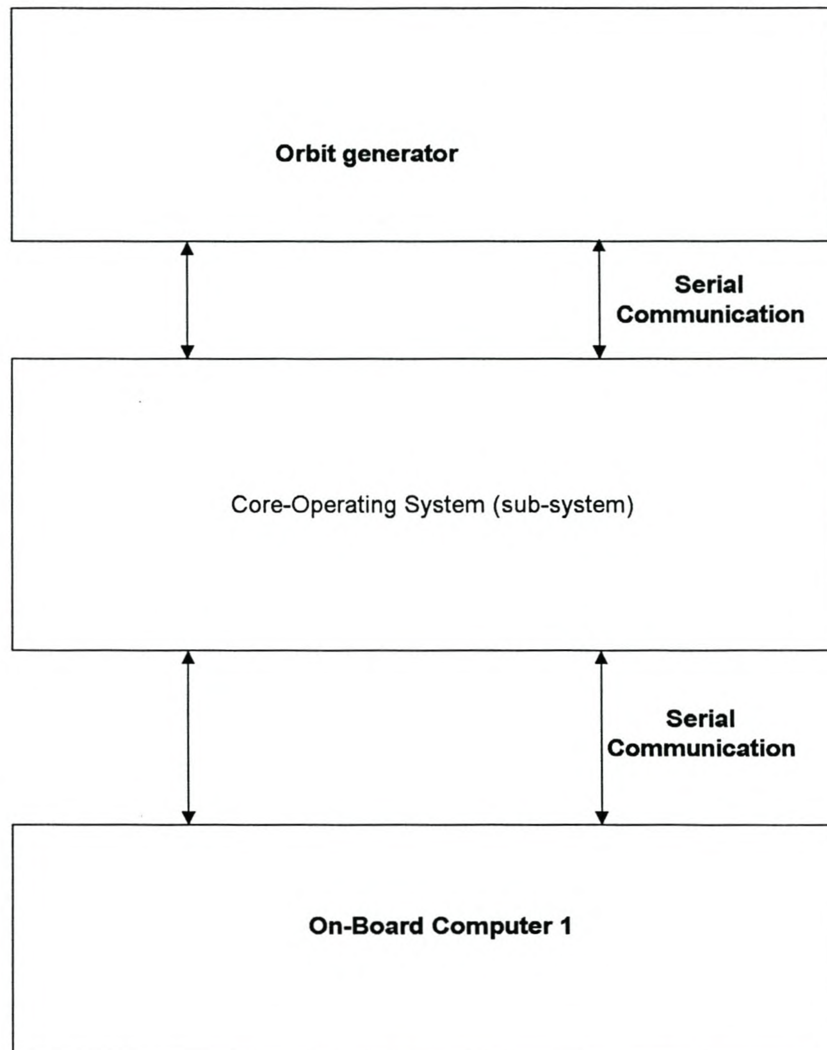


Figure 2: Hardware-in-the-loop operating system.

The orbit generator and the subsystem will communicate with each other via the serial communication. Therefore, a **communication protocol** will also have to be developed.

Stellenbosch University http://scholar.sun.ac.za

Furthermore, this subsystem will also communicate with the Onboard Computer 1 (OBC1) through serial communication. It therefore is a **distributed operating system**, consisting of the orbit propagator and the subsystem. Simulations will be added for the **Star sensor**, **Magnetic field sensor** and **Sun sensor.**

2.1 The Orbit Generator

The **orbit generator,** which sometimes is referred to as the **propagator,** is a portion of the operating system that is designed to simulate the positioning of a satellite in orbit. The results from the simulation are sent to the Core-Operating System. The Core-Operating System treats the propagator as the sensors and will communicate with the OBC1 (On-board Computer 1).

The programming language used for this project is JAVA. The first part of the process takes the input from the user and calculates the position of the satellite. The position of the satellite will be defined, in this system, by its height above the earth, the true anomaly, and the longitude and latitude of the satellite at time intervals after the epoch. The orbit, based on the eccentricity, semi-major and semi-minor axis, will be drawn on the output screen. In addition, the radial distance from the centre of the earth to the satellite orbit position is also drawn. As the satellite approaches the furthest point from the earth, i.e. the apogee, it is expected to slow down.

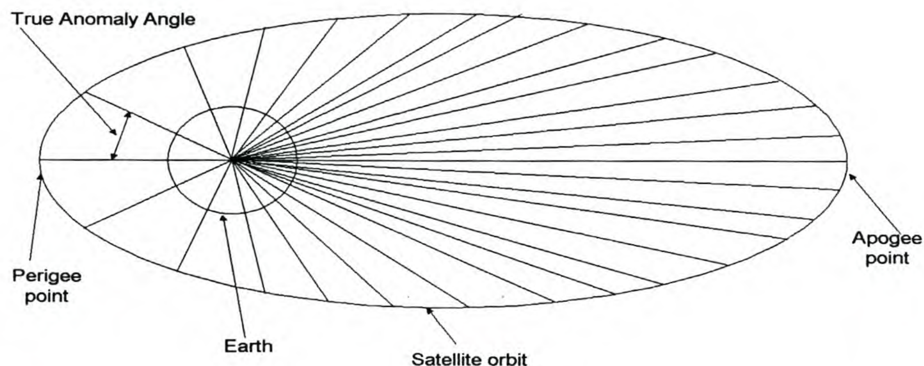


Figure 2.1: True anomaly angle decreases as the satellite approaches the apogee point.

Stellenbosch University http://scholar.sun.ac.za

Therefore, as shown in Figure 2.1, the radial distances must be clustered very close to each other as the satellite approaches the apogee. Figure 2.1 also shows that the true anomaly angle is big at the perigee point and that the true anomaly angle is small at the apogee.

2.2 The Core-Operating System

The Core-Operating System has to simulate the orientation of a satellite and send the simulated results to the orbit propagator. It will also obtain the simulated positioning parameters from the orbit propagator, as well as communicate with the On-Board Computer1. In fact, the Core-Operating System emulates the OBC1 computer, as it also generates magnetic field data and orientation data. The magnetic fields that are generated by the OBC1 are dependent on the position of the satellite. They are sent to the orbit propagator, which will receive and display them. Section 3.4.1 explains the magnetic field simulation in greater detail. The orientation data is simulated randomly and consists of the yaw, pitch and roll angle values. These values, together, provide the orientation of the satellite. The orientation and the magnetic fields data will be displayed and sent to the orbit propagator. The Core-Operating System therefore coordinates virtually the entire process of controlling the satellite.

2.3 The Communications Protocol

The orbit propagator and the OBC1 will have to be able to communicate with the Core-Operating System, as shown in Figure 2 above. In other words, data and some commands will flow through the serial port to and from the Core-Operating System. This means that communication between these systems will be serial-port communication. Hence, a protocol will have to be developed that will help standardise this type of communication between these systems.

In any serial communication, the two computers have to be synchronised and perform all the handshaking procedures before they can start communicating with each other. This helps to

Stellenbosch University http://scholar.sun.ac.za

ensure that no data will be lost in the process of communication. One computer must know whether the other one is going to be the receiver or the sender. It also needs to know how long the data packet will be. Therefore, if the data is sent to one computer, this computer should be able to detect the start of the packet and must be able to validate the data that it has received whether the data is correct. It must then inform the server computer to send the next packet. A server computer is a computer that processes requests and sends responses or data.

The communications protocol that will be implemented here is very simple. One computer will send, character-by-character, data to the other computer. The data packet will consist of the start of the packet and the end of the packet. When the packet is fully received, the server computer will wait for response data. A new data packet will then flow from the recipient to the server computer. If data is flowing from the orbit propagator, the data packet will consist of the radial distance, height, longitude and latitude. These are the positioning data parameters and they will be separated from each other with a special character. Also, if the data is flowing from the core-operating system, the data packet will consist of the same positioning data parameters, the orientation data and the magnetic fields data.

The orientation data consists of the yaw, the pitch and the roll. The magnetic fields data consists of the dipole field given in local tangent coordinates. This dipole field is actually the magnetic field force with three components pointed in the directions of the geocentric distance, the coelevation angle and the longitude East of Greenwich. A special character is also used to separate these data from each other. Therefore, a packet from the Core-Operating System will consist of nine data parameters. The magnetic field is explained further in section 2.4.2.

2.4 Sensors

Sensors are specialised devices that are used by the satellite to determine its attitude when in orbit [4]. There are many kinds of sensors, such as sun sensors, star sensors, horizon sensors, magnetic fields sensors also called magnetometers and gyroscopic sensors, as well

Stellenbosch University http://scholar.sun.ac.za

as other sensors that are not covered in this thesis [4]. For this system, there are three sensors that will be discussed briefly. These are the star sensors, the magnetic field sensors and the sun sensors. However, only one sensor, the magnetic field sensor, will be emulated with the software.

2.4.1 Star sensors

Star sensors are used for high-accuracy missions. They measure the brightness of the stars and compare the pattern to a catalogue to determine which type of star pattern has been captured. These star sensors have the ability to compare their orientation output with the inertial reference [4]. This makes the attitude determination software simpler, because the star sensor has done most of the work [4]. However, for the star sensors to work effectively accurately, the vehicle must be stabilised [4]. There are **two types of star sensors**: the **scanning** type and the **tracking** type.

Scanners are used on spinning spacecraft [4]. These scanners **allow the stars to pass through multiple slits of their field of view**. Thereafter, the attitude of the satellite will be determined after several star crossings.

Trackers are used in three-axis attitude stabilised spacecraft to track one or more stars. This helps in determining the two and three-axis attitude information [4].

2.4.2 Magnetic field sensor (Magnetometers)

Magnetometers are the specialised sensors that are used in spacecraft attitude determination [1]. They are vector sensors and they measure both the direction and the magnitude of the earth's magnetic fields [1]. When compared to the earth's known magnetic field, their output helps to determine the attitude of the spacecraft [4]. They are reliable, lightweight and have low power requirements [1]. Another advantage of using magnetometers is that they can operate over a wide range of temperatures and have no moving parts [1].

Stellenbosch University http://scholar.sun.ac.za

However, the accuracy of the magnetometers is not as good as that of the star or horizon sensor [4]. The problem is that the magnetic fields are not completely known, because the earth's magnetic fields can shift with time [4]. Hence, the models used to predict the direction and magnitude of the magnetic field at the spacecraft's position are subject to error [1]. In addition, the strength of the earth's magnetic field is inversely proportional to the distance from the earth [1]. Hence, the use of magnetometers is generally limited to spacecraft flying below an altitude of 1000 km [1]. Therefore, to improve the accuracy of the magnetometers, the data from the magnetometers is often combined with data from the sun sensors or horizon sensors [4].

2.4.3 Sun sensor

Sun sensors are visible light detectors, which measure one or two angles between their mounting base and the incident light [4]. They are the most widely used sensors because they are very accurate and reliable and have a low power consumption [1]. This low power consumption is because of the brightness of the sun [1]. Furthermore, sun sensors are used to protect sensitive equipment, such as star trackers, to provide a reference for onboard attitude control, and also to position solar power arrays [1].

Sun sensors also play a vital role in initial acquisition and failure recovery systems and could also be part of an independent solar array orientation system [4]. The sun sensor can provide very accurate measurements of up to less than 0.01 degrees accuracy [4]. However, it is not always possible to take advantage of this accuracy feature [4]. To obtain an unobstructed field of view, the sun sensors are usually mounted near the end of the vehicle [4]. The data collected is either processed on-board or sent to the ground [4].

Stellenbosch University http://scholar.sun.ac.za

3 IMPLEMENTATION OF THE SYSTEM

This chapter presents the approach that was used to implement an orbit propagator and core-operating system. The communications protocol, together with the magnetic field emulator, is also discussed and implemented. The orbit generator will be discussed first, before a discussion of the entire system.

3.1 The orbit generator/propagator

The orbit propagator reads the initial conditions from the keyboard and determines the position of the satellite at time intervals. This means that the propagator calculates the height above the earth, the true anomaly, and the longitude and latitude of the satellite at time intervals. The orbit, based on the eccentricity, semi-major and semi-minor axis, will be drawn on the output screen. In addition, the radial distance from the centre of the earth to the satellite orbit position is also drawn. As the satellite approaches the furthest point from the earth, i.e. the apogee, it is expected to slow down. Therefore, the radial distances must be clustered very close to one another as the satellite approaches the apogee. Figure 3 below illustrates this concept and also shows that, at the perigee point the true anomaly angle is big, while the true anomaly angle is small at the apogee.

Initial program testing showed that the radial distances are clustered in the second quadrant and in the third quadrant, as shown in Figure 3. If the results are analysed and interpreted, they would show that the satellite reverses after a certain instant of time, both in the second quadrant and in the third quadrant. This is not what was expected. The problem therefore had to be investigated thoroughly. The first attempt was by plotting a graph of true anomaly versus the mean anomaly $\boldsymbol{M}$. The range of the true anomaly angle was from zero to 2π.

Stellenbosch University http://scholar.sun.ac.za

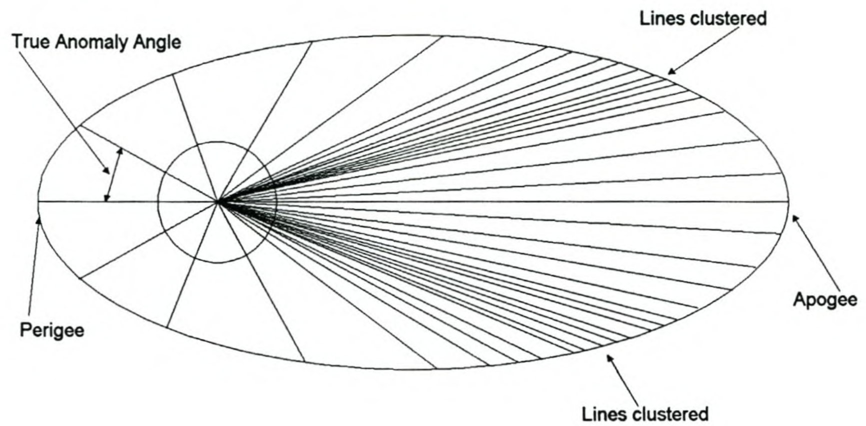


Figure 3: Radial lines closely packed to each other in the second and third quadrant.

The sketch that was drawn using Matlab illustrated something that seemed to be very interesting. In Figure 3.1 it can be seen that, initially, as the mean anomaly increases, the true anomaly slope is very steep. This indicates that the true anomaly increases very quickly, with only a small increment in mean anomaly. After a certain magnitude of M, the slope of the true anomaly reaches a turning point and becomes negative.

Stellenbosch University http://scholar.sun.ac.za

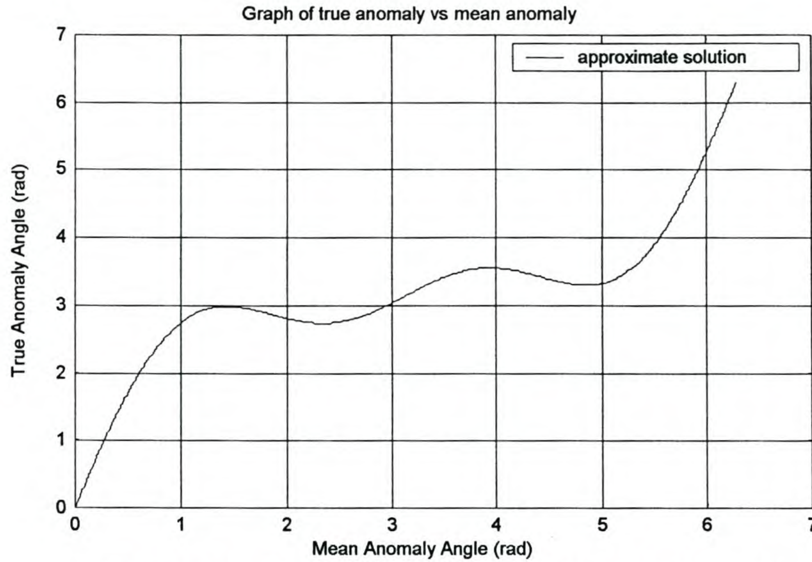


Figure 3.1: True anomaly versus mean anomaly angle.

This indicates that the true anomaly is in the region where the increments are very small. The slope turns again and becomes positive. If one looks at the sketch in Figure 3.1, it is possible to observe a sinusoidal oscillation. After a certain range of true anomaly, the slope becomes steep and positive again. This is an indication that the True anomaly increases faster and that it is passing the apogee region.

These results prompted an investigation into a better method of finding the best solution. The approximate solution was being used for the true anomaly angle:

$$\upsilon \approx M + 2e\sin 2M + \frac{5}{4}e^2 \sin 2M + \vartheta\left(e^3\right) \quad (1)$$

Equation (1) is the approximated true anomaly angle. What follows is another way of finding the true anomaly by using Kepler's equation, which relates, for an ecliptic orbit, the mean anomaly, M, at some time t, to the eccentric anomaly, E:

$$M = E - esinE \quad (2)$$

where e is the eccentricity. Now, the mean anomaly at epoch can be found by $M_0 \approx M\ (t = t_0)$ where $t_0 = 0$. Consequently, the mean anomally can be found using the mean anomaly at epoch by $M \approx M_0 + n\,(t - t_0)$ (3)

Stellenbosch University http://scholar.sun.ac.za

where $n \equiv 2\pi/period$ is the mean motion. Given these equations, it is possible to solve Kepler's equation numerically to find E, before or after the epoch time. Iterative solutions will be used to do so. Thereafter, ν can be obtained from E by:

$$Cos\ (v) = \frac{\cos(E) - e}{1 - e\cos E} \tag{4}$$

First, equation (2) needs to be solved for each instant of time, as follows:

$$f(E) = \quad E - esinE - M \tag{5}$$

From equation (5), a solution for the value of E is possible if, and only if, the value of $f(E) = 0$. But the solution could be found by using three algorithms. These are

- **True solution**
- **Halving algorithm**
- **Newtonian method**

3.1.1 True solution

Matlab has a function called linspace, which can used to generate a row vector of eccentric anomaly, E_n, with elements between $E_0=0$ and $E_n = 2\pi$, and where E_n is its *n-th* element.

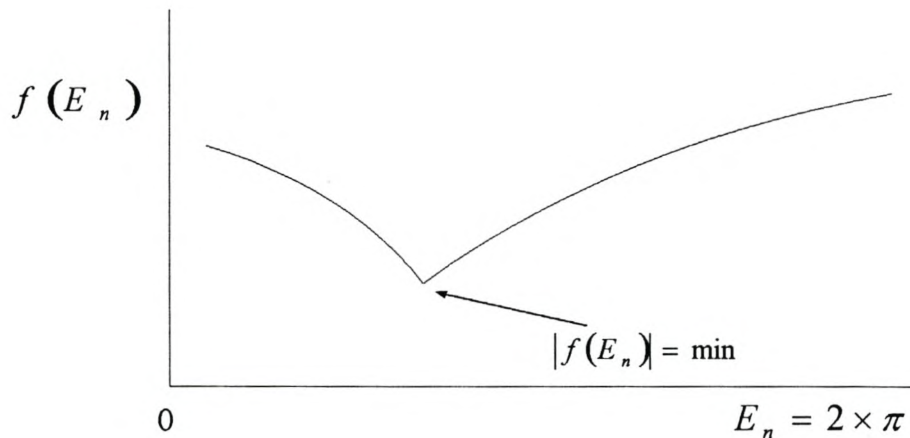


Figure 3.2: The True Solution method.

Stellenbosch University http://scholar.sun.ac.za

Now, it is possible to start searching for a value for $|f(E_n)|=min$. In Figure 3.2, the graph of $f(E_n)$ ***versus*** E_n is drawn. After finding $|f(E_n)|=min$ and also because the values of $f(E_n)$ are stored in the array, it is easy to find the value of E. Figure 3.2 shows what the graph looks like and it is possible to clearly see $|f(E_n)|=min$ on the graph.

Once the value of E has been found, equation (4) is used to find the true anomaly angle. Thereafter, a graph of true anomaly versus mean anomaly was drawn using MATLAB. This is shown in Figure 3.3 below.

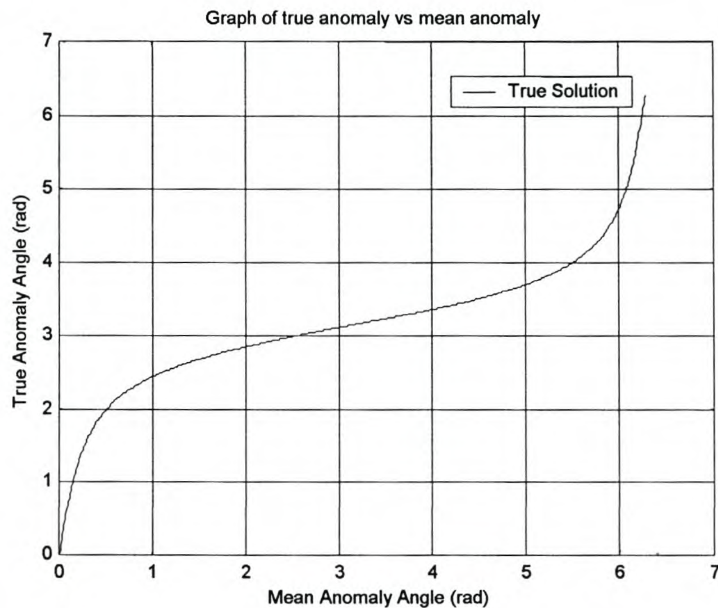


Figure 3.3: Graph of true anomaly versus mean anomaly using True Solution

From the graph in Figure 3.3, it can be seen that the speed of the satellite is high in the region between values 0 and 1 of mean anomaly. In the region between the values 1 and 5 of mean anomaly, the speed is slow. From the value of 5, the speed increases again. This is what was expected from the simulation. However, all the algorithms need to be tested, so that the best one could be chosen. Therefore, the true solution will be used as a reference for other algorithms for finding the eccentric anomaly.

Stellenbosch University http://scholar.sun.ac.za

3.1.2 Halving algorithm

In order to investigate what the outcome will be when using the halving algorithm [2], it is necessary to draw a sketch, as shown in Figure 3.4, to explain the process thoroughly.

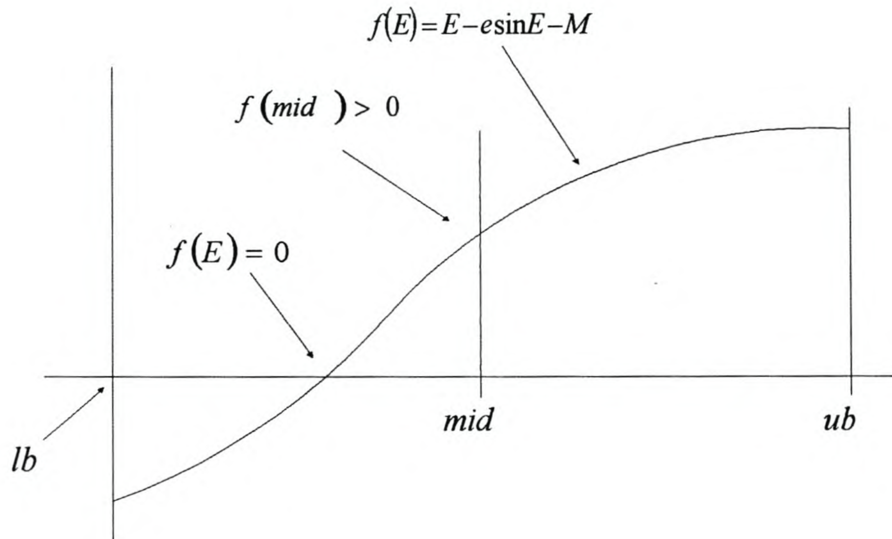


Figure 3.4: The halving algorithm that iterates until the value of $f(E) = 0$, where $E = median$

Now it is necessary to find a value for E where

$f(E) \leq limit,$

with $limit < 1 \times 10^{-8}$

First find $\boldsymbol{f(E)}$, where $\boldsymbol{E = \{lb, mid, ub\}}$. If $\boldsymbol{f(mid) \geq limit}$, the upper bound (***ub***) of the graph must be positioned at the point where $\boldsymbol{E = mid}$. The middle point (***mid***) will then be shifted to a certain position towards the left-hand side. If the $\boldsymbol{f(mid) \leq -limit}$, then the lower bound (***lb***) must be changed to the position where $\boldsymbol{E = mid}$. This process must continue until $\boldsymbol{|f(mid)| < limit}$. The solution for **E**, therefore will be where $\boldsymbol{E = mid}$ and $\boldsymbol{|f(mid)| < limit}$. Once ***E*** is found, the true anomaly angle (ν) needs to be solved using equation (4):

$$Cos\ (\nu) = \frac{\cos(E) - e}{1 - e\cos(E)} \qquad (4)$$

Stellenbosch University http://scholar.sun.ac.za

$$\Rightarrow \nu = Cos^{-1}\left[\frac{\cos(E)-e}{1-e\cos(E)}\right] \qquad (6)$$

Figure 7 below illustrates the results from using this algorithm. The graph looks smooth and has a region where the slope is flat. This means that the speed of the satellite is slow in this region. These are the results that were expected from the satellite motion.

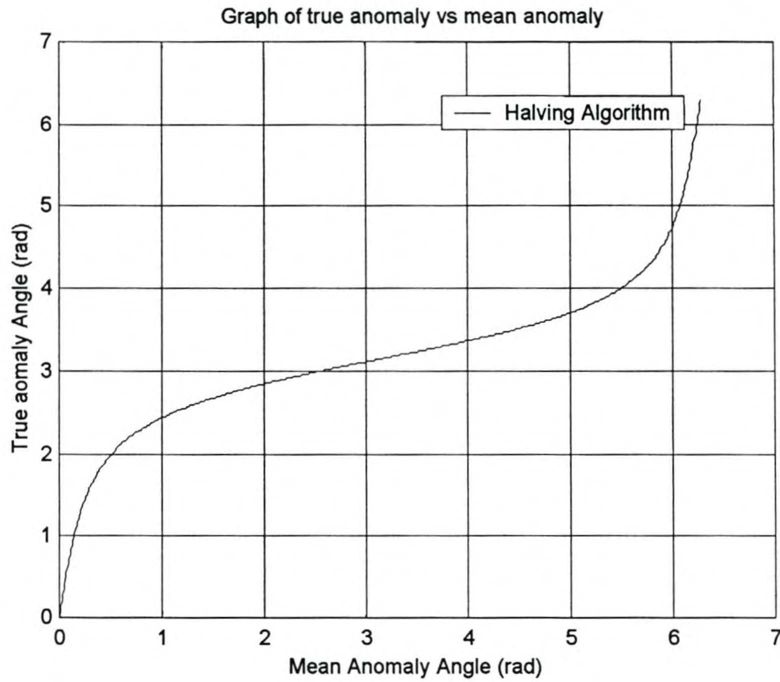


Figure 3.5: True anomaly versus mean anomaly angle for the halving algorithm.

3.1.3 Newtonian method [3]

The Newtonian method is one of the best methods for finding the roots of an equation. The equation of which the roots must be solved is equation (5):

$$f(E) = E - esinE - M \qquad (5)$$

Firstly, it is necessary to find the derivative of $\boldsymbol{f(E)}$, i.e. $\boldsymbol{f'(E)}$, start from $\boldsymbol{E_0 = M}$ to $\boldsymbol{E_n}$ until $|f(E_n)| = 0$. However, iteration will be used until $|f(E_n)| < limit = 1x10^{-8}$. The equation is :

Stellenbosch University http://scholar.sun.ac.za

$$E_n = E_{n-1} - \frac{f(E_{n-1})}{f'(E_{n-1})} \quad (7)$$

The first approximation is used to get a second, the second to get a third, and so on. To go from ***(n-1)th*** approximation $\boldsymbol{E_{n-1}}$ to the next approximation, $\boldsymbol{E_n}$, use equation (7):

$$E_n = E_{n-1} - \frac{f(E_{n-1})}{f'(E_{n-1})} \quad (7)$$

where $f'(E_{n-1})$ is the derivative of f at $E_n = 2 \times \pi$.

The value of ***E*** that is found by using this method ([3]) is used in equation 4 to calculate the true anomaly angle.

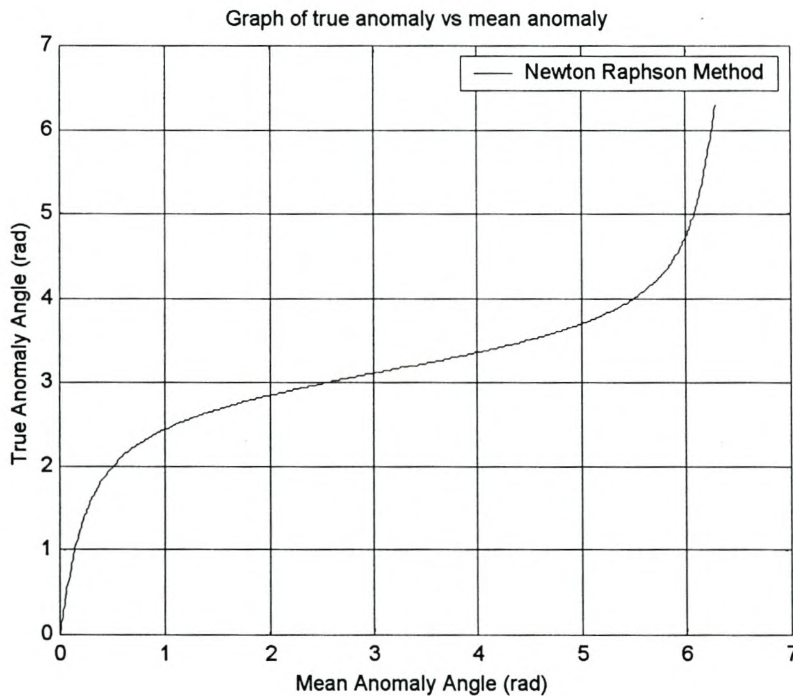


Figure 3.6: True anomaly versus mean anomaly angle for the Newton Raphson method.

The true anomaly angle is then plotted against the mean anomaly, and the graph thereof is shown in Figure 3.6 above.

Stellenbosch University http://scholar.sun.ac.za

3.1.4 Analysis based on the algorithms and with the True Solution as a reference

The analysis of the results will help to determine whether the intended output was obtained, and whether the system is working according to the design plan. It will also help in selecting the best and the most efficient algorithm for this application.

3.1.5 Comparing the approximate solution with the True Solution

The most important thing that had to be done was to check and compare the first method of finding the true anomaly angle, the **approximate method**, with the True Solution. This was done by drawing the graph of the latter method in the same system of axes as the other graph which has been labeled the **True Solution Algorithm**. In **Figure 3.7** below, the approximation graph looks oscillatory, whereas the other one looks smooth. The one that appears smooth is the solution that was looked for, because it is the most accurate one of all.

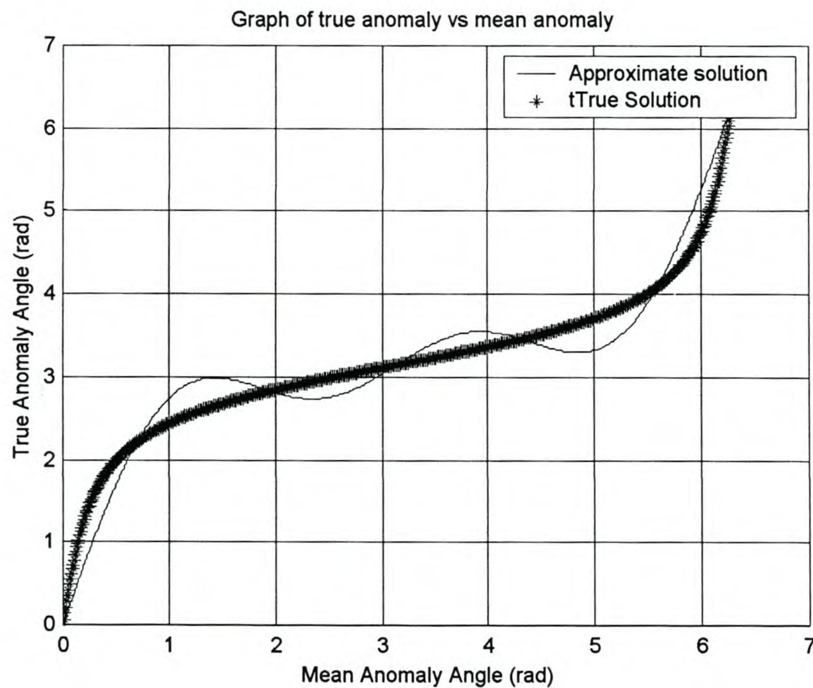


Figure 3.7: Comparing the approximate solution with the True Solution.

Stellenbosch University http://scholar.sun.ac.za

The approximate solution comes from equation (1). The original equation for the true anomaly angle comes from Gauss' Equation relating true anomaly ν to the eccentric anomaly *E*, as follows:

$$\tan\left(\frac{\nu}{2}\right) = \left(\frac{1+e}{1-e}\right)^{1/2} \tan\left(\frac{E}{2}\right) \qquad \text{...(7)}$$

Using power expansion series, **equation (7)** is combined with equation (2), and expressed directly as a function of *M*. This gives rise to **equation (1).** This equation works fine for small eccentricities up to $e = 0.3$. From visual inspection of the graph of true anomaly versus *e*, it was clear that the graph starts to oscillate more when the eccentricity goes higher than $e = 0.3$, i.e the graph starts to oscillate for higher eccentricities. However, if the equation is expanded into more higher order terms, it will also become accurate.

3.1.6 Finding the error measure between the approximation and the True Solution

It is possible to determine the degree of error by which the oscillating graph deviates from the True Solution by observing the increase in eccentricity. This is done by plotting a graph of the error measure against eccentricity, in figure 3.8 below.

3.1.6.1 Graph of error measure versus eccentricity

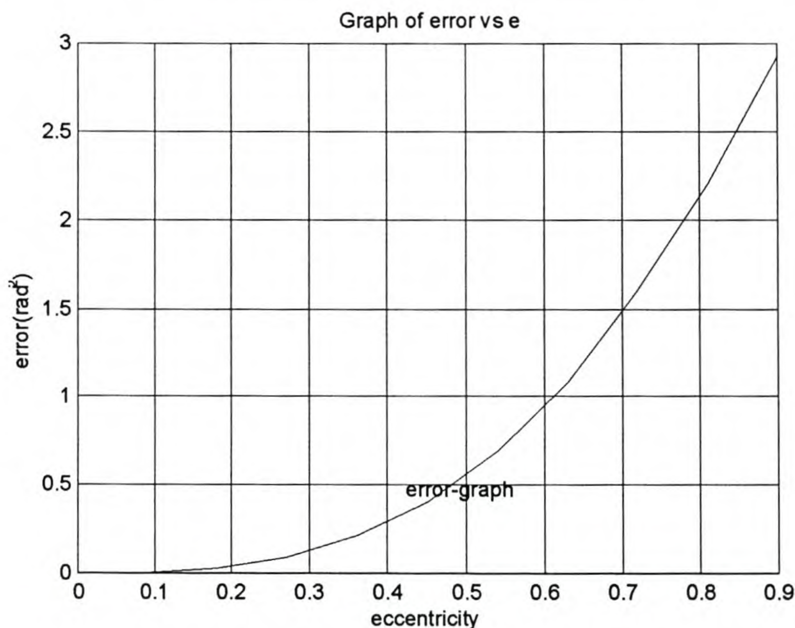


Figure 3.8: Finding the error measure between the approximation and the True Solution

Stellenbosch University http://scholar.sun.ac.za

The error measure is obtained by calculating the areas of the graph that deviate from the true solution. Those areas are then summed and taken as the error measure. The graph below shows the **error** measure as a function of e. From the graph in Figure 3.8 above it can be seen that the slope becomes very steep as the eccentricity approaches **1**. This shows that the error measure increases as eccentricity approaches **1**.

Stellenbosch University http://scholar.sun.ac.za

3.1.7 Halving algorithm and the True Solution

The halving algorithm and True Solution are very **accurate,** but the True Solution is even more accurate than the other algorithms. The two graphs that are obtained, when these two algorithms are used, appear in the same system of axes as a single graph. The reason for this is that both graphs are drawn almost on top of one another, as can be seen in Figure 3.9 below.

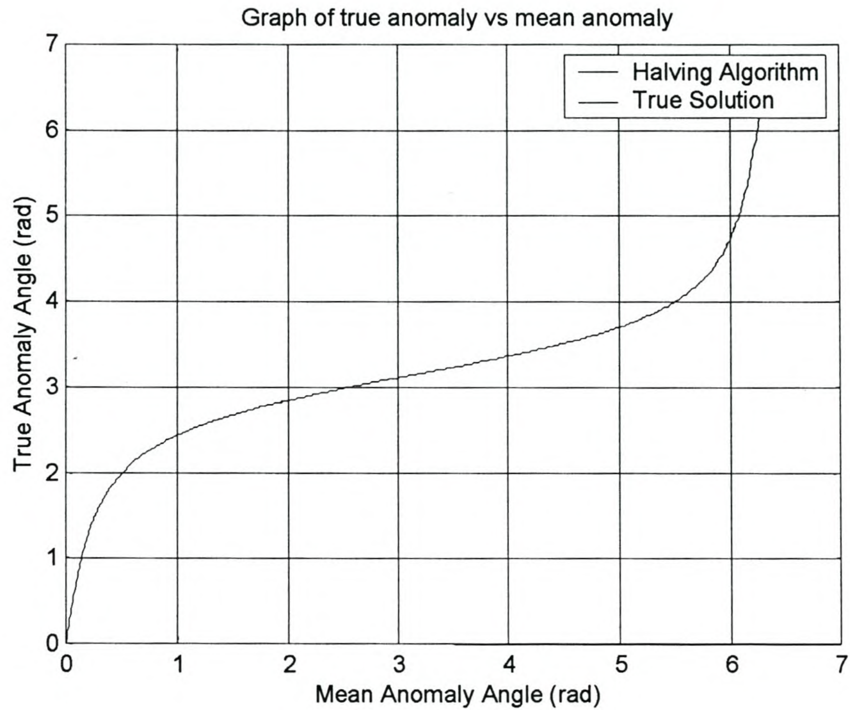


Figure 3.9: Comparing the halving algorithm with the True Solution.

Stellenbosch University http://scholar.sun.ac.za

3.1.8 Finding the error measure between the halving algorithm and the true solution

If the error measure is calculated as above and a graph is plotted of the error measure against eccentricity, the error is extremely small up to the value of $\times 10^{-4}$. As shown in Figure 3.10, the error measure decreases as the eccentricity increases.

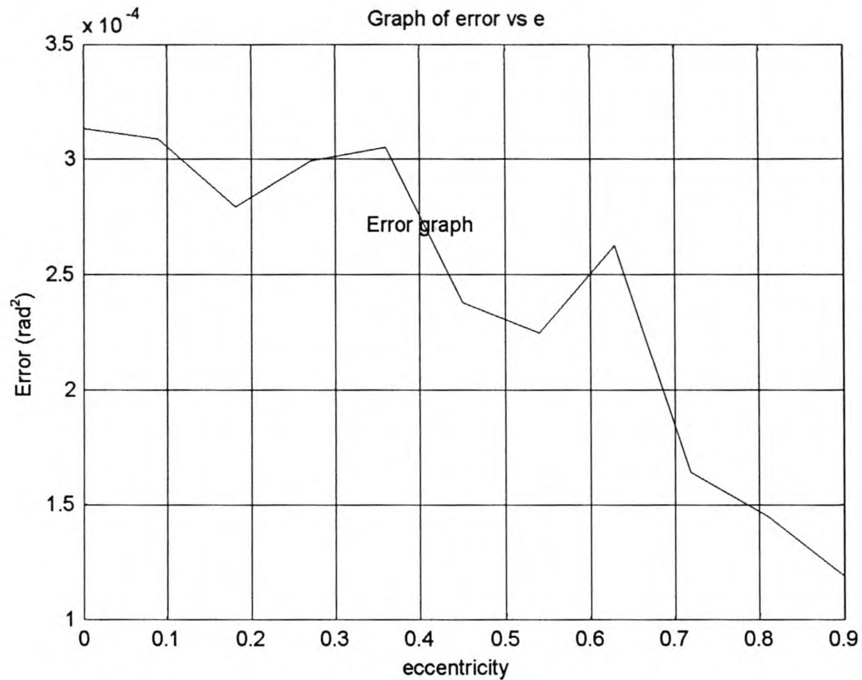


Figure 3.10: Finding the error measure between the halving algorithm and the True Solution graphs.

3.1.9 Comparing the Newton Raphson method with the True Solution

The Newton Raphson method gives the same results as the halving algorithm and the true solution. One difference is that the Newton Raphson method is faster than the other methods. Figure 3.11 below shows a graph of the Newton Raphson method together with the graph of the true solution. They are both drawn on the same system of axes and are on top of one another which is why only one graph can be seen.

Stellenbosch University http://scholar.sun.ac.za

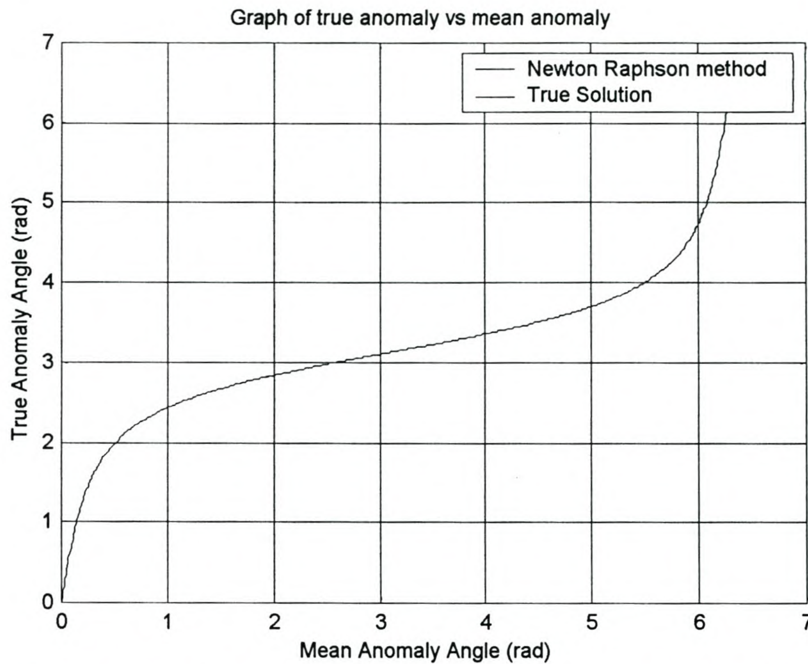


Figure 3.11: Comparing the Newton Raphson method with the True Solution.

3.1.10 Finding the error measure between the Newton Raphson method and the True Solution

The error measure for this method is also extremely small, with the scale being up to $x10^{-4}$ units. This shows that this algorithm is almost as accurate as the True Solution. Figure 3.12 below shows a graph of this error-measure plotted against eccentricity, $\boldsymbol{e}$.

Stellenbosch University http://scholar.sun.ac.za

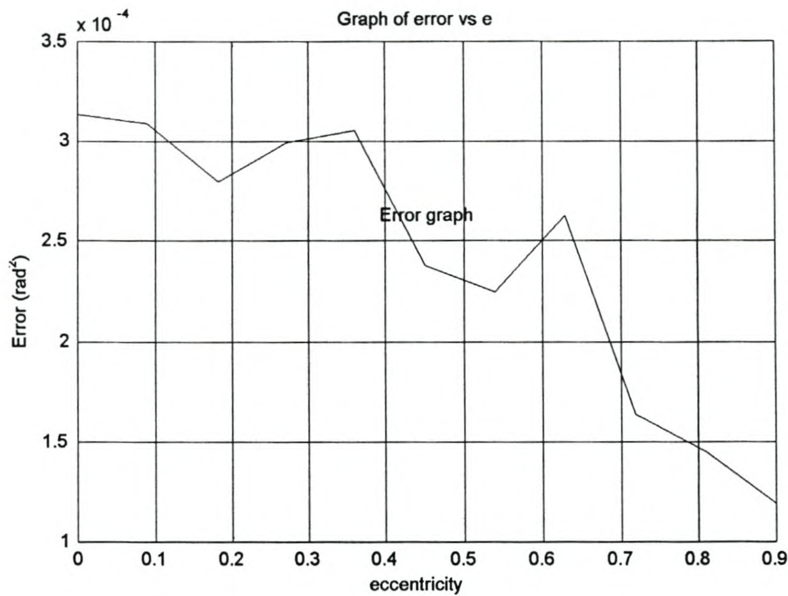


Figure 3.12: Finding the error measure between the Newton Raphson method and the true solution graphs.

3.1.11 Conclusion based on the error measure graphs

It has become clear that the two algorithms, the Halving algorithm and the Newton Raphson method, are the best algorithms to choose, as they are more accurate. Furthermore, their error measures are approximately $\times 10^{-4}$ scale units, and become small as the eccentricity tends to 0. However, the Newton Raphson method has the advantage because it is faster. Therefore, the Newton Raphson algorithm will be used to calculate the eccentric anomaly angle. The further work was done with JAVA because the MATLAB was used only to solve the eccentric anomaly angle.

Stellenbosch University http://scholar.sun.ac.za

3.1.12 Software Design

For this system, the overall design used is given in the flowchart shown in Figure 3.13 below.

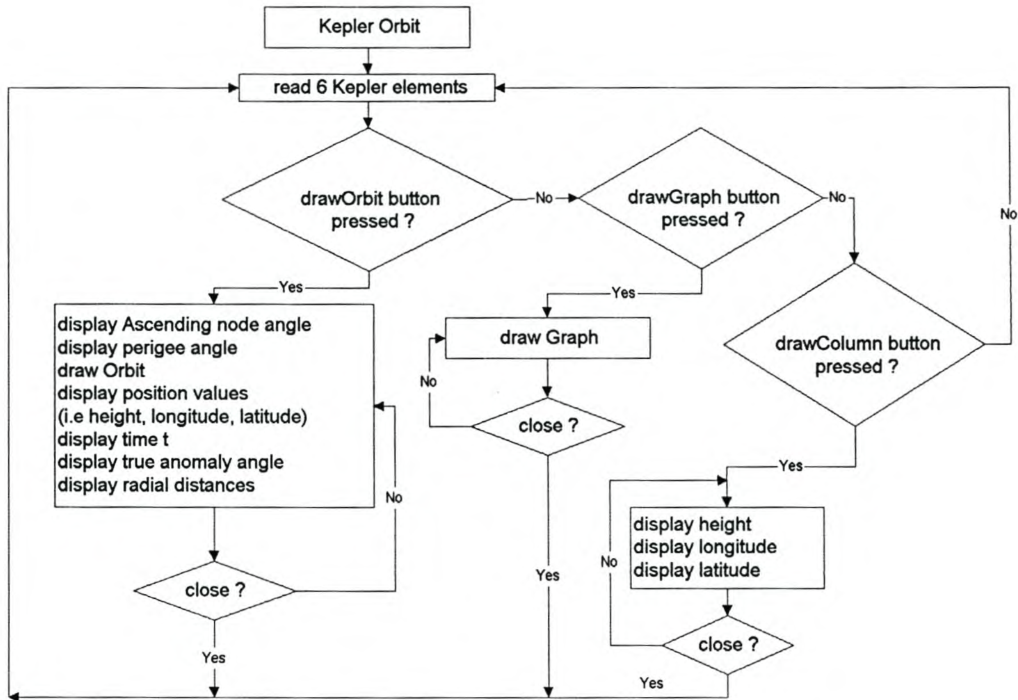


Figure 3.13: Flowchart for the orbit generator

The flowchart in **Figure 3.13** shows an algorithm for the orbit generator. The user enters the six Kepler elements and the program reads these values and waits for the drawGraph, the drawOrbit or the displayColumn buttons to be pressed. When this happens, an event-handler will be executed, and a graph or orbit or the positioning values will be displayed in a separate window. If the drawGraph button is pressed, a graph of longitude versus latitude is also drawn in a separate window. Also, if the drawOrbit button is pressed, an orbit with the radial distances drawn from the centre of the earth to the orbit is drawn in a new window. The values of height, longitude and latitude are also drawn in a new window if the drawColumn button is pressed.

Stellenbosch University http://scholar.sun.ac.za

Another window is drawn when the drawGraph button is pressed. In this window, the orientation parameters that are read from the serial port are displayed in the text fields. These are the yaw, pitch and roll values. This window can also be closed or minimized. This does not affect the running of the whole program, because it is an independent thread.

3.2 The Core-Operating System

The Core-Operating System is also called the simulator. It simulates the orientation angles for the spacecraft, namely the yaw, the pitch and the roll. The Core-Operating System will display all these orientation values in a window that has text fields for the yaw, pitch and roll values. It will then send these orientation values to the orbit generator via the serial port. The orbit generator will then capture the values from the serial port and send the positioning parameters, together with the orientation parameters, back to the Core-Operating System. The core-operating system will then display all six of these values. Figure 3.14 below shows a flowchart of the Core-Operating System with the communication protocol.

The Core-Operating System consists of four classes. These are the **OBC1_simulation**, the **Receive**, **Send** and **Simulation** classes. The OBC1_simulation class is used to simulate all the orientation values. It inherits all characteristics from the Thread class. After simulating the values using random numbers, it will then use some methods from the simulator class to display these orientation values in text fields. It will also use the class Send to send these simulated values through the serial port to the orbit generator.

The Simulator class is the one that draws together all the text fields and displays all the results. It actually invokes the OBC1_simulation to provide all the simulation values for the satellite orientation. The Receive class invokes the Send class so that it can listen to the port whenever new data is received. When the data has been received, it is displayed in the text fields that are drawn by the Simulator. All these classes are threads, except the Simulator class. The Simulator class inherits some characteristics of the Frame from the library.

Stellenbosch University http://scholar.sun.ac.za

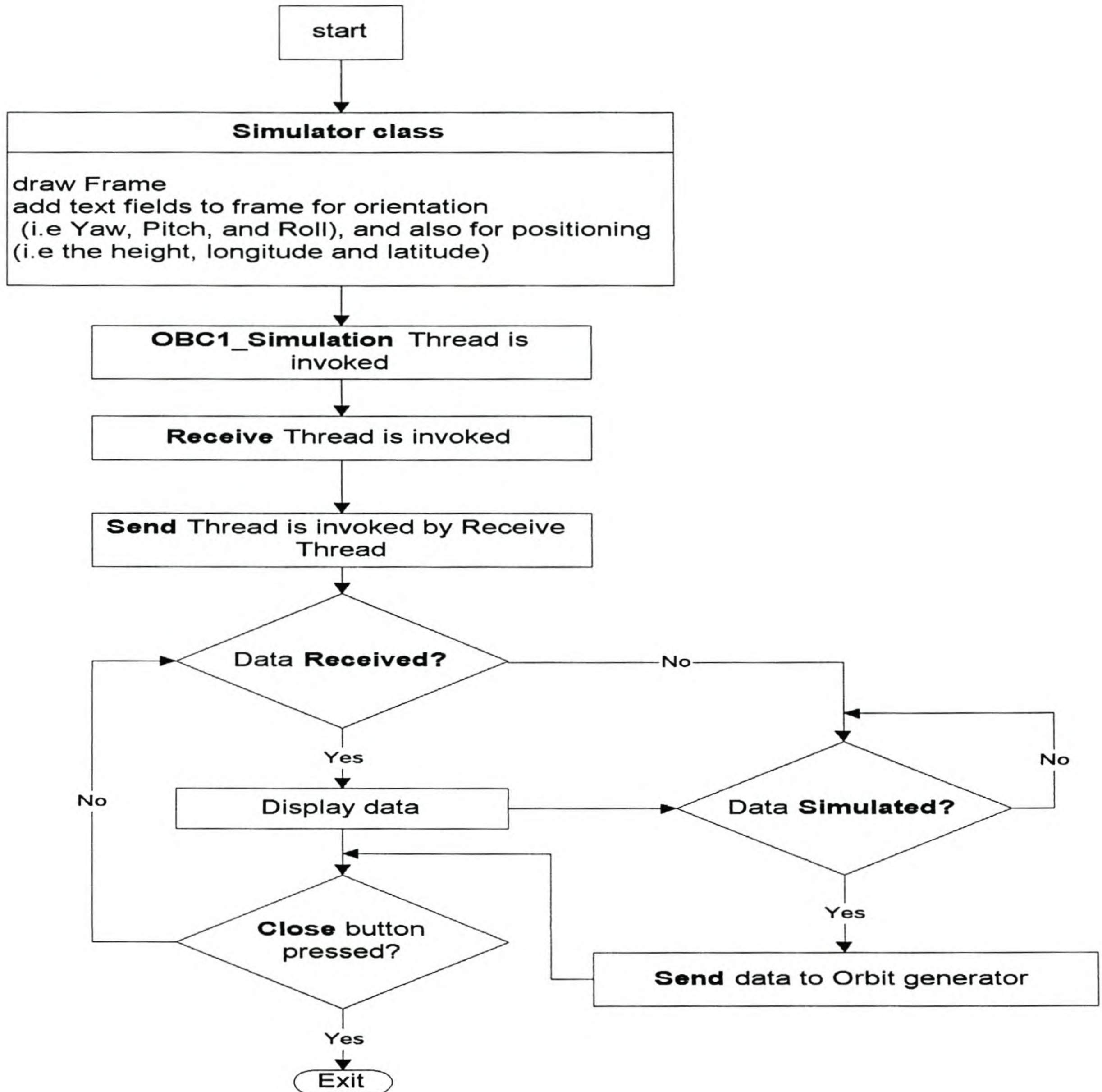


Figure 3.14: Flowchart for the core-operating system.

3.3 The Communications protocol

The communication protocol consists of the Receive and the Send thread. It was designed to establish a serial communication between the orbit propagator and the OBC1 emulator. The Receive monitors the serial port for any data that is received. It consists of a serial port event listener, which enables it to listen to the serial port for any data coming in, and the Send thread which sends the data to the client computer. These threads keep running

Stellenbosch University http://scholar.sun.ac.za

independently of each other and when the Receive thread has new data, it takes it and displays it on the screen. Some of the data is used to calculate the magnetic fields.

3.3.1 The Receive thread

The Receive thread inherits the properties of a thread class and implements the serial port event listener. When this thread starts running, it first searches for the Comm port that is available and takes control of it. If no is port available, an exception will be thrown to indicate that there is no port to set up communication. But, if the port has been found, the thread will identify the name of the port and displays its name on the screen. Once the port is identified, the thread will open it and add event listener to it, so that it can be notified of any data available in the port. It will also initialise the port by calling the method serialPortParams that initialises the data bits, baud rates, parity bits and the stop bits. Once all that is done, the thread instantiates Send thread. Upon instantiating the Send thread, it will also pass the serial port handle to the Send using the Send constructor. It will then activate its start method to begin running the thread.

A flowchart of the Receive thread is shown in Figure 3.15. The Receive thread also consists of a number of flags, which are used to identify the type of data that is received. These are YawF, PitchF, RollF, heightF, longitudeF and latitudeF flags. There are also a number of array variables, which are used to store the data. They are the yaw, pitch, roll, height, longitude and latitude. The status of these flags changes when a certain control variable, called the separator, is received.

Stellenbosch University http://scholar.sun.ac.za

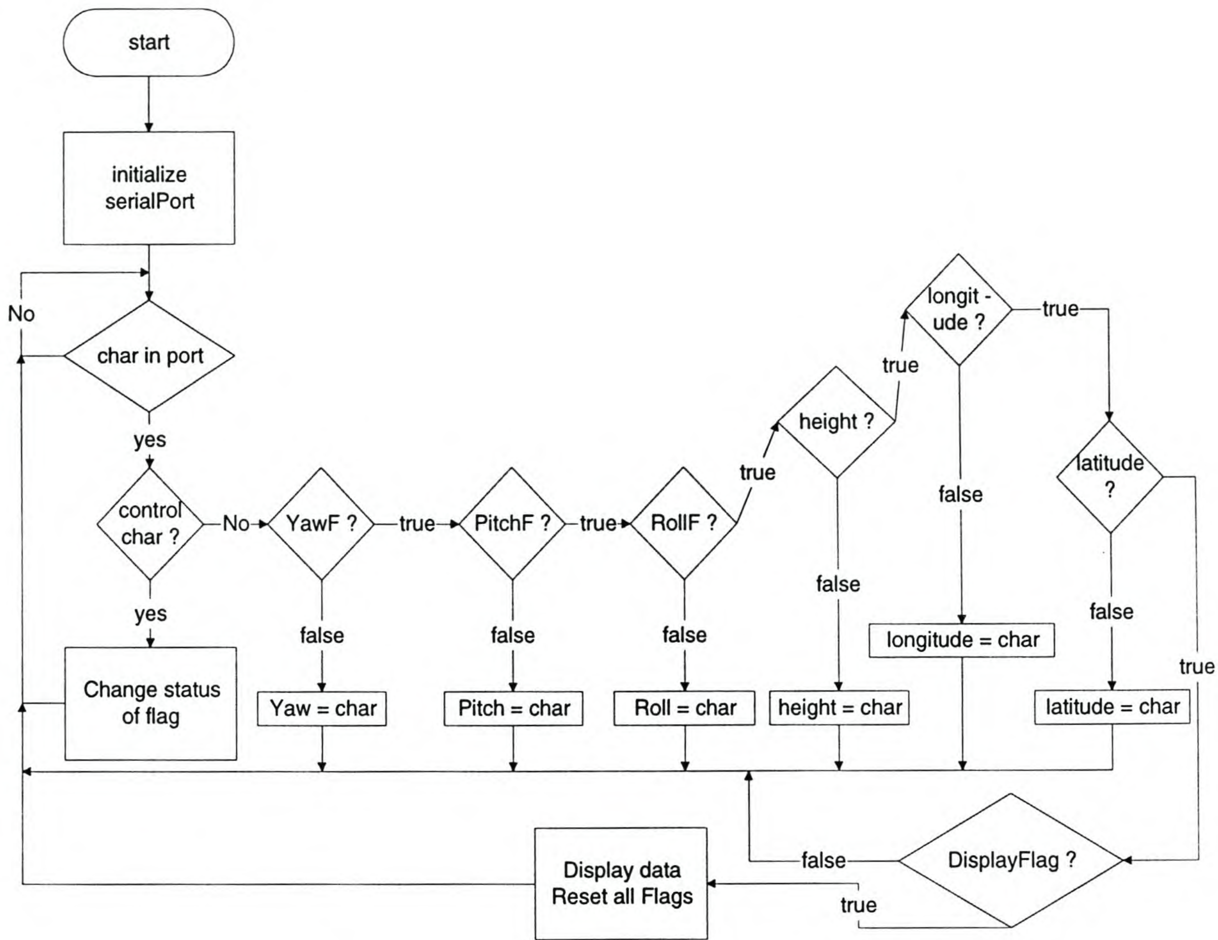


Figure 3.15: A flowchart for the Receive thread

When the program starts, all flags are initialized with false. Data is received in the port, character by character. Whenever there is data in the port, the data is checked to make sure whether it is not any of the control variables. If not, and also if the YawF flag is false, the data will be added to the Yaw array variable. This continues until a separator variable is received and, when that happens, the status of the YawF will be set to true. The incoming data will now be added to the next variable and the same process will be followed for other data variables.

When the control variable indicating the end of a packet is received, the Display flag is set to true. Now all the data will be displayed on the screen and the magnetic fields are also calculated and displayed. Before displaying, the data is formatted into a double. If the data is

Stellenbosch University http://scholar.sun.ac.za

calculated and displayed. Before displaying, the data is formatted into a double. If the data is corrupted, the variable containing that data will be set to zero values inside the exception handler that catches the NumberFormatException thrown, and formatted to double. Thereafter, all the flags will be reset to false and the TxFlag is set to true. This enables the Send thread to start sending the data.

3.3.2 The Send thread

The Send thread inherits the properties of a thread class. When it starts, it obtains the serial port handle from its constructor. The handle will be used to call the methods used to write to the port. A flowchart for the Send thread is shown in Figure 3.16, below. The Send thread first reads the status of each and every response flag, then sends a character that is used as a response according to the status of the flag. These response flags are TxAck, which is used for acknowledging the reception of a character, StartPacket, for acknowledging the reception of start of packet, and EndPacketAck, which is used to acknowledge that the end of packet has been received.

The Send thread keeps checking the status of TxFlag. This flag is used to tell the Send thread that the data is ready to be sent to the client computer. Therefore, if this flag is false, the program goes back to read the status of the Response flags. This process continues until the TxFlag is set to true. When this flag is set to true, the Send thread starts sending the packet, character by character. It first sends the character that indicates the start of a packet, until that character is acknowledged. Thereafter, it sends the data starting with yaw, followed by pitch, then roll, etc. Separator characters are inserted between the data, for example of a yaw and pitch, so that the receiving computer can identify the data that is meant for yaw, pitch, roll, etc. Therefore, while this thread is sending the data, it also sends the separator character, and keeps sending it until it is acknowledged. After all the data is sent, the thread sends a character that indicates the end of the packet and continues sending this character until it is acknowledged.

Stellenbosch University http://scholar.sun.ac.za

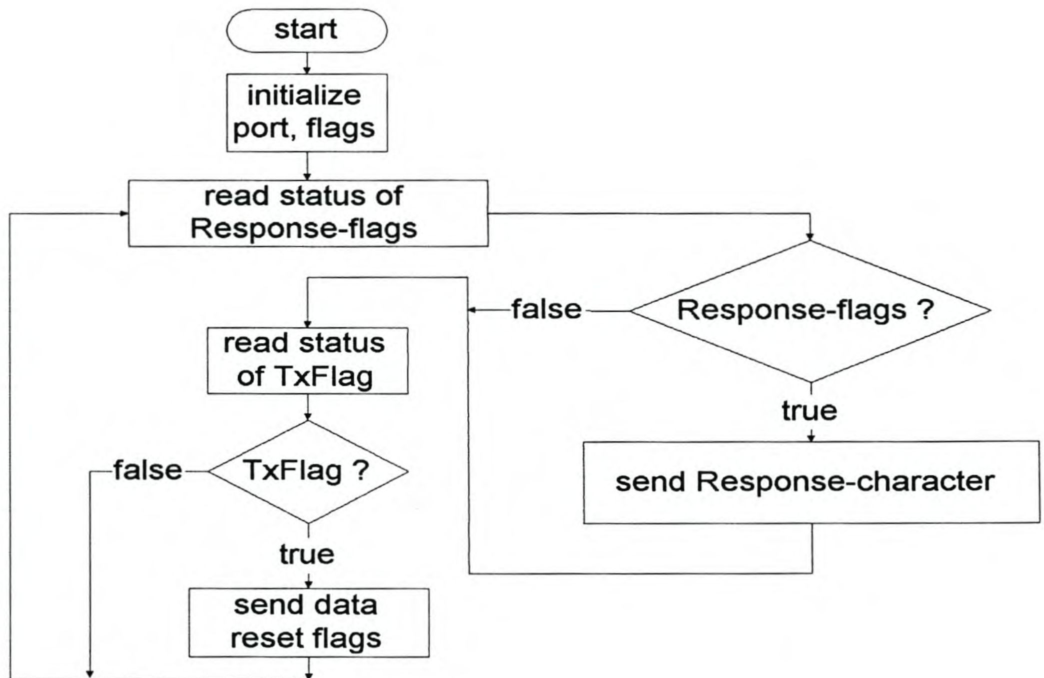


Figure 3.16: A flowchart for the Send thread.

Once it is acknowledged, that indicates the packet has been received on the other side. The thread will then reset all the flags back to their original status. It then goes back to the beginning, reads the status of the response flags and repeats the whole process again.

3.3.3 The major problems encountered while testing the protocol

The protocol presented lots of problems: among them, were the loss of some characters and the data getting mixed up. When the data got mixed up, some of the data variables were left empty, because the data had been appended to another data variable. Another problem was that sometimes, when the data characters were lost, only a comma would be left if the data were a double number. Therefore, when the data is assembled and formatted into a double, the program throws NumberFormatExceptions.

Stellenbosch University http://scholar.sun.ac.za

3.3.4 Solution to the protocol problems

A strategy was planned to get rid of these problems. This was to first identify the cause of the problems. It was then identified that there should be a time delay before sending each character. This helped to reduce the problems.

However, problems were still experienced with NumberFormatExceptions being thrown. This problem was attributed to the fact that some characters including separator characters, were being lost. These separator characters help the receiving program to identify which data belongs to yaw, which data belongs to pitch, and so on. Therefore, the best solution was to continue sending the separator character until it had been received and acknowledged. An additional solution was to make sure that, if the NumberFormatException is thrown, zero value was assigned to the data variable involved. This was the best solution for the protocol, as it kept the program running very well and handling the exceptions whenever they were thrown.

3.4 Magnetic field emulator

The magnetic field sensor measures the magnetic fields of the earth. The measurements give the direction and size of the magnetic fields. They help in determining the attitude of a satellite in a specified position. The magnetic field emulator emulates the magnetic field measurements. It generates the magnetic field dipole using the positioning data values received from the orbit propagator. The data are then displayed and also sent to the orbit propagator. Section 2.4.2.1 discusses the magnetic fields calculations, as presented by James R. Werts [1].

3.4.2 Software design for magnetic field emulator

The software design of the magnetic field emulator takes the radial distance, the longitude and the latitude as the inputs for the vehicle position. It then calculates the magnetic field dipoles based on these positioning parameters. In other words, it emulates the magnetic field measurement, produces the results and sends them to the propagator.

Stellenbosch University http://scholar.sun.ac.za

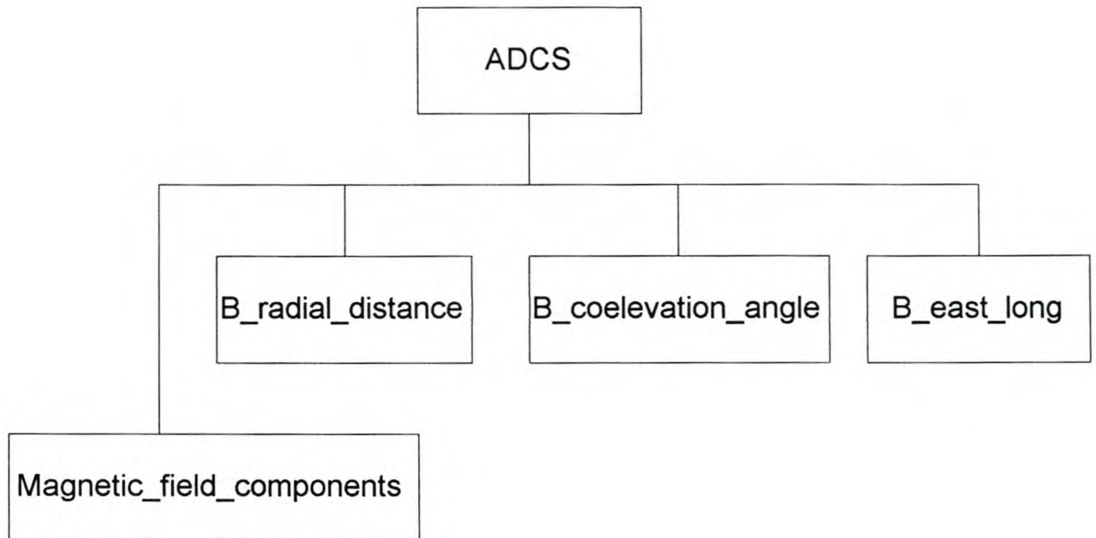


Figure 3.17: Earth's magnetic field emulator

The OBC1 emulator uses the data received from the orbit propagator to calculate the magnetic field of the earth in that position. Figure 3.17 above shows a sketch of the magnetic field emulator program. It is a class called ADCS and consists of four methods. Magnetic_field_components is a method used to display the magnetic field data. B_radial_distance calculates and returns the field dipole value in the direction of the geocentric distance. B_coelevation_angle calculates the field dipole value in the direction of the coelevation angle. The B_east_long method calculates and returns the dipole value in the direction of the longitude, East of Greenwich. The magnetic field data is then passed on to the Transmitter program. The Transmitter program takes the data and sends it to the orbit propagator.

3.4.3 Implementation of magnetic field emulator

After the magnetic field emulator was implemented, data was received via the serial port and was used to calculate the magnetic field for each and every specified satellite position. The position is specified in terms of height, longitude and latitude. The magnetic field dipole B_r which is aligned in the direction of geocentric distance r, is given by equation 1.2.34, where g_1^0 is a Gaussian coefficient of order $m=0$ and degree $n=1$, and g_1^1 and h_1^1 are

Stellenbosch University http://scholar.sun.ac.za

Gaussian coefficients of order *m=1* and degree *n=1*. The values for these coefficients are obtained in table H-1 of James R. Werts [1].

$$B_r = 2\left(\frac{a}{r}\right)^3 \left[g_1^0 \cos\theta + \left(g_1^1 \cos\theta + h_1^1 \sin\phi\right)\sin\theta\right] \qquad \text{1.2.34}$$

The position parameters for the satellite have also been used in equation 1.2.34 to compute B_r. Once the height of the satellite has been obtained from the orbit propagator, the geocentric distance, *r*, was calculated by adding the radius, *R*, of the earth to the height of the satellite. The angles, ϕ and θ, are the longitude and latitude, and *a* is the mean radius of the earth with a value of 6371.2 km. The piece of a Java code, which uses equation 1.2.34 to determine the value for B_r, has been given figure 3.18. It is a function, which takes three input parameters: phi, theta and r.

```
public String B_radial_distance(double phi, double theta, double r)
        {
                B_radian = 2*(a/r)*(a/r)*(a/r)*(g1_0*Math.cos(theta*Math.PI/180) +
                                        (g1_1*Math.cos(phi*Math.PI/180) +
                                        h1_1*Math.sin(phi*Math.PI/180))*
                                        Math.sin(theta*Math.PI/180));

                B_r     = String.valueOf(Text.format(B_radian,6,4));
                B_rad.setText(B_r);
                return B_r;
        }
```

Figure 3.18: A Java code for determining B_r which is the magnetic field dipole aligned in the direction of geocentric distance *r*.

The value that is returned by this function is converted into a String, as needed for displaying in the text field. Equation 1.2.35 has been used to compute B_θ, which is the

Stellenbosch University http://scholar.sun.ac.za

magnetic field dipole aligned in the direction of the co-elevation angle.

$$B_\theta = \left(\frac{a}{r}\right)^3 \left[g_1^0 \sin\theta - \left(g_1^1 \sin\theta + h_1^1 \sin\phi\right)\cos\theta\right] \quad \text{.............................. 1.2.35}$$

The Java code for equation 1.2.35 is given in figure 3.19. The angles phi, theta and r are the input parameters. The values of these parameters are longitude, latitude and geocentric distance.

```
public String B_coelevation_angle(double phi, double theta,double r)
        {
                Btheta = (a/r)*(a/r)*(a/r)*
                                (g1_0*Math.sin(theta*Math.PI/180) -
                                (g1_1*Math.cos(phi*Math.PI/180) +
                                 h1_1*Math.sin(phi*Math.PI/180))*
                                 Math.cos(theta*Math.PI/180));

                BTheta= String.valueOf(Text.format(Btheta,6,4));
                B_theta.setText(BTheta);

                return BTheta;
        }
```

Figure 3.19: A Java code for computing B_θ, which is the magnetic field dipole aligned in the direction of the co-elevation angle.

The returned value for this function is converted into a string to make it easier to display in a text field. Equation 1.2.36 has been used to calculate B_ϕ which is the magnetic field dipole aligned in the direction of longitude, ϕ, East of Greenwich and its Java code is given in figure 3.20.

$$B_\phi = \left(\frac{a}{r}\right)^3 \left[g_1^1 \sin\phi - h_1^1 \cos\phi\right] \quad \text{.. 1.2.36}$$

Stellenbosch University http://scholar.sun.ac.za

The function has two input parameters and it returns a value, which is also converted into a string. The input parameters are the longitude, phi or ϕ, and the geocentric distance, r.

```
public String B_east_long(double phi, double r)
        {
                Bphi = (a/r)*(a/r)*(a/r)*
                        (g1_1*Math.sin(phi*Math.PI/180) -
                         h1_1*Math.cos(phi*Math.PI/180));
                BPhi = String.valueOf(Text.format(Bphi,6,4));
                B_phi.setText(BPhi);
                return BPhi;
        }
```

Figure 3.20: A Java code for calculating B_ϕ which is the magnetic field dipole aligned in the direction of longitude, ϕ, East of Greenwich.

These values of the magnetic field dipoles represent the magnetic field at that position of the satellite. The magnetic field data is taken by the OBC1 emulator and combined into a bundle of data. The OBC1 emulator calls the Send thread of the communication protocol to send the data to the orbit propagator. The propagator takes the magnetic field data and displays it on the screen. The Receive and Send threads are invoked when the OBC1 emulator starts executing, and the Receive thread listens for any incoming data. The data received are the position parameters of the satellite, which are used to calculate the magnetic fields.

Stellenbosch University http://scholar.sun.ac.za

4. EVALUATION OF THE SYSTEM, CONCLUSION AND RECOMMENDATIONS

When the system was planned, test criteria were also specified that could help in evaluating the system. The first criterion was to draw the orbit and observe its shape as the eccentricity changes its value between 0 and 1. The expected outcome was that the orbit would become more circular as eccentricity approaches 0, and more elliptical as eccentricity approaches 1. This was tested, and the expected outcome was demonstrated.

The second test criteria was to find out if the positioning of the satellite in the orbit was exactly the way stated by Kepler's second law. According to Kepler's second law, the satellite or an object is supposed to move faster as it approaches perigee and to slow down as it approaches apogee. It was going to be tested by drawing radial distances from the centre of the earth to the satellite point or centre. The radial distances are supposed to be more spread out as the satellite approaches the perigee point. When the program was tested, the radial distances were exactly as expected. The only problem was that, upon approaching the apogee, some of the radial distances were clustered together before the apogee point and after the apogee point, as shown in Figure 3 in Section 3.1. After a careful analysis, it was realised that the reason for this behaviour was the way in which the calculations had been carried out. Looking at equation 1 in Section 3.1, it is clear that the true anomaly is determined as a function of the mean anomaly. A number of techniques therefore were selected to find a solution to this problem.

A graph of the true anomaly angle versus the mean anomaly angle was plotted. It then was observed that the true anomaly angle increases more rapidly for values for the mean anomaly angle that are close to zero and also around 1. But for values from 1 to between 5 and 6, the true anomaly angle increases slowly, and instead starts to oscillate sinusoidally, as shown in Figure 3.1 in Section 3.1. As the mean anomaly approaches one, the true anomaly angle increases rapidly again, as shown in Figure 3.1. The reason why the graph oscillates is that the equation that was used to calculate the true anomaly angle was the approximate solution, and it only works properly for the lower values of eccentricity. Hence, an alternative equation had to be used. Equation 4 of section 3.1 has, therefore, been used as an

Stellenbosch University http://scholar.sun.ac.za

alternative, and it relates to true anomaly as a function of eccentric anomaly angle and eccentricity.

A number of iterative algorithms were used to find the eccentric anomaly angle from equation 2. Those algorithms were the halving algorithms, the True Solution and the Newtonian method. All three algorithms gave accurate results. The test method still was to plot a graph of true anomaly angle versus mean anomaly angle. This time, the graph increased rapidly, with values of mean anomaly from zero to one, and became partially flattened from 1 up to values between 5 and 6. Thereafter, it again increases rapidly as the mean anomaly approaches 1. This was how it was expected to behave. This time, when the orbit was plotted using one of these algorithms to calculate true anomaly, the test results were very accurate, as expected. The radial distances formed the expected pattern, for an accurate calculation. See Figure 2.1 in Section 2.1.

The communication protocol also encountered problems, with some of the characters being lost or overwritten. The technique that was used to solve the problem was to first use a time delay, as well as to ensure that the sending of each character was acknowledged. If no acknowledgement was received, the character had to be retransmitted. After testing and debugging, the system began to work properly. The system still works well and both simulators are communicating through the serial port.

The manner in which the data is transferred makes the system partially slower, because each and every character that is received is acknowledged. However, the data rate of transfer is less than one second, the data is transfered successfully in milliseconds. Therefore, the communication protocol works fine, although few errors are experienced by losing characters, but is rectified by using a zero value to avoid an exception. However, the data is being transferred successfully.

In this thesis, it was possible to demonstrate the basic structure of a multiprocessor based HIL system successfully, including the basic operating system kernel, the kinematic orbit propagator as well as the infrastructure for the sensors and torquers orientation control

Stellenbosch University http://scholar.sun.ac.za

dynamics. The basic infrastructure components that were demonstrated were the magnetic field emulator and the implementation of the emulation for the orientation angle of a satellite. User friendly software with a graphical user interface was also designed and demonstrated. The magnetic field emulator was also designed and demonstrated successfully by using the magnetic field dipole equations as explained in Section 3.4. More ADCS components could have been included in the system if there had been enough time, and hence are recommended for future reseach. This includes components such as the star sensor and the sun sensor, and possibly also the entire Attitude Determination and Control Subsystem (ADCS).

Stellenbosch University http://scholar.sun.ac.za

REFERENCES

[1]. James R. Werts, "Spacecraft Attitude Determination and Control", 1999.

[2] Steven C. Chapra and Raymond P. Canale. "Numerical Methods for Engineers" with Personal computer applications: 1985.

[3] George B. Thomas and Ross L. Finney, "Calculus and Analytic Geometry", 1988, 7th Edition.

[4] James R. Werts and Wiley J. Larson, "Space Mission Analysis and Design", 1999, 3rd Edition.

[5] Giorgio C. Buttazzo, "Hard Real-Time Computing Systems.", Predictable Scheduling Algorithms and Applications, Kluwer Academic Publishers.

[6] Hermann Kopetz, "Real-Time Systems", Design Principles for Distributed Embedded Applications, Kluwer Academic Publishers.

[7] John A. Stankovic, Marco Spuri, Krithi Ramamritham, Giorgio C. Buttazzo, "Deadline Scheduling for Real-Time Systems", EDF and Related Algorithms, Kluwer Academic Publishers.

[8] Bruce A. Campbell and Walter McCandles, Jr, "Introduction To Space Sciences And Spacecraft Applications", Gulf Publishing Company.

[9] Harvey M. Deitel, "Operating Systems", second edition.

[10] Andrew S. Tannenbaum, "Computer Networks".

[11] Uyless D. Black, "Data Communications And Distributed Networks", third edition.

Stellenbosch University http://scholar.sun.ac.za

[12] Editor: Wushow Chou, Contributors: Wushow Chou, Ira W. Cotton, Gilbert Falk, Simon S. Lam, Patric V. McGregor, R. Andrew Pickens and Hellen M. Wood, "Computer Communications", Volume 1.

Stellenbosch University http://scholar.sun.ac.za

APPENDIX A: Listing of Java programs

This appendix contains all the software code written for the thesis in Java and some in Matlab. A brief explanation is given for each piece of code.

A.1 NEWTON RAPHSON METHOD

The Newtonian method is the algorithm that uses iterations to solve the roots of an equation. The NewtonRaphsonMethod function below implements the Newtonian method. It has two parameters as shown below and it does the iterations until the specified limit has been reached. When that happens, the solution for the roots has been found.

```
function EC = NewtonRaphsonMethod(M, e)
   n=1;
   E(n) = M;                              % initial value of E
   fE(n) = E(n)-e*sin(E(n))-M;            % Definition of a function f(E)

   while(abs(fE(n)) >= (1*10^(-8)))       % the program stops when this
                                          % condition is satisfied.
      n=n+1;                              % we increment n by 1 so that we
                                          % can test for the next value of E.

      E(n) = E(n-1) + (M+e*sin(E(n-1))-E(n-1))/(1-e*cos(E(n-1)));
                                          % Newtonian method for next value

      fE(n) = E(n)-e*sin(E(n))-M;         % the function f(E)
   end
EC=E(n);
```

The following code is the **Java code** for the Newtonian method:

```
public class NewtonRaphsonMethod
{
        public double eccentricAnomally(double M, double e)
        {
                double En_1 = 0;
                double En = M;  //initial value of E
                double fEn = En-e*Math.sin(En)-M;          // Definition of a function f(E)
                while (Math.abs(fEn) >= 1E-8)              // the program stops when this
                                                           // condition is satisfied.
                {
                        En_1 = En;
                        En = En_1 + (M+e*Math.sin(En_1)-En_1)/(1-e*Math.cos(En_1));
                                                     // Newton Method for next value
                        fEn = En-e*Math.sin(En)-M;  //the function f(E)
                }
                return En;
```

Stellenbosch University http://scholar.sun.ac.za

```
        }
}
```

A.2 TRUE SOLUTION

Matlab has a function called linspace, which can used to generate a row vector of eccentric anomaly, E_n, with elements between $\mathbf{E_0=0}$ and $\mathbf{E_n}= 2\pi$, and where $\mathbf{E_n}$ is its n-th element. The following program uses a linspace to generate **16384** values of E between **0** and 2π**.** Each value of E is used to solve the function f(E). When f(E) is less than or equal to the limit, the iteration stops and the value of such E is taken as the solution. A graph of f(E) versus E is plotted by this program.

```
a=320.0; w = 0; Mo = 0.34096; to = 1; e = 0.5; mu = 398600; t = 12;

n = sqrt(mu/(a*a*a));
M = Mo + n*(t-to);
E=linspace(0,2*pi,16384);
fE = E - e*sin(E) - M;
limit = 1*10^(-8)
%x = find(abs(fE)<=limit);
for x=1:16384
    if(fE(x)<limit)
        y = [x,fE(x),E(x)];
        fprintf('%2.0f %12.7f %12.4f\r\n\',y);
        if(fE(x)> (-limit))
            y = [x,fE(x),E(x)];
            fprintf('%2.0f %12.7f %12.4f\r\n\',y);
            %break;
            c=x
        end
    end
    if(c==x)
        fprintf('Yebo GoGo\r\n\',c);
        break;
    end
end
disp(c)
y = [c,fE(c),E(c)];
fprintf('%2.0f %12.7f %12.4f\r\n\',y);

plot(E,fE)
title('Kepler Vessel')
xlabel('E(radians)')
ylabel('f(E)')
grid on
```

Stellenbosch University http://scholar.sun.ac.za

A.3 HALVING ALGORITHM

Section 3.1.2 explains the theory about the halving algorithm. This program implements the halving algorithm to solve for the eccentric anomaly, **E**. This EccentricAnomaly function uses two functions, Loop1 and Loop2, to shift the median point and to find the Eccentric anomaly.

```
function E = EccentricAnomally(M, ub, e)
lb = 0;
mid = pi;
Emid = mid - e*sin(mid) - M;
if(Emid >= 0.000000001)
   mid = Loop2(Emid, mid, ub, lb, M, e);
end
if(Emid <= -0.000000001)
   mid = Loop1(Emid, mid, ub, lb, M, e);
end
E = mid;
```

```
function Emedian = Loop1(Emid, mid, ub, lb, M, e)

while (Emid<=-0.000000001)
   lb = mid;
   mid = lb + (ub-lb)/2;
   Emid = mid - e*sin(mid) - M;

   if(Emid >= 0.000000001)
      mid = Loop2(Emid, mid, ub, lb, M, e);
      Emid = mid - e*sin(mid) - M;
   end
end
Emedian = mid;
```

```
function Emedian = Loop2(Emid, mid, ub, lb, M, e)

while (Emid >= 0.000000001)
      ub = mid;
      mid = lb + (ub-lb)/2;
      Emid = mid - e*sin(mid) - M;

      if(Emid <= -0.000000001)
         mid = Loop1(Emid, mid, ub, lb, M, e);
         Emid = mid - e*sin(mid) - M;
      end
end
Emedian = mid;
```

Stellenbosch University http://scholar.sun.ac.za

A.4 ADCS CLASS

The ADCS class draws a frame that contains the textfields for the geocentric distance, the coelevation angle, the longitude in the East of the Greenwich, and all other parameters.

```
package ADCS;
import java.awt.*;
import java.awt.event.*;
import ADCS.*;
import javagently.*;

public class ADCS extends Frame
{
        //************************ variable declaration area ******************
        private  TextField geo_r      = new TextField(12);
        private  TextField coel_angle = new TextField(12);
        private  TextField east_long  = new TextField(12);

        private  TextField B_rad   = new TextField(12);
        private  TextField B_theta = new TextField(12);
        private  TextField B_phi   = new TextField(12);


        private  Label geo_rad = new Label("geocentric distance");
        private  Label co_el_angle = new Label("coelevation angle");
        private  Label east_longt = new Label("east longitude");

        private  Label B_Rad = new Label("   B_radius");
        private  Label B_Phi = new Label("   B_phi");
        private  Label B_Theta = new Label("   B_theta");

        private Panel field_Panel  = new Panel();
        private Panel fieldPanel   = new Panel();

        private Panel SuperPanel  = new Panel();
        private static Canvas middleCanvas  = new Canvas();
        private String B_r, BPhi, BTheta;
        private double g1_0, g1_1, h1_1, a = 6371.2, B_radian = 0.0;
        private double Btheta = 0.0, Bphi = 0.0;
        //**************** variable declaration area ends ********************


        public ADCS(String s, double g1_0, double g1_1, double h1_1 )
        {
```

Stellenbosch University http://scholar.sun.ac.za

```
super(s);
setTitle("     Magnetic Fields as calculated by the ADCS");
setSize(400,200);
setBackground(Color.orange);

setLayout(new FlowLayout());
field_Panel.setLayout(new GridLayout(2, 5, 3, 3));
field_Panel.setBackground(Color.cyan);
field_Panel.setSize(200, 100);
field_Panel.add(geo_rad);
field_Panel.add(co_el_angle);
field_Panel.add(east_longt);
field_Panel.add(geo_r);
field_Panel.add(coel_angle);
field_Panel.add(east_long);

middleCanvas.setSize(200, 5);
middleCanvas.setBackground(Color.cyan);


fieldPanel.setLayout(new GridLayout(2, 5, 3, 3));
fieldPanel.setBackground(Color.cyan);
fieldPanel.setSize(200, 50);
fieldPanel.add(B_Rad);
fieldPanel.add(B_Theta);
fieldPanel.add(B_Phi);
fieldPanel.add(B_rad);
fieldPanel.add(B_theta);
fieldPanel.add(B_phi);

SuperPanel.setLayout(new GridLayout(7, 1));
SuperPanel.setBackground(Color.cyan);

SuperPanel.add(field_Panel);
SuperPanel.add(middleCanvas);
SuperPanel.add(fieldPanel);

add(SuperPanel);

setVisible(true);
addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent e){
                e.getWindow().setVisible(false);
                dispose();
        }
});
```

Stellenbosch University http://scholar.sun.ac.za

```
        B_r = new String();
        BTheta = new String();
        BPhi = new String();

        this.g1_0 = g1_0;
        this.g1_1 = g1_1;
        this.h1_1 = h1_1;
}


public void Magnetic_field_components(double phi, double theta, double r)
{
        geo_r.setText(String.valueOf(r));

        coel_angle.setText(String.valueOf(theta));

        east_long.setText(String.valueOf(phi));
}

public String B_radial_distance(double phi, double theta, double r)
{
        B_radian = 2*(a/r)*(a/r)*(a/r)*(g1_0*Math.cos(theta*Math.PI/180) +

                                (g1_1*Math.cos(phi*Math.PI/180)  +

                                 h1_1*Math.sin(phi*Math.PI/180))*

                                 Math.sin(theta*Math.PI/180));

        B_r      = String.valueOf(Text.format(B_radian,6,4));
        B_rad.setText(B_r);
        return B_r;
}

public String B_coelevation_angle(double phi, double theta,double r)
{
        Btheta = (a/r)*(a/r)*(a/r)*

                        (g1_0*Math.sin(theta*Math.PI/180) -

                        (g1_1*Math.cos(phi*Math.PI/180)  +

                         h1_1*Math.sin(phi*Math.PI/180))*

                         Math.cos(theta*Math.PI/180));
```

Stellenbosch University http://scholar.sun.ac.za

```
            BTheta= String.valueOf(Text.format(Btheta,6,4));
            B_theta.setText(BTheta);
            return BTheta;
        }

        public String B_east_long(double phi, double r)
        {
            Bphi = (a/r)*(a/r)*(a/r)*

                        (g1_1*Math.sin(phi*Math.PI/180) -

                         h1_1*Math.cos(phi*Math.PI/180));

            BPhi = String.valueOf(Text.format(Bphi,6,4));
            B_phi.setText(BPhi);
            return BPhi;
        }
}
```

A.5 OBC1 SIMULATION CLASS

This program contains methods for simulating the yaw, pitch, and roll by using random numbers.

```
import javagently.*;
import java.io.*;
import java.util.*;
import javax.comm.*;
import ADCS.*;

public class OBC1_Simulation extends Thread
{
        Receive Serial;
        public static String Data = new String();
        public static boolean TxFlag = false;
        public static boolean DataFlag = false,
                                        dispFlag = true;

        public OBC1_Simulation(ADCS adcs)
        {
            Serial = new Receive(adcs);
        }

        public double Roll()
        {
            double Roll = Math.random();
```

Stellenbosch University http://scholar.sun.ac.za

```
        return Roll;
}

public double Pitch()
{
        double Pitch = Math.random();
        return Pitch;
}

public double Yaw()
{
        double Yaw = Math.random();
        return Yaw;
}

public double[] SimulateOrientation()
{
                double orientation[] = new double[3];
                orientation[0] = Yaw();
                orientation[1] = Pitch();
                orientation[2] = Roll();
                return orientation;
}

public static void delay(int k){
        //delays for k sec
        try{ Thread.currentThread().sleep(k); }
        catch(InterruptedException e){}
}

public void sendOrientation()
{
        double orientation[] = new double[3];
        orientation = SimulateOrientation();

        if (dispFlag)
        {
                dispFlag = false;
                Simulator.Yaw.setText(String.valueOf(

Text.format(orientation[0]*100,6,4)));
                Simulator.Pitch.setText(String.valueOf(

Text.format(orientation[1]*100,6,4)));
                Simulator.Roll.setText(String.valueOf(
```

Stellenbosch University http://scholar.sun.ac.za

```
        Text.format(orientation[2]*100,6,4)));
                    if (TxFlag == false)
                    {
                            Send.YAW = new String();
                            Send.PITCH = new String();
                            Send.ROLL = new String();

                            Send.YAW = Text.format(orientation[0]*100,6,3);
                            Send.PITCH = Text.format(orientation[1]*100,6,3);
                            Send.ROLL = Text.format(orientation[2]*100,6,3);

                            TxFlag = true;
                            DataFlag = true;
                    }
            }
        }

        public void run()
        {
                Serial.start();
                while (true)
                {
                        sendOrientation();
                        delay(100);
                }
        }
}
```

A.6 SIMULATION CLASS

This program draws a frame with textfields for the yaw, pitch, roll, height, longitude and latitude. It also activates the OBC class and calculates the magnetic fields.

```
import javagently.*;
import java.io.*;
import java.awt.*;
import java.awt.event.*;
import java.util.*;
import javax.comm.*;
import ADCS.*;



public class Simulator extends Frame
{
```

Stellenbosch University http://scholar.sun.ac.za

```
//************************* Data area ****************************
public  static TextField Yaw      = new TextField(12);
public  static TextField Pitch    = new TextField(12);
public  static TextField Roll     = new TextField(12);

public  static TextField Yaw1     = new TextField(12);
public  static TextField Pitch1   = new TextField(12);
public  static TextField Roll1    = new TextField(12);

public  static TextField height   = new TextField(12);
public  static TextField longitude = new TextField(12);
public  static TextField latitude = new TextField(12);


private  Label L1 = new Label("Yaw");
private  Label L2 = new Label("Pitch");
private  Label L3 = new Label("Roll");

private  Label L4 = new Label("Yaw");
private  Label L5 = new Label("Pitch");
private  Label L6 = new Label("Roll");

private  Label L7 = new Label("longitude");
private  Label L8 = new Label("latitude");
private  Label L9 = new Label("height");

private Panel Panel1      = new Panel();
private Panel Panel2      = new Panel();
private Panel Panel3      = new Panel();

private Panel SuperPanel  = new Panel();
private Panel LeftPanel   = new Panel();
private Panel MiddlePanel = new Panel();
private Panel RightPanel  = new Panel();


private static Canvas topCanvas       = new Canvas();
private static Canvas middleCanvas  = new Canvas();
private static Canvas bottomCanvas  = new Canvas();
private static Canvas leftCanvas    = new Canvas();
private static Canvas rightCanvas   = new Canvas();
private static Canvas labelCanvas   = new Canvas();


private Canvas TxCanvas = new Canvas();
```

Stellenbosch University http://scholar.sun.ac.za

```
        private Canvas RxCanvas = new Canvas();
        private static ADCS adcs;
        private static String g1_0, g1_1, h1_1;
        private static BufferedReader keyboard;
        private OBC1_Simulation OBC;

//***************** Data area ends ***************************************


        public Simulator(String s)
        {
                super(s);
                setTitle("Simulation for the Satellite Orientation");
                setSize(500,550);
                setBackground(Color.orange);

                setLayout(new FlowLayout());
                Panel1.setLayout(new GridLayout(2, 5, 10, 20));
                Panel1.setBackground(Color.orange);
                Panel1.setSize(200, 200);
                Panel1.add(L1);
                Panel1.add(L2);
                Panel1.add(L3);
                Panel1.add(Yaw);
                Panel1.add(Pitch);
                Panel1.add(Roll);

                setLayout(new FlowLayout());
                Panel2.setLayout(new GridLayout(2, 20, 10, 20));
                Panel2.setBackground(Color.orange);
                Panel2.setSize(200, 200);
                Panel2.add(L4);
                Panel2.add(L5);
                Panel2.add(L6);
                Panel2.add(Yaw1);
                Panel2.add(Pitch1);
                Panel2.add(Roll1);

                setLayout(new FlowLayout());
                Panel3.setLayout(new GridLayout(2, 20, 10, 20));
                Panel3.setBackground(Color.orange);
                Panel3.setSize(200, 100);
                Panel3.add(L7);
                Panel3.add(L8);
                Panel3.add(L9);
                Panel3.add(height);
```

Stellenbosch University http://scholar.sun.ac.za

```
Panel3.add(longitude);
Panel3.add(latitude);

setLayout(new FlowLayout());
topCanvas.setSize(200, 50);
middleCanvas.setSize(200, 70);
bottomCanvas.setSize(200, 60);
leftCanvas.setSize(50, 500);
rightCanvas.setSize(50, 500);
labelCanvas.setSize(200, 20);

topCanvas.setBackground(Color.green);
middleCanvas.setBackground(Color.cyan);
bottomCanvas.setBackground(Color.cyan);
leftCanvas.setBackground(Color.cyan);
rightCanvas.setBackground(Color.cyan);
labelCanvas.setBackground(Color.cyan);

setLayout(new FlowLayout());
SuperPanel.setLayout(new GridLayout(7, 1));
SuperPanel.setBackground(Color.orange);

SuperPanel.add(topCanvas);
SuperPanel.add(labelCanvas);
SuperPanel.add(Panel1);
SuperPanel.add(middleCanvas);
SuperPanel.add(Panel2);
SuperPanel.add(bottomCanvas);
SuperPanel.add(Panel3);

setLayout(new FlowLayout());
LeftPanel.setSize(50, 400);
MiddlePanel.setSize(200, 400);
RightPanel.setSize(50, 400);

LeftPanel.add(leftCanvas);
MiddlePanel.add(SuperPanel);
RightPanel.add(rightCanvas);

add(LeftPanel);
add(MiddlePanel);
add(RightPanel);

setVisible(true);
addWindowListener(new WindowAdapter(){
        public void windowClosing(WindowEvent e){
```

Stellenbosch University http://scholar.sun.ac.za

```
                        e.getWindow().setVisible(false);
                        dispose();
                        OBC.stop();
                }
        });
}

public void writeToCanvas(Graphics g)
{
        g.setFont(new Font("Monospaced", Font.ITALIC, 14));
        g.drawString(" Satellite orienation: This simulates OBC1", 5, 10);
        g.drawString("Designer: Lungile L. Grungxu", 20, 30);
        g.drawString("Stellenbosch", 30, 50);
}

public void writeToMiddleCanvas(Graphics g)
{
        g.setFont(new Font("Monospaced", Font.BOLD, 15));
        g.drawString(" Received Orientation parameters", 10, 60);
}

public void writeToCanvasBottom(Graphics g)
{
        g.setFont(new Font("Monospaced", Font.BOLD, 15));
        g.drawString(" Transmitted Orientation parameters", 0, 60);
}

public void writeToBottomCanvas(Graphics g)
{
        g.setFont(new Font("Monospaced", Font.BOLD, 15));
        g.drawString("Satellite positioning parameters", 10, 30);
        g.drawString("simulated by orbit-propagator:", 20, 45);
        g.drawString("They are displayed below", 30, 60);


}

public void displaytoBottomCanvas(Simulator sim)
{
        sim.writeToBottomCanvas(bottomCanvas.getGraphics());
        sim.writeToBottomCanvas(bottomCanvas.getGraphics());
        sim.writeToBottomCanvas(bottomCanvas.getGraphics());
        sim.writeToBottomCanvas(bottomCanvas.getGraphics());
}

public void displaytoTopCanvas(Simulator sim)
```

Stellenbosch University http://scholar.sun.ac.za

```
{
        sim.writeToCanvas(topCanvas.getGraphics());
        sim.writeToCanvas(topCanvas.getGraphics());
        sim.writeToCanvas(topCanvas.getGraphics());
}

public void displaytoLabelCanvas(Simulator sim)
{
        sim.writeToCanvasBottom(labelCanvas.getGraphics());
}

public void displaytoMiddleCanvas(Simulator sim)
{
        sim.writeToMiddleCanvas(middleCanvas.getGraphics());
        sim.writeToMiddleCanvas(middleCanvas.getGraphics());
        sim.writeToMiddleCanvas(middleCanvas.getGraphics());
        sim.writeToMiddleCanvas(middleCanvas.getGraphics());
}


public static void main(String [] args) throws IOException
{
        g1_0 = new String();
        g1_1 = new String();
        h1_1 = new String();
        keyboard = new BufferedReader(new InputStreamReader(System.in));

        System.out.print("\n\nEnter the ff values:\n\ng1_0: ");
        g1_0 = keyboard.readLine();
        System.out.println();

        System.out.print("g1_1:  ");
        g1_1= keyboard.readLine();
        System.out.println();

        System.out.print("h1_1:  ");
        h1_1= keyboard.readLine();
        System.out.print("\n");

        adcs = new ADCS("Magnetic Fields",new Double(g1_0).doubleValue(),
                        new Double(g1_1).doubleValue(),
                        new Double(h1_1).doubleValue());

        Simulator sim = new Simulator("Simulation for the Satellite Orientation");
        sim.displaytoTopCanvas(sim);
        sim.displaytoLabelCanvas(sim);
```

Stellenbosch University http://scholar.sun.ac.za

```
                sim.displaytoMiddleCanvas(sim);
                sim.displaytoBottomCanvas(sim);
                sim.OBC = new OBC1_Simulation(adcs);
                sim.OBC.start();
        }
}
```

A.7 SEND THREAD FOR ORIENTATION

This thread is used to send the data through the serial port. The data to be sent is yaw, pitch, and roll

```
import java.io.*;
import java.util.*;
import javax.comm.*;

public class Send extends Thread{
        static OutputStream outputStream;
        static SerialPort serialPort;
        public static String data;

        public static String YAW = new String();
        public static String PITCH = new String();
        public static String ROLL = new String();

        public static boolean flag = false;
        public static boolean TxFlag = false;


        /*******************        beginning of dataTx   *******************/
        private String DataString = new String();
        private boolean DataFlag, flagData=true;

        /*******************        end of dataTx  ***********************/

        public Send(SerialPort serialPort)
        {
                //constructor
                this.serialPort = serialPort;
        }

        public void run() {

                System.out.println("\n\n");
```

Stellenbosch University http://scholar.sun.ac.za

```
try {outputStream = serialPort.getOutputStream();}
catch(IOException e){}

try
{
        while (true)
        {
                if (Receive.TxAck==true)
                {
                        outputStream.write((byte)'@');
                        Receive.TxAck=false;
                }

                if (Receive.StartPacket==true)
                {
                        outputStream.write((byte)'y');
                        Receive.StartPacket=false;
                }

                if (Receive.EndPacketAck==true)
                {
                        outputStream.write((byte)'%');
                        Receive.EndPacketAck=false;
                }

                DataFlag = OBC1_Simulation.DataFlag;
                if (DataFlag == true)
                {
                        OBC1_Simulation.DataFlag = false;
                        if (flagData==true)
                        {
                                Receive.TxFlag=true;
                                flagData=false;
                        }
                }

                boolean TxFlag = Receive.TxFlag;
                if (TxFlag==true)
                {
                        System.out.println("\nSENT:");

                        System.out.print("s");
                        while (true)
                        {
                                outputStream.write((byte)'s');
```

Stellenbosch University http://scholar.sun.ac.za

```
        sleep(10);
        if (Receive.AckStart==true)
        {
                break;
        }
}
Receive.AckStart=false;


for(int i=0; i<YAW.length(); i++)
{
        outputStream.write((byte)YAW.charAt(i));
        System.out.print(YAW.charAt(i));
        sleep(20);
}

System.out.print("#");
while (true)
{
        outputStream.write((byte)'#');
        sleep(60);
        if (Receive.Ack==true)break;
}
Receive.Ack=false;

for(int i=0; i<PITCH.length(); i++)
{
        outputStream.write((byte)PITCH.charAt(i));
        System.out.print(PITCH.charAt(i));
        sleep(20);
}

System.out.print("#");
while (true)
{
        outputStream.write((byte)'#');
        sleep(60);
        if (Receive.Ack==true)break;
}
Receive.Ack=false;

for(int i=0; i<ROLL.length(); i++)
{
        outputStream.write((byte)ROLL.charAt(i));
        System.out.print(ROLL.charAt(i));
        sleep(20);
```

Stellenbosch University http://scholar.sun.ac.za

```
                    }

                    System.out.print("#");
                    while (true)
                    {
                            outputStream.write((byte)'#');
                            sleep(60);
                            if (Receive.Ack==true)break;
                    }
                    Receive.Ack=false;

                    for(int i=0; i<Receive.B_radial_distance.length(); i++)
                    {

outputStream.write((byte)Receive.B_radial_distance.charAt(i));

System.out.print(Receive.B_radial_distance.charAt(i));
                            sleep(20);
                    }

                    System.out.print("#");
                    while (true)
                    {
                            outputStream.write((byte)'#');
                            sleep(60);
                            if (Receive.Ack==true)break;
                    }
                    Receive.Ack=false;

            for(int i=0; i<Receive.B_coelevation_angle.length(); i++)
                    {

outputStream.write((byte)Receive.B_coelevation_angle.charAt(i));

System.out.print(Receive.B_coelevation_angle.charAt(i));
                            sleep(20);
                    }

                    System.out.print("#");
                    while (true)
                    {
                            outputStream.write((byte)'#');
                            sleep(60);
                            if (Receive.Ack==true)break;
                    }
                    Receive.Ack=false;
```

Stellenbosch University http://scholar.sun.ac.za

```
                                        for(int i=0; i<Receive.B_east_longitude.length(); i++)
                                        {
                    outputStream.write((byte)Receive.B_east_longitude.charAt(i));

                    System.out.print(Receive.B_east_longitude.charAt(i));
                                                sleep(20);
                                        }

                                        System.out.print("&");
                                        while (true)
                                        {
                                                outputStream.write((byte)'&');
                                                sleep(20);
                                                if (Receive.AckEndPacket==true)break;
                                        }

                                        Receive.AckEndPacket=false;
                                        OBC1_Simulation.TxFlag = false;
                                        System.out.println("\n");

                                        Receive.Yaw = new char[20];
                                        Receive.Pitch = new char[20];
                                        Receive.Roll = new char[20];
                                        Receive.height = new char[20];
                                        Receive.longitude = new char[20];
                                        Receive.latitude = new char[20];

                                        Receive.TxFlag = false;
                                }
                                else
                                        sleep(2);
                        }
                }catch (IOException e) {}
                catch(InterruptedException e){}
        }
}
```

A.8 RECEIVE THREAD FOR ORIENTATION CLASS

This thread listens to the serial port and reads all the data coming in.

```
import javagently.*;
import java.io.*;
import java.util.*;
```

Stellenbosch University http://scholar.sun.ac.za

```
import javax.comm.*;
import java.lang.Byte;
import java.lang.String;
import ADCS.*;


public class Receive extends Thread implements SerialPortEventListener
{
        static CommPortIdentifier portId;
        static Enumeration portList;
        private InputStream inputStream;
        private SerialPort serialPort;
        private String string = new String();

        public static char Yaw[] = new char[20];
        public static char Pitch[] = new char[20];
        public static char Roll[] = new char[20];
        public static char height[] = new char[20];
        public static char longitude[] = new char[20];
        public static char latitude[] = new char[20];

        private int count=0, AckCount=1; String Char = new String();
        public static String B_radial_distance, B_coelevation_angle, B_east_longitude;

        private String phi, theta, a, rd, ht;
        private ADCS adcs;

        public static boolean RxFlag = true;
        public static boolean TxFlag = false;
        public static boolean TxAckFlag = false, AckStart = false;
        public static boolean AckFlag = false, StartFlag = false;
        public static boolean Ack = false, TxAck = false, StartPacket = false,
                              EndPacketAck = false, AckEndPacket = false,
                              DisplayFlag = false;

        private boolean YawF=false, PitchF = false;
        private boolean RollF=false, heightF = false;
        private boolean longitudeF=false, latitudeF = false;


        public void run()
        {
                try
                {
                        sleep(1);
                }
```

Stellenbosch University http://scholar.sun.ac.za

```
        catch(InterruptedException e){}
}

public Receive(ADCS adcs)
{
        this.adcs = adcs;
        portList = CommPortIdentifier.getPortIdentifiers();
        while (portList.hasMoreElements())
        {
                portId = (CommPortIdentifier) portList.nextElement();
                if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL)
                {
                        if (portId.getName().equals("COM1"))
                        {
                                System.out.println("Connectedthrough
                                "+portId.getName());
                                break;
                        }
                }
        }

        try
        {
                serialPort = (SerialPort) portId.open("SimpleReadApp", 2000);
        }
        catch (PortInUseException e) {}

        try
        {
                inputStream = serialPort.getInputStream();
        }
        catch (IOException e) {}

        try
        {
                serialPort.addEventListener(this);
        }
        catch (TooManyListenersException e) {}

        serialPort.notifyOnDataAvailable(true);
        try
        {
                serialPort.setSerialPortParams(9600,
                        SerialPort.DATABITS_8,
                        SerialPort.STOPBITS_1,
                        SerialPort.PARITY_NONE);
```

Stellenbosch University http://scholar.sun.ac.za

```
        }
        catch (UnsupportedCommOperationException e) {}
        Send Tx = new Send(serialPort);
        Tx.start();
}

public synchronized void serialEvent(SerialPortEvent event)
{
        switch(event.getEventType())
        {
                case SerialPortEvent.BI:
                case SerialPortEvent.OE:
                case SerialPortEvent.FE:
                case SerialPortEvent.PE:
                case SerialPortEvent.CD:
                case SerialPortEvent.CTS:
                case SerialPortEvent.DSR:
                case SerialPortEvent.RI:
                case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
                        break;
                case SerialPortEvent.DATA_AVAILABLE:
                        byte[] readBuffer = new byte[20];

                        try
                        {
                                while (inputStream.available() > 0)
                                {
                                        int numBytes = inputStream.read(readBuffer);
                                        string = new String(readBuffer);

                                        if ((string.charAt(0)!='#')&&
                                        (string.charAt(0)!='&')&&
                                        (string.charAt(0)!='@')&&
                                        (string.charAt(0)!='%')&&
                                        (string.charAt(0)!='y')&&
                                        (string.charAt(0)!='s'))
                                        {
                                                if (YawF==false)
                                                {
                                                        Yaw[count]=string.charAt(0);
                                                        count++;
                                                }
                                                else if (PitchF==false)
                                                {
                                                        Pitch[count]=string.charAt(0);
                                                        count++;
```

Stellenbosch University http://scholar.sun.ac.za

```
                        }
                        else if (RollF==false)
                        {
                                Roll[count]=string.charAt(0);
                                count++;
                        }
                        else if (heightF==false)
                        {
                                height[count]=string.charAt(0);
                                count++;
                        }
                        else if (longitudeF==false)
                        {
                        longitude[count]=string.charAt(0);
                        count++;
                }
                else
                {
                        latitude[count]=string.charAt(0);
                        count++;
                }
        }
        else if (string.charAt(0)=='@')
                Ack=true;

        else if (string.charAt(0)=='#') //if separator
        {
                string = new String();
                if (AckCount==1)
                {
                        if (YawF==false)
                        {
                                YawF=true;
                                AckCount=2;
                        }
                }

                else if (AckCount==2)
                {
                        if (PitchF==false)
                        {
                                PitchF=true;
                                AckCount=3;
                        }
                }
                else if (AckCount==3)
```

Stellenbosch University http://scholar.sun.ac.za

```
        {
                if (RollF==false)
                {
                        RollF=true;
                        AckCount=4;
                }
        }

        else if (AckCount==4)
        {
                if (heightF==false)
                {
                        heightF=true;
                        AckCount=5;
                }
        }

        else if (AckCount==5)
        {
                if (longitudeF==false)
                {
                        longitudeF=true;
                }
        }
        TxAck = true;
        count=0;
}

else if (string.charAt(0)=='%')
{
        AckEndPacket=true;
}

else if (string.charAt(0)=='y')
{
        AckStart=true;
}

else if (string.charAt(0)=='&') //if end of each packet.
{
        EndPacketAck = true;
        DisplayFlag = true;
}
else if (string.charAt(0)=='s')
{
        if (DisplayFlag==false)
```

Stellenbosch University http://scholar.sun.ac.za

```
                    {
                            StartPacket=true;
                    }
                    else
                    {
                            DisplayFlag=false;
                    }
            }

            if (DisplayFlag == true)

            {
                            DisplayFlag=false;
//****** search and remove the 2nd/3rd/etc decimal point *******
    int y=0, p=0, r=0, h=0, lg=0, lt=0;     //initialize counters
    for(int i=0; i<=17; i++)
    {
        if (Yaw[i]=='.')
        {
            y++;
            if (y>=2)
            {
                Yaw[i] = Char.charAt(0);
                y=1;
            }
        }
        if (Pitch[i]=='.')
        {
            p++;
            if (p>=2)
            {
                Pitch[i]= Char.charAt(0);
                p=1;
            }
        }
        if (Roll[i]=='.')
        {
            r++;
            if (r>=2)
            {
                Roll[i]= Char.charAt(0);
                r=1;
            }
        }
        if (height[i]=='.')
        {
```

Stellenbosch University http://scholar.sun.ac.za

```
                h++;
                if (h>=2)
                {
                        height[i]= Char.charAt(0);
                        h=1;
                }
        }
        if (longitude[i]=='.')
        {
                lg++;
                if (lg>=2)
                {
                        longitude[i]= Char.charAt(0);
                        lg=1;
                }
        }
        if (latitude[i]=='.')
        {
                lt++;
                if (lg>=2)
                {
                        latitude[i]= Char.charAt(0);
                        lt=1;
                }
        }
}
y=0; p=0; r=0; h=0; lg=0; lt=0;

//***********  change character arrays to strings  ************

AckCount=1;
String YAW = (new String(Yaw)).trim();
String PITCH = (new String(Pitch)).trim();
String ROLL = (new String(Roll)).trim();
String HEIGHT = (new String(height)).trim();
String LONGITUDE = (new String(longitude)).trim();
String LATITUDE = (new String(latitude)).trim();

Date date = new Date();
System.out.println( "Time := "+date.getHours()+" :
                "+date.getMinutes()+" : "+date.getSeconds()+"");


//***********     display to textFields   *****************
try
{
```

Stellenbosch University http://scholar.sun.ac.za

```
        System.out.println("Yaw: "+String.valueOf(
                Text.format(new Double(YAW).doubleValue(),6,4)));
        Simulator.Yaw1.setText(String.valueOf(
                Text.format(new Double(YAW).doubleValue(),6,4)));

}
catch(NumberFormatException e)
{
        YAW="0.0";
        System.out.println("Yaw: "+String.valueOf(
                Text.format(new Double(YAW).doubleValue(),6,4)));

        Simulator.Yaw1.setText(String.valueOf(
                Text.format(new Double(YAW).doubleValue(),6,4)));
}

try
{
        System.out.println("Pitch: "+String.valueOf(
        Text.format(new Double(PITCH).doubleValue(),6,4)));

        Simulator.Pitch1.setText(String.valueOf(
        Text.format(new Double(PITCH).doubleValue(),6,4)));
}

catch(NumberFormatException e)
{
        PITCH="0.0";
        System.out.println("Pitch: "+String.valueOf(
        Text.format(new Double(PITCH).doubleValue(),6,4)));

        Simulator.Pitch1.setText(String.valueOf(
        Text.format(new Double(PITCH).doubleValue(),6,4)));
}

try
{
        System.out.println("Roll: "+String.valueOf(
                Text.format(new Double(ROLL).doubleValue(),6,4)));

        Simulator.Roll1.setText(String.valueOf(
                Text.format(new Double(ROLL).doubleValue(),6,4)));
}
catch(NumberFormatException e)
{
        ROLL="0.0";
```

Stellenbosch University http://scholar.sun.ac.za

```
        System.out.println("Roll: "+String.valueOf(
                Text.format(new Double(ROLL).doubleValue(),6,4)));

        Simulator.Roll1.setText(String.valueOf(
                Text.format(new Double(ROLL).doubleValue(),6,4)));
}

try
{
        ht = Text.format(new Double(HEIGHT).doubleValue(),6,4);
        System.out.println("Height: "+String.valueOf(ht));
        Simulator.height.setText(String.valueOf(ht));
}
catch(NumberFormatException e)
{
        HEIGHT="0.0";
        ht = Text.format(new Double(HEIGHT).doubleValue(),6,4);
        System.out.println("Height: "+String.valueOf(ht));
        Simulator.height.setText(String.valueOf(ht));
}

try
{
        phi = Text.format(new Double(
                        LONGITUDE).doubleValue(),6,4);
        System.out.println("Longitude: "+String.valueOf(phi));

        Simulator.longitude.setText(String.valueOf(phi));
}
catch(NumberFormatException e)
{
        LONGITUDE="0.0";
        phi = Text.format(new Double(
                        LONGITUDE).doubleValue(),6,4);
        System.out.println("Longitude: "+String.valueOf(phi));
        Simulator.longitude.setText(String.valueOf(phi));
}

try
{
        theta = Text.format(new Double(
                        LATITUDE).doubleValue(),6,4);
        System.out.println("Latitude: "+String.valueOf(theta));
        Simulator.latitude.setText(String.valueOf(theta));
}
```

Stellenbosch University http://scholar.sun.ac.za

```
                    catch(NumberFormatException e)
                    {
                            LATITUDE="0.0";
                            theta = Text.format(new Double(
                                                    LATITUDE).doubleValue(),6,4);
                            System.out.println("Latitude: "+String.valueOf(theta));
                            Simulator.latitude.setText(String.valueOf(theta));
                    }




                    adcs.Magnetic_field_components(new Double(phi).doubleValue(),
                                            new Double(theta).doubleValue(),
                                            new Double(rd).doubleValue());

                    B_radial_distance = adcs.B_radial_distance(
                                            new Double(phi).doubleValue(),
                                            new Double(theta).doubleValue(),
                                            new Double(rd).doubleValue());

                    B_coelevation_angle = adcs.B_coelevation_angle(
                                            new Double(phi).doubleValue(),
                                            new Double(theta).doubleValue(),
                                            new Double(rd).doubleValue());

                    B_east_longitude = adcs.B_east_long(
                                            new Double(phi).doubleValue(),
                                            new Double(rd).doubleValue());

                    //***********     reset the flags     ******************
                    YawF=false;
                    PitchF=false;
                    RollF=false;
                    heightF=false;
                    longitudeF=false;
                    TxFlag=true;                          //end of packet flag.
                    OBC1_Simulation.dispFlag = true;
                }
            }
        }
        catch (IOException e) {}
        break;
    }
}
```

Stellenbosch University http://scholar.sun.ac.za

```
}
```

A.9 DisplayArea class

The DisplayArea receives parameter values through its initialize methods. It activate the SatelliteOrbit class and pass the parameter values to this class.

```
import javagently.*;
import java.io.*;
import java.awt.*;
import java.awt.event.*;

public class DisplayArea extends Frame
{
        public  static TextArea area = new TextArea("", 1000000, 20,
TextArea.SCROLLBARS_VERTICAL_ONLY);

//************************ Declaration area **********************
        public  static int x, y, t, dt;
        private static boolean flag1, flag2;
        static  double  T, i, n, r, temp, C;
        static  double  w, Mo, to , e , u, M;
        static  double  mu;                                //gravitationalParameter
        static  double  a, R, longitude, latitude, height, b, xx;
        public  static double  AsN;              // AsN is the longitude of ascending node

//***************** Constructor     ******************************
        public DisplayArea(String s)
        {
                super(s);
                //setTitle("Display Kepler orbit results");
                setSize(800,600);
                setBackground(Color.orange);
                add(area, BorderLayout.CENTER);
                area.setFont(new Font("TimesRoman", Font.PLAIN, 12));
```

Stellenbosch University http://scholar.sun.ac.za

```
            area.setBackground(Color.cyan);
            setVisible(true);
    }

    public void initialise(double i, double Mo, double e, double R, int dt,
                           double w, double to, double T, double AsN,
                           boolean flag1, boolean flag2)
    {
            this.i = i;
            this.Mo = Mo;
            this.e = e;
            this.R = R;
            this.w = w;
            this.dt = dt;
            this.to = to;
            this.T = T;
            this.AsN = AsN;
            this.flag1 = flag1;
            this.flag2 = flag2;
    }

    public void go()
    {
            final SatelliteOrbit sat = new SatelliteOrbit();
            Frame f = new DisplayArea("Display Kepler orbit results");
            f.addWindowListener(new WindowAdapter(){
                    public void windowClosing(WindowEvent e){
                            e.getWindow().setVisible(false);
                            dispose();
                            sat.stop();
                    }
            });
            sat.initialise(i, Mo, e, R, dt, w, to, T, AsN, flag1, flag2);
            sat.start();
    }
}
```

A.10 KeplerOrbit class

This code draws a frame with textfields for the Kepler elements and buttons for drawing a graph, drawing orbit and viewing in column format.

```
import javagently.*;
import java.io.*;
```

Stellenbosch University http://scholar.sun.ac.za

```
import java.awt.*;
import java.awt.Font;
import java.awt.event.*;

public class KeplerOrbit extends Frame implements ActionListener
{
        public static int     dt;
        public static double  T, i ;
        public static double  w, Mo, to, e ;
        public static double  R;
        public static double  AsN;// AsN is the longitude of ascending node


        private static TextField period = new TextField(12);
        private static TextField inclination = new TextField(12);
        private static TextField epochTime = new TextField(12);
        private static TextField MeanAnomally_at_epoch = new TextField(12);
        private static TextField eccentricity = new TextField(12);
        private static TextField argumentOf_perigee = new TextField(12);
        private static TextField longitude_Of_Ascending_Node = new TextField(12);
        private static TextField timeIncrement_dt = new TextField(12);
        private static TextField earthRadius = new TextField(12);


        private  Label L1 = new Label("Orbital period(sec)");
        private  Label L2 = new Label("inclination angle(deg)");
        private  Label L3 = new Label("epoch time(sec)");
        private  Label L4 = new Label("Mean anomally at epoch (rad)");
        private  Label L5 = new Label("eccentricity");
        private  Label L6 = new Label("argument of perigee (deg)");
        private  Label L7 = new Label("longitude of ascending node(deg)");
        private  Label L8 = new Label("time increment, dt (sec)");
        private  Label L9 = new Label("earth radius, R (km)");

        private  Button graphButton = new Button("draw graph");
        private  Button orbitButton = new Button("draw Orbit");
        private  Button displayButton = new Button("view in column format" );

        private  Panel dataPanel = new Panel();
        private  Panel buttonPanel = new Panel();

        public  static Canvas buttonCanvas = new Canvas();
        public  static KeplerOrbit orbit = new KeplerOrbit("Kepler orbit");

        private static boolean flag1, flag2 = false;
```

Stellenbosch University http://scholar.sun.ac.za

```
/************ BEGINING OF A CONSTRUCTOR ****************/
public KeplerOrbit(String s)
{
        super(s);
        setTitle("Kepler Orbit");
        setSize(800,500);
        setBackground(Color.green);
        setLayout(new BorderLayout());

        dataPanel.setSize(400, 300);
        dataPanel.setLayout(new GridLayout(8, 3, 100, 1));
        dataPanel.setBackground(Color.cyan);

        buttonPanel.setSize(600, 100);
        buttonPanel.setLayout(new GridLayout(1, 3, 50, 0));
        buttonPanel.setBackground(Color.green);

        buttonCanvas.setBackground(Color.orange);
        buttonCanvas.setSize(600, 200);

        dataPanel.add(L1);
        dataPanel.add(L2);
        dataPanel.add(L3);
        dataPanel.add(period);
        dataPanel.add(inclination);
        dataPanel.add(epochTime);
        dataPanel.add(L4);
        dataPanel.add(L5);
        dataPanel.add(L6);
        dataPanel.add(MeanAnomally_at_epoch);
        dataPanel.add(eccentricity);
        dataPanel.add(argumentOf_perigee);
        dataPanel.add(L7);
        dataPanel.add(L8);
        dataPanel.add(L9);
        dataPanel.add(longitude_Of_Ascending_Node);
        dataPanel.add(timeIncrement_dt);
        dataPanel.add(earthRadius);

        buttonPanel.add(orbitButton);
        buttonPanel.add(graphButton);
        buttonPanel.add(displayButton);


        period.addActionListener(this);
```

Stellenbosch University http://scholar.sun.ac.za

```
        inclination.addActionListener(this);
        epochTime.addActionListener(this);
        MeanAnomally_at_epoch.addActionListener(this);
        eccentricity.addActionListener(this);
        argumentOf_perigee.addActionListener(this);
        longitude_Of_Ascending_Node.addActionListener(this);
        timeIncrement_dt.addActionListener(this);
        earthRadius.addActionListener(this);
        dataPanel.setVisible(true);


        graphButton.addActionListener(this);
        orbitButton.addActionListener(this);
        displayButton.addActionListener(this);
        buttonPanel.setVisible(true);


        add(dataPanel, BorderLayout.NORTH);
        setLayout(new FlowLayout());
        add(buttonCanvas);
        add(buttonPanel);
        setVisible(true);

        R=6378.0;
        i = 86.0;  w  =  0.0;
        T = 120.0; AsN=30.0;
        Mo = 0.0;  to = 0.0;
        dt = 60;   e  =  0.2;
        Mo = Mo*Math.PI/180;
}
/**********    END OF A CONSTRUCTOR        ***************/


 public void getInitValues()
 {
        orbit.writeToCanvas(buttonCanvas.getGraphics());
        period.setText(String.valueOf(Text.format(T,6,4)));
        inclination.setText(String.valueOf(Text.format(i,6,4)));
        epochTime.setText(String.valueOf(Text.format(to,6,4)));
        MeanAnomally_at_epoch.setText(String.valueOf(Text.format(Mo,6,4)));
        eccentricity.setText(String.valueOf(Text.format(e,6,4)));
        argumentOf_perigee.setText(String.valueOf(Text.format(w,6,4)));

        longitude_Of_Ascending_Node.setText(
                                String.valueOf(Text.format(AsN,6,4)));
        timeIncrement_dt.setText(String.valueOf(dt));
```

Stellenbosch University http://scholar.sun.ac.za

```
        earthRadius.setText(String.valueOf(Text.format(R,6,4)));
        orbit.writeToCanvas(buttonCanvas.getGraphics());
}

public static void getNewValues()
{
        orbit.writeToCanvas(buttonCanvas.getGraphics());
        T = (Double.valueOf(period.getText())).doubleValue();
        i = (Double.valueOf(inclination.getText())).doubleValue();
        to = (Double.valueOf(epochTime.getText())).doubleValue();
        Mo = (Double.valueOf(MeanAnomally_at_epoch.getText())).doubleValue();
        e = (Double.valueOf(eccentricity.getText())).doubleValue();
        w = (Double.valueOf(argumentOf_perigee.getText())).doubleValue();

        AsN = (Double.valueOf(longitude_Of_Ascending_Node.getText()
                                                            )).doubleValue();
        dt = (Integer.valueOf(timeIncrement_dt.getText())).intValue();
        R = (Double.valueOf(earthRadius.getText())).doubleValue();
        orbit.writeToCanvas(buttonCanvas.getGraphics());
}

public void  actionPerformed(ActionEvent event)
{
        if (event.getActionCommand().equals("draw graph"))
        {
                getNewValues();
        }
        else if (event.getActionCommand().equals("draw Orbit"))
        {
                getNewValues();
                SatOrbit p = new SatOrbit("Satellite Orbit");
                p.initialise(i, Mo, e, R, dt, w, to, T, AsN);
                p.go();
        }
        else if (event.getActionCommand().equals("view in column format"))
        {
                getNewValues();
                DisplayArea p = new DisplayArea("Display Kepler orbit results");
                p.initialise(i, Mo, e, R, dt, w, to, T, AsN, true, true);
                p.go();
        }
}

public void writeToCanvas(Graphics g)
{
        g.setFont(new Font("Monospaced", Font.ITALIC, 24));
```

Stellenbosch University http://scholar.sun.ac.za

```
                g.drawString(" Satellite-Orbit positioning simulation", 30, 50);
                g.drawString("Designer: Lungile L. Grungxu", 100, 100);
                g.drawString("Stellenbosch", 160, 150);
        }

        public static void main(String[]args)
        {
                orbit.writeToCanvas(buttonCanvas.getGraphics());
                orbit.getInitValues();
                orbit.writeToCanvas(buttonCanvas.getGraphics());
                orbit.addWindowListener(new WindowAdapter(){
                        public void windowClosing(WindowEvent e){
                                System.exit(0);
                        }
                });
                orbit.writeToCanvas(buttonCanvas.getGraphics());
        }
}
```

A.11 Orbit class

This code contains the methods for drawing the circle, drawing orbit, calculating longitude, latitude and so on

```
import java.awt.*;

public class Orbit{//draws the orbit of the satellite.
        int x1, y1, x2, y2;


        public void drawCircles(Graphics g, int x, int y, double length)
        {
                g.drawOval(x, y, (int)length, (int)length);
                g.drawLine(x+(int)(length/2), y, x+(int)(length/2), y+(int)(length));
                g.drawLine(x, y+(int)(length/2), x+(int)(length), y+(int)(length/2));
        }

        public void drawAscendingNode(Graphics g, int x, int y, double length,
                                                        double AscendingNode_angle)
        {
                g.drawLine(x+(int)(length/2), y+(int)(length/2),
                        x+(int)length/2+Math.abs((int)(length/2*Math.cos(
                                AscendingNode_angle*Math.PI/180))),
                        y+(int)length/2+Math.abs((int)(length/2*Math.sin(
                                AscendingNode_angle*Math.PI/180))));
```

Stellenbosch University http://scholar.sun.ac.za

```
}


public void drawEarth(Graphics g,int x, int y, double r, double R, double b, double e,
double a)
{
        //This draws the earth.
        g.drawOval(x+(int)(a*(1-e)-R), y+(int)(b-R), (int)(2*R), (int)(2*R));
}

public void drawEarth2(Graphics g, int x, int y, double b, double ra, double R)
{
        //This draws the earth.
        g.drawOval(x+(int)(ra-R), y+(int)(b-R), (int)(2*R), (int)(2*R));
}


public void drawOrbit(Graphics g, int x, int y, double a, double b)
{
        //This draws the elliptical orbit.
        g.drawOval(x, y, 2*(int)a, 2*(int)b);
}

public void drawRadialDistance(Graphics g,double a, int x, int y, double r,
double b,double e, double anom)
{
        /* This draws the radial distance from the centre of the earth to the
          satellite. i.e  it draws r=R+h.*/
        int x1 = x+(int)(a*(1-e));
        int y1 = y+(int)b;

        double xr = r*Math.cos(anom*Math.PI/180);
        double yr = r*Math.sin(anom*Math.PI/180);

        int x2 = x+(int)(xr+(a*(1-e)));
        int y2 = y+(int)(b+yr);
        g.drawLine(x1, y1, x2, y2);
}


public void rD(Graphics g, double ra, int x, int y, double r,
double b, double trueAnomally)
{
        /* This draws the radial distance from the centre of the earth to the
          satellite. i.e  it draws r=R+h.*/
        x1 = x+(int)ra;
```

Stellenbosch University http://scholar.sun.ac.za

```
                y1 = y+(int)b;
                x2 = x1+(int)(r*Math.cos(-trueAnomally*Math.PI/180));
                y2 = y1+(int)(r*Math.sin(-trueAnomally*Math.PI/180));
                g.drawLine(x1, y1, x2, y2);
        }

        public double longitude(double r, double w, double u, double AsN, double i)
        {
                double x=r*(Math.cos(w+u)*Math.cos(AsN)-Math.sin(w+u)*
                                Math.sin(AsN)*Math.cos(i));
                double y=r*(Math.cos(w+u)*Math.sin(AsN)+Math.sin(w+u)*
                                Math.cos(AsN)*Math.cos(i));
                double longtude = Math.atan(y/x);
                return longtude;
        }

        public double latitude(double r, double w, double u, double i)
        {
                double z = r*(Math.sin(w+u)*Math.sin(i));
                double latitudes = Math.asin(z/r);
                return latitudes;
        }
}
```

A.12 Orientation class

This code draws a frame with three textfields for the display of yaw, pitch and roll values.

```
import javagently.*;
import java.io.*;
import java.awt.*;
import java.awt.event.*;

public class Orientation extends Frame
{

        //*****************          Declaration area          *****************
         public  static TextField Yaw = new TextField(12);
         public  static TextField Pitch = new TextField(12);
         public  static TextField Roll = new TextField(12);


         private  Label L1 = new Label("Yaw");
         private  Label L2 = new Label("Pitch");
         private  Label L3 = new Label("Raw");
```

Stellenbosch University http://scholar.sun.ac.za

```
 private Panel LabelPanel = new Panel();
 private Canvas SatCanvas = new Canvas();
Receive Serial = new Receive();
//**************     Declaration area ends          *****************


 public Orientation(String s)
 {
        super(s);
        setTitle("Satellite Orientation");
        setSize(400,200);
        setBackground(Color.green);

        setLayout(new FlowLayout());
        LabelPanel.setLayout(new GridLayout(2, 8, 10, 10));
        LabelPanel.setBackground(Color.green);
        LabelPanel.setSize(340, 20);
        LabelPanel.add(L1);
        LabelPanel.add(L2);
        LabelPanel.add(L3);
        LabelPanel.add(Yaw);
        LabelPanel.add(Pitch);
        LabelPanel.add(Roll);

        SatCanvas.setBackground(Color.yellow);
        SatCanvas.setSize(340, 100);

        add(LabelPanel);
        add(SatCanvas);
        setVisible(true);

        addWindowListener(new WindowAdapter(){
               public void windowClosing(WindowEvent e){
                      e.getWindow().setVisible(false);
                      dispose();
                      Serial.stop();
               }
        });
 }

 public void go()
 {
        Serial.start();
 }
}
```

Stellenbosch University http://scholar.sun.ac.za

A.13 SatelliteOrbit class

This class calls on the method to initialize the data variables and then activates the NewtonRaphson algorithm to calculate eccentric anomaly.

```
import javagently.*;
import java.io.*;
import java.awt.*;
import java.awt.event.*;

public class SatelliteOrbit extends Thread
{

        //***************     Declaration area      ***********************
        public int x, y, t, dt;
        private String data = new String();
        public static String Data = new String();
        public static boolean TxFlag = false;
        public static boolean RxFlag = false, dispFlag=true;
        public static boolean PositionFlag=false;
        private boolean flag1, flag2; Text disp;
        private double T, i, n, r=0, temp, C;
        private double w, Mo, to , e , u, M;
        private double mu; //gravitationalParameter
        private double a, R, longitude, latitude, height, b, xx;
        private double AsN;// AsN is the longitude of ascending node
        //************        Declaration area ends        *******************


        public SatelliteOrbit()
        {
                x=60;
                y=30;
                longitude=0.0;
                height = 0.0;
                mu = 398600.0;
                temp = 0;
                C = 2.0/3.0;
                Mo = Mo*Math.PI/180;
        }

        public void initialise(double i, double Mo, double e, double R, int dt, double w,
                        double to, double T, double AsN, boolean flag1, boolean flag2)
        {
                this.i = i;
                this.Mo = Mo;
```

Stellenbosch University http://scholar.sun.ac.za

```
        this.e = e;
        this.R = R;
        this.w = w;
        this.dt = dt;
        this.to = to;
        this.T = T;
        this.AsN = AsN;
        this.flag1 = flag1;
        this.flag2 = flag2;
}


public double rDist(int t, double n, double aa)
{
        //calculates the radial distance between earth and satellite.
        u = trueAnom(t, e, n);
        double rd = aa*(1 - e*e)/(1 + e*Math.cos(u));
        return rd;
}


public double trueAnom(int t, double e, double n)
{       //calculates true anomaly angle.
        M = Mo + n*(t - to);
        //double ub = 6*Math.PI;

        NewtonRaphsonMethod eA = new NewtonRaphsonMethod();
        //instantiates the object of type NewtonRaphsonMethod and eA is a handle
        //in the stack pointing to the NewtonRaphsonMethod object located at the
        //heap.

        double E = eA.eccentricAnomally(M, e);
        //the handle eA is used to call the method eccentricAnomally(M, e)

        double u = Math.acos((Math.cos(E)-e)/(1-e*Math.cos(E)));
        int count = 0;
        while (true){
                if (count==0)
                        if(M <= Math.PI)break;
                count = count+2;
                if ((M >= (count-1)*Math.PI)&&(M < count*Math.PI)){
                        u = count*Math.PI - u;
                        count = 0;
                        break;
                }
```

Stellenbosch University http://scholar.sun.ac.za

```
                        if ((M >= count*Math.PI)&&(M < (count+1)*Math.PI)){
                                u = count*Math.PI + u;
                                count = 0;
                                break;
                        }
                        if (M==count*Math.PI){
                                count = 0;
                                break;
                        }
                }
                if(dispFlag)
                        if ((flag1==false)&&(flag2==false))
                                SatOrbit.displayTrueAn.setText(String.valueOf(

Text.format(u*180/Math.PI,6,4)));
                return u;
}


public double Angle(int t, double n)
{
                //adds true anomally and the argument of perigee together.
                u = trueAnom(t, e, n);
                double angle = w + u*180/Math.PI;
                return angle;
}


public void doCalculations()
{               //does calculations
                a = 331.24915*Math.pow(T, C);
                t = (int)to;
                n = Math.sqrt(mu/(a*a*a));
                xx = a/170.0;
                a = a/xx;
                b = a*Math.sqrt(1-e*e);
}

//************ Graphics method    *******************
public void displayOrbit(Graphics g)
{                       //displays the earth and the orbit.
                        R = R/100;
                        SatOrbit.orbit.drawEarth(g, x, y, r, R, b, e, a);
                        SatOrbit.orbit.drawOrbit(g, x, y, a, b);
                        R = R*100;
}
```

Stellenbosch University http://scholar.sun.ac.za

```
public void displayRadialDistances(Graphics g)
{       //draws radial distances in the orbit.
        int x1 = 10, y1 = 100;
        double length = 150.0;
        Graphics graphics = SatOrbit.AsNcanvas.getGraphics();
        if ((flag1==false)&&(flag2==false))
        {
                graphics.setColor(Color.blue);
        }
        else
        {
        DisplayArea.area.append("time   "+ "(sec)" +  "      " +"radial dist  " +
                                "(km) " +  "          " +"height   "    + "(km) " +
                                "    " + "Longitude   "  + "(deg)" +  "     " +
                                "Latitude    "  + "(deg)" +  "       " +
                                "Ascend_Node  "  + "(deg)" +  "    " +
                                "Perigee angle " + "(deg)" + "\n\n");
        }
        while(true)
        {

                //if(dispFlag)
                        if ((flag1==false)&&(flag2==false))
                        {
                                graphics.drawString("Angle of the Ascending Node",
                                                                        10, 80);
                                displayOrbit(g);
                                graphics.setColor(Color.green);
                                SatOrbit.orbit.drawAscendingNode(graphics, x1, y1,
                                                                length, AsN);
                                graphics.setColor(Color.blue);
                                SatOrbit.orbit.drawCircles(graphics, x1, y1, length);
                        }
                a = a*xx;
                r = rDist(t, n, a);
                r = r/xx;
                a = a/xx;
                if (dispFlag)
                {
                        if ((flag1==false)&&(flag2==false))
                               SatOrbit.orbit.drawRadialDistance(
                                       g, a, x, y, r, b, e, Angle(t, n)+ 180-w);
                }
                r = r*xx;
                height = r - R; // height above the earth.
```

Stellenbosch University http://scholar.sun.ac.za

```
longitude = (SatOrbit.orbit.longitude(r, w*Math.PI/180, u,
        AsN*Math.PI/180, i*Math.PI/180) + 1.57)*180/Math.PI;

if (longitude > 360)
                longitude = longitude - 360;

if (longitude > temp)
                temp = longitude;

if (longitude < temp){
                longitude = longitude + temp;
                if (longitude >= 6*180/Math.PI)
                {
                        temp = 0;
                }
}
longitude = longitude - 360/(24*60*60)*dt;
latitude = SatOrbit.orbit.latitude(r, w*Math.PI/180, u,
                                i*Math.PI/180)*180/Math.PI;
if (dispFlag)
{
        if ((flag1==false)&&(flag2==false))
        {
                SatOrbit.displayHeight.setText(
                        String.valueOf(Text.format(height,6,4)));
                SatOrbit.displayRadialDist.setText(
                        String.valueOf(Text.format(r,6,4)));
                SatOrbit.displayLongitude.setText(
                        String.valueOf(Text.format(longitude,6,4)));
                SatOrbit.displayLatitude.setText(
                        String.valueOf(Text.format(latitude,6,4)));
                SatOrbit.displayTime.setText(String.valueOf(t));
        }
}

//--------------Orbit perturbations due to sun and moon------------------
-

double no_rev = 24*60/T;

double dw_moon = 0.00169*(4-5*Math.sin(i)*Math.sin(i))/no_rev;
double dw_sun = 0.00077*(4-5*Math.sin(i)*Math.sin(i))/no_rev;

double dAsN_moon = -0.00338*Math.cos(i)/no_rev;
double dAsN_sun = -0.00154*Math.cos(i)/no_rev;

w = w + (dw_moon + dw_sun)*dt;
```

Stellenbosch University http://scholar.sun.ac.za

```
                AsN = AsN + (dAsN_moon + dAsN_sun)*dt;
                if (dispFlag)
                {
                        dispFlag=false;
                        if ((flag1==false)&&(flag2==false))
                        {

                                SatOrbit.AscendingNode.setText(
                                        String.valueOf(Text.format(AsN, 6, 4)));
                                SatOrbit.PerigeeAngle.setText(
                                        String.valueOf(Text.format(w, 6, 4)));
                                SatOrbit.orbit.drawAscendingNode(graphics, x1, y1,
                                        length, AsN);
                        }
                        //-----------------        perturbations          ------------------
                        else
                        {

                                DisplayArea.area.append(String.valueOf(t)
                                        +"              "+ Text.format(r,6,4)
                                        +"               "+ Text.format(height,6,4)
                                        +"              "+ Text.format(longitude,6,4)
                                        +"               "+ Text.format(latitude,6,4)
                                        +"               "+ Text.format(AsN, 6, 4)
                                        +"                "+ Text.format(w, 6, 4)
                                        +"\n\n");
                                if(Angle(t, n) > 360) break;
                        }
                        if (PositionFlag==false)
                        {
                                Send.TxHeight = Text.format(height,6,4)
                                                +new String();
                                Send.TxLongitude = Text.format(longitude,6,4)
                                                +new String();
                                Send.TxLatitude = Text.format(latitude,6,4)
                                                +new String();
                                PositionFlag=true;
                        }
                        t = t + dt;        // time increment.
                        delay(100);
                }
        }
        stop();
}

public static void delay(int k)
```

Stellenbosch University http://scholar.sun.ac.za

```
        {
                //delays for k sec
                try
                {
                        Thread.currentThread().sleep(k);
                }
                catch(InterruptedException e){}
        }


        public void drawGraphics(Graphics g)
        {       //calls all the graphics methods in this class.
                doCalculations();
                displayRadialDistances(g);
        }
        //**************      End of Graphics methods      *************


        public void run()
        {
                Graphics g = SatOrbit.OrbitCanvas.getGraphics();
                if ((flag1==false)&(flag2==false))
                {
                        drawGraphics(g);
                }
                else
                {
                        doCalculations();
                        displayRadialDistances(g);
                }
        }
}
```

A.14 SatOrbit CLASS

The SatOrbit class below draws a frame with textfields for displaying the height, longitude, latitude, radial distance, angle of the ascending node and time.

```
import javagently.*;
import java.io.*;
import java.awt.*;
import java.awt.event.*;

public class SatOrbit extends Frame
{
```

Stellenbosch University http://scholar.sun.ac.za

```
//*****************        Declaration area        ***************
public static int dt;
static double  T, i;
static double  w, Mo, to , e ;
static double  R;
public static double  AsN;    // AsN is the longitude of ascending node


public  static TextField displayTime = new TextField(12);
public  static TextField displayTrueAn = new TextField(12);
public  static TextField displayRadialDist = new TextField(12);
public  static TextField displayHeight = new TextField(12);
public  static TextField displayLongitude = new TextField(12);
public  static TextField displayLatitude = new TextField(12);


private  Label L1 = new Label("time(sec)");
private  Label L2 = new Label("true Anomally angle(deg)");
private  Label L3 = new Label("Radial distance(km)");
private  Label L4 = new Label("height (km)");
private  Label L5 = new Label("longitude (deg)");
private  Label L6 = new Label("latitude (deg)");

static     Orbit orbit  = new Orbit();
public  static Canvas OrbitCanvas = new Canvas();
private  Panel DataPanel = new Panel();

public static TextField AscendingNode = new TextField(12);
public static TextField PerigeeAngle  = new TextField(12);

private Label La = new Label("Ascending Node Angle(deg)");
private Label Lp = new Label("Angle of Perigee(deg)");

private Panel PerturbationPanel = new Panel();
private Panel PertPanel = new Panel();

private Panel CanvasPanel = new Panel();
public static Canvas AsNcanvas = new Canvas();

private static GridBagLayout gridbag = new GridBagLayout();
private static GridBagConstraints c = new GridBagConstraints();
final SatelliteOrbit p = new SatelliteOrbit();
//**************  Declaration area ends        ****************


public SatOrbit(String s)
```

Stellenbosch University http://scholar.sun.ac.za

```
{
        super(s);
        setTitle("Satellite Orbit");
        setSize(800,600);
        setBackground(Color.orange);

        setLayout(new FlowLayout());
        PerturbationPanel.setLayout(new GridLayout(2, 2, 40, 10));
        PerturbationPanel.setBackground(Color.orange);
        PerturbationPanel.setSize(300, 40);

        PerturbationPanel.add(La);
        PerturbationPanel.add(Lp);
        PerturbationPanel.add(AscendingNode);
        PerturbationPanel.add(PerigeeAngle);

        setLayout(null);
        PertPanel.setBackground(Color.orange);
        PertPanel.setSize(800, 40);
        PertPanel.add(PerturbationPanel);



        setLayout(new FlowLayout());
        OrbitCanvas.setBackground(Color.cyan);
        OrbitCanvas.setSize(600, 380);

        setLayout(new FlowLayout());
        AsNcanvas.setBackground(Color.green);
        AsNcanvas.setSize(200, 380);

        setLayout(new FlowLayout());
        CanvasPanel.setLayout(new GridBagLayout());
        CanvasPanel.setBackground(Color.orange);
        CanvasPanel.setSize(800, 380);

        c.gridwidth = 3;                //reset to the default
        c.gridheight = 1;
        gridbag.setConstraints(OrbitCanvas, c);
        CanvasPanel.add(OrbitCanvas);
        CanvasPanel.add(AsNcanvas);


        setLayout(new FlowLayout());
        DataPanel.setLayout(new GridLayout(4, 12, 40, 2));
        DataPanel.setBackground(Color.orange);
```

Stellenbosch University http://scholar.sun.ac.za

```
        DataPanel.setSize(400, 140);

        DataPanel.add(L1);
        DataPanel.add(L2);
        DataPanel.add(L3);
        DataPanel.add(displayTime);
        DataPanel.add(displayTrueAn);
        DataPanel.add(displayRadialDist);
        DataPanel.add(L4);
        DataPanel.add(L5);
        DataPanel.add(L6);
        DataPanel.add(displayHeight);
        DataPanel.add(displayLongitude);
        DataPanel.add(displayLatitude);

        add(PertPanel);
        add(CanvasPanel);
        add(DataPanel);
        setVisible(true);

        addWindowListener(new WindowAdapter(){
                public void windowClosing(WindowEvent e){
                        p.stop();
                        e.getWindow().setVisible(false);
                        dispose();
                }
        });
}

public void initialise(double i, double Mo, double e, double R, int dt,
                        double w, double to, double T, double AsN)
{
   this.i = i;
   this.Mo = Mo;
   this.e = e;
   this.R = R;
   this.w = w;
   this.dt = dt;
   this.to = to;
   this.T = T;
   this.AsN = AsN;
}

public void go()
{
        p.initialise(i, Mo, e, R, dt, w, to, T, AsN, false, false);
```

Stellenbosch University http://scholar.sun.ac.za

```
                Orientation Orr = new Orientation("Satellite Orientation");
                Orr.go();
                p.start();
        }
}
```

A.15 Send Thread for the propagator

This Send thread is used to send the positioning parameter through the serial port. Each parameter is separated with a character, "#", from the other data parameters. The data packet also goes with a character that indicates the end of a packet.

```
import java.io.*;
import javax.comm.*;

public class Send extends Thread{

        static OutputStream outputStream;
        static SerialPort serialPort;

        public static String TxHeight, TxLongitude, TxLatitude;
        public static boolean PositionFlag = false;


        public Send(SerialPort serialPort)
        {
                //constructor
                this.serialPort = serialPort;
        }

        public void run()
        {
                try
                {
                   outputStream = serialPort.getOutputStream();
                }
                catch (IOException e) {}
                try
                {
                     while (true)
```

Stellenbosch University http://scholar.sun.ac.za

```
{
            if (Receive.TxAck==true)
            {
                outputStream.write((byte)'@');//send ack
                Receive.TxAck=false;
            }

            if (Receive.StartPacket==true)
            {
                outputStream.write((byte)'y');
                Receive.StartPacket=false;
            }

            if (Receive.EndPacketAck==true)
            {
                outputStream.write((byte)'%');
                Receive.EndPacketAck=false;
            }

            boolean TxFlag = Receive.TxFlag;
            if (TxFlag == true)
            {
                String Yaw = Receive.YAW,
                 Pitch = Receive.PITCH,
                 Roll = Receive.ROLL;

                while (PositionFlag = false)
                {
                    PositionFlag =
                        SatelliteOrbit.PositionFlag;
                    sleep(5);
                }

                System.out.print("s");

                while (true)
                {
                    outputStream.write((byte)'s');
                    sleep(10);
                    if (Receive.AckStart==true)
                    {
                        break;
                    }
                }
                Receive.AckStart=false;
```

Stellenbosch University http://scholar.sun.ac.za

```
for (int i=0; i<Yaw.length(); i++)
{
        outputStream.write(
                (byte)Yaw.charAt(i));
        System.out.print(Yaw.charAt(i));
        sleep(20);
}

System.out.print("#");
while (true)
{
        outputStream.write((byte)'#');
        sleep(60);
        if (Receive.Ack==true)break;
}
Receive.Ack=false;


for (int i=0; i<Pitch.length(); i++)
{
        outputStream.write(
                (byte)Pitch.charAt(i));
        System.out.print(Pitch.charAt(i));
        sleep(20);
}

System.out.print("#");
while (true)
{
        outputStream.write((byte)'#');
        sleep(60);
        if (Receive.Ack==true)break;
}
Receive.Ack=false;

for (int i=0; i<Roll.length(); i++)
{
        outputStream.write(
                (byte)Roll.charAt(i));
        System.out.print(Roll.charAt(i));
        sleep(20);
}

System.out.print("#");
while (true)
{
```

Stellenbosch University http://scholar.sun.ac.za

```
        outputStream.write((byte)'#');
        sleep(60);
        if (Receive.Ack==true)break;
}
Receive.Ack=false;

for (int i=0; i<TxHeight.length(); i++)
{
        outputStream.write(
                (byte)TxHeight.charAt(i));
        System.out.print(TxHeight.charAt(i));
        sleep(20);
}

System.out.print("#");
while (true)
{
        outputStream.write((byte)'#');
        sleep(60);
        if (Receive.Ack==true)break;
}
Receive.Ack=false;

for (int i=0; i<TxLongitude.length(); i++)
{
        outputStream.write(
                (byte)TxLongitude.charAt(i));
        System.out.print(
                TxLongitude.charAt(i));
        sleep(20);
}

System.out.print("#");
while (true)
{
        outputStream.write((byte)'#');
        sleep(60);
        if (Receive.Ack==true)break;
}
Receive.Ack=false;

for (int i=0; i<TxLatitude.length(); i++)
{
        outputStream.write(
                (byte)TxLatitude.charAt(i));
        System.out.print(TxLatitude.charAt(i));
```

Stellenbosch University http://scholar.sun.ac.za

```
                                sleep(20);
                        }

                        System.out.print("&");
                        while (true)
                        {
                                outputStream.write((byte)'&');
                                sleep(20);
                                if (Receive.AckEndPacket==true)break;
                        }

                        Receive.AckEndPacket=false;
                        System.out.println("\n\n");

                        Receive.Yaw = new char[20];
                        Receive.Pitch = new char[20];
                        Receive.Roll = new char[20];

                        SatelliteOrbit.PositionFlag = false;
                        Receive.TxFlag = false;
                }
                else
                        sleep(2);
            }
        }
        catch (IOException e) {}
        catch(InterruptedException e){}
    }
}
```

A.16 Receive Thread for the propagator

This thread receives all the data and strips it into pieces according to the separators. It listens to the serial port and captures the data as it comes. The data to be received is the yaw, pitch and roll. The following code is the implementation of the thread.

```
import javagently.*;
import java.io.*;
import java.util.*;
import javax.comm.*;

public class Receive extends Thread implements SerialPortEventListener
{
    static CommPortIdentifier portId;
    static Enumeration portList;
```

Stellenbosch University http://scholar.sun.ac.za

```
InputStream inputStream;
SerialPort serialPort;

public static  char  Yaw[] = new char[20],
                     Pitch[] = new char[20],
                     Roll[] = new char[20];

public static String YAW = new String(),
                     PITCH = new String(),
                     ROLL = new String();

int count=0, AckCount = 1; String Char = new String();

private boolean RollF=false, YawF=false, PitchF=false;
static boolean TxFlag = false, StartDisplay = false;
public static boolean Ack = false, TxAck = false, AckStart=false,
              EndPacketAck = false, AckEndPacket=false,
              StartPacket = false,  DisplayFlag = false;
private String string = new String();


public void run()
{
     try
     {
          sleep(1);
     }
     catch(InterruptedException e){}
}

public Receive()
{
        portList = CommPortIdentifier.getPortIdentifiers();
        while (portList.hasMoreElements())
        {
             portId = (CommPortIdentifier) portList.nextElement();
             if (portId.getPortType() == CommPortIdentifier.PORT_SERIAL)
             {
                    if (portId.getName().equals("COM1"))
                    {
                           System.out.println("Connected through "+portId.getName());
                           break;
                    }
             }
        }
```

Stellenbosch University http://scholar.sun.ac.za

```
        try
        {
              serialPort = (SerialPort) portId.open("SimpleReadApp", 2000);
         }
        catch (PortInUseException e) {}

        try
        {
              inputStream = serialPort.getInputStream();}
              catch (IOException e) {}

              try
              {
                      serialPort.addEventListener(this);
              }
              catch (TooManyListenersException e) {}

              serialPort.notifyOnDataAvailable(true);
              try
              {
                      serialPort.setSerialPortParams(9600,
                              SerialPort.DATABITS_8,
                              SerialPort.STOPBITS_1,
                              SerialPort.PARITY_NONE);
   }
              catch (UnsupportedCommOperationException e) {}
              Send Tx = new Send(serialPort);
              Tx.start();
}

public void serialEvent(SerialPortEvent event) {
   switch(event.getEventType()) {
   case SerialPortEvent.BI:
   case SerialPortEvent.OE:
   case SerialPortEvent.FE:
   case SerialPortEvent.PE:
              case SerialPortEvent.CD:
   case SerialPortEvent.CTS:
   case SerialPortEvent.DSR:
   case SerialPortEvent.RI:
   case SerialPortEvent.OUTPUT_BUFFER_EMPTY:
                      break;
   case SerialPortEvent.DATA_AVAILABLE:
                      byte[] readBuffer = new byte[20];
```

Stellenbosch University http://scholar.sun.ac.za

```
//*********************************************
try {

        while (inputStream.available() > 0) {
                int numBytes = inputStream.read(readBuffer);
                char Character = (new String(readBuffer)).charAt(0);

                if((Character!='#')&&(Character!='&')&&
                        (Character!='@')&&(Character!='%')&&
                        (Character!='y')&&(Character!='s'))
                {
                        if (YawF==false)
                        {
                                Yaw[count]=Character;
                                count++;
                        }
                        else if (PitchF==false)
                        {
                                Pitch[count]=Character;
                                count++;
                                AckCount=2;
                        }
                        else
                        {
                                Roll[count]=Character;
                                count++;
                        }
                }

                else if (Character=='@')
                {
                        Ack=true;
                }

                else if (Character=='#')        //if separator

                {
                        if (AckCount==1)
                        {
                                if (YawF==false)
                                        YawF=true;
                        }

                        if (AckCount==2)
                        {
                                if (PitchF==false)
```

Stellenbosch University http://scholar.sun.ac.za

```
                                        PitchF=true;
                        }
                        TxAck = true;
                        count = 0;
                }

                else if (Character=='%')
                {
                        AckEndPacket=true;
                        AckCount=1;
                }

                else if (Character=='y')
                {
                        AckStart=true;
                }

                else if (Character=='&') //if end of each packet.
                {
                        EndPacketAck = true;
                        DisplayFlag = true;
                }

                else if (Character=='s')
                {
                        if (DisplayFlag==false)
                        {
                                StartPacket=true;
                        }

                        else
                        {
                                DisplayFlag = false;
                        }
                }

                if (DisplayFlag == true)
                {
                                DisplayFlag = false;
                                System.out.println("\n");
//****** search and remove the 2nd/3rd/etc decimal point *******
                                int y=0, p=0, r=0; //initialize counters
                                for(int i=0; i<=17; i++)
                                {
                                 if (Yaw[i]=='.')
                                  {
```

Stellenbosch University http://scholar.sun.ac.za

```
                    y++;
                    if (y>=2)
                    {
                            Yaw[i] =Char.charAt(0);
                            y=1;
                    }
            }
            if (Pitch[i]=='.')
            {
                    p++;
                    if (p>=2)
                    {
                            Pitch[i]=Char.charAt(0);
                            p=1;
                    }
            }
            if (Roll[i]=='.')
            {
                    r++;
                    if (r>=2)
                    {
                            Roll[i]= Char.charAt(0);
                            r=1;
                    }
            }
    }

    y=0; p=0; r=0;

//***********  change character arrays to strings  ************
    YAW = (new String(Yaw)).trim();
    PITCH = (new String(Pitch)).trim();
    ROLL = (new String(Roll)).trim();


    try
    {
            Orientation.Yaw.setText(
                    String.valueOf(
                    Text.format(
                    new Double(
                    YAW).doubleValue(),6,
                                    4)));
            System.out.println("\nYaw:
                                    "+YAW);
```

Stellenbosch University http://scholar.sun.ac.za

```
}
catch(NumberFormatException e)
{
        YAW = "0.00";
        Orientation.Yaw.setText(
        String.valueOf(Text.format(
        new Double(
        YAW).doubleValue(),6,4)));
        System.out.println("\nYaw:
                        "+YAW);
}

try
{
        Orientation.Pitch.setText(
        String.valueOf(Text.format(
                New Double(PITCH).
                   doubleValue(),6,4)));
        System.out.println("Pitch:
                        "+PITC);
}
catch(NumberFormatException e)
{
        PITCH="0.00";
        Orientation.Pitch.setText(
        String.valueOf(Text.format(
        new Double(
        PITCH).doubleValue(),6,4)));
        System.out.println("Pitch:
                    "+PITCH);
}

try
{
        Orientation.Roll.setText(
        String.valueOf(Text.format(
        new Double(
        ROLL).doubleValue(),6,4)));
        System.out.println("Roll:
                    "+ROLL+"\n");
}
catch(NumberFormatException e)
{
        ROLL="0.00";
        Orientation.Roll.setText(
        String.valueOf(Text.format(
```

Stellenbosch University http://scholar.sun.ac.za

```
                                                          new Double(
                                                          ROLL).doubleValue(),6,4)));
                                                          System.out.println("Roll:
                                                                   "+ROLL+"\n");
                                                    }



                                                    YawF=false;
                                                    PitchF=false;
                                                    RollF=false;
                                                    SatelliteOrbit.dispFlag=true;

                                                    TxFlag=true;   //end of packet flag.
                                              }
                                  }
                        }
                        catch (IOException e) {}
                        break;
            }
      }
}
```