DEVELOPMENT OF REAL-TIME ORBITAL PROPAGATOR SOFTWARE FOR A CUBESAT'S ON-BOARD COMPUTER

by

THINAWANGA TSHILANDE

Thesis submitted in partial fulfillment of the requirements for the degree

Master of Technology: Electrical Engineering

in the Faculty of Engineering

at the Cape Peninsula University of Technology

Supervisor: Prof. B.D.L. Opperman
Co-supervisor: Prof. R.R. van Zyl

Bellville
February 2015

CPUT copyright information

The thesis may not be published either in part (in scholarly, scientific or technical journals), or as a whole (as a monograph), unless permission has been obtained from the University.

DECLARATION

I, Thinawanga Tshilande, declare that the contents of this thesis represent my own unaided work, and that the thesis has not previously been submitted for academic examination towards any qualification. Furthermore, it represents my own opinions and not necessarily those of the Cape Peninsula University of Technology.

Signed	**Date**

ABSTRACT

A precise orbit propagator was developed for implementing on a CubeSat's on-board computer for real-time orbital position and velocity determination and prediction. Knowledge of the accurate orbital position and velocity of a Low Earth Orbit (LEO) Cubesat is required for various applications such as antenna and imager pointing. Satellite motion is governed by a number of forces other than Earth's gravity alone. The inclusion of perturbation forces such as Earth's aspheric gravity, third body attraction (e.g. Moon and Sun), atmospheric drag and solar radiation pressure, is subsequently required to improve the accuracy of an orbit propagator.

Precise orbit propagation is achieved by numerically integrating a set of coupled second order differential equations derived from satellite's perturbed equations of motion. For the purpose of this study two numerical integrators were selected: RK4 - Fourth order Runge Kutta method and RKF78 - results from embedding RK7 into RK8. The former is a single-step integrator while the latter is a multi-step integrator. These integrators were selected for their stability, high accuracy and computational efficiency.

An orbit propagation software tool is presented in this study. Considering the processing power of Central Processing Unit (CPU) of CubeSat's on-board computer and a trade-off between precision and computational cost, the 10×10 and 20×20 gravity field models, the Exponential atmospheric model and Jacchia 70 static atmospheric model, were implemented. A 60×60 gravity field model is also investigated for reference. For validation purpose the developed software tool results were compared with results from Systems Tool Kit (STK) and Satellite Laser Ranging (SLR) using SUNSAT satellite reference orbit.

Keywords: Orbit, Propagation, RKF78, RK4, Software, Perturbations

ACKNOWLEDGEMENTS

I wish to thank:

- Prof. Ben Opperman for his guidance, support, and for believing in me.
- Prof. Robert van Zyl for giving me an opportunity to be part of the F'SATI group.
- The South African National Space Agency for making me part of their amazing team and allowing me to use their facilities to pursue my research.
- Singo Alpheous for his fatherly support throughout my academic life. May your soul rest in eternal peace.
- Nigussie Giday, Thabang Matladi, Malan Ahoua, Matamba Tshimangadzo, Lifa Mbuli, Vumile Tyalimpi, Amin Mahmud, Nicholas Ssessanga and Electdom Siziba for their inputs, and wonderful support.
- Dr Fhumulani Nemulodi for overseeing the final execution of this document.
- Singo Cynthia, Tshilande Rendani and Christina, Marubini Florah and Divhani for their support and tender parental care.
- Everyone who made a contribution to my work, be it guidance, academic or financial support.

The financial assistance of the National Research Foundation towards this research is acknowledged. Opinions expressed in this thesis and the conclusions arrived at, are those of the author, and are not necessarily to be attributed to the National Research Foundation.

Dedicated

to

Mampofu Catherine Singo

and
Tshinakaho Betty Tshilande

Contents

List of Figures

List of Tables

Glossary

Terms/Acronyms/ Abbreviations	**Definition/Explanation**
ADCS	**A**ttitude and **D**etermination **C**ontrol **S**ystem
ASCII	**A**merican **S**tandard **C**ode for **I**nformation **I**nterchange
AU	**A**stronomical **U**nit
C&DH	**C**ommand and **D**ata **H**andling
COSPAR	**CO**mmittee on **SPA**ce **R**esearch
COTS	**C**ommercial-**O**ff-**T**he-**S**helf
CPU	**C**entral **P**roccessing **U**nit
CPUT	**C**ape **P**eninsula **U**niversity of **T**echnology
CubeSat	**Cube Sat**ellite
DRAO	**D**ominion **R**adio **A**strophysical **O**bservatory
ECEF	**E**arth **C**entred **E**arth **F**ixed
ECI	**E**arth **C**entred **I**nertial
EGM96	1996 **E**arth **G**ravity **M**odel
FSATI	**F**rench **S**outh **A**frican **I**nstitute of **T**echnology
GEO	**G**eosynchronous **E**arth **O**rbit
GPS	**G**lobal **P**ositioning **System**
GAST	**G**reenwich **A**pparent **S**idereal **T**ime
GCC	**G**NU **C**ompiler **C**ollection
GMST	**G**reenwich **M**ean **S**idereal **T**ime
HPOP	**H**igh **P**resicion **O**rbit **P**ropagator
IC	**I**ntegrated **C**ircuit

IDE	**I**ntegrated **D**evelopment **E**nvironment
INPE	**I**nstituto **N**acional de **P**esquisas **E**spaciais
ISGI	**I**nternational **S**ervice of **G**eomagnetic **I**ndices
JD	**J**ulian **D**ate
LEO	**L**ow **E**arth **O**rbit
MEO	**M**edium **E**arth **O**rbit
PCB	**P**rinted **C**ircuit **B**oard
PRECurSOr	**PREC**ise **S**atellite **O**rbit Propagato**r**
RK	**R**unge-**K**utta
RKF	**R**unge-**K**utta-**F**ehlberg
SANSA	**S**outh **A**frican **N**ational **S**pace **A**gency
SAST	**S**outh **A**frican **S**tandard **T**ime
SLR	**S**atellite **L**aser **R**anging
SPIDR	**S**pace **P**hysics **I**nteractive **D**ata **R**esources
STK	**S**ystems **T**ool **K**it
SUNSAT	**S**tellenbosch **UN**iversity **SAT**ellite
TAI	**I**nternational **A**tomic **T**ime
TT&C	**T**elemetry **T**racking and **C**ommand
US	**U**nited **S**tates
UT	**U**niversal **T**ime

Physical Constants

Earth's gravity potential second degree term	J_2	=	0.0010826269
Eccentricity of Earth orbit	e	=	0.081819221456
Gravitational constant	G	=	$6.672 \times 10^{-11} m^3 kg^{-1} s^{-2}$
Gravitational parameter (Earth)	μ	=	$3.986\ 004\ 415 \times 10^{14}\ m^3 s^{-2}$
Gravitational parameter (Sun)	$\mu_\odot$	=	$1.327\ 124 \times 10^{14}\ m^3 s^{-2}$
Gravitational parameter (Moon)	$\mu_☾$	=	$4.902\ 799 \times 10^{6}\ m^3 s^{-2}$
Radius of Earth	$R_\oplus$	=	$6378136.3\ m$
Rotation rate of Earth	$\omega_\oplus$	=	$4.178\ 074\ 216 \times 10^{-3}\ ^\circ/s$
Solar pressure	P_{SR}	=	$4.51 \times 10^{-6} kg\ m^{-1} s^{-2}$

Chapter 1

Introduction

1.1 Problem Statement

Satellites move at very high speeds when they orbit the Earth. They experience different perturbation forces along their path, some of which pose complex challenges to the development of a precise propagator. Commercial-off-the-shelf (COTS) software packages are capable of delivering precise orbit propagation results, but come at a price. The source code for software packages are however, not available and they cannot be customised. This necessitates the development of in-house software for cases where an orbit propagator for a CubeSat on-board computer (OBC) is needed. The OBC carries out different tasks depending on the mission objectives. The OBC Central Processing Unit (CPU) processing and memory resources must subsequently be utilised wisely. Thus, a fairly precise propagator was developed which uses less complex models for the sake of OBC CPU resources whilst maintaining sufficient accuracy for operational applications.

This study focuses on the development of orbit propagator software, using the C++ programming language, for the OBC of a Low Earth Orbit (LEO) CubeSat. The orbit propagator's accuracy over time is tested, and a trade-off between the computational cost and the complexity of the model is investigated.

1.2 Overview

Astrodynamics, the science that studies the motion of man-made objects in space, developed about 60 years ago. It was preceded by and developed on the basis of Astronomy, which dates back to the time when the first human gazed at the sky. As human curiosity and the urge to boldly go where no one had gone before grew, humans saw the need to invent instruments and objects that would help them understand and study the heavens in detail. This led to the invention of telescopes, sounding balloons, spacecraft and satellites. Scientists began to appreciate the potential of space and are eager to explore it to uncover its mysteries. The first artificial satellite was launched successfully on 4 October 1957 (Sputnik 1).

It is important that the real-time and future positions of satellites be predicted and monitored precisely, such as is required for remote sensing satellites. Since it is not always feasible to precisely track a satellite from the ground, precise orbit determination software, which predict the position of a satellite within a few centimetres of errors, have been developed. In this study we present an orbital propagator for a LEO (Low Earth Orbit) CubeSat satellite which was developed using numerical integration. This was done by modelling all the major forces that affect the propagation of a satellite.

As will be shown, the propagator yields sufficiently good results up to a certain epoch. With the help of an on-board GPS (Global Positioning System) receiver, the initial conditions of this propagator can be re-initialised when the errors observed in the state vector become unacceptably large compared to the real-time state vector. The accuracy of the *PRECurSOr* (Precise Satellite Orbit propagator), the software developed in this study, was tested against precise satellite laser ranging measurements (SLR) of a micro satellite and compared with the High-Precision Orbit Propagator (HPOP), a COTS package of the Systems Tool Kit (STK).

1.3 Research Objectives

The general objective of this work was to develop software to predict the orbital propagation of a LEO CubeSat using the C++ programming language (procedural programming only) and which can be used on a CubeSat OBC. The software runs on a real-time Linux operating system known as PiBang Linux.
The orbit propagator is able to determine the real-time immediate and future state vector of a LEO CubeSat.

1.4 Research Questions

The research addresses the following questions:

- How far in advance and with what accuracy does this software tool predict the satellite's state vector?
- Which factors affect the accuracy of the software tool?
- What is the trade-off between the complexity of the model and its accuracy?

1.5 Research Background

An orbit propagator can successfully predict a satellite's position with great accuracy. Some of the precision propagators that presently exist can predict a satellite's position within a few centimetres of error. Three categories of propagators exist. These are listed below with examples of software for each category:

- Analytic Propagators

  PPT2 (Position and Partials as functions of Time) (Brouwer, 1959) and

  SGP (Simplified General Perturbations) (Kozai, 1962)

- Semi-Analytic Propagators

  SALT (Semi-Analytic Liu Theory) (Liu, 1974) and

  DSST (Draper Semi-Analytical Satellite Theory) (San-Juan *et al.*, 2011)

- Numerical Propagators

  Oblitz (Opperman, 2003) and

  Picard-Chebyshev (Fukushima, 1997)

1.5.1 Analytic Propagators

Analytic propagators approximate the motion of a satellite by solving equations of motion analytically. The future state vector or ephemeris depends on the forces that are modelled. Vallado (1997) discussed three types of simplified analytical techniques:

1.5.1.1 PKEPLER

This technique determines the classical orbital elements, updates the elements during the desired time and then reform the future state vector. It also takes into account the effects of perturbations while updating the orbital elements.

1.5.1.2 Kozai's Method

This technique uses Langrange's Variations of Parameters(VOP) equations. It was the first analytical technique to include the J_2 to J_5 Earth's aspherical effects, but neglected atmospheric drag effects which in turn, limited its accuracy.

1.5.1.3 Brouwer's Method

When it was first proposed in 1959, this technique was not that accurate as the drag effects were not included and it had limited forces. Since then the technique has been improved and is now perceived as a superb analytical theory.

1.5.2 Semi-Analytical Propagators

The semi-analytical technique produces propagators with the best speed and accuracy. Instead of making many approximations like the analytical theories do, it incorporates numerical techniques. The technique is very accurate for long-term studies.

1.5.3 Numerical Propagators

Numerical propagators numerically integrate the equations of motion taking into account all the necessary perturbation effects. Numerical techniques like *Runge-Kutta (RK) methods* are typically employed. The speed of a propagator is sacrificed for the sake of accuracy.

1.6 Methodology

The laws of Kepler and Newton form the basis of this research. By using relevant RK methods to numerically integrate a system of coupled second-order differential equations describing the perturbed acceleration of a satellite in an inertial frame, the orbit propagator software for the OBC of a LEO CubeSat was developed. The C++ programming language was used. A special perturbation technique known as Cowell's method, was used to numerically integrate two first-order differential equations which define a satellite's perturbed motion. The *Classical Runge-Kutta* (RK4) and *Runge-Kutta-Fehlberg7(8)* (RKF78) numerical integration techniques were selected for numerical integration. The RKF78 method combines the seventh-order Runge-Kutta (RK7) and eighth-order Runge-Kutta (RK8) methods taking the solution of RK8 to update the state vector. The perturbed satellite motion can be described by coupled second-order non-linear differential equations as:

$$\begin{aligned} \dot{\mathbf{r}} &= \mathbf{v} \\ \dot{\mathbf{v}} &= -\frac{\mu}{r^3}\mathbf{r} + \mathbf{a}_p \end{aligned} \tag{1.1}$$

where $\mathbf{a}_p$ [1] represents the sum of all the perturbation accelerations. The velocity and positon vectors are represented by the symbols $\mathbf{v}$ and $\mathbf{r}$ respectively. The Earth's gravitational parameter is represented by μ. The forces which define acceleration of a satellite in a perturbed orbit are as follows:

- Aspherical Earth : This perturbation effect results from the Earth not being a perfect spherical body. The gravitational force subsequently varies with geographic location. This effect is modelled with spherical harmonics and gravitational constants.
- Atmospheric Drag : This perturbation effect is caused by the drag that the atmosphere exerts on a satellite and this force is opposite to the direction of satellite movement. To accurately model this force, solar conditions and the geomagnetic state need to be known in order to determine the atmospheric density.
- Third-Body Perturbations : This is caused by bodies other than the Earth exerting gravitational attraction on the satellite. The bodies can be the Sun, planets or other spacecraft. For a LEO satellite, the Earth dominates other bodies and their effects are not significant for short orbit propagation periods.
- Solar Radiation Pressure : Solar radiation exerts pressure on the satellite. This pressure causes a perturbation in the motion of a satellite. Similar to drag, solar cycles and variations must be well understood to accurately model this effect.

1.7 Research Delineation

Very small perturbation forces that affect a satellite's orbit, such as thrust and tides, are not covered by this study. The research also does not cover hyperbolic and parabolic types of satellite trajectories. Medium Earth Orbit (MEO) and Geostationary Earth Orbit (GEO) satellites are also not considered.

[1] See Appendix A for vectors and their representations

1.8 Summary of Chapters

In Chapter 2 the literature that was reviewed for this dissertation are discussed. The third chapter discusses orbit perturbations and their equations of motion. Time, coordinate systems and transformation essential for the development of orbit propagator are covered in the fourth chapter. The fifth chapter discusses considerations taken into account to develop the software and gives relevant background information on OBC. The sixth chapter compares the software's performance with other propagators. Finally, the seventh chapter offers conclusions.

Chapter 2

Literature Review

2.1 Introduction

Johannes Kepler, with the help of Tycho Brahe's data, laid the basic foundation of astrodynamics by his laws of planetary elliptical orbit motion around the Sun. Isaac Newton formalized the laws of astrodynamics. This chapter covers the dawn of the space era, as well as basic laws and equations that govern the motion of bodies in space.

2.2 Satellites History

The word 'satellite' sounds familiar to most people today. It is a subject of interest not only to engineers and scientists, but to everyone because of their application in various areas of our everyday life. Satellite Applications are numerous and include television broadcasting, intercontinental communication services, remote sensing, atmospheric monitoring, space exploration, navigation, spying, etc. Arthur C. Clarke first proposed the concept of geostationary communication satellites in his paper which was published in the *Wireless World* magazine in October 1945 (Clarke, 1945).

Several experimental sounding rockets were launched in 1945-1955 followed by United States (US) and Soviet Union plans to launch artificial satellites. The Soviet Union

initiated the space age and the space race with the US by successfully launching their first artificial satellite, Sputnik-1, on 4 October 1957. Sputnik-2 and Sputnik-3 followed shortly thereafter.

The US then launched Explorer-1 successfully on 31 January 1958. Explorer-1 was the first spacecraft to detect the Van Allen radiation belts (Maini and Agrawal, 2007). Following launch failures of Explorer-1 and Vanguard-1, the US launched Vanguard-1(TV-4) successfully on 17 March 1958. It carried out geodetic studies and revealed that the Earth was pear-shaped (Maini and Agrawal, 2007).

As the "Space Race" escalated, the two countries extended the use of satellites to areas such as communication, weather forecasting, navigation, spying, etc. The first weather satellite, TIROS-1, was launched on 1 April 1960. TIROS-1 was the first satellite to provide pictures of the Earth. The first experimental infrared surveillance satellite MIDAS-2, the first experimental passive communications satellite Echo-1 and the active repeater communications satellite Courier-1B were also launched in 1960. The first true communications satellite, which also happened to be commercially funded, was launched on 10 July 1962 and was dubbed Teslar-1. This was followed a year later by Teslar-2 on 7 May 1963.

The idea of Clarke (1945) became a reality when SYNCOM-2 (Synchronous Communication Satellite) became the first operational communications satellite in geosynchronous orbit. It was followed by launches SYNCOM-3 and another series of communications satellites known as INTELSAT (International Telecommunications Satellite Organization). The Soviets replied to the space race by launching their Molniya series of communications satellites, beginning April 1965. A Molniya orbit is a highly elliptical orbit with an inclination of 63.4°. This inclination minimizes the advance of the argument of perigee due to the even zonal harmonic coefficients of the Earth's gravity field, so that the argument of perigee remains at -90°.

Other countries including European countries, Indonesia, India, China, Saudi Arabia, Brazil, Mexico and Japan followed with the launch of domestic communications satellites. Satellites for other applications were also being developed and launched during the

1960s. This includes satellites for meteorological studies, navigation, surveillance and Earth observation.

CubeSats were introduced in 1999 as a university engineering development platform. The idea was proposed by Prof. Jordi Puig-Suari of CalPoly and Bob Twiggs of Stanford University. These CubeSats are about 10 cm×10 cm×10 cm in size, known as 1-Unit (1U) CubeSats. A 1U CubeSat can be combined to form a 2U and 3U CubeSat (Auret, 2012). The first CubeSats were launched in June 2003 on a Russian commercial spacecraft, Eurockot (Panajaya *et al.*, 2003). Since the introduction of the CubeSat concept, a large number of CubeSats, which have performed everything from remote sensing to studies of the upper atmosphere, have been launched from the US, Asia, Europe and Latin America.

2.2.1 South Africa's Contribution to Satellite Technology

South Africa is also a global contributor in satellite technology. On 23 February 1999, SUNSAT (Stellenbosch University Satellite) was launched by NASA (National Aeronautics and Space Administration) from Vandenberg Airforce Base on a Delta II rocket. This first remote sensing and packet communications microsatellite was developed by graduate students at Stellenbosch University[1] in South Africa (Mostert and Koekemoer, 1997). SumbandilaSat was the next satellite to be developed in South Africa. This satellite was manufactured at SunSpace & Information Systems, a company of the Stellenbosch University(Steyn, 2008). SumbandilaSat was launched on 17 September 2009 on a Soyuz-2 launch vehicle from the Baikonur Cosmodrome.

South Africa and Africa's first CubeSat, TshepisoSAT (initially called ZACUBE-1), shown in Figure 2.1, was launched from the Yasny launch base in Russia on 21 November 2013 at 09:10:11 SAST (South African Standard Time). TshepisoSAT was developed by graduate students of Cape Peninsula University of Technology (CPUT)[2] in Cape Town, South Africa. TshepisoSAT is a collaboration between the French South African Institute of Technology (F'SATI), CPUT's CubeSat Programme and the Space Science Directorate

[1]See www.sun.ac.za
[2]See www.cput.ac.za

of the South African National Space Agency (SANSA)[3]. The development of ZACUBE-2 is already underway. This will be a 3U CubeSat, developed by CPUT students.

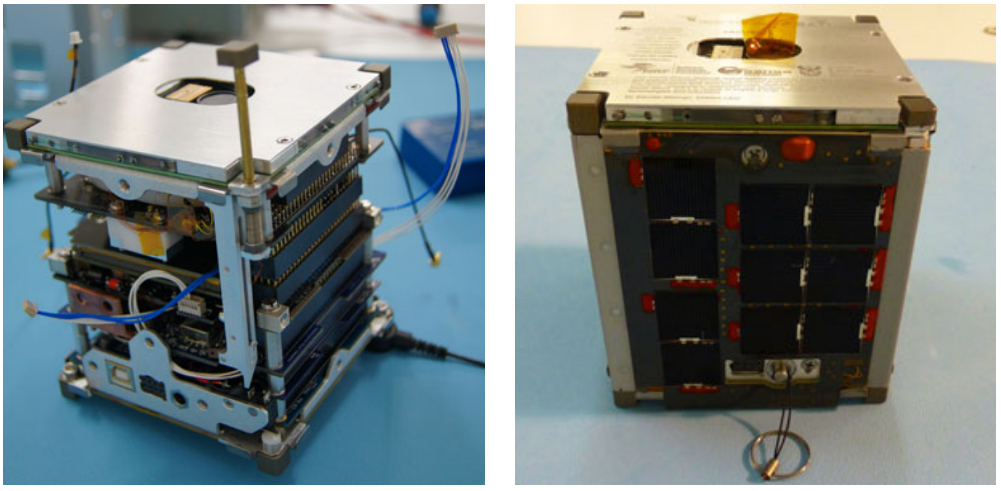

Figure 2.1: South Africa's first CubeSat, TshepisoSAT, originally known as ZACUBE-1

2.3 Historical Background of Equations of Motion

Man's interest in space began with understanding motions of celestial objects which later led to time measurement and navigation. The earliest evidence of man's interest in the universe dates back to 1650 B.C. in Babylon and Egypt (Chobotov, 2002).

Aristotle first postulated the geocentric model of the universe in 350 B.C. He also postulated that all heavenly bodies moved in circular motion (Chaisson and McMillan, 2008). Although Aristarchus postulated the heliocentric model in 300 B.C., in which the Sun and stars are fixed and the Earth orbits the Sun, it was the Aristotelian theory of a geocentric universe that was to dominate for the next 2000 years (Opperman, 2003). Hipparchus introduced the epicyclical motion of the planets in 130 B.C., to explain the retrogade motion of the planets beyond the Earth's orbit, which was further elaborated on by Ptolemy in A.D. 150 (Chobotov, 2002).

[3]See www.sansa.org.za

The Aristotelian theory of a geocentric universe dominated until the time of Nicholas Copernicus (1467-1543), who postulated the Sun at the center of the universe (Opperman, 2003). Planets were seen as moving in epicycles around the Sun, with the Moon orbiting the Earth. He hypothesized that the stars lay on a sphere of very large radius (Chobotov, 2002). His theory was not well received in his day. Johannes Kepler (1571 - 1630) was one of the few people who read Copernicus' book, *On the Revolutions of the Heavenly Spheres (1543)* (Opperman, 2003). In 1609, Kepler postulated his first 2 laws and the third one followed in 1619:

First Law: *The orbit of each planet is an ellipse, with the sun at focus.*
Second Law: *The line joining the planet to the sun sweeps out equal areas in equal times.*
Third Law: *The square of the period of a planet is proportional to the cube of its mean distance from the sun.*

Galileo Galilei (1546 - 1642) observed the four moons of Jupiter which led to acceptance of the Copernican heliocentric (Chaisson and McMillan, 2008). Kepler's laws were only a description, not an explanation of planetary motion. Isaac Newton (1642 - 1727) formulated the law of gravitation and the laws of motion, which form the foundation for modern space flight. He also developed the fundamental concepts of differential calculus in 1666. Newton's work was published in 1687 as the *Mathematical Principles of Natural Philosophy*, or, more simply, *Principia*, undoubtedly one of the supreme achievements of human kind (Bate *et al.*, 1971). The three laws of motion and the law of gravitation state:
First Law: *Every body continues in its state of rest or of uniform motion in a straight line unless it is compelled to change that state by forces impressed upon it.*
Second Law: *The rate of change of momentum is proportional to the force impressed and is in the same direction as that force.*
This can be expressed mathematically (Bate *et al.*, 1971 :4) as follows:

$$\sum \mathbf{F} = m\ddot{\mathbf{r}}$$

Third Law: *To every action there is always opposed an equal reaction.*

Law of Universal Gravitation: *Two bodies attract one another with a force proportional to the product of their masses and inversely propotional to the square of the distance between them.*

This can be expressed mathematically (Bate *et al.*, 1971 :4) as follows:

$$\mathbf{F}_g = -\frac{GMm}{r^2}\frac{\mathbf{r}}{r}$$

After Newton, Joseph Louis Lagrange (1736- 1813) showed the first solutions for the three-body problem in *Essai sur le probleme des trois corps* in 1772 (Vallado, 1997). Numerous other scientists contributed to creating the required mathematical tools needed to advance astrodynamic thought and study.

2.4 Keplerian Orbits

2.4.1 General and Restricted Two-Body Problem

Consider a system of two bodies of masses M and m, with $\mathbf{r}_M$ and $\mathbf{r}_m$ as their position vectors respectively in an initial frame as shown in Figure 2.2. For this case, M is the

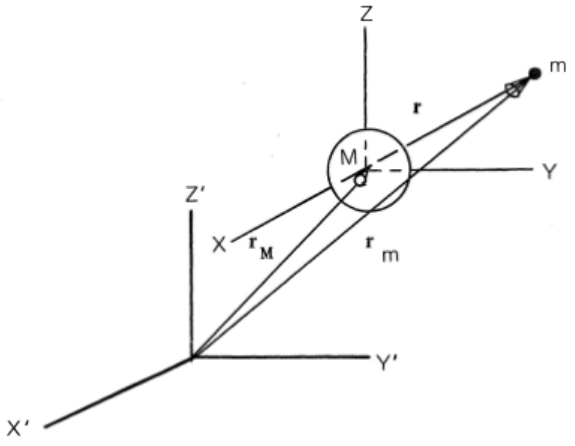

Figure 2.2: Relative motion of two bodies (Bate *et al.*, 1971 :13)

mass of the Earth and m is the mass of a satellite. From Newton's second law and law of

gravitation it can be shown that

$$\ddot{\mathbf{r}} = -\frac{G(M+m)}{r^3}\mathbf{r} \tag{2.1}$$

where

$\ddot{\mathbf{r}}$ = the acceleration of body m relative to M,

G = gravitational constant, $6.672 \times 10^{-11} m^3 kg^{-1} s^{-2}$, and

$\mathbf{r}$ = position of body m relative to M.

By assuming the satellite's mass is significantly smaller than the mass of the Earth, the satellite's mass may be ignored and GM be replaced with μ ($= 3.986\,004\,415 \times 10^{14}\ m^3 s^{-2}$), which is known as the Earth's gravitational parameter. Equation (2.1) then becomes

$$\ddot{\mathbf{r}} + \frac{\mu}{r^3}\mathbf{r} = 0 \tag{2.2}$$

which is known as the restricted two-body equation of motion and it represents the motion of a satellite (mass m) in a gravitational field of the Earth (mass M). The results of this equation are valid subject to the following assumptions:

- The principal mass M is assumed to be fixed in inertial space, which implies that m does not affect the motion of M ($M \gg m$).
- The coordinate system chosen for a particular problem is inertial.
- The bodies of the satellite and the Earth are spherically symmetrical, with uniform density.
- Gravity is the only force acting on the system along the line joining the centres of two bodies.

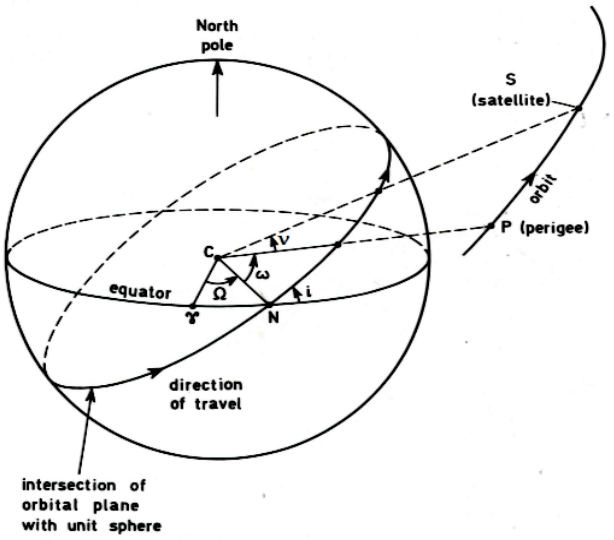

Figure 2.3: Classical Orbital elements (King-Hele, 1987)

2.4.2 Classical Orbital Elements

Five parameters are used to describe the shape, size and orientation of an elliptic orbit. The sixth orbital element defines the angular position of a satellite in its orbit. The *semi-major axis* (a) and *eccentricity* (e) are used to define the size and shape of the orbit. The *true anomaly* (ν) describes the location of a satellite in its orbit by the angular distance it has travelled.

The two angles, namely the *inclination* (i) and the *right ascension of the ascending node* Ω (RAAN) define the orbital plane's orientation in space. The last element is called the *argument of perigee* (ω), the angle between the RAAN and perigee. These orbital elements are shown geometrically in Figure 2.3 below.

2.4.3 Orbital Parameters of a Satellite

With the help of Figure 2.4, satellite parameters in orbital plane together with their equations are listed in Table 2.1.

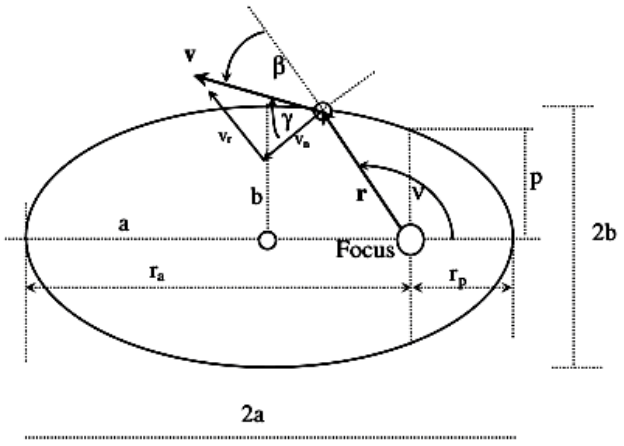

Figure 2.4: Geometry of an ellipse (Opperman, 2003 :2-4)

Table 2.1: Orbital parameters of a satellite in a conic section.

Parameter	Description	Equation
a	*Semi-major axis*	$a = \frac{r_a + r_p}{2}$
b	*Semi-minor axis*	$b = \sqrt{a^2(1-e^2)}$
e	*Eccentricity*	$e = \frac{r_a - r_p}{r_a + r_p}$
r_a	*Apoapsis radius*	$r_a = a(1+e)$
r_p	*Periapsis radius*	$r_p = a(1-e)$
p	*Semi-latus rectum*	$p = a(1-e^2)$ $p = r_a(1-e)$ $p = r_p(1+e)$
γ	*Flight path angle*	$\gamma = \pi/2 - \beta$
h	*Angular momentum*	$h = r\nu\cos\gamma$

Continued on next page

Table 2.1 – continued from previous page

Parameter	Description	Equation
v_r	*Radial velocity component*	$v_r = \sqrt{\frac{\mu}{p}} e \sin\nu$
v_n	*Normal velocity component*	$v_n = \sqrt{\frac{\mu}{p}}(1 + e\cos\nu)$
v	*Velocity at any position* - Ellipse - Circular orbit - Parabola	 $v = \sqrt{\mu\left(\frac{2}{r} - \frac{1}{a}\right)}$ $v = \sqrt{\mu/r}$ $v = \sqrt{2v_c}$
n	*Mean motion*	$n = \sqrt{\frac{\mu}{a^3}}$
P	*Orbital Period*	$P = 2\pi/n$

2.5 The Trajectory Equation

The solution of the equation of motion (Equation 2.2) which is called the trajectory equation[4] is represented by

$$r = \frac{h^2/\mu}{1 + (B/\mu)\cos\nu} \tag{2.3}$$

where h is the specific angular momentum[5] and B is the magnitude of the vector integration constant. This equation tells us the size and shape of the orbit and is expressed in this context in polar coordinates. The general equation which determines the kind of curve Equation (2.3) represents, is written as:

$$r = \frac{p}{1 + e\cos\nu} \tag{2.4}$$

[4]See Appendix C, for the derivation
[5]See Appendix B for equation

This is called the polar equation of a conic section. The origin of a conic section (circle, ellipse, hyperbola or parabola) is located at a focus and p is the semi-latus rectum, simply called *parameter*, e is the eccentricity which determines the type of the conic section and ν is the polar angle as shown in Figure 2.5. The classification of conic sections in terms of their eccentricities is illustrated in Table 2.2.

Table 2.2: Classification of conic sections in terms of eccentricity.

Eccentricity	Orbit
$e = 0$	Circle
$0 < e < 1$	Ellipse
$e = 1$	Parabola
$e > 1$	Hyperbola

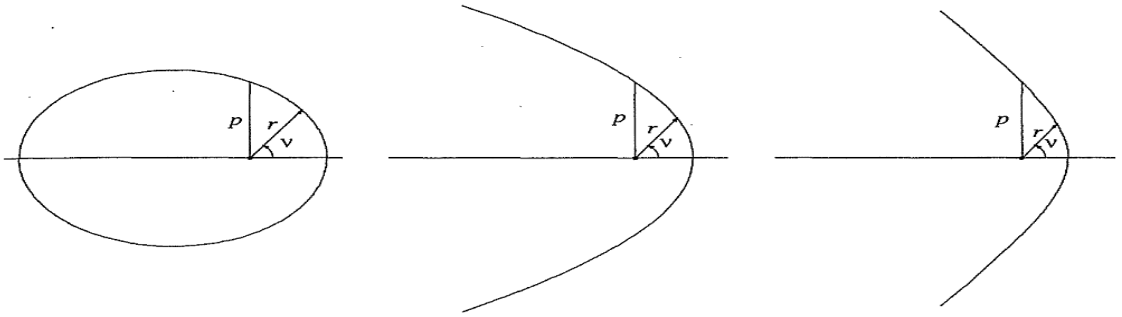

Figure 2.5: Conic sections with equal semi-latus (p) rectum and different eccentricities (e) (Montenbruck and Gill, 2001 :19)

2.6 Analytical Solution to the Restricted Two-Body Problem

The geometrical form of satellite orbits is summarised by Equation (2.4). The problem of determining the future (or past) state of a satellite in orbit at a specific time t, given the state at some initial time t_0, is known as Kepler's Problem. This problem can be solved by formulating an analytical solution. The position of a satellite at time t can be found by determining its angular distance (ν) in orbit and determining (x,y) from Equation (2.4). Note that other orbital elements do not change in Keplerian orbit (Chobotov, 2002). The relationship between the true anomaly (ν) and its counterpart, called the eccentric anomaly (E), is represented geometrically in Figure 2.6. The transformation between the

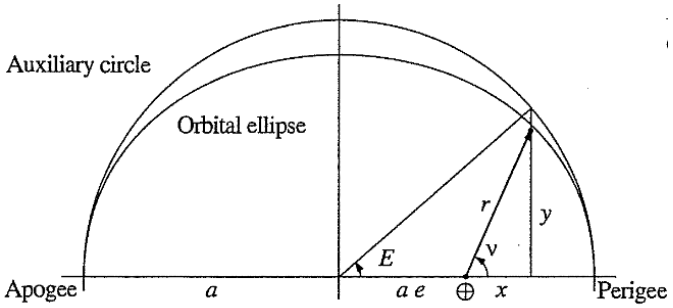

Figure 2.6: Geometrical Relationship of ν and E (Montenbruck and Gill, 2001)

two angular distances (ν and E) is derived in detail in Appendix D. Just as the angular distance ν in the ellipse changes in a non-uniform way, the motion of its image E in the auxiliary circle is also non-uniform. The satellite's position is defined by the following relations (Montenbruck and Gill, 2001 :22)

$$\begin{aligned} x &= r\cos\nu = a(\cos E - e) \\ y &= r\sin\nu = a\sqrt{1-e^2}\sin E \end{aligned} \tag{2.5}$$

The relations for its velocity($\dot{x}$ and $\dot{y}$) can be found by direct differentiation,

$$\begin{aligned} \dot{x} &= -\frac{\sqrt{\mu}}{p}\sin\nu = -a\dot{E}\sin E \\ \dot{y} &= \frac{\mu}{p}(e+\cos\nu) = a\dot{E}\sqrt{1-e^2}\cos E \end{aligned} \tag{2.6}$$

Kepler's equation is represented by

$$E - e\sin E = n(t-T) \tag{2.7}$$

where n is the mean motion[6] and T is the time of perifocal passage. The mean anomaly M, which defines uniform motion along another auxiliary circle, is defined by the expression on the right of Equation (2.7)

$$M = n(t-T) \tag{2.8}$$

In order to obtain the position of a satellite at time t, the time of periforcal passage and the semi-major axis must be known to calculate M. The solution of Kepler's equation is

[6]See Table 2.1 for the equation

equivalent to finding the root of $f(E)$ which is expressed by,

$$f(E) = E - e \sin E - M \tag{2.9}$$

Kepler's equation can only be solved by iterative methods. If the starting value of $E_0 = M$ is approximated, which is recommended for small eccentricities, or $E_0 = \pi$, which is recommended for high eccentricities, Kepler's equation can be iterated using Newton-Raphson iteration techniques as follows (Montenbruck and Gill, 2001)

$$E_{i+1} = E_i - \frac{E_i - e \sin E_i - M}{1 - e \cos E_i} \tag{2.10}$$

Kepler orbits are not a complete reflection of real-life situations. Non-Keplerian influences known as perturbative effects are discussed in the next chapter.

Chapter 3

Perturbation Equations

3.1 Introduction

Keplerian orbits are formed on the assumptions that the Earth or the main attracting object is a perfect sphere, the masses of the attracting body and that of a satellite are uniformly distributed and gravity is the only force acting on a satellite. Although this is a good approximation of the actual satellite motion (position and velocity), the accuracy decreases as the propagation time increases. These deviations are caused by non-Keplerian influences such as the attracting body's aspheric gravity, luni-solar and planetary effects, atmospheric drag and solar radiation pressure. These effects are called orbit perturbations and are discussed in the following section.

3.2 Equation of Motion with Perturbations

Since the propagation of the two-body motion result in increase of errors as the time of propagation increases, it is better to replace the two-body equation, which is expressed by Equation (2.2), with a new equation which includes all perturbation forces.

$$\dot{\mathbf{v}} = -\frac{\mu}{r^3}\mathbf{r} + \mathbf{a}_p \tag{3.1}$$

where $\mathbf{a}_p$ is the sum of all perturbing accelerations. These can be classified as gravitational (luni-solar and planetary attractions, and aspheric gravity) or non-gravitational (atmospheric drag and solar pressure). These perturbations can also be grouped as conservative or non-conservative. For conservative accelerations, $\mathbf{a}_p$ is an explicit function of position only, while for non-conservative accelerations, it is an explicit function of position and velocity (Chobotov, 2002).

3.3 Perturbation Methods

Bate *et al.* (1971) describe a perturbation as a deviation from some normal or expected motion. Two approaches to solving the equations of motion are offered: *general perturbation* and *special perturbation*. The former involves an analytical integration of series expansions of the perturbing accelerations and the latter uses step-by-step numerical integration of the equations of motion.

3.3.1 General Perturbation Techniques

The integration of series (which are expansions based on perturbation equations) analytically, term by term is the core of the general perturbation theory. The expansions are obtained by means of methods of variation of parameters. The general perturbation techniques won't be covered in details in this study. For further reading see (Vallado, 1997 : 539-644).

3.3.1.1 Variation of Parameters

A two-body state vector allows us to determine orbital elements which remain constant anywhere along this orbit. This allows the computation of the state vector of a satellite at any time along the orbit. However, in the presence of perturbations the orbital elements are not constant. The concept of variation of parameters is subsequently introduced. This

concept allows orbital elements to vary while the state vector is computed from a unique set of two-body elements.

3.3.2 Special Perturbation Techniques

Two basic special perturbation techniques exist: Cowell's method and Encke's method.

3.3.2.1 Cowell's Method

This method was developed by P.H. Cowell in the early 20th century. It's been used to predict the return of Halley's Comet and to determine the orbit of the eighth satellite of Jupiter (Bate *et al.*, 1971). This is a very simple method compared to other perturbation methods and is now used most frequently, since the power of computing is increasing. Cowell's method takes the equations and motion with all perturbations and then step-wise integrates them numerically. This is achieved by splitting Equation (3.1) into two coupled first-order non-linear differential equations

$$\begin{aligned} \dot{\mathbf{r}} &= \mathbf{v} \\ \dot{\mathbf{v}} &= -\frac{\mu}{r^3}\mathbf{r} + \mathbf{a}_p \end{aligned} \tag{3.2}$$

where $\mathbf{r}$ and $\mathbf{v}$ are position vector and velocity of a satellite respectively. The advantages of this method is that its very accurate and very easy to implement. Its disadvantages are that it is slow and can be subject to instabilities in some situations. Cowell's method forms the backbone of this study and was selected for its simplicity and accuracy.

3.3.2.2 Encke's Method

Encke's method is a very complex method but its quicker than Cowell's method. This method integrates the differences between the reference and true orbit. The reference orbit is the orbit that would results if there were no perturbations on the satellite at a particular time. At that instant the two orbits coincide. When the reference orbit deviates

too far from the true orbit the process of rectification takes place. A new starting point and epoch are chosen and the new reference orbit is calculated from the radius and velocity of the true orbit. If we let $\mathbf{r}$ and ρ be the radius vectors of the true and reference orbit respectively, we can represent the difference between the reference and true orbit $\delta\ddot{\mathbf{r}}$ with the following equation (Chobotov, 2002 :197)

$$\delta\ddot{\mathbf{r}} = \frac{\mu}{\rho^3}(fq\mathbf{r} - \delta\mathbf{r}) + \mathbf{a}_p \tag{3.3}$$

This method was not used in this study and will not be discussed.

3.4 Perturbations due to Earth's Oblateness

Contrary to the assumptions made in the formulation of the two-body equation of motion, the Earth is not a perfect sphere with symmetrical mass distribution. Instead, the shape of the rotating Earth is oblate with the bulge at the equator, flattened at the poles and not symmetrical in terms of its mass distribution. The bulge at the equator results in the radius of the Earth at the equator being larger by about 22 km than through the poles (Sellers *et al.*, 2004). A number of sources have been identified as causes of this oblateness, but the main sources include the hydrostatic balance between the dominant gravitational force and the centrifugal force due to Earth's rotation, tides, atmosphere-ocean circulation, earthquakes, post-glacial rebound and core flows (Chao, 2006). The gravity potential function of the Earth is defined by the following formula (Vallado, 1997 :492):

$$\Phi = \frac{\mu}{r}\left[1 + \sum_{n=2}^{\infty}\sum_{m=0}^{n}\left(\frac{R_\oplus}{r}\right)^n P_{nm}[\sin(\phi_{sat})]\{C_{nm}\cos(m\lambda_{sat}) + S_{nm}\sin(m\lambda_{sat})\}\right] \tag{3.4}$$

where

$R_\oplus$ = Radius of the Earth

ϕ_{sat} = Geocentric latitude of a satellite

λ_{sat} = Geographic longitude of a satellite

n = Degree of gravity model

m = Order of gravity model

P_{nm} = Associated Legendre functions

C_{nm} and S_{nm} = The gravitational coefficients

For the origin of spherical coordinates to coincide with the centre of mass of the Earth, $C_{10} = C_{11} = S_{11} = 0$ (Rim and Schutz, 2002).

The coefficients can be classified into the following groups of spherical harmonics, as illustrated in Figure 3.1:

- Zonal harmonics ($m = 0$, $-C_{n,0} = J_n$) represent bands along latitude. J_2 is the strongest perturbation and is almost a 1000 times larger than J_3. For any $P_{n,0}$ there are n circles of latitude along which $P_{n,0}$ is zero.
- Sectorial harmonics ($n = m$) represent bands along longitude. It focuses on the mass distributed along the longitude. Legendre functions are only equal to zero at the poles. This type of harmonics is used to compare the difference in density of the ocean and continents.
- Tesseral harmonics ($n \neq m \neq 0$) divide the sphere up into a checkerboard array. The number of curves along which these harmonics vanish are $n - m$ parallels of latitude and $2m$ meridians. They are a key to account for great terrestrial concentrations like mountains.

The acceleration due to this oblateness effect can be obtained by finding the partial derivative or gradient of the potential function

$$\mathbf{a} = \frac{\delta\Phi}{\delta x}\hat{\mathbf{i}} + \frac{\delta\Phi}{\delta y}\hat{\mathbf{j}} + \frac{\delta\Phi}{\delta z}\hat{\mathbf{k}} \tag{3.5}$$

This results in the following acceleration expressions (Vallado, 1997 :497)

$$\begin{aligned}
\mathbf{a}_I &= \left\{ \frac{1}{r}\frac{\partial\Phi}{\partial r} - \frac{r_K}{r^2\sqrt{r_I^2 + r_J^2}}\frac{\partial\Phi}{\partial\phi_{sat}} \right\} r_I - \left\{ \frac{1}{r_I^2 + r_J^2}\frac{\partial\Phi}{\partial\lambda_{sat}} \right\} r_J \\
\mathbf{a}_J &= \left\{ \frac{1}{r}\frac{\partial\Phi}{\partial r} - \frac{r_K}{r^2\sqrt{r_I^2 + r_J^2}}\frac{\partial\Phi}{\partial\phi_{sat}} \right\} r_J + \left\{ \frac{1}{r_I^2 + r_J^2}\frac{\partial\Phi}{\partial\lambda_{sat}} \right\} r_I \\
\mathbf{a}_K &= \frac{1}{r}\frac{\partial\Phi}{\partial r} r_K + \frac{\sqrt{r_I^2 + r_J^2}}{r^2}\frac{\partial\Phi}{\partial\lambda_{sat}}
\end{aligned} \tag{3.6}$$

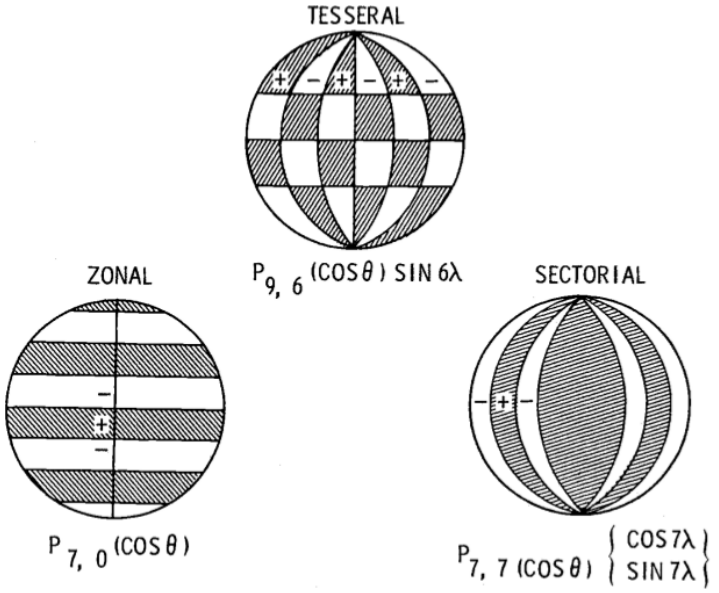

Figure 3.1: Classification of different types of spherical harmonics (Chobotov, 2002 :207)

where the three partials are given by

$$\begin{aligned}
\frac{\partial \Phi}{\partial r} &= -\frac{\mu}{r^2}\sum_{n=2}^{\infty}\sum_{m=0}^{n}\left(\frac{R_\oplus}{r}\right)(n+1)\bar{P}_{nm}[\sin\phi_{sat}]\{\bar{C}_{nm}\cos(m\lambda_{sat})+\bar{S}_{nm}\sin(m\lambda_{sat})\} \\
\frac{\partial \Phi}{\partial \phi_{sat}} &= \frac{\mu}{r}\sum_{n=2}^{\infty}\sum_{m=0}^{n}\left(\frac{R_\oplus}{r}\right)\bar{P}_{n,m+1}[\sin\phi_{sat}] - m\tan(\phi_{sat})\bar{P}_{nm}[\sin\phi_{sat}] \\
&\quad \times\{\bar{C}_{nm}\cos(m\lambda_{sat})+\bar{S}_{nm}\sin(m\lambda_{sat})\} \\
\frac{\partial \Phi}{\partial \lambda_{sat}} &= \frac{\mu}{r}\sum_{n=2}^{\infty}\sum_{m=0}^{n}\left(\frac{R_\oplus}{r}\right)m\bar{P}_{nm}[\sin\phi_{sat}]\{\bar{S}_{nm}\cos(m\lambda_{sat})-\bar{C}_{nm}\sin(m\lambda_{sat})\}
\end{aligned} \tag{3.7}$$

The Legendre functions and the gravitational coefficients are now normalised. It is important to normalise the Legendre functions when using normalised gravitational coefficients. The normalisation is achieved using (Vallado, 1997 :493):

$$\bar{C}_{nm} = \frac{C_{nm}}{\Pi_{nm}} \qquad \bar{S}_{nm} = \frac{S_{nm}}{\Pi_{nm}} \qquad \bar{P}_{nm} = \Pi_{nm}P_{nm} \tag{3.8}$$

where the overall transformation, Π_{nm} is given by

$$\Pi_{nm} = \sqrt{\frac{(n+m)!}{(n-m)!k(2n+1)}} \tag{3.9}$$

$$k = \begin{cases} 1 & \text{if } m = 0 \\ 2 & \text{if } m \neq 0 \end{cases}$$

The Legendre functions are computed by recursion

$$\begin{aligned} P_{n,0}[\alpha] &= \frac{(2n-1)\alpha P_{n-1,0}[\alpha] - (n-1)P_{n-2,0}[\alpha]}{n} && , n \geq 2 \\ P_{n,m}[\alpha] &= P_{n-2,m}[\alpha] + (2n-1)\cos(\phi_{sat})P_{n-1,m-1}[\alpha] && , 0 < m < n \\ P_{n,n}[\alpha] &= (2n-1)\cos(\phi_{sat})P_{n-1,n-1}[\alpha] && , m \neq 0 \end{aligned} \tag{3.10}$$

where $\alpha = \sin(\phi_{sat})$ and the starting values for $P_{nm}[\alpha]$ are

$$\begin{aligned} P_{0,0}[\alpha] &= 1 \\ P_{1,0}[\alpha] &= \alpha \\ P_{1,1}[\alpha] &= \cos(\phi_{sat}) \end{aligned}$$

The expressions for the longitude terms or the trigonometric functions are also computed recursively.

$$\begin{aligned} \sin(m\lambda) &= 2\cos(\lambda)\sin\{(m-1)\lambda\} - \sin\{(m-2)\lambda\} \\ \cos(m\lambda) &= 2\cos(\lambda)\cos\{(m-1)\lambda\} - \cos\{(m-2)\lambda\} \\ m\tan(\phi_{sat}) &= (m-1)\tan(\phi_{sat}) + \tan(\phi_{sat}) \end{aligned} \tag{3.11}$$

3.4.1 Gravity Field Models

The internal mass distribution of the Earth is not well known. This makes it hard to determine gravitational constants (C_{nm} and S_{nm}) using their defining equations. A number of gravity models have been developed since the launches of Sputnik I and Vanguard 1. Ground-based satellite tracking such as radiometric Doppler tracking and *Satellite Laser Ranging* (SLR) systems have improved the knowledge of the gravitational field. The 1996 Earth Gravity Model (EGM96) with a field of 20×20 was implemented in this study. This field retains good accuracy. A 60×60 gravity model was implemented to evaluate

the trade-off between cost and accuracy when compared to the 20×20 gravity model. The EGM96 model has a complete field of 360×360 degree and order of normalised coefficients. A representation of the geoid as determined by the EGM96 gravity model is shown in Figure 3.2. Daniel Hanish [1] stated that, a 18×18 gravity field model has uncertainty of

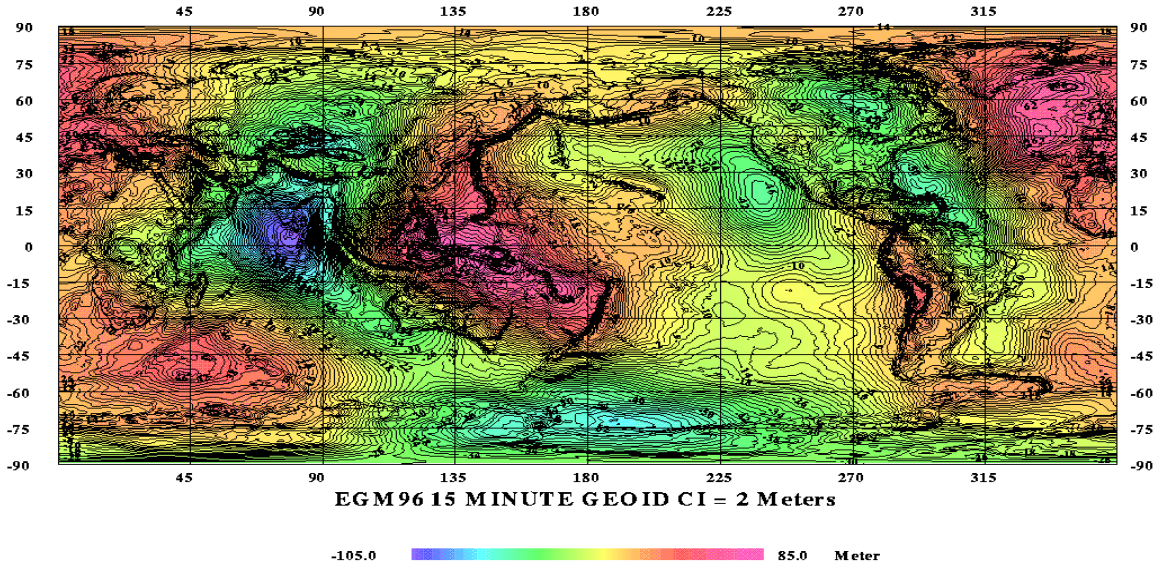

Figure 3.2: 1996 Earth Gravity Model 15 minute geoid (Credit: Daniel Hanish)

around 3 - 4 metres, while a field of 360×360 will yield an uncertainty of 0.5 - 1.0 metres in the geoid correction. Therefore, one can use a 20×20 gravity model and still retain good results without implementing a full gravity model.

3.5 Atmospheric Drag

Another dominant force is the atmospheric drag. This force acts opposite to the velocity of the satellite motion, and hence decelerates the satellite. It's actually the main cause of LEO satellites falling back to the Earth and the most dominant force during the final stages of the lifetime of the satellite. Drag affects all satellites at all altitudes (Gaposchkin and Coster, 1988). The main cause of drag in LEO is the Earth's atmosphere which interacts with solar particles and the geomagnetic field. Drag acceleration is a product of four

[1] See http://www.pha.jhu.edu/~hanish/EGM.html

factors: C_d, A/m, ρ and v_{rel}, and is defined by (Vallado, 1997 :498)

$$\mathbf{a}_{drag} = -\frac{1}{2}\frac{C_D A}{m}\rho v_{rel}^2 \frac{\mathbf{v}_{rel}}{v_{rel}} \tag{3.12}$$

The *coefficient of drag* is defined by the symbol, C_D and is dimensionless. This coefficient describes the interaction of the atmosphere and the surface material of a satellite. Its value depends on the interaction on surface material of a spacecraft, the chemical composition of the atmosphere, the molecular weight, and the temperature of the particles that interact with a spacecraft (Montenbruck and Gill, 2001). The value $C_D = 2.2$, which is appropriate for a sphere or rotating cylinder and many other convex bodies was used in this study (King-Hele, 1987).

The cross-sectional area perpendicular to the direction of the satellite motion is A. For a spherical satellite, the area can be determined with ease. However, it is not a simple task to determine the cross-sectional area of other shapes of satellites. If the satellite is attitude-controlled, A and consequently the area-to-mass ratio can be calculated (King-Hele, 1987). The mass of the satellite is m. A typical 1U CubeSat has a mass of 1.1 kg. SUNSAT's mass of 62 kg was used in this study.

The velocity of a satellite relative to the rotating atmosphere, v_{rel}, is the vector sum of the velocity of the satellite and that of the atmosphere. This is calculated with the assumption that the atmosphere co-rotates with the Earth and is obtained using this expression (Vallado, 1997 :499):

$$\begin{aligned} \mathbf{v}_{rel} &= \frac{d\mathbf{r}}{dt} - \omega_{\oplus} \times \mathbf{r} \\ &= \begin{bmatrix} \frac{dx}{dt} + \omega_{\oplus} y \\ \frac{dy}{dt} - \omega_{\oplus} x \\ \frac{dz}{dt} \end{bmatrix} \end{aligned} \tag{3.13}$$

The *atmospheric density* ρ, is the cause of drag. It is the most difficult variable to calculate in drag evaluation. The atmospheric density changes due to atmospheric molecular composition, geomagnetic variations and the solar flux. Thermospheric models for calculating atmospheric density have been derived since the launch of Sputnik I. Dynamic and

static thermospheric models exist. These models are complex and computationally intensive, but they are often simplified to economise computer run time (Opperman, 2003). Existing thermospheric models include:

- **Jacchia models**[2]: These models contain analytical expressions for determining exospheric temperature as a function of time, solar activity and geomagnetic activity. The computed temperature is then used to empirically determine temperature profiles or the diffusion equation (Vallado, 1997). These models include Jacchia 70, 71 and 77, and Jacchia-Roberts. Most of these models use inputs such as geomagnetic index (k_p) and the $F_{10.7}$ which is the solar flux of radiation at 2.800 MHz (Gaposchkin and Coster, 1988). For the purpose of this study Jacchia 70 was used, because it is well tested, relatively simple to implement and its source code is readily available.

- **Harris-Priester**: This is a static model. It is computationally efficient and gives fairly good results (Vallado, 1997). The Harris-Priester model is based on the properties of the upper atmosphere as determined by the solution of the heat conduction equation and quasi-hydrostatic conditions (Montenbruck and Gill, 2001).

- **Soviet Cosmos**: This model uses an analytical method to obtain atmospheric density in an aspherical upper atmosphere, using observations of Soviet Cosmos satellites (Vallado, 1997). The Soviet Cosmos model is valid for satellites at altitudes of 160-600 km (Vallado, 1997).

Other existing thermospheric models include the *Mass Spectrometer and Incoherent Scatter Model* (MSIS), Exponential model, *COSPAR International Reference Atmosphere* (CIRA), etc. (Vallado, 1997).

A drag calculation error could be the result of an error in the calculation of any of these four factors (Gaposchkin and Coster, 1988). Gaposchkin and Coster (1988) computed the time it took for drag to change a satellite's position by 12 km and found 0.92 days for a satellite at an altitude of 300 km, 22.8 days for a satellite at an altitude of 800 km and 38.9 days for a satellite at 2800 km, with the area-to-mass ratio of 0.01 m^2/kg.

[2]See (Vallado, 1997 :843-854) for details of the formulas

3.5.1 Cross-Sectional Area

The cross sectional area of a satellite is calculated by considering the mean of three possible maximum areas (Opperman, 2003). This is shown in Figure 3.3. The mean cross-sectional

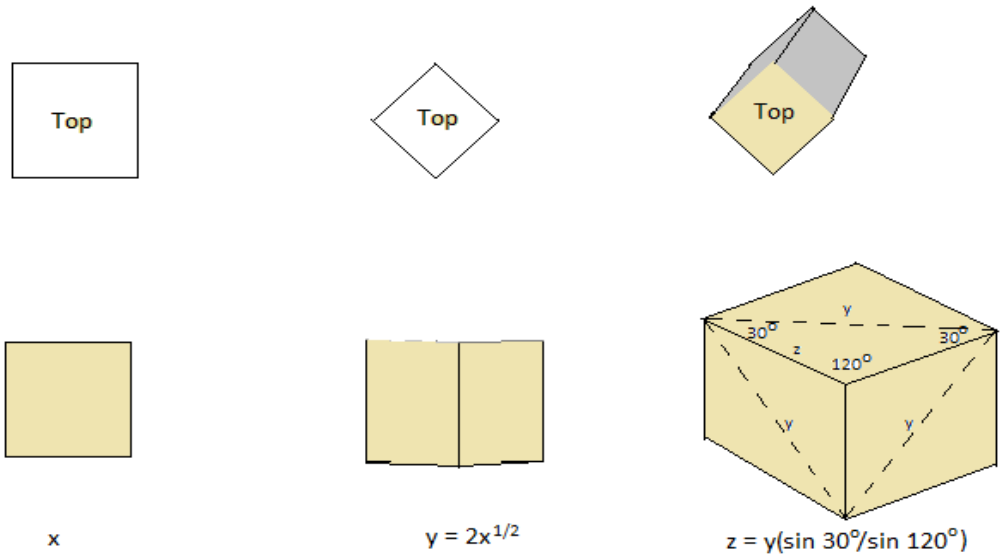

Figure 3.3: Possible cross-sectional areas of a cubic satellite (Opperman, 2003)

area is calculated using (Opperman, 2003):

$$A = \frac{1}{3}(A_1 + A_2 + A_3) \tag{3.14}$$
$$= \frac{1}{3}(x^2 + xy + 3z^2 \sin 60)m^2$$

The equation yields the mean cross-sectional area of 0.27987 m^2 for SUNSAT, with dimensions 45 cm × 45 cm × 45 cm, and 0.01382 m^2 for a 1U CubeSat.

3.6 Third-Body Perturbations

Unlike restricted two-body attraction, a LEO satellite experiences gravitational attraction from any large body in the solar system. The Moon and Sun are the main causes of third body perturbations on LEO satellites, though planets may be included as well.

Perturbation effects by these two bodies become noticeable when drag begin to diminish (Vallado, 1997), e.g. for very high orbit satellites. The basic equation for third-body perturbations is given by (Vallado, 1997 :515)

$$\mathbf{a}_{3body} = -\frac{Gm_{\oplus}}{r^3_{\oplus sat}}\mathbf{r}_{\oplus sat} + Gm_3\left(\frac{\mathbf{r}_{sat3}}{r^3_{sat3}} - \frac{\mathbf{r}_{\oplus 3}}{r^3_{\oplus 3}}\right) \tag{3.15}$$

The first term on the right hand side is from the two-body equation while the second term indicates the addition of the third body which is represented by the subscript 3. This equation can be numerically unstable in a case where the distance from the third body, the Sun for instance, to the satellite and the distance from the Earth to the Sun are almost equal. Vallado (1997) presented the final stable third-body equation, after some manipulation of the Taylor series expansion and the Legendre functions as

$$\mathbf{a}_{3body} = -\frac{Gm_{\oplus}}{r^3_{\oplus sat}}\mathbf{r}_{\oplus sat} - \sum_{k=1}^{n}\frac{Gm_k}{r^3_{\oplus k}}(\mathbf{r}_{\oplus sat} - \beta_k\mathbf{r}_{satk}) \tag{3.16}$$

where,

$$\begin{aligned} \beta_k &= 3B_k + 3B_k^2 + B_k^3 \\ B &= \sum_{j=1}^{\infty} P_j[\cos(\varepsilon)]h^j \\ h &= \frac{r_{\oplus sat}}{r_{\oplus 3}}. \end{aligned} \tag{3.17}$$

The angle between the third body and satellite as seen from the Earth is represented by ε and $P_j[\cos(\varepsilon)]$ are called Legendre polynomials of order j and are computed by the first expression ($P_{n,0}$) in Equation (3.10).

3.7 Solar Radiation Pressure

Satellites in LEO experience a force from absorption or reflection of photons from the Sun. This force always points in the direction away from the Sun just like comet tails. The

influence of this force depends on altitude and the solar activity. It is very important to quantify the effect of solar radiation pressure during periods of intense solar storms as it can dominate all other perturbation forces, especially at higher altitudes. The acceleration due to solar radiation pressure depends on the mass and surface area of the satellite and is defined by (Vallado, 1997 :520):

$$\mathbf{a}_{SR} = -\frac{P_{SR} c_R A_\odot}{m} \frac{\mathbf{r}_{\odot sat}}{|\mathbf{r}_{\odot sat}|} \tag{3.18}$$

where

$P_{SR} = 4.51 \times 10^{-6} kg\ m^{-1} s^{-2}$ is the solar pressure,

$C_R =$ the radiation pressure reflectivity coefficient.

The value of C_R is always between 0.0 and 2.0 and it is hard to determine as it changes over time. A value of 0.0 means that the satellite experiences no solar pressure as the satellite permits all light to pass through. A value of 1.0 means that all the solar radiation is absorbed, while a value of 2.0 means that all the solar radiation is reflected (Vallado, 1997). The difficulties in modelling solar-radiation pressure lie in determining the area exposed to the Sun's radiation, which is defined by $A_\odot$. However, for the purpose of calculating drag, this area is often assumed to be the maximum possible cross-sectional area of the satellite (Du Toit, 1997). Solar radiation pressure interacts with satellites on the Sun side. When the satellite is in the Earth's shadow this force need not be considered. To achieve that a switching function is used, which turns solar radiation pressure on or off when it is necessary.

3.7.1 Shadow Analysis

A satellite in orbit experiences periodic eclipses behind the Earth. A function to control the situation where a satellite goes in and out of the Earth's shadow when computing radiation pressure is necessary. The angular separation between the Sun and the satellite is represented by their dot product (Vallado, 1997 :521)

$$\cos(\varphi) = \frac{\mathbf{r}_\odot \cdot \mathbf{r}_{sat}}{\mathbf{r}_\odot \mathbf{r}_{sat}} \tag{3.19}$$

where φ is the angular separation between the Sun and satellite. Equation (3.19) can be written in terms of the true anomaly to give precise information about the satellite's position in orbit for an accurate switching function

$$\cos(\varphi) = \beta_1 \cos(\nu) + \beta_2 \sin(\nu) \tag{3.20}$$

The shadow function[3] is then found by squaring Equations (3.19) and (3.20) and using Equation (2.4)

$$S = R_{\oplus}^2(1 + e\cos(\nu))^2 + p^2\{\beta_1 \cos(\nu) + \beta_2 \sin(\nu)\}^2 - p^2 \tag{3.21}$$

This function vanishes only if

$$1 - \left(\frac{R_{\oplus}}{a(1-e)}\right)^2 < \beta_1^2 < 1 - \left(\frac{R_{\oplus}}{a(1+e)}\right)^2$$

3.8 Precision Modelling

The perturbation forces described above are sufficient for a precise satellite orbit. In cases where higher precision of radial position of 10 cm or lower is required, other perturbation forces such as the radiation pressure of the Earth, planetary attraction, tidal forces that modify the Earth's gravity field, as well as general relativistic deviations to the Newtonian equations of motion need to be taken into account (Montenbruck and Gill, 2001). A spacecraft with an on-board thruster system may experience additional perturbation forces due to this system, and should also be modelled for high-precision results. Finally, for effects that cannot be described by physical models, empirical accelerations may be introduced (Montenbruck and Gill, 2001).

[3]For derivation see (Escobal, 1985 :155-159) and (Vallado, 1997 :520-523)

3.9 Summary

The theory of Keplerian orbits alone cannot explain the motion of satellites. Non-Keplerian orbits were introduced and discussed in this chapter. Perturbation methods (general and special perturbations) were also discussed.

In Chapter 4, time, coordinate systems and transformations applicable in the development of *PRECurSOr* are discussed.

Chapter 4

Time, Coordinate Systems and Transformations

4.1 Introduction

An orbit is suitably defined using a frame of reference and time. This means an inertial coordinate system must be found. Time on the other hand is used to define the moment of a phenomenon with precision. This moment is referred to as the epoch (Vallado, 1997). This chapter discusses the time systems, coordinate systems and the transformations implemented in developing *PRECurSOr*.

4.2 Time Systems

The following time systems are of utmost necessity for developing an orbit propagator. A brief introduction follows:

- **International Atomic Time** (TAI) is the most precise time standard. It is based on the specific quantum transition of the electrons in a cesium-133 atom (Vallado, 1997).

- **Coordinated Universal Time** (UTC) is derived from atomic time and is the most commonly used time system.

- **Julian Date** (JD) is defined as the continuous amount of time measured in days from the epoch of January 1, 4713 B.C., 12:00 (Greenwich) (Vallado, 1997). One JD is measured from noon to noon. Julian Date for any known date and time can be calculated with the following general formula (Opperman, 2003; Vallado, 1997 :4-4,68):

  if $month = 1$ or 2, set $year = year$ - 1 and $month = month + 12$

  Set $day = day + (hour + minutes/60 + seconds/3600)/24$

  Let

$$B = 2 - INT\left(\frac{year}{100}\right) + INT\left(\frac{INT\left(\frac{year}{100}\right)}{4}\right)$$

  Therefore,

$$JD = INT\{365.25(year + 4716)\} + INT\{30.6001(month + 1)\} + day + B - 1524.5 \tag{4.1}$$

- **Greenwich Mean Sidereal Time** (θ_{GMST}) is the sidereal time associated with the Greenwich meridian. It uses the vernal equinox as the reference point and the following equation defines it (Vallado, 1997):

$$\theta_{GMST} = \theta_{GST0} + \omega_{\oplus} UT1 \tag{4.2}$$

  where

$$\theta_{GST0} = 100.4606184^{\circ} + 36000.77005361 T_{UT1} + 0.00038793 T_{UT1}^2 - 2.6 \times 10^{-8} T_{UT1}^3$$

  $\omega_{\oplus}$ = Earth's mean angular rotation rate

  T_{UT1} = Number of Julian centuries elapsed from epoch J2000 given by:

$$T_{UT1} = \frac{JD_0 - 2451545.0}{36525}$$

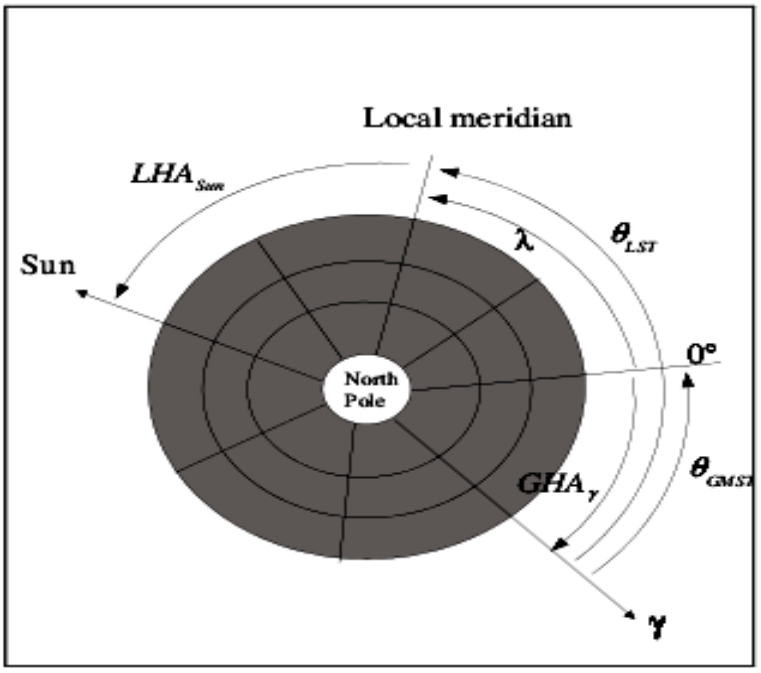

Figure 4.1: Sidereal time (Vallado, 1997; Opperman, 2003 : 4-5)

- **Greenwich Apparent Sidereal Time** (GAST) differs from GMST by the Equation of the Equinoxes due to nutation. It introduces programming complexity due to the functions required in its computation.

- **Local Sidereal Time**(θ_{LST}) is the sidereal time at a particular longitude (λ) and is related to θ_{GMST} at a particular longitude by (Vallado, 1997):

$$\theta_{LST} = \theta_{GST} + \lambda \tag{4.3}$$

4.3 Coordinate systems

4.3.1 Earth Centred Inertial (ECI) System

This is a non-rotating geocentric-equatorial coordinate system with its origin at the centre of Earth (Figure 4.2). Its fundamental plane is the equator. The positive X axis points in the direction of the vernal equinox and the positive Z axis points in the direction of the

North Pole. *Right ascension* (α) and *declination* (δ) are used to define the location of a satellite along some direction from the origin of the celestial sphere.

Right ascension refers to the angle measured eastward in the plane of the equator from a fixed inertial axis in space (vernal equinox) to a plane of the equator (meridian) which contains the object ($0° \leq \alpha \leq 360°$). Declination, on the other hand, refers to the angle between the object and equatorial plane measured (positive above the equator) in the meridional plane containing the object ($-90° \leq \delta \leq 90°$). A satellite's position and velocity vectors, denoted by $\mathbf{r}$ and $\mathbf{v}$ respectively, are also used in this inertial system. The magnitude of vector $\mathbf{r}$ is defined as the radial distance between the origin of the coordinate system and the satellite's location.

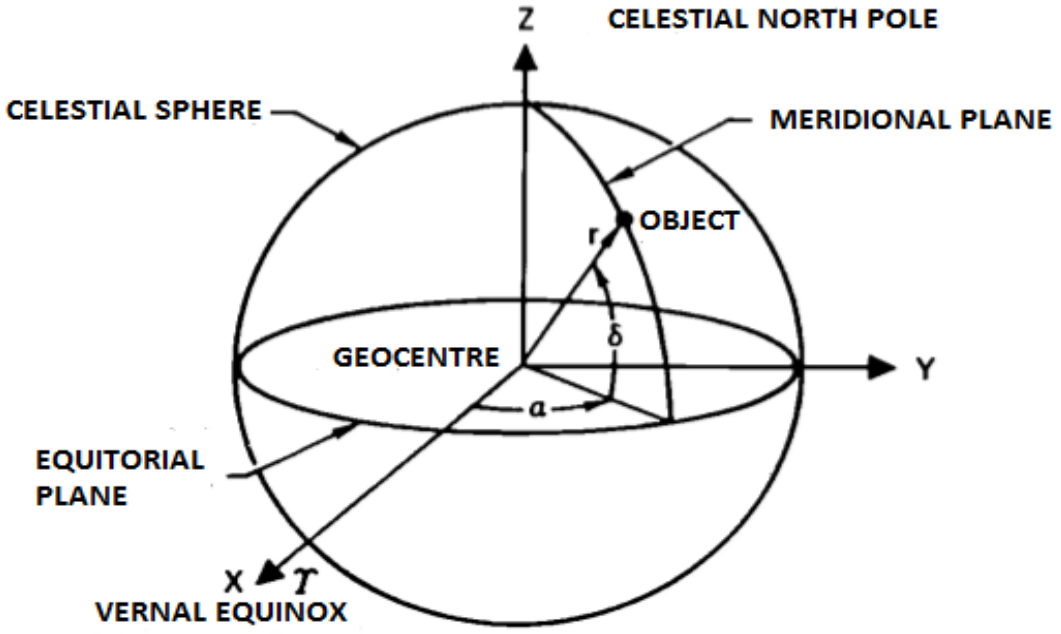

Figure 4.2: Earth-centred inertial system (Chobotov, 2002)

4.3.2 Earth Centred Earth Fixed (ECEF) System

This coordinate system is the same as the ECI system except that the coordinate system rotates with the Earth. In this system, the primary axis is always aligned with a particular meridian.

4.3.3 Geographic Coordinate System

A satellite can be located relative to Earth by its *longitude* (λ), *latitude* (ϕ) and *altitude* above the reference ellipsoid. The origin of this coordinate system is the Earth's centre (Figure 4.3). The fundamental plane is the equator. The Z axis points in the direction of the North Pole. The principal axis (X axis) points toward the Greenwich meridian. The latitude is defined as the angle measured perpendicular to the equatorial plane between the equator and a ray connecting geocentre with a point on the Earth's surface ($-90° \leq \phi \leq 90°$). The *east longitude* (λ_E) is defined as the angle measured eastward from the prime meridian in the equatorial plane to the meridian containing the surface point ($0° \leq \lambda_E \leq 360°$)

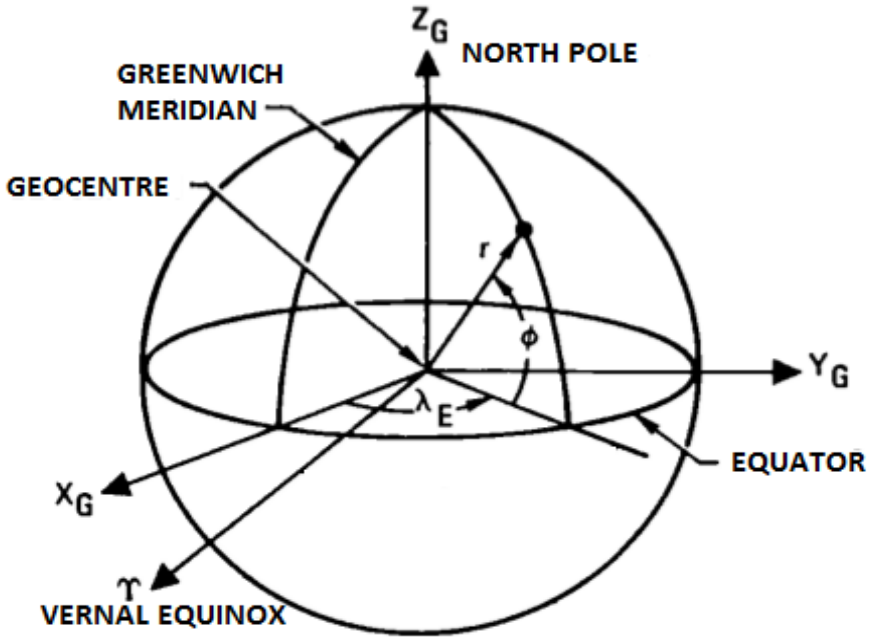

Figure 4.3: Geographic coordinate system (Chobotov, 2002)

4.3.4 Perifocal Coordinate System(PQW)

The fundamental plane of the perifocal coordinate system is the satellite's orbit. The origin is the centre of the Earth. The W-axis is normal to the orbit, p-axis points towards perigee, and q-axis completes the orthogonal setup. This coordinate is not suited for

circular orbits. In presence of perturbations, this coordinate system is not inertial. The coordinate system is shown in Figure 4.4.

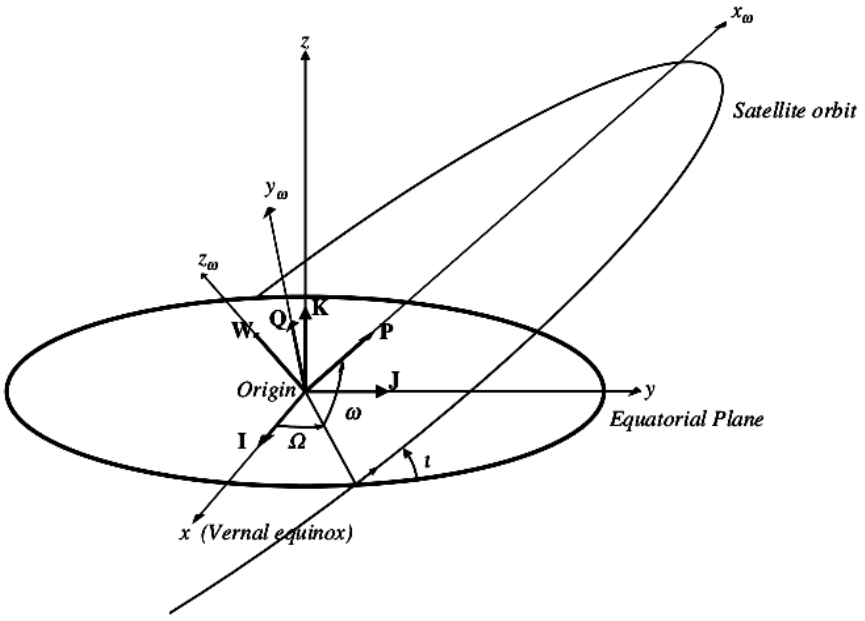

Figure 4.4: Perifocal coordinate system (Escobal, 1985 :77)

4.4 Coordinate Transformations

When transforming one coordinate system to another, two concepts apply: corrections for rotations within a system and translation between systems having different origins (Vallado, 1997). Common transformations which are used in the process of developing an orbital propagator are briefly discussed below.

4.4.1 Transformation between ECI and ECEF

Since the ECI and ECEF systems share the same z-axis and the fundamental plane, transformation from one to the other only require rotation of an angle $\theta_{GMST} = \theta_{GMST,2000} + \omega_\oplus \times t$, where θ_{GMST} is the Greenwich Mean Sidereal time, $\omega_\oplus = 0.0000729212$ rad/s is the rotation rate of the Earth and t is the elapsed time since ECEF and ECI frames are

separated by an angle of $\theta_{GMST,2000} = 1.7447672$ rad, the value taken on 1 January 2000 at 00 : 00 : 00 (Ilyas, 2011). The conversion between these frames which neglects the effects of precession, nutation and polar motion is given by:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{ECI} = \begin{pmatrix} \cos(\theta_{GMST}) & -\sin(\theta_{GMST}) & 0 \\ \sin(\theta_{GMST}) & \cos(\theta_{GMST}) & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix}_{ECEF} \tag{4.4}$$

The inverse transformation can be obtained by using the inverse of the rotation matrix used above (Ilyas, 2011).

4.4.2 Transformation from Classical Orbital Elements to ECI

The transformation is successfully done by first transforming orbital elements to the PQW frame, then transforming the PQW frame to the ECI frame (Chobotov, 2002 : 62-65). The first transformation is accomplished by :

$$\mathbf{r} = r\cos\nu\hat{\mathbf{P}} + r\sin\nu\hat{\mathbf{Q}} \tag{4.5}$$
$$\mathbf{v} = \sqrt{\frac{\mu}{p}}[(-\sin\nu)\hat{\mathbf{P}} + (e + cos\nu)\hat{\mathbf{Q}}]$$

Finally, transformation from PQW to ECI frame is given by:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix}_{ECI} = [R] \begin{pmatrix} r\cos\nu \\ r\sin\nu \\ 0 \end{pmatrix}_{PQW} \tag{4.6}$$

and

$$\begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}_{ECI} = [R] \begin{pmatrix} -\sqrt{\frac{\mu}{p}}\sin\nu \\ \sqrt{\frac{\mu}{p}}(e + \cos\nu) \\ 0 \end{pmatrix}_{PQW} \tag{4.7}$$

where

$$[R] = \begin{pmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{pmatrix} \tag{4.8}$$

and

$$\begin{aligned} R_{11} &= \cos\Omega\cos\omega - \sin\Omega\sin\omega\cos i \\ R_{12} &= -\cos\Omega\sin\omega - \sin\Omega\cos\omega\cos i \\ R_{13} &= \sin\Omega\sin i \\ R_{21} &= \sin\Omega\cos\omega + \cos\Omega\sin\omega\cos i \\ R_{22} &= -\sin\Omega\sin\omega + \cos\Omega\cos\omega\cos i \\ R_{23} &= -\cos\Omega\sin i \\ R_{31} &= \sin\omega\sin i \\ R_{32} &= \cos\omega\sin i \\ R_{33} &= \cos i \end{aligned} \tag{4.9}$$

4.5 Summary

The time systems, coordinate systems and transformations used in the development of *PRECurSOr* were covered in this chapter.

Chapter 5 details the considerations taken into account to develop the software and gives relevant background information on OBC.

Chapter 5

OBC Emulator and Software Considerations

5.1 Introduction

The process of software development requires careful considerations, since a small error may result in operational failure. Choosing the right programming language is very important. Some programming languages are slower than others, depending on the problem being solved. This chapter details the considerations taken into account to develop *PRECurSOr* - Precise Satellite Orbit propagator. To understand the limited processing resources of an OBC, relevant background information is given.

5.2 On-Board Computer (OBC)

Lumbwe (2013) defines the OBC as an embedded computer dedicated to command and data handling (C&DH) of the satellite. The OBC is part of five CubeSat subsystems. The other four subsystems are the electrical power system, communication, ADCS (Attitude and Determination Control System) and the payload subsystem.

The electrical power system is responsible for power distribution and control. The communication subsystem, which is also referred to as the Telemetry Tracking and Command (TT&C) subsystem, provides the communication link between the satellite and the ground segment. The ADCS subsystem controls the attitude of the satellite. Finally, the payload subsystem hosts the payload required for a specific CubeSat mission (Lumbwe, 2013).

The OBC is the heart of the satellite where the subsystems communicate via a serial bus interface. The OBC hosts a memory subsystem for additional functions such as recording housekeeping parameters (temperature and power consumption) and telemetry payload data. These are collected at given timescales or coordinates, before initiation of the transmission to the ground station during an overpass of the satellite (Lumbwe, 2013). The OBC subsystem is composed of both hardware and software components as shown in Figure 5.1. *PRECurSOr* runs under the application(s) software component of the OBC subsystem.

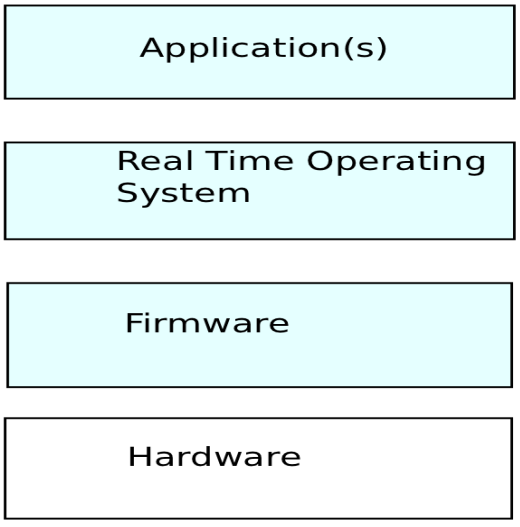

Figure 5.1: Model of the OBC subsystem (Catsoulis, 2005; Lumbwe, 2013)

- **The hardware component** contains the physical electronic components that are located on a printed circuit board (PCB). It consists essentially of a microcontroller with various peripheral interfaces and supporting hardware with a memory module

to store programmes and data (Lumbwe, 2013). A microcontroller is defined as a processor, memory and some Input/Output (I/O) devices all contained within a single integrated circuit (IC), intended for use in embedded systems (Catsoulis, 2005). Microcontrollers are available as 8-bit, 16-bit and 32-bit, depending on the manufacturer.

- **The software component** controls the operation and functionality of the computer. It consists of three layers (application, operating system and firmware), where each layer will only interact with the layers immediately above or below it (Catsoulis, 2005). Firmware is software responsible for initializing other hardware subsystems to a known state and for configuring the OBC for correct operation when the computer first powers up. The operating system controls the operation of the computer and the application software contains programmes that provide the functionality of the OBC (Catsoulis, 2005).

5.2.1 Typical OBCs and their Processing Speeds

A Raspberry Pi[1] computer, shown in Figure 5.2, was used as an OBC emulator to test *PRECurSOr*'s performance and memory usage. The first operating system of choice for study was FreeRTOS, but PiBang Linux[2] was used instead. FreeRTOS was ruled out due to its present installation complexity on a Raspberry Pi computer. PiBang Linux presents simplicity in installation because it is based on Raspbien, an operating system based on Debian, which can be easily optimized for Raspberry hardware. The specifications of the implemented Raspberry Pi are listed in Table 5.1.

An HDMI-to-HDMI lead connector was used for this study. Alternately, a standard RCA composite video lead can be connected to the HDMI output. Raspberry Pi requires an SD card with a minimum memory of 4GB to give the user at least 2GB free space to install additional packages or to create programmes. An 8GB SD card was selected for this study. Low-powered USB keyboards and mice must be used for a Raspberry Pi to function normally. Ethernet cable was not not necessary for this study. A Raspberry Pi

[1]See http://www.raspberrypi.org
[2]See http://pibanglinux.org/

Table 5.1: Raspberry Pi computer specifications

Manufacturer	Raspberry Pi Foundation
Processor	Broadcom 700 MHz
RAM	512 MB
Graphics Processor	VideoCore IV
Peripherals	10/100 Ethernet port 2 × USB 2.0 jacks HDMI port Micro-USB port SD card slot Audio Jack RCA video
Supply Line Voltage	5.0 V

requires a good micro-USB power supply that can provide at least 700mA at 5V. For the purpose of this study a micro-USB power supply providing 1000mA at 5V was selected.

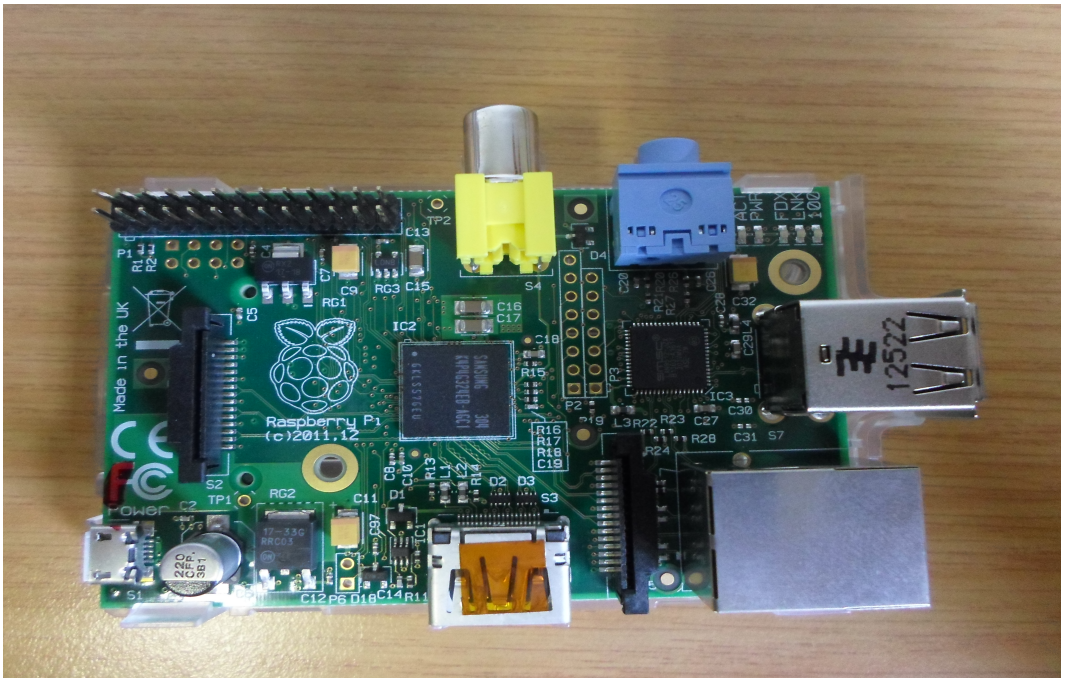

Figure 5.2: The Raspberry Pi computer used as OBC emulator in this study

5.3 Programming Language

Several aspects were considered in the selection of an appropriate programming language. The availability of an Integrated Development Environment (IDE) at no cost, execution

speed and the ease of implementation were the major factors in this selection process. Another aspect was the availability of source code for complex libraries such as calculating atmospheric density (Opperman, 2003). All factors considered, led to the selection of C++ as the programming language. The portability of the C++ language across different platforms and operating systems makes it an appropriate language to use for this study. It should see relative hassle-free implementation on the CubeSat OBC. Initially, the code was written and compiled under the Linux operating system on a desktop PC (Personal Computer) with a GCC (GNU Compiler Collection) compiler using a terminal. The absence of a debugger, however, led to the selection of NetBeans IDE.

5.4 Algorithm Description

In the process of development of the orbit propagator, different algorithms were implemented. This section and its subsections lists and discusses some vital concepts considered in the development of *PRECurSOr*.

5.4.1 Orbit Dynamics Overview

As discussed in chapter 4, the following gravitational and non-gravitational perturbation models were considered in the development of *PRECurSOr*.

- Gravitational perturbations: Aspheric gravity and luni-solar perturbations
- Non-gravitational perturbations: Solar radiation pressure and atmospheric drag

5.4.2 Initial Conditions

The orbit propagator was tested using a one-day section of SUNSAT's SLR-derived precision orbit as reference, with the initial conditions for the position and velocity given in the ECI coordinate system. The epoch of propagation was selected as 06 February 2000 at 00:00:00 UTC with the following initial conditions for the state vector:

Table 5.2: Initial conditions of SUNSAT corresponding to the epoch of 06 February 2000 at 00:00:00 UTC

Orbital parameters		Position and Velocity	
a(m)	7137884.390	X(m)	-611359.693
e	0.014205	Y(m)	6818312.960
i(°)	96.469	Z(m)	1885999.168
Ω(°)	273.334	V_X(m/s)	705.897
ω(°)	233.749	V_Y(m/s)	1956.499
ν(°)	290.747	V_Z(m/s)	-7218.130

5.4.3 Data Files

The software includes ASCII (American Standard Code for Information Interchange) data files which were used in the aspheric gravity and drag models. The files and sources are listed in Table 5.3 below. From the EMG96[3] gravity file, two multi-dimensional

Table 5.3: PRECurSOr data files

File name	**Description**	**Source**
egm96_to360.ascii	EMG96 gravity file	NASA
expmod.dat	Exponential Atmospheric Model density computation file	(Vallado, 1997 :510)
flux_mean.dat	Stores the geomagnetic activity, the mean solar flux, observed and adjusted daily flux values	INPE (Instituto Nacional de Pesquisas Espaciais)

arrays containing spherical harmonics coefficients (C_{nm} and S_{nm}) of 20×20 field were created for computational efficiency. The *flux_mean.dat*[4] data file is a product of a compilation of daily solar flux data from the Dominion Radio Astrophysical Observatory (DRAO) and geomagnetic activity data from the International Service of Geomagnetic Indices (ISGI) provided by the Space Physics Interactive Data Resources (SPIDR). The *flux_mean.dat* file contains geomagnetic activity and solar flux data from 1958-1-1 to 2008-12-31, useful for calculating atmospheric drag. This file was expanded to incorporate flux and geomagnetic data up to 2013-12-31, using algorithms provided on the INPE website[5]. The *expmod.dat* file contains data on atmospheric parameters which enable

[3]See ftp://cddis.gsfc.nasa.gov/pub/egm96/general_info/egm96_to360.ascii
[4]See http://www2.dem.inpe.br/val/atmod/flux_mean.dat
[5]See http://www2.dem.inpe.br/val/atmod/default.html

atmospheric density calculation using the following equation (Vallado, 1997):

$$\rho = \rho_0 EXP\left(-\frac{h_{ellp} - h_0}{H}\right) \tag{5.1}$$

where ρ_0 is the reference density, h_{ellp} is the reference altitude, h_0 is the satellite altitude and H is the scale height.

5.4.4 Numerical Integration

Numerical methods can be used for computation of high precision orbits. A number of numerical integrators have been developed and some of them have been applied in the field of astrodynamics. The Runge-Kutta methods, Adams-Bashforth-Moulton, Gauss-Jackson and many others are examples of numerical integrators. Most numerical integrators were derived from the Taylor series given by (Vallado, 1997 :476)

$$y(t) = y(t_0) + \dot{y}(t_0)(t - t_0) + \frac{\ddot{y}(t_0)(t - t_0)^2}{2!} + \frac{\dddot{y}(t_0)(t - t_0)^3}{3!} + ... \tag{5.2}$$

Two types of numerical integration methods exist: single-step and multi-step methods. Only the Runge-Kutta methods will be discussed, as they were used in this study.

5.4.4.1 Runge-Kutta Methods

The Runge-Kutta (RK) methods were originally developed by Carl Runge (1856-1927) in 1895 and Wilhelm Kutta (1867 - 1944) in 1901 (Vallado, 1997). The methods were later expanded to Runge-Kutta-Fehlberg (also called Embedded Runge-Kutta) by Erwin Fehlberg.

5.4.4.2 Single-Step RK Methods

Single-step RK methods derive from the Euler method given by

$$y_{n+1} = y_n + hf(x_n, y_n) \tag{5.3}$$

The Euler method is also called RK1. The increment function f is an average of the derivative dy/dx over time interval x to $x_n + h$. Runge-Kutta schemes require the first derivative $f(x_n, y_n)$ only at the beginning of the interval. There are many orders of RK methods. These orders reflect the accuracy to which the function f is computed, compared to a Taylor series expansion (Curtis, 2010). The most widely used RK method is the classical fourth-order Runge-Kutta method or RK4 which was used in this study to verify results of the multi-step integrator. This is represented mathematically by (Press *et al.*, 2002 :711)

$$\begin{aligned} k_1 &= hf(x_n, y_n) \\ k_2 &= hf(x_n + \frac{h}{2}, y_n + \frac{k_1}{2}) \\ k_3 &= hf(x_n + \frac{h}{2}, y_n + \frac{k_2}{2}) \\ K_4 &= hf(x_n + h, y_n + k_3) \\ y_{n+1} &= y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(h^5) \end{aligned} \tag{5.4}$$

This is a fourth order method since its error term is $O(h^5) = O(h^{n+1})$. The order is represented by n.

5.4.4.3 Multi-Step RK Methods

Multi-step methods are sometimes called *predictor-corrector* methods. A number of multi-step methods based on the *embedded Runge-Kutta* exist. These methods were originally invented by Erwin Fehlberg. RK methods keep the step-size, h, constant which results in waste of computation time and loss of accuracy in the intervals where the solution changes rapidly. *Runge-Kutta-Fehlberg* (RKF) methods use adaptive step-size control to scale the step-size according to the change in solution. In cases where the solution changes rapidly, small step-sizes are required, while in cases where the solution changes slowly, large step-sizes are required. This gives the RKF methods the advantage of taking larger steps on average compared to other multi-step methods (Cash and Karp, 1990). In numerical integration, the distance divided by the cost is important. Another advantage of RKF methods is that of flexibility of the step-size selection (Cash and Karp, 1990).

The scaling of h is possibly achieved by combining two adjacent-order RK methods into one and keeping the local truncation errors within some tolerance. The local truncation errors are obtained from the difference between the two adjacent-order RK methods. If the step is accepted, the solution of the higher order method is used for the next integration step. An example of RKF method is the embedding of RK7 into RK8 to produce the RKF7(8) method. The general form of function evaluations of RKF methods is

$$\begin{aligned} k1 &= f(t_n, y_n) \\ k2 &= f(t_n + a_2 h, y_n + b_{21} k1) \\ k3 &= f(t_n + a_3 h, y_n + b_{31} k1 + b_{32} k2) \\ &... \\ k_s &= f(t_n + a_s h, y_n + b_{s1} k1 + b_{s2} k2 + ... + b_{s,s-1} k_{s-1}) \end{aligned}$$

where $s = \{4, 5..., 13\}$ for the RKF7(8) method. Unlike the Runge-Kutta methods for orders less than four, the number of function evaluations for higher order RKF methods does not correspond to the order of the method. The formulas for the two adjacent-order RK methods are

$$y_{n+1} = y_n + h \sum_{i=1}^{s-2} c_i k_i + O(h^{p+2})$$

and

$$y^*_{n+1} = y_n + h \sum_{i=1}^{s} c^*_i k_i + O(h^{p+1}) \tag{5.5}$$

where p is the order of the higher RK method. The error estimate is

$$e = \| \, y_{n+1} - y^*_{n+1} \, \| = \sum_{i=1}^{s} (c_i - c^*_i) k_i \tag{5.6}$$

The values of the various constants are given in Table 5.5. The step-size is calculated using

$$h_{new} = 0.8 * h_{old} \left(\frac{tol}{e} \right)^{\frac{1}{p+1}} \tag{5.7}$$

The vector norm in e is due to the fact that y may be a vector. If that's the case, then h_{new} is rescaled according to the needs of the "worst-offender" equation (Press *et al.*, 2002).

5.5 Software Structure

Figure 5.3 outlines the programme structure of *PRECurSOr* followed by the list of functions and their description in Table 5.4. Appendix F documents a few important functions. Some of the functions, e.g. **app_sidereal_time**, **nutation**, **obliquity**, **sfd70** and **jsmade** were obtained from existing libraries. Although some of them were modified, the need to create new functions serving the same operation serves no meaningful purpose, since these functions are well-tried.

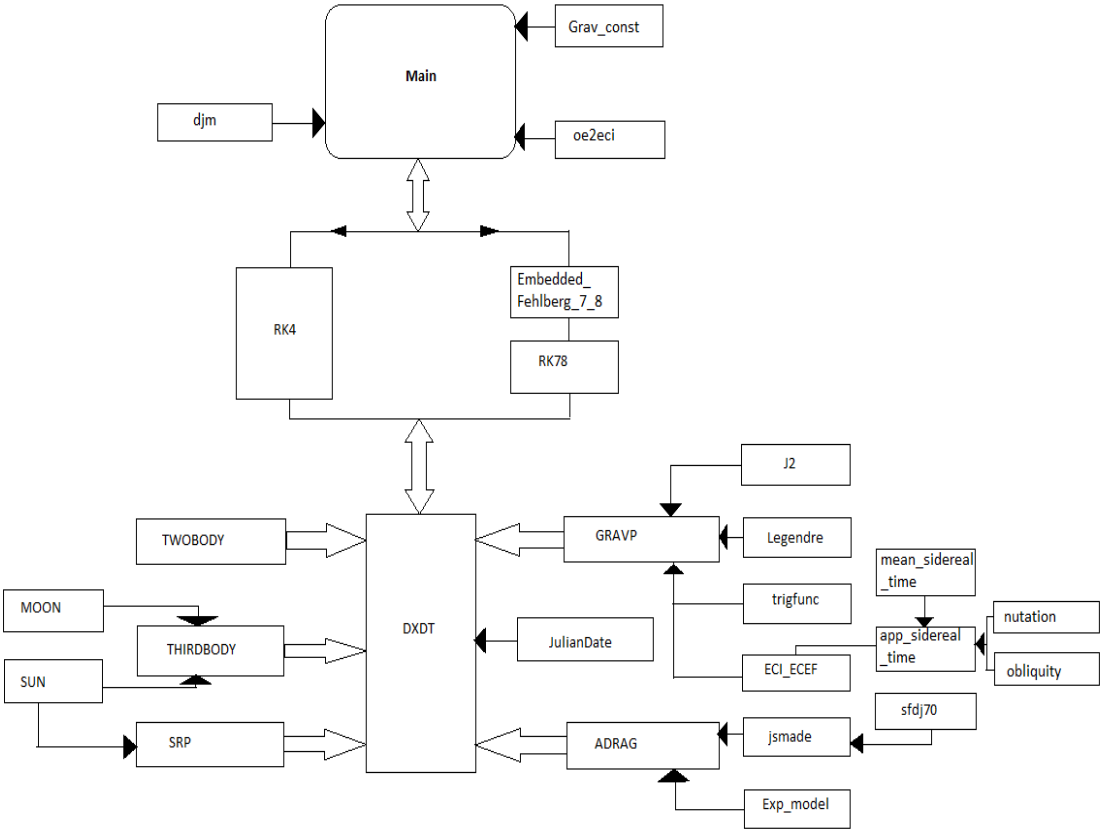

Figure 5.3: PRECurSOr programme structure

5.6 Summary

The initial conditions to validate the *PRECurSOr* and data files considered were described in this chapter. The C++ programming language running on NetBeans IDE was selected for the development of *PRECurSOr*. A Raspberry Pi was implemented as a representation of an OBC.

Chapter 6 presents the results in the form of software validation to demonstrate the performance of *PRECurSOr* as compared to other propagators.

Table 5.4: Functions used in PRECurSOr

Function	Description
djm	*Calculates modified Julian date for a given (Gregorian) calender date and time*
JulianDate	*Calculates Julian date for a given (Gregorian) calender date and time*
Grav_const	*Reads and unnormalise gravity constants*
oe2eci	*Transforms classical orbital elements to ECI coordinate system*
RK4	*Classical Runge-Kutta method*
RK78	*Runge-Kutta 7(8) method*
Embedded_Fehlberg_7_8	*Computes the Fehlberg coefficients for RK78*
DXDT	*Computes derivatives of the equations of motion*
MOON	*Computes the position vector of the moon in ECI system*
SUN	*Computes the position vector of the Sun in ECI system*
Legendre	*Computes Legendre and associated Legendre functions*
trigfunc	*Computes latitude and longitude trigonometric terms by recursion*
ECI_ECEF	*Transforms ECI system to ECEF for latitude and longitude computation*
Exp_model	*Computes atmospheric density for altitudes between 90-100 km using the exponential model*
mean_sidereal_time	*Computes mean sidereal time at the meridian of Greenwich*
app_sidereal_time	*Computes apparent sidereal time at the meridian of Greenwich*
nutation	*Computes the nutation of longitude and nutation of obliquity of the ecliptic*
obliquity	*Computes the mean and true obliquity of the ecliptic*
jsmade	*Computes atmospheric density for heights from 100 to 2000 km, using the jacchia 70 static model*
sfdj70	*Retrieves the mean solar flux and geo-physical data at modified JD*
TWOBODY	*Computes the Cartesian ECI position of the initial two-body motion*
GRAVP	*Computes aspherical Earth perturbation using spherical harmonics*
THIRDBODY	*Computes luni-solar third body perturbations*
ADRAG	*Computes atmospheric drag perturbation*
SRP	*Computes solar radiation pressure perturbation*

Table 5.5: Parameters for the Embedded Runge-Kutta 7(8) Method.

	a_k	b_{kl}												c_k	c_k^*
l/k		1	2	3	4	5	6	7	8	9	10	11	12		
1	0	0												$\frac{41}{480}$	0
2	$\frac{2}{27}$	$\frac{2}{27}$												0	0
3	$\frac{1}{9}$	$\frac{1}{36}$	$\frac{1}{12}$											0	0
4	$\frac{1}{6}$	$\frac{1}{24}$	0	$\frac{1}{8}$										0	0
5	$\frac{5}{12}$	$\frac{5}{12}$	0	$-\frac{25}{16}$	$\frac{25}{16}$									0	0
6	$\frac{1}{2}$	$\frac{1}{20}$	0	0	$\frac{1}{4}$	$\frac{1}{5}$								$\frac{34}{105}$	$\frac{34}{105}$
7	$\frac{5}{6}$	$-\frac{25}{108}$	0	0	$\frac{125}{108}$	$-\frac{65}{27}$	$\frac{125}{54}$							$\frac{9}{35}$	$\frac{9}{35}$
8	$\frac{1}{6}$	$\frac{31}{300}$	0	0	0	$\frac{61}{225}$	$-\frac{2}{9}$	$\frac{13}{900}$						$\frac{9}{35}$	$\frac{9}{35}$
9	$\frac{2}{3}$	2	0	0	$-\frac{53}{6}$	$\frac{704}{45}$	$-\frac{107}{9}$	$\frac{67}{90}$	3					$\frac{9}{280}$	$\frac{9}{280}$
10	$\frac{1}{3}$	$-\frac{91}{108}$	0	0	$\frac{23}{108}$	$-\frac{976}{135}$	$\frac{311}{54}$	$-\frac{19}{60}$	$\frac{17}{6}$	$-\frac{1}{12}$				$\frac{9}{280}$	$\frac{9}{280}$
11	1	$\frac{2383}{4100}$	0	0	$-\frac{341}{164}$	$\frac{4496}{1025}$	$-\frac{301}{82}$	$\frac{2133}{4100}$	$\frac{45}{82}$	$\frac{45}{164}$	$\frac{18}{41}$			$\frac{41}{840}$	0
12	0	$\frac{3}{205}$	0	0	0	0	$-\frac{6}{41}$	$-\frac{3}{205}$	$-\frac{3}{41}$	$\frac{3}{41}$	$\frac{6}{41}$	0		0	$\frac{41}{840}$
13	1	$-\frac{1777}{4100}$	0	0	$-\frac{341}{164}$	$\frac{4496}{1025}$	$-\frac{289}{82}$	$\frac{2193}{4100}$	$\frac{51}{82}$	$\frac{33}{164}$	$\frac{12}{41}$	0	1	0	$\frac{41}{840}$

Chapter 6

Results : Software Validation

6.1 Introduction

The validation of software is important in ensuring the developed software correctly performs its required tasks. This is achieved by comparing results produced by the propagator with those of its counterparts. The results of the *PRECurSOr* software were validated against HPOP (a package of STK) and SLR-derived reference orbit of SUNSAT. According to Vallado (1997), modern laser-tracking systems e.g. SLR, permit very accurate analysis of how a well propagation routine models the real world. The validation of *PRECurSOr* is performed by considering the sequential addition of perturbation forces to the initial two-body solution.

6.2 Force Models Versus Processing Time

Table 6.1 presents the trade-off between the model complexity and its accuracy. The perturbation forces were added sequentially to the initial two-body equation of motion. The ptrocessing time (t_p) was recorded from PiBang Linux on its terminal window. The position and velocity absolute errors are represented by Δr and Δv, respectively. These

are calculated using the SLR-derived orbit as reference. Figure 6.1 is the graphical representation of the absolute errors in position compared to the processing time of Raspberry Pi computer.

i

Table 6.1: One-day PRECurSOr force models versus processing time (t_p) evaluation using different integrators and SLR-derived orbit as reference

	RK4 ($h = 60$ sec)			RK4 ($h = 30$ sec)			RKF78		
	t_p sec	Δr (m)	Δv (m/s)	t_p sec	Δr (m)	Δv (m/s)	t_p sec	Δr (m)	Δv (m/s)
TWOBODY	11.26	274985.31	286.22	12.15	276327.53	287.63	11.58	276376.91	287.68
MOON	11.26	274985.31	286.22	12.30	276327.53	287.63	11.69	276376.91	287.68
SUN	11.35	274985.31	286.22	12.65	276327.53	287.63	11.97	276376.91	287.68
DRAG	18.44	274929.33	286.16	27.76	276271.58	287.57	22.39	276320.97	287.62
J2	18.53	4584.23	5.21	27.91	3206.46	3.76	22.64	3155.75	3.70
10×10	19.13	1331.11	1.41	29.01	56.14	0.05	23.16	105.67	105.6
20×20	20.04	1117.32	1.17	31.60	266.82	0.29	24.38	317.43	0.34
60×60	30.06	1008.25	1.06	51.18	375.04	0.40	40.21	426.15	0.45
SRP	30.73	1007.41	1.06	51.73	375.65	0.40	42.13	427.05	0.45

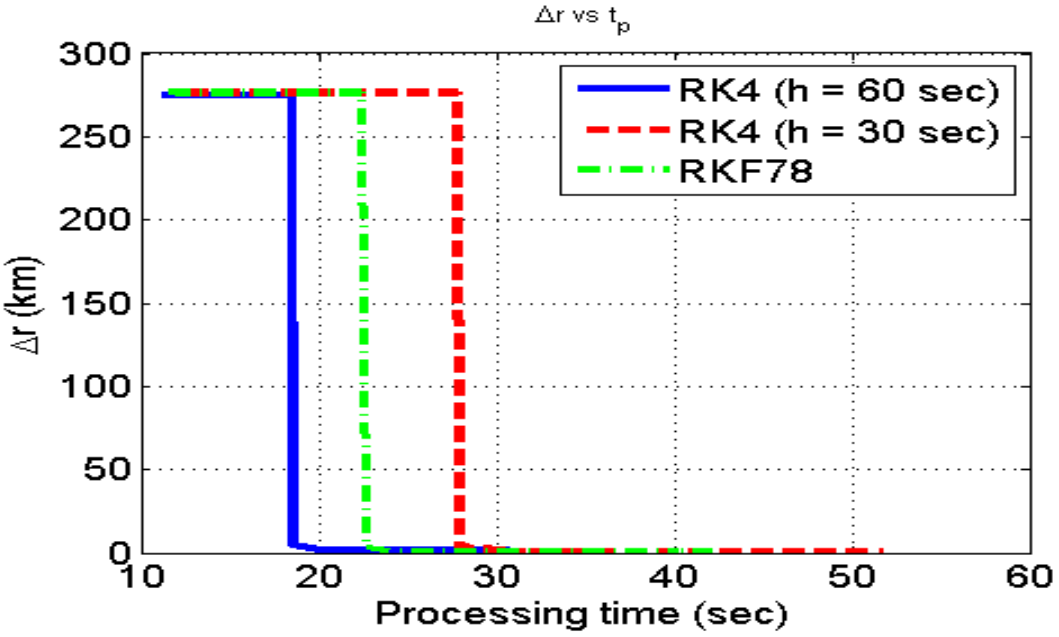

Figure 6.1: PRECurSOr processing time versus accuracy.

6.3 Validation of *PRECurSOr*

6.3.1 Two-body and Perturbation Forces

In Figure 6.2 *PRECurSOr* with a two-body solution is validated by comparing its absolute errors in position and velocity with that of STK. This is a one-day propagation by *PRECurSOr* (two-body) using the RK4 ($h = 60$ *sec*) integrator. Absolute errors are obtained using SLR-derived orbit from processed data as reference.

Figure 6.3 presents results of absolute errors in position and velocity of *PRECurSOr* and STK's HPOP, using SLR-derived data as the reference orbit. Both propagators include J_2 perturbation and a 20×20 gravity field model to the initial two-body solution. Results of *PRECurSOr* using a 60×60 gravity model are also shown. The RK4 ($h = 60$ *sec*) integrator was used for this propagation.

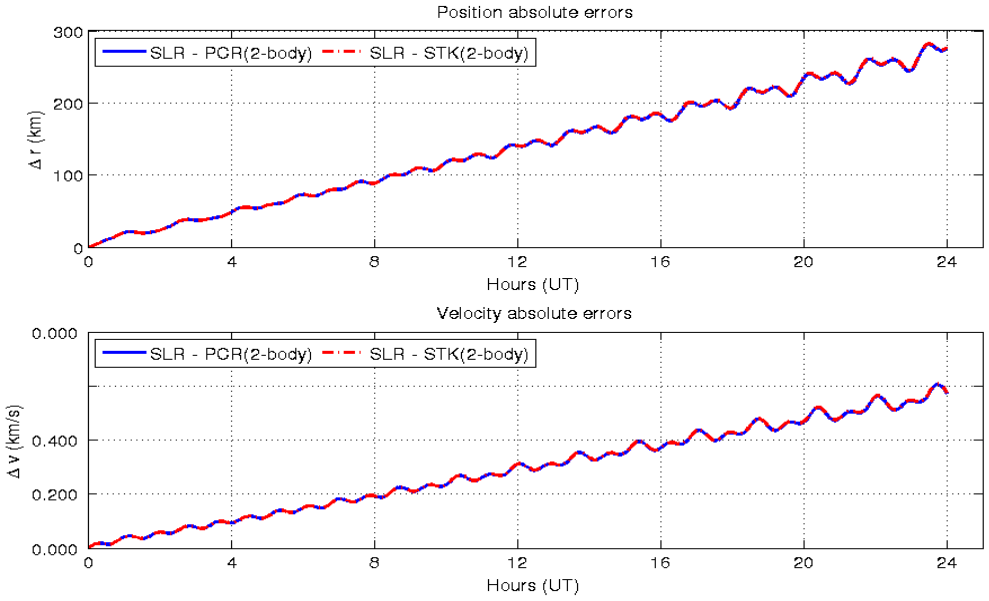

Figure 6.2: Absolute errors in position and velocity of two-body propagators compared to SUNSAT SLR-derived data

The results of atmospheric drag validation are shown in Figure 6.4. The Jacchia-Roberts atmospheric model was used for density calculations in STK, while for *PRECurSOr*, the Jacchia 70 static atmospheric model was used. In addition, results by *PRECurSOr* using

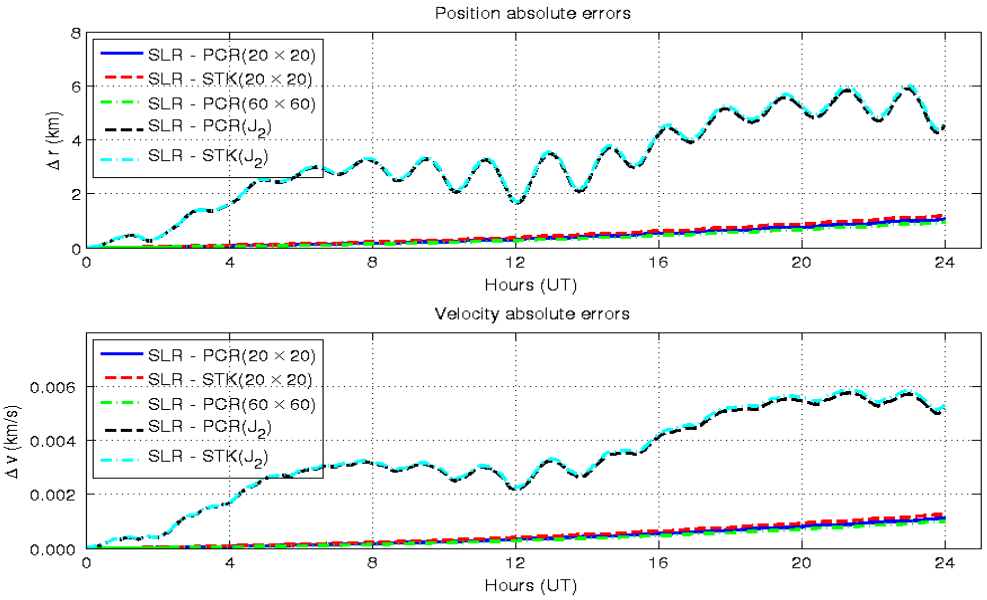

Figure 6.3: Absolute errors in position and velocity of PRECurSOr and STK's HPOP with J_2 perturbation and 20×20 gravity field model

10×10 and 60×60 gravity fields with drag are shown. A comparison of the state vectors of the two propagators (using 20×20 gravity models) is tabulated in Table 6.2.

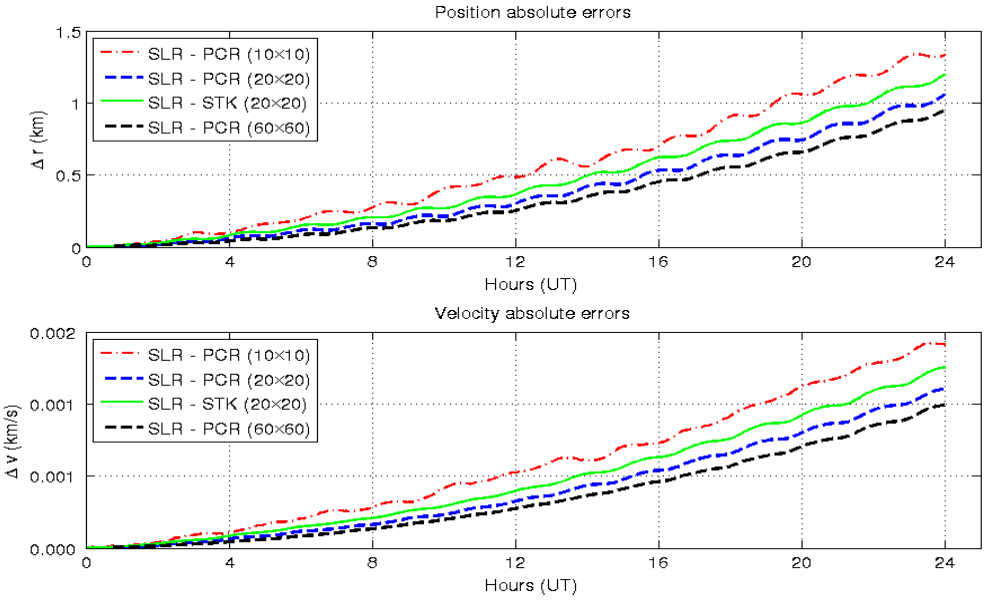

Figure 6.4: Absolute errors in position and velocity by PRECurSOr and STK with the presence of drag

6.3.2 SUNSAT Orbit

Figure 6.5 illustrates the ground track of one orbit of SUNSAT as determined by STK, SLR and *PRECurSOr*. A 3D space orbit of SUNSAT as determined by SLR and *PRECurSOr* follows in Figure 6.6. The positions of SUNSAT after one day of propagation using *PRECurSOr* compared to that obtained from the SLR-derived orbit are also marked on the figure.

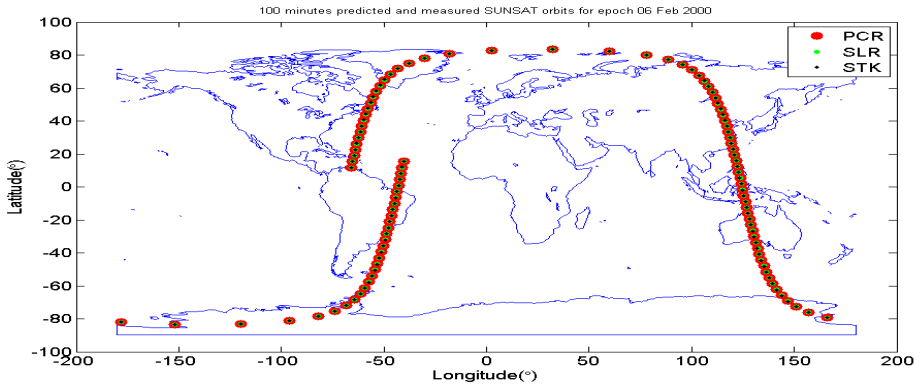

Figure 6.5: SUNSAT ground track comparison of the measured SLR orbit against predicted STK and *PRECurSOr* orbits

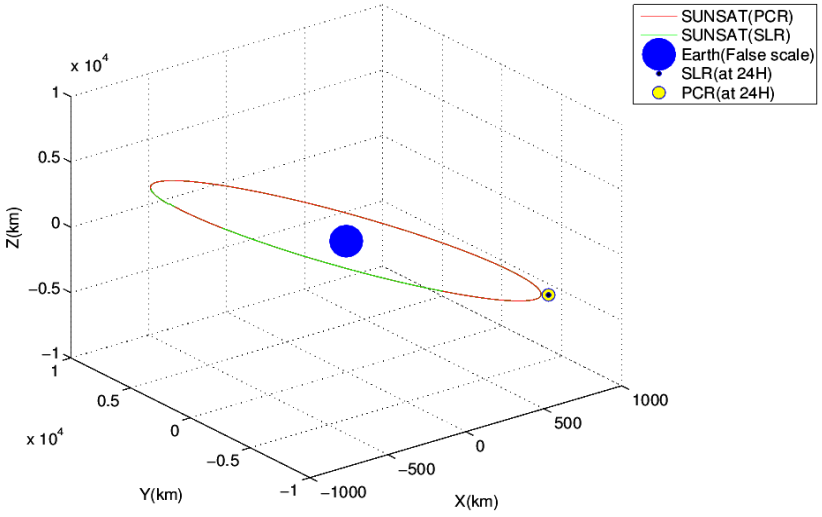

Figure 6.6: 100 minutes measured and predicted SUNSAT orbit with the position of the satellite after 24 hours as determined by SLR and *PRECurSOr*

6.4 Analysis of Integrators

The absolute errors in position and velocity of *PRECurSOr* after one day, using the RK4 (h = 30 *sec*), RK4 (h = 60 *sec*) and RKF78 integrators are shown graphically in Figure 6.7. The results of each integrator are plotted using perturbation forces returning minimum absolute errors. Since RKF78 is a multi-step integrator, MATLAB spline interpolation function was used for off-line interpolation and plotting of RKF78 data. Figure 6.8 indicates how the RKF78 integrator step-size changes over a one-day propagation period.

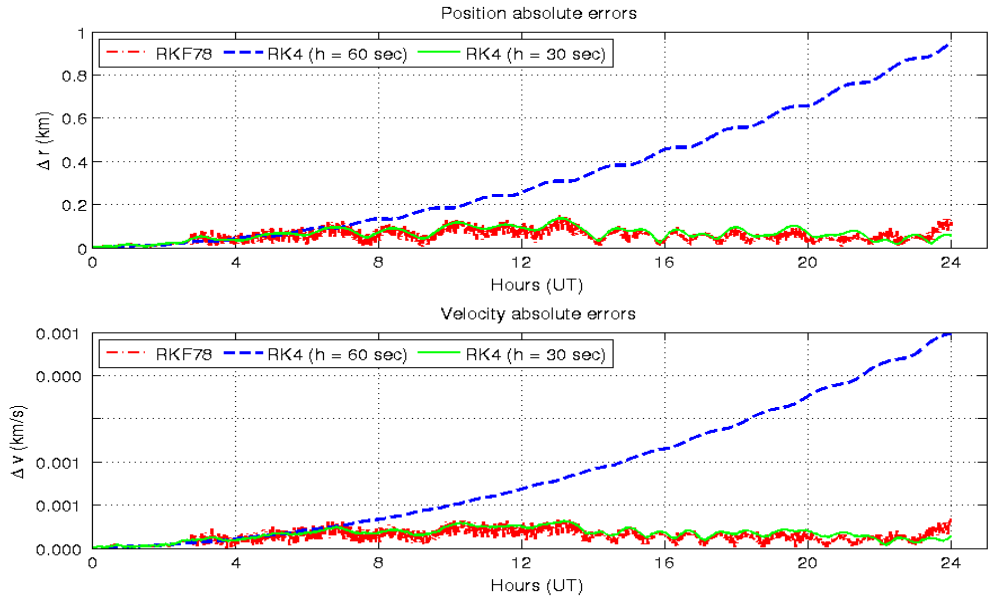

Figure 6.7: Absolute error in position and velocity of *PRECurSOr* compared to SUNSAT SLR-derived orbit, using step sizes of 20, 30 and 60 seconds

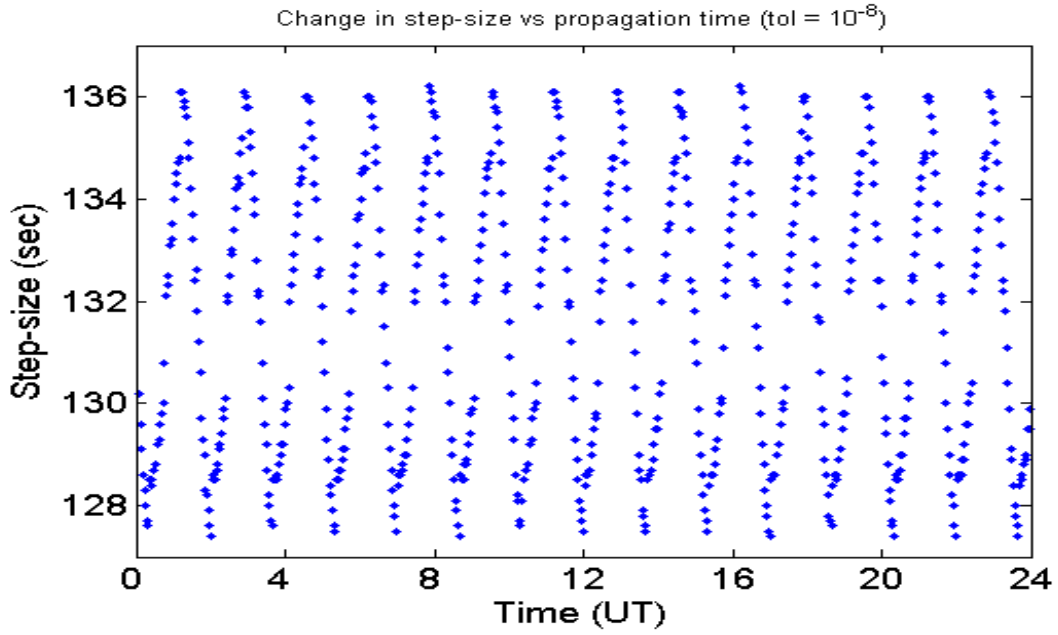

Figure 6.8: Step-size (h) changes in RKF78 with the error tolerance of 10^{-8}, over a one-day propagation period

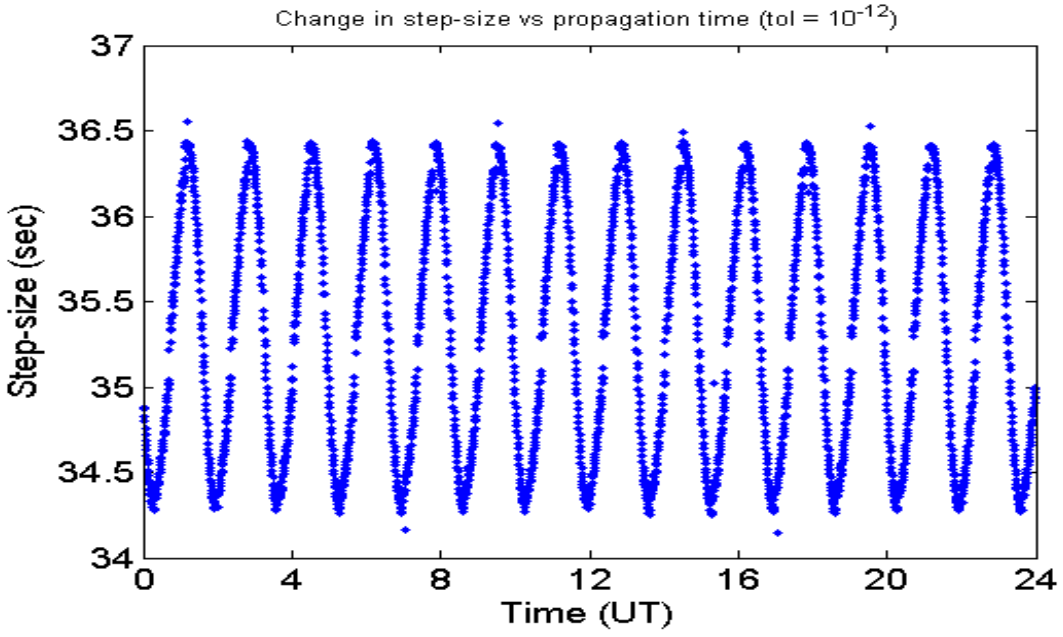

Figure 6.9: Step-size (h) changes in RKF78 with the error tolerance of 10^{-12}, over a one-day propagation period

6.5 Discussion

As shown in Table 6.1 and Figure 6.1, the RK4 ($h = 60$ *sec*) integrator processes the results with all perturbation forces considered, in the shortest period compared to the RK4 ($h = 30$ *sec*) and RKF78 integrators. The RK4 ($h = 30$ *sec*) integrator yields better accuracy

(smaller absolute errors in position and velocity) in 29.01 seconds, with luni-solar, J_2 and 10×10 perturbations considered. Luni-solar perturbations could subsequently be ignored for a one-day propagation, due to the negligible improvement in accuracy and the relatively high computational cost.

Propagation done by the RK4 ($h = 30$ *sec*) integrator with J_2, 10×10 and SRP perturbations is ultimately the best option compared to the RKF78 and RK4 ($h = 60$ *sec*) integrators, for this one-day test case, considering the propagator's accuracy and computation time. These results are tabulated in Table 6.3. However, if better accuracy is required within the first 4 hours of propagation, the RK4 ($h = 30$ *sec*) integrator with a 20×20 gravity field model and atmospheric drag may be used. This is because the errors obtained by the RK4 ($h = 30$ *sec*) integrator with J_2, 10×10 and SPR perturbations don't increase linearly as shown in Figure 6.7.

The adaptive step-size RKF78 integrator yields smaller absolute errors propagating with step-sizes between 127 and 136 seconds, with the error tolerance of 10^{-8}, as shown in Figure 6.8. When the error tolerance is decreased to 10^{-12} it is found that the step-sizes decrease to 32 - 37 seconds (Figure 6.9), but the solution's accuracy does not significantly improve. The step-size increases when the satellite's orbital speed decreases (towards apogee) and decreases towards perigee where the orbital speed is at a maximum. For the given SUNSAT orbital period of about 100 minutes, the number of positive and negative peaks in both figures correspond to the number of times SUNSAT orbits the Earth through apogee and perigee respectively over the integration period of one day (i.e. about 14 times per day each).

The introduction of a 60×60 gravity field model using the RK4 ($h = 30$ *sec*) and RKF78 integrators does not improve the accuracy of *PRECurSOr*. This is due to the accumulation of errors by integrators and the step-size selection in case of the RK4 integrator. It is evident from this discrepancy that in numerical integration, for this test case and the propagators used, higher order or smaller step-size does not necessarily render better accuracy or performance for one-day propagation. The validation of *PRECurSOr* using STK with SLR-derived orbit as reference indicates that *PRECurSOr* is more accurate than STK's HPOP.

Table 6.2: Position and velocity absolute errors of PRECurSOr (RK4, h = 60 *sec*) and STK (RK4, h = 60 *sec*)

Time (EpochHr)	Program	X(m)	Y(m)	Z(m)	V_x(m/s)	V_y(m/s)	V_z(m/s)	$\Delta\mathbf{r}$(m)	$\Delta\mathbf{v}$(m/s)
1 Hour	SLR	73029.63066	-6656914.39	2798448.421	-933.056169	2779.862946	6777.740693		
	PCR	73027.12247	-6656911.54	2798450.423	-933.0565583	2779.868175	6777.740988	4.28914	0.00525
	STK	73027.209	-6656907.012	2798456.893	-933.054901	2779.874611	6777.739729	11.4894	0.01177
2 Hours	SLR	453429.1835	3738183.610	-5938731.48	830.5028808	-6401.54511	-3969.92157		
	PCR	453435.3635	3738168.526	-5938736.339	830.5053541	-6401.558493	-3969.906913	17.0095	0.02000
	STK	453432.894	3738160.761	-5938741.209	830.502948	-6401.563476	-3969.898033	25.1095	0.02986
3 Hours	SLR	-833949.512	342335.4591	7155952.778	-387.961040	7361.554612	-463.959568		
	PCR	-833959.8703	342373.0969	7155948.931	-387.9554048	7361.552923	-463.9966656	39.2262	0.03756
	STK	-833955.457	342390.663	7155947.904	-387.954619	7361.552108	-464.014139	55.7366	0.05500
4 Hours	SLR	882458.4015	-4690605.05	-5270183.71	-210.585211	-5686.73168	4881.027073		
	PCR	882462.7803	-4690645.128	-5270136.978	-210.6019434	-5686.696316	4881.077731	61.7129	0.06401
	STK	882457.954	-4690658.602	-5270123.952	-210.598659	-5686.683435	4881.095293	80.2369	0.08463
5 Hours	SLR	-627911.147	6820981.821	1865602.940	702.2448718	1937.867783	-7225.03629		
	PCR	-627905.2444	6821000.783	1865530.213	702.2669666	1937.787937	-7225.05514	75.3900	0.08496
	STK	-627902.538	6821007.678	1865504.051	702.259904	1937.760661	-7225.063787	102.576	0.11161
6 Hours	SLR	88987.01876	-6650580.02	2815144.524	-940.682367	2794.998693	6769.481014		
	PCR	88963.56969	-6650529.846	2815249.077	-940.6977952	2795.107585	6769.438509	118.315	0.11791
	STK	88966.336	-6650516.193	2815277.160	-940.689020	2795.137223	6769.428643	148.639	0.14825
7 Hours	SLR	445878.9457	3720185.794	-5950586.40	846.6741233	-6411.33533	-3950.54040		
	PCR	445908.2439	3720077.964	-5950643.044	846.6728863	-6411.413954	-3950.42521	125.277	0.13947
	STK	445900.153	3720048.443	-5950662.823	846.665948	-6411.432538	-3950.393730	158.606	0.17615
8 Hours	SLR	-836177.037	358428.8881	7153804.333	-405.944999	7360.383301	-484.093557		
	PCR	-836206.7621	358591.8599	7153791.924	-405.9198706	7360.372284	-484.2591822	166.124	0.16788
	STK	-836194.891	358633.005	7153789.619	-405.919391	7360.370880	-484.300833	205.424	0.20922
9 Hours	SLR	895401.5684	-4705605.32	-5256659.08	-197.976076	-5671.28784	4897.062326		
	PCR	895408.623	-4705750.828	-5256511.917	-198.0210696	-5671.157556	4897.226179	207.076	0.21411
	STK	895398.471	-4705781.707	-5256485.730	-198.012799	-5671.128003	4897.261417	247.333	0.25794
10 Hours	SLR	-644552.285	6823913.005	1842610.834	699.6225659	1916.995141	-7232.75563		
	PCR	-644533.413	6823967.136	1842400.5	699.6705761	1916.76728	-7232.812615	218.006	0.23974
	STK	-644528.558	6823979.849	1842351.555	699.656230	1916.716248	-7232.828738	268.806	0.29028
11 Hours	SLR	104847.6662	-6643350.13	2833744.264	-948.964750	2812.325319	6760.186593		
	PCR	104791.6852	-6643232.705	2834006.64	-948.994208	2812.593876	6760.073574	292.855	0.29285
	STK	104797.668	-6643211.187	2834052.982	-948.978474	2812.643208	6760.056421	342.217	0.34378
12 Hours	SLR	438663.3430	3699094.493	-5964326.49	862.7264156	-6422.89251	-3928.04697		
	PCR	438728.2933	3698826.314	-5964469.618	862.7168385	-6423.080826	-3927.766592	310.843	0.33789
	STK	438714.195	3698778.186	-5964502.162	862.705824	-6423.110699	-3927.715889	365.370	0.39705
13 Hours	SLR	-838138.166	376386.5011	7151721.988	-423.637199	7358.866662	-506.161339		
	PCR	-838190.8791	376750.5463	7151694.178	-423.5813118	7358.842969	-506.5357927	368.891	0.37934
	STK	-838171.504	376811.422	7151690.730	-423.581648	7358.841020	-506.597656	427.371	0.44058
14 Hours	SLR	907736.7345	-4721806.41	-5242146.74	-185.138074	-5654.67849	4914.244797		
	PCR	907743.3574	-4722122.269	-5241843.704	-185.2250681	-5654.39187	4914.585718	437.767	0.45382
	STK	907728.101	-4722167.413	-5241806.923	-185.211417	-5654.348336	4914.635346	495.854	0.51664
15 Hours	SLR	-661134.755	6826960.566	1820214.839	696.2149934	1896.250676	-7239.97616		
	PCR	-661093.1141	6827070.066	1819775.588	696.2999314	1895.78054	-7240.098609	454.606	0.49319
	STK	-661086.450	6827087.457	1819707.761	696.278123	1895.709816	-7240.120906	524.942	0.56344
16 Hours	SLR	120992.5493	-6636208.09	2851769.651	-956.606129	2828.857451	6751.294353		
	PCR	120897.6885	-6635988.719	2852272.316	-956.6481674	2829.373625	6751.071778	556.593	0.56369
	STK	120907.185	-6635960.797	2852333.043	-956.625559	2829.438729	6751.048948	621.171	0.63126
17 Hours	SLR	430790.6428	3680808.750	-5976571.19	879.0767519	-6432.57954	-3907.95884		
	PCR	430896.9636	3680300.862	-5976851.911	879.0548732	-6432.925604	-3907.426794	589.961	0.63507
	STK	430876.739	3680237.698	-5976894.907	879.039958	-6432.964274	-3907.360641	662.043	0.71219
18 Hours	SLR	-840071.032	393089.6939	7149592.025	-442.016582	7357.399224	-526.634761		
	PCR	-840153.3399	393746.9661	7149541.766	-441.9242889	7357.355478	-527.3103423	664.309	0.68326
	STK	-840126.451	393823.574	7149537.339	-441.925572	7357.353142	-527.388169	737.998	0.76028
19 Hours	SLR	920599.0886	-4735524.35	-5229345.32	-172.263718	-5639.93041	4929.468963		
	PCR	920601.6978	-4736071.883	-5228828.178	-172.4027328	-5639.429378	4930.054303	753.152	0.78293
	STK	920581.346	-4736128.154	-5228783.434	-172.383697	-5639.374711	4930.115052	824.993	0.86059
20 Hours	SLR	-677950.830	6829647.345	1798712.771	693.5216679	1876.595024	-7246.84054		
	PCR	-677878.3837	6829837.012	1797963.21	693.6486873	1875.7948	-7247.045673	776.572	0.83581
	STK	-677869.929	6829858.020	1797880.433	693.619474	1875.708414	-7247.072913	862.389	0.92176
21 Hours	SLR	136542.7592	-6629013.54	2869409.092	-965.110952	2845.048921	6742.506767		
	PCR	136395.2022	-6628660.053	2870217.796	-965.1680567	2845.88265	6742.14599	894.834	0.91023
	STK	136408.135	-6628627.123	2870289.328	-965.138594	2845.959881	6742.118988	970.699	0.99045
22 Hours	SLR	423912.5491	3660989.721	-5989106.17	895.3884566	-6443.22398	-3886.72854		
	PCR	424067.6255	3660188.095	-5989557.78	895.3454844	-6443.762163	-3885.88098	933.061	1.00491
	STK	424041.251	3660113.212	-5989609.020	895.326682	-6443.807338	-3885.802964	1018.67	1.09582
23 Hours	SLR	-842667.618	410715.4574	7147333.001	-459.541019	7355.714865	-548.394316		
	PCR	-842780.3469	411734.721	7147254.644	-459.4018379	7355.645641	-549.4386904	1028.47	1.05588
	STK	-842745.958	411823.196	7147249.416	-459.404102	7355.643074	-549.528430	1113.65	1.14460
24 Hours	SLR	933315.9401	-4751747.19	-5214530.62	-160.155418	-5623.09584	4946.559678		
	PCR	933316.3829	-4752559.898	-5213765.086	-160.3545332	-5622.342443	4947.42957	1116.49	1.16789
	STK	933291.017	-4752624.172	-5213715.048	-160.329975	-5622.279410	4947.498143	1197.86	1.25608

Table 6.3: Position and velocity absolute errors of PRECurSOr (RK4 (h = 30 *sec*) with a 10×10 gravity field model and drag model)

Time (EpochHr)	Program	X(m)	Y(m)	Z(m)	V_x(m/s)	V_y(m/s)	V_z(m/s)	$\Delta\mathbf{r}$(m)	$\Delta\mathbf{v}$(m/s)
1 Hour	SLR	73029.63066	-6656914.39	2798448.421	-933.056169	2779.862946	6777.740693		
	PCR	73019.63271	-6656916.876	2798438.664	-933.0581866	2779.857225	6777.744214	14.19	0.007015
2 Hours	SLR	453429.1835	3738183.610	-5938731.48	830.5028808	-6401.54511	-3969.92157		
	PCR	453443.0839	3738173.731	-5938737.313	830.5124295	-6401.550187	-3969.911738	18.023	0.01462
3 Hours	SLR	-833949.512	342335.4591	7155952.778	-387.961040	7361.554612	-463.959568		
	PCR	-833966.3568	342382.3784	7155945.76	-387.9621888	7361.554625	-464.0005261	50.343	0.04097
4 Hours	SLR	882458.4015	-4690605.05	-5270183.71	-210.585211	-5686.73168	4881.027073		
	PCR	882473.1998	-4690625.756	-5270160.446	-210.6003845	-5686.710518	4881.051358	34.475	0.03561
5 Hours	SLR	-627911.147	6820981.821	1865602.940	702.2448718	1937.867783	-7225.03629		
	PCR	-627909.8683	6820995.641	1865540.851	702.2683663	1937.811087	-7225.052703	63.622	0.06353
6 Hours	SLR	88987.01876	-6650580.02	2815144.524	-940.682367	2794.998693	6769.481014		
	PCR	88980.06765	-6650554.774	2815196.441	-940.7036954	2795.062443	6769.456112	58.146	0.07169
7 Hours	SLR	445878.9457	3720185.794	-5950586.40	846.6741233	-6411.33533	-3950.54040		
	PCR	445902.6618	3720112.606	-5950635.966	846.6850642	-6411.377057	-3950.459382	91.52	0.09179
8 Hours	SLR	-836177.037	358428.8881	7153804.333	-405.944999	7360.383301	-484.093557		
	PCR	-836202.4037	358491.1572	7153786.44	-405.9434872	7360.390776	-484.1637445	69.578	0.0706
9 Hours	SLR	895401.5684	-4705605.32	-5256659.08	-197.976076	-5671.28784	4897.062326		
	PCR	895420.675	-4705647.748	-5256630.453	-197.9939633	-5671.242226	4897.098442	54.635	0.06087
10 Hours	SLR	-644552.285	6823913.005	1842610.834	699.6225659	1916.995141	-7232.75563		
	PCR	-644550.5666	6823932.127	1842502.779	699.6526297	1916.899099	-7232.788317	109.75	0.1058
11 Hours	SLR	104847.6662	-6643350.13	2833744.264	-948.964750	2812.325319	6760.186593		
	PCR	104823.424	-6643318.407	2833820.557	-948.9859825	2812.415886	6760.147088	86.109	0.1011
12 Hours	SLR	438663.3430	3699094.493	-5964326.49	862.7264156	-6422.89251	-3928.04697		
	PCR	438697.8955	3699024.803	-5964376.214	862.739926	-6422.935198	-3927.958858	92.318	0.09884
13 Hours	SLR	-838138.166	376386.5011	7151721.988	-423.637199	7358.866662	-506.161339		
	PCR	-838180.9355	376511.3626	7151701.423	-423.6164591	7358.868534	-506.2791574	133.58	0.1196
14 Hours	SLR	907736.7345	-4721806.41	-5242146.74	-185.138074	-5654.67849	4914.244797		
	PCR	907765.822	-4721833.843	-5242124.183	-185.1851355	-5654.6406	4914.275493	45.906	0.06777
15 Hours	SLR	-661134.755	6826960.566	1820214.839	696.2149934	1896.250676	-7239.97616		
	PCR	-661134.8487	6826978.274	1820139.814	696.2831159	1896.184878	-7239.988038	77.087	0.09545
16 Hours	SLR	120992.5493	-6636208.09	2851769.651	-956.606129	2828.857451	6751.294353		
	PCR	120960.2467	-6636189.598	2851789.573	-956.667192	2828.893217	6751.279448	42.219	0.07232
17 Hours	SLR	430790.6428	3680808.750	-5976571.19	879.0767519	-6432.57954	-3907.95884		
	PCR	430847.7433	3680791.105	-5976588.151	879.0924413	-6432.592654	-3907.919377	62.123	0.04445
18 Hours	SLR	-840071.032	393089.6939	7149592.025	-442.016582	7357.399224	-526.634761		
	PCR	-840127.0251	393164.6931	7149578.91	-442.0009215	7357.397907	-526.6986111	94.51	0.06576
19 Hours	SLR	920599.0886	-4735524.35	-5229345.32	-172.263718	-5639.93041	4929.468963		
	PCR	920627.4786	-4735550.934	-5229301.538	-172.3000421	-5639.904109	4929.518366	58.564	0.06672
20 Hours	SLR	-677950.830	6829647.345	1798712.771	693.5216679	1876.595024	-7246.84054		
	PCR	-677957.7026	6829676.013	1798654.703	693.5656879	1876.527993	-7246.838391	65.123	0.08022
21 Hours	SLR	136542.7592	-6629013.54	2869409.092	-965.110952	2845.048921	6742.506767		
	PCR	136523.9387	-6628985.217	2869464.894	-965.1536716	2845.104483	6742.484576	65.347	0.07352
22 Hours	SLR	423912.5491	3660989.721	-5989106.17	895.3884566	-6443.22398	-3886.72854		
	PCR	423940.9208	3660988.65	-5989116.154	895.4131527	-6443.22376	-3886.70794	30.095	0.03216
23 Hours	SLR	-842667.618	410715.4574	7147333.001	-459.541019	7355.714865	-548.394316		
	PCR	-842713.7382	410753.1994	7147327.729	-459.5399664	7355.713136	-548.4199293	59.827	0.02569
24 Hours	SLR	933315.9401	-4751747.19	-5214530.62	-160.155418	-5623.09584	4946.559678		
	PCR	933340.4851	-4751708.24	-5214562.742	-160.1803242	-5623.127237	4946.52371	56.136	0.05385

Table 6.4: Position and velocity absolute errors of PRECurSOr (RKF78, with a 10×10 gravity field model and drag model)

Time (EpochHr)	Program	X(m)	Y(m)	Z(m)	V_x(m/s)	V_y(m/s)	V_z(m/s)	$\Delta\mathbf{r}$(m)	$\Delta\mathbf{v}$(m/s)
1 Hour	SLR	73029.63066	-6656914.39	2798448.421	-933.056169	2779.862946	6777.740693		
	PCR	73020.00023	-6656918.15	2798436.085	-933.0581234	2779.854436	6777.745102	16.09	0.00978
2 Hours	SLR	453429.1835	3738183.610	-5938731.48	830.5028808	-6401.54511	-3969.92157		
	PCR	453442.9917	3738174.572	-5938736.939	830.5124907	-6401.549492	-3969.912645	17.382	0.01383
3 Hours	SLR	-833949.512	342335.4591	7155952.778	-387.961040	7361.554612	-463.959568		
	PCR	-833966.1479	342378.5557	7155945.936	-387.9626126	7361.554564	-463.9967484	46.7	0.03721
4 Hours	SLR	882458.4015	-4690605.05	-5270183.71	-210.585211	-5686.73168	4881.027073		
	PCR	882472.7922	-4690640.513	-5270149.01	-210.6028088	-5686.697847	4881.065979	51.654	0.05448
5 Hours	SLR	-627911.147	6820981.821	1865602.940	702.2448718	1937.867783	-7225.03629		
	PCR	-627907.7064	6821001.908	1865518.459	702.270569	1937.787574	-7225.059485	86.905	0.08736
6 Hours	SLR	88987.01876	-6650580.02	2815144.524	-940.682367	2794.998693	6769.481014		
	PCR	88985.37647	-6650571.111	2815158.529	-940.7031471	2795.022885	6769.472743	16.679	0.03295
7 Hours	SLR	445878.9457	3720185.794	-5950586.40	846.6741233	-6411.33533	-3950.54040		
	PCR	445898.9998	3720140.434	-5950618.924	846.6872506	-6411.358207	-3950.488989	59.309	0.05778
8 Hours	SLR	-836177.037	358428.8881	7153804.333	-405.944999	7360.383301	-484.093557		
	PCR	-836202.2951	358488.4912	7153786.955	-405.9437842	7360.391044	-484.1612501	67.026	0.06815
9 Hours	SLR	895401.5684	-4705605.32	-5256659.08	-197.976076	-5671.28784	4897.062326		
	PCR	895420.6635	-4705648.923	-5256629.715	-197.9941018	-5671.240654	4897.098766	55.932	0.06228
10 Hours	SLR	-644552.285	6823913.005	1842610.834	699.6225659	1916.995141	-7232.75563		
	PCR	-644554.9643	6823920.365	1842548.111	699.6481554	1916.94693	-7232.775747	63.21	0.05817
11 Hours	SLR	104847.6662	-6643350.13	2833744.264	-948.964750	2812.325319	6760.186593		
	PCR	104828.5381	-6643334.362	2833784.545	-948.9853654	2812.378279	6760.162974	47.298	0.06154
12 Hours	SLR	438663.3430	3699094.493	-5964326.49	862.7264156	-6422.89251	-3928.04697		
	PCR	438697.601	3699027.356	-5964375.06	862.7401043	-6422.933211	-3927.961601	89.664	0.09556
13 Hours	SLR	-838138.166	376386.5011	7151721.988	-423.637199	7358.866662	-506.161339		
	PCR	-838179.9735	376493.1649	7151703.431	-423.6186423	7358.870045	-506.260741	116.06	0.1012
14 Hours	SLR	907736.7345	-4721806.41	-5242146.74	-185.138074	-5654.67849	4914.244797		
	PCR	907766.3903	-4721818.985	-5242137.849	-185.1823894	-5654.654026	4914.259055	33.416	0.05259
15 Hours	SLR	-661134.755	6826960.566	1820214.839	696.2149934	1896.250676	-7239.97616		
	PCR	-661139.5845	6826965.849	1820188.836	696.278173	1896.236653	-7239.974802	26.971	0.06473
16 Hours	SLR	120992.5493	-6636208.09	2851769.651	-956.606129	2828.857451	6751.294353		
	PCR	120965.2439	-6636205.014	2851754.626	-956.6664401	2828.856693	6751.294447	31.318	0.06032
17 Hours	SLR	430790.6428	3680808.750	-5976571.19	879.0767519	-6432.57954	-3907.95884		
	PCR	430844.4618	3680815.812	-5976573.943	879.0943195	-6432.576387	-3907.945695	54.35	0.02217
18 Hours	SLR	-840071.032	393089.6939	7149592.025	-442.016582	7357.399224	-526.634761		
	PCR	-840126.3857	393153.8908	7149579.771	-442.0021552	7357.398329	-526.6879148	85.647	0.05508
19 Hours	SLR	920599.0886	-4735524.35	-5229345.32	-172.263718	-5639.93041	4929.468963		
	PCR	920627.5547	-4735553.221	-5229300.877	-172.3002926	-5639.902025	4929.519247	60.16	0.06835
20 Hours	SLR	-677950.830	6829647.345	1798712.771	693.5216679	1876.595024	-7246.84054		
	PCR	-677960.0999	6829669.001	1798680.156	693.5629842	1876.554982	-7246.831027	40.233	0.05832
21 Hours	SLR	136542.7592	-6629013.54	2869409.092	-965.110952	2845.048921	6742.506767		
	PCR	136528.508	-6628999.578	2869433.433	-965.1529047	2845.07156	6742.498121	31.472	0.04845
22 Hours	SLR	423912.5491	3660989.721	-5989106.17	895.3884566	-6443.22398	-3886.72854		
	PCR	423936.5609	3661020.681	-5989097.588	895.4155244	-6443.202563	-3886.742069	40.11	0.03707
23 Hours	SLR	-842667.618	410715.4574	7147333.001	-459.541019	7355.714865	-548.394316		
	PCR	-842711.7766	410720.7335	7147330.747	-459.5438541	7355.715368	-548.3869699	44.53	0.00789
24 Hours	SLR	933315.9401	-4751747.19	-5214530.62	-160.155418	-5623.09584	4946.559678		
	PCR	933341.7596	-4751670.049	-5214598.058	-160.1731281	-5623.162998	4946.482758	105.66	0.1036

Chapter 7

Conclusions and Future Work

7.1 Conclusions

A precise orbit propagator, *PRECurSOr*, was presented. The propagator was developed using Cowell's method of special perturbations for short-term precision orbit propagation. *PRECurSOr* uses two integrators, the RK4 and RKF78, and includes an aspherical Earth, atmospheric drag, luni-solar and solar radiation pressure perturbative effects.

Considering the effect of an aspherical Earth improves the accuracy of *PRECurSOr* significantly, making it a very important perturbation to quantify. This significant improvement of the accuracy is indicated in Table 6.1. Results in Table 6.1 indicate that luni-solar perturbations have a negligible impact on the accuracy of the propagator and may be neglected. In addition, solar radiation pressure only improves *PRECurSOr*'s accuracy when RK4 ($h = 60$ *sec*) integrator is used. Inclusion of atmospheric drag does not improve the solution significantly over a one-day propagation and may also be neglected, except in the case where the RK4 ($h = 30$ *sec*) integrator with a 10×10 gravity model is used.

The worst of absolute errors in position by *PRECurSOr* for this test case at a chosen epoch for a one-day propagation were 1331.11 m, which result in an angular separation of 0.13° for a satellite like SUNSAT, with a perigee of about 600 km. Alternatively, the smallest absolute errors in position were 56.14 m which correspond to an angular

separation of 0.005°. This corresponds to a very small pointing error, which means that position measurements by *PRECurSOr* are reliable for imaging.

It is recommended that for a one-day propagation, the RKF78 integrator with all perturbation forces included be used. However, if accurate position of a satellite is required within the first 4 hours of propagation, the RK4 ($h = 30$ *sec*) with a 20×20 gravity field model or the RK4 ($h = 60$ *sec*) integrator with a 60×60 gravity field model is recommended for precision propagation. With the help of an on-board GPS (Global Positioning System) receiver, the initial conditions of this propagator can be re-initialised or uploaded from ground station, when the errors observed in the state vector become unacceptably large compared to the real-time state vector.

7.2 Future Work

PRECurSOr was developed under minimal number of perturbations forces enabling it to retain sufficient accuracy. Perturbations such as the Earth's radiation pressure, planetary attraction (third body attraction other than the Moon and Sun), tidal forces and general relativistic deviations to the Newtonian equation of motion could be incorporated into this propagator. The dynamics of a thruster system could also be included. Instead of using static atmospheric models for density calculations, dynamic atmospheric models like Jacchia 70 and 71, Jacchia-Roberts and MSIS could be implemented to improve the accuracy of the propagator. The 20×20 gravity field model could be extended to a full gravity model. The performance of the propagator could be tested for satellites at other orbit altitude regimes, such as MEO and GEO.

Appendix A

Vectors

The following is a summary of (Vallado, 1997 :855-856) and (Curtis, 2010 :2-9).

A vector is defined as a quantity with both magnitude and direction. We represent a vector graphically by an arrow pointing to the direction of the vector, where the length of the arrow signifies its magnitude. A vector can be denoted by a bold letter(e.g. $\mathbf{A}$) or just a plain letter with an arrow on top(e.g. $\vec{a}$). The former will be used to denote vectors in this study. Vectors come in different multiple dimensions but here we will use vectors with three components only.
In this section we'll only use three vectors $\mathbf{r}_{\oplus}$, $\mathbf{r}_{\odot}$ and $\mathbf{r}_{sat}$ which will be used several times in this study. These two vectors are represented in cartesian components as follows:

$$\mathbf{r}_{\oplus} = x_{\oplus}\hat{\mathbf{i}} + y_{\oplus}\hat{\mathbf{j}} + z_{\oplus}\hat{\mathbf{k}}$$

and

$$\mathbf{r}_{\odot} = x_{\odot}\hat{\mathbf{i}} + y_{\odot}\hat{\mathbf{j}} + z_{\odot}\hat{\mathbf{k}}$$

where $\hat{\mathbf{i}}$, $\hat{\mathbf{j}}$ and $\hat{\mathbf{k}}$ are unit vectors.

The magnitude of a vector, say $\mathbf{r}_\oplus$, which is denoted by $|\mathbf{r}_\oplus|$ or $r_\oplus$, is found by taking the square root of the sum of the squares of each component.

$$|\mathbf{r}_\oplus| = r_\oplus = \sqrt{x_\oplus^2 + y_\oplus^2 + z_\oplus^2}$$

Addition of two vectors is performed by adding the corresponding components of the vectors involved. Vector addition is commutative i.e. $\mathbf{r}_\oplus + \mathbf{r}_\odot = \mathbf{r}_\odot + \mathbf{r}_\oplus$.

Similarly, vectors subtraction is also performed by subtracting the corresponding components of the vectors involved. But, vector subtraction is not commutative. Therefore, $\mathbf{r}_\oplus - \mathbf{r}_\odot = -(\mathbf{r}_\odot - \mathbf{r}_\oplus)$. The vectors $\mathbf{r}_{\oplus\odot}$ and $\mathbf{r}_{\odot\oplus}$ will represent subtraction of vectors $\mathbf{r}_\oplus$ and $\mathbf{r}_\odot$ depending on which vector is getting subtracted from another vector: $\mathbf{r}_{\oplus\odot} = \mathbf{r}_\odot - \mathbf{r}_\oplus$ or $\mathbf{r}_{\odot\oplus} = \mathbf{r}_\oplus - \mathbf{r}_\odot$.

There are not set of defined rules for vector division and multiplication. However, there are two binary operations on vectors which are closely related to division and multiplication of vectors. The operations are the dot product and the cross product. The dot product of two vectors is defined as follows:

$$\mathbf{r}_\odot \cdot \mathbf{r}_\oplus = r_\odot r_\oplus \cos\theta$$

Where θ is the angle between the two vectors. The dot product of two vectors is a scalar and is also commutative(i.e. $\mathbf{r}_\odot \cdot \mathbf{r}_\oplus = \mathbf{r}_\oplus \cdot \mathbf{r}_\odot$) Finally, the cross product of two vectors is defined as follows:

$$\mathbf{r}_\odot \times \mathbf{r}_\oplus = r_\odot r_\oplus \sin\theta$$

The cross product of two vectors is another vector. Commutativity does not apply in cross product, thus, $\mathbf{r}_\odot \times \mathbf{r}_\oplus = -\mathbf{r}_\oplus \times \mathbf{r}_\odot$
Some important vector identities are listed below,

$$\mathbf{r}_{sat} \cdot (\mathbf{r}_{\oplus} \times \mathbf{r}_{\odot}) = (\mathbf{r}_{sat} \times \mathbf{r}_{\oplus}) \cdot \mathbf{r}_{\odot} \tag{A.1}$$

$$\mathbf{r}_{sat} \times (\mathbf{r}_{\oplus} \times \mathbf{r}_{\odot}) = (\mathbf{r}_{sat} \cdot \mathbf{r}_{\odot})\mathbf{r}_{\oplus} - (\mathbf{r}_{sat} \cdot \mathbf{r}_{\oplus})\mathbf{r}_{\odot} \tag{A.2}$$

$$(\mathbf{r}_{sat} \times \mathbf{r}_{\oplus}) \times \mathbf{r}_{\odot} = (\mathbf{r}_{sat} \cdot \mathbf{r}_{\odot})\mathbf{r}_{\oplus} - (\mathbf{r}_{\oplus} \cdot \mathbf{r}_{\odot})\mathbf{r}_{sat} \tag{A.3}$$

Appendix B

Mechanical Energy and Angular Momentum

Conservation of Mechanical Energy

The *specific mechanical energy* is found by dot-multiplying equation (2.2) by $\dot{\mathbf{r}}$ (Chobotov, 2002 :23-24):

$$\dot{\mathbf{r}} \cdot \ddot{\mathbf{r}} + \frac{\mu}{r^3}\mathbf{r} \cdot \dot{\mathbf{r}} \tag{B.1}$$

Knowing that $\mathbf{r} \cdot \mathbf{r} = r^2$ and $\frac{d}{dt}(1/r) = (-1/r^2)\frac{dr}{dr}$ we can further simplify this equation as follows:

$$\begin{aligned} 0 &= \frac{1}{2}\frac{d}{dt}(\dot{\mathbf{r}} \cdot \dot{\mathbf{r}}) + \frac{\mu}{r^3} r\dot{r} \\ &= \frac{1}{2}\frac{d}{dt}(v \cdot v) + \frac{\mu}{r^2}\dot{r} \\ &= \frac{1}{2}\frac{d}{dt}(v^2) - \frac{d}{dt}\frac{\mu}{r} \end{aligned} \tag{B.2}$$

$$\left(\frac{v^2}{2} - \frac{\mu}{r}\right) = \varepsilon \tag{B.3}$$

Where $(v^2/2)$ is the specific kinetic energy per unit mass and $(-\mu/r)$ is the specific potential energy per unit mass. The specific potential energy is the same as the gravitational potential per unit mass. The specific mechanical enegry(ε) always remains constant. This means when a satellite is farthest from the Earth its kinetic enegry decreases while its potential energy increases and when a satellite is closest to the Earth its kinetic energy increases and its potential energy decreases.

Conservation of Angular Momentum

The specific angular momentum can be found by vector-multiplying equation (2.2) by $\mathbf{r}$ (Chobotov, 2002 :24):

$$\mathbf{r} \times \ddot{\mathbf{r}} + \frac{\mu}{r^3}\mathbf{r} \times \mathbf{r} = 0. \tag{B.4}$$

The second term vanishes because $\mathbf{r} \times \mathbf{r} = 0$, so that,

$$\mathbf{r} \times \ddot{\mathbf{r}} = 0. \tag{B.5}$$

Using, $\frac{d}{dt}(\mathbf{r} \times \dot{\mathbf{r}}) = \dot{\mathbf{r}} \times \dot{\mathbf{r}} + \mathbf{r} \times \ddot{\mathbf{r}} = \mathbf{r} \times \ddot{\mathbf{r}}$ we end up with the following expression:

$$\frac{d}{dt}(\mathbf{r} \times \mathbf{v}) = 0, \tag{B.6}$$

where the specific angular momentum is represented by $\mathbf{r} \times \mathbf{v} = \mathbf{h}$. The term $\frac{d\vec{h}}{dt} = 0$ implies that the specific angular momentum of a satellite remains constant along its orbit.

Appendix C

The Trajectory Equation

The following work presents the derivation of the trajectory equation as presented by (Vallado, 1997 :110-111)

We start off by vector-multiplying Equation (2.2) by $\mathbf{h}$:

$$\ddot{\mathbf{r}} \times \mathbf{h} + \frac{\mu}{r^3}(\mathbf{r} \times \mathbf{h}) = 0$$

The first term can be written as a derivative, $\frac{d}{dt}(\dot{\mathbf{r}} \times \mathbf{h})$ like we did in Section 2.4.3.However, The second term can be simplified using a vector identity listed in A.3 as follows:

$$\frac{\mu}{r^3}\mathbf{r} \times (\mathbf{r} \times \mathbf{v}) = \frac{\mu}{r^3}((\mathbf{r} \cdot \mathbf{v})\mathbf{r} - r^2\mathbf{v})$$

This will further simplify to,

$$\begin{aligned} \frac{\mu}{r^3}(\mathbf{r} \times \mathbf{h}) &= \frac{\mu}{r^2}\dot{r}\mathbf{r} - \frac{\mu}{r}\mathbf{v} \\ &= -\mu\frac{d}{dt}\left(\frac{\mathbf{r}}{r}\right) \end{aligned} \tag{C.1}$$

Substituting the simplified expressions for both the first and second term yields,

$$\frac{d}{dt}(\dot{\mathbf{r}} \times \mathbf{h}) - \mu \frac{d}{dt}\left(\frac{\mathbf{r}}{r}\right) = 0$$

Rewriting,

$$\frac{d}{dt}(\dot{\mathbf{r}} \times \mathbf{h}) = \mu \frac{d}{dt}\left(\frac{\mathbf{r}}{r}\right)$$

Integrating both sides,

$$\dot{\mathbf{r}} \times \mathbf{h} = \mu \frac{\mathbf{r}}{r} + \mathbf{B}$$

dot-multiplying by $\mathbf{r}$,

$$\mathbf{r} \cdot (\dot{\mathbf{r}} \times \mathbf{h}) = \mathbf{r} \cdot \left(\mu \frac{\mathbf{r}}{r} + \mathbf{B}\right)$$

The left-hand side can be simplified using Equation (A.1) as follows,

$$\mathbf{r} \cdot (\dot{\mathbf{r}} \times \mathbf{h}) = (\mathbf{r} \times \dot{\mathbf{r}}) \cdot \mathbf{h} = \mathbf{h} \cdot \mathbf{h} = h^2$$

Substituting we get,

$$h^2 = \mu r + rB \cos \nu$$

Solving for the position yields the solution of the two-body equation:

$$r = \frac{h^2/\mu}{1 + (B/\mu) \cos \nu}$$

Appendix D

Eccentric and True Anomaly Transformation

The following appendix covers the work of (Vallado, 1997 :210-215);(Opperman, 2003 :B1-5)

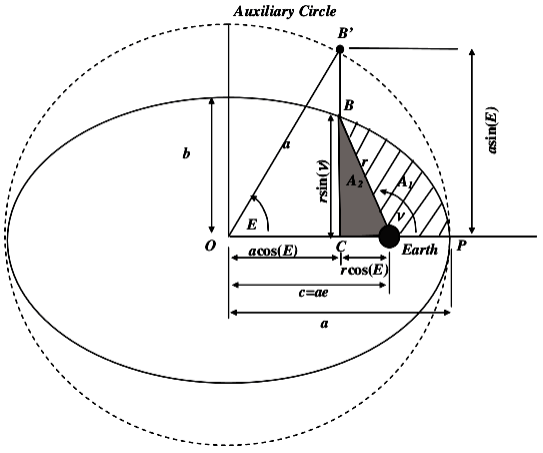

Figure D.1: Geometry of Kepler's Equation(Opperman (2003),Pp B-1)

From Figure D.1 we can directly obtain expressions relating E and ν, which are the following

$$\sin E = \frac{r \sin \nu}{a\sqrt{1-e^2}} \tag{D.1}$$

$$\cos E = \frac{ae + r\cos\nu}{a} \tag{D.2}$$

Substituting the trajectory equation in place of r and writing p as it is listed in Table 2.1 gives

$$\sin E = \frac{a(1 - e^2\sin\nu)}{a(1 + e\cos\nu)\sqrt{1 - e^2}}$$

and

$$\cos E = e + \frac{\frac{a(1-e^2)}{(1+e\cos\nu)}\cos\nu}{a}$$

After simplifying we get

$$\sin E = \frac{\sqrt{1 - e^2}\sin\nu}{1 + e\cos\nu}$$

and

$$\cos E = \frac{e + \cos\nu}{1 + e\cos\nu}$$

Solving for $\cos\nu$ in Equation (D.2) yields

$$\cos\nu = \frac{\cos E - e}{1 - e\cos E} \tag{D.3}$$

Solving for r in Equation (D.2) and substituting for $\cos\nu$ gives

$$r = a(1 - e\cos E) \tag{D.4}$$

and solving for $\sin\nu$ in Equation (D.1) and substituting the value of r yields

$$\sin\nu = \frac{\sqrt{1 - e^2}\sin E}{1 - e\cos E} \tag{D.5}$$

Dividing Equation (D.3) by Equation (D.5) and using tangent half-angle formula we obtain a single equation for true anomaly

$$\tan\left(\frac{\nu}{2}\right) = \frac{\frac{\sqrt{1-e^2}\sin E}{1-e\cos E}}{\frac{\cos E - e}{1-e\cos E} + 1}$$

$$\tan\left(\frac{\nu}{2}\right) = \frac{\sqrt{1 - e^2}\sin E}{\cos E - e + 1 - e\cos E}$$

The right-hand side denominator is the product of $(1 - e)(cosE + 1)$

$$\tan\left(\frac{\nu}{2}\right) = \frac{\sqrt{1 - e^2}\sin E}{(1 - e)(\cos E + 1)}$$

Using the tangent half-angle formula to reduce the terms containing E we get

$$\tan\left(\frac{\nu}{2}\right) = \sqrt{\frac{1 + e}{1 - e}}\tan\left(\frac{E}{2}\right)$$

Which can finally be written as

$$\tan\left(\frac{E}{2}\right) = \sqrt{\frac{1 - e}{1 + e}}\tan\left(\frac{\nu}{2}\right)$$

Appendix E

Lunar and Solar Position

Position of The Sun

The geocentric position of the Sun can be calculated by assuming a purely elliptical motion of the Earth with a low accuracy of 0.01°(Meeus, 1991). Given the Julian Day the time T measured in Julian centuries of 36525 ephemeris days from the epoch J2000.0, is obtained from

$$T = \frac{JD - 2451545.0}{36525} \tag{E.1}$$

The position vector may be computed using the following (Meeus, 1991, :163-164):

$$\mathbf{r}_\odot = \begin{pmatrix} r_\odot \cos(\odot) \\ r_\odot \cos(\varepsilon) \sin(\odot) \\ r_\odot \sin(\varepsilon) \sin(\odot) \end{pmatrix} (AU) \tag{E.2}$$

Where

- $r_\odot$ is the radial distance of the Sun from Earth in AU

$$r_\odot = \frac{1.000001018(1 - e^2)}{1 - e \cos \nu}$$

The eccentricity of the Earth's orbit, e is

$$e = 0.016708634 - 0.000042037T - 0.0000001267T^2$$

The Sun's true anomaly ν is: $\nu = M + C$

C is the Sun's equation of the centre given by:

$$\begin{aligned} C = &+ (1.914602^\circ - 0.004817^\circ T - 0.000014^\circ T^2)\sin(M) \\ &+ (0.019993^\circ - 0.000101^\circ T)\sin(2M) + 0.000289^\circ \sin(3M) \end{aligned}$$

M, the mean anomaly of the Sun is

$$M = 357.52911^\circ + 35999.05029^\circ T - 0.0001537^\circ T^2$$

- $\odot$ is the Sun's true longitude given by: $\odot = L_0 + C$

  L_0 is the geometric mean longitude of the Sun referred to the mean equinox of date and is given by:

$$L_0 = 280.46646^\circ + 36000.76983^\circ T + 0.0003032^\circ T^2$$

- ε is the mean obliquity of ecliptic given by

$$\varepsilon = 23.4392911^\circ - 0.0130041^\circ T - 1.64 \times 10^{-7} T^2 + 5.04 \times 10^{-7} T^3$$

In addition, the Sun's right ascension α and declination δ are obtained from:

$$\tan(\alpha) = \frac{\cos(\varepsilon)\sin(\odot)}{\cos(\odot)} \tag{E.3}$$

$$\sin(\delta) = \sin(\varepsilon)\sin(\odot) \tag{E.4}$$

Position of The Moon

The motion of the moon is very complex and to obtain the position vector using exact formulas would require a lot of computations. Vallado (1997) presented shortened formulas to compute the position of the Moon with a satisfactory accuracy to our problem. The position vector is given by:

$$\mathbf{r}_{moon} = r_{moon} \begin{pmatrix} \cos(\phi_{ecliptic})\cos(\lambda_{ecliptic}) \\ \cos(\varepsilon)\cos(\phi_{ecliptic})\sin(\lambda_{ecliptic}) - \sin(\varepsilon)\sin(\phi_{ecliptic}) \\ \sin(\varepsilon)\cos(\phi_{ecliptic})\sin(\lambda_{ecliptic}) + \cos(\varepsilon)\sin(\phi_{ecliptic}) \end{pmatrix} \tag{E.5}$$

Where

- r_{moon} is the distance between the centres of the Earth and Moon in Earth radii(ER)

$$r_{moon} = \frac{1}{\sin(\psi)}$$

ψ is the equatorial horizontal parallax of the moon given by

$$\begin{aligned} \psi =& 0.9505^\circ + 0.0518\cos(M_{moon}) + 0.0095\cos(M_{moon} - 2D_\odot) \\ &+ 0.0078\cos(2D_\odot) + 0.0028\cos(2M_{moon}) \end{aligned}$$

M_{moon} is the Moon's mean anomaly($r = 360^\circ$)

$$\begin{aligned} M_{moon} =& 134.9629814^\circ + (1325r + 198.8673981)T + 0.0086972T^2 \\ &+ 1.778 \times 10^{-5}T^3 \end{aligned}$$

$D_\odot$ is the mean elongation of the Sun and is obtained from

$$\begin{aligned} D_\odot =& 297.8503631^\circ + (1236r + 307.111480)T - 0.00191417T^2 \\ &+ 5.28 \times 10^{-6}T^3 \end{aligned}$$

- $\phi_{ecliptic}$ is the ecliptic latitude

$$\begin{aligned}\phi_{ecliptic} =& 5.13^\circ \sin(93.3 + 483202.03T) + 0.28 \sin(228.2 + 960400.87T) \\ & - 0.28 \sin(318.3 + 6003.18T) - 0.17 \sin(217.6 - 407332.20T)\end{aligned}$$

- $\lambda_{ecliptic}$, ecliptic longitude is given by

$$\begin{aligned}\lambda_{ecliptic} =& 218.32^\circ + 481267.8813T + 6.29 \sin(134.9 + 477198.85T) \\ & - 1.27 \sin(257.2 - 413335.38T) + 0.66 \sin(235.7 + 890534.23T) \\ & + 0.21 \sin(269.9 + 954397.70T) - 0.19 \sin(357.5 + 35999.05T) \\ & - 0.11 \sin(186.6 + 966404.05T)\end{aligned}$$

The Moon's right ascension(α) and declination(δ) are obtained from:

$$\sin(\alpha) = \frac{-\sin(\phi_{ecliptic}) \sin(\varepsilon) + \cos(\phi_{ecliptic}) \cos(\varepsilon) \sin(\lambda_{ecliptic})}{\cos(\delta)} \tag{E.6}$$

$$\sin(\delta) = \sin(\phi_{ecliptic}) \cos(\varepsilon) + \cos(\phi_{ecliptic}) \sin(\varepsilon) \sin(\lambda_{ecliptic}) \tag{E.7}$$

Appendix F

C++ Source Code - PRECurSOr

```
/*********************************************************************
 Name: DXDT.cpp
 Description: The differential equation y'=f(x,y)
 Input(s) _____________________________________________
       X[6]    = 6-D current cartesian state X,Y,Z,XD,YD,ZD,(M,M/SEC)
       t       = Current time(s)
 Output(s) ____________________________________________
        DX     = 6-D Derivatives of position and velocity
 Date Last Modified: 24 July 2013
 Author: Tshilande T
 Reference: Opperman 2003
*********************************************************************/
void DXDT(double t, double X[], double DX[])
 {
    int im,in;
    double r_sat[3],sa[3],P[61][61] = {0.0},CN[n+1],SN[n+1],TN[n+1];
    double JD,Rij,SRij,R2,Dr,Dp,Dl,dPhi_dr,dPhi_dlambda,dPhi_dphi;
    double idr,idphi,idlambda,C1,C2,C3;
    double radius,r_cubed,r5;
    double Lon, Lat,sphi;
```

```
  double PCSR,rdrs,ratm,theta;
  double r_satmoon[3],r_satsun[3];
  double r_sun[3],r_moon[3],rs_unit[3],rss_unit[3],su[2],rad_sun,
         rad_moon,mag_rsun,rm3,rsm,rsm3,rs3,rss,rss3;
  double v_rel[3],omega,altitude,vr2,vr_mag;
  static double an[6], wmol, rho;
  static double sf[3], te[2];

  /*...Create position vector*/
  double r_x = X[0],
         r_y = X[1],
         r_z = X[2];
  r_sat[0] = X[0],
  r_sat[1] = X[1],
  r_sat[2] = X[2];
/*...Compute the radial distance from Earth to Sat*/
R2 = r_x*r_x + r_y*r_y + r_z*r_z;
radius = sqrt(R2);
r_cubed = radius*R2;
/*...Transfer velocity from satellite state vector*/
DX[0] = X[3];
DX[1] = X[4];
DX[2] = X[5];
/*...initialize acceleration for this integration step*/
for(int i=3; i<=5; i++){
    DX[i] = 0.0;
}
/*...Compute Julian date*/
JD = JulianDate(year,month,day,t);
//compute longitude, latitude and altitude
ECI_ECEF(JD,r_sat,sa);
```

```
Lon = sa[0];
Lat = sa[1];
sphi = sin(Lat);
if(GRAVP == 1){
    if(n == 2 && m == 0){
        //Compute J2
        r5 = R2 * r_cubed;
        C1 = -1.5*j2*R_earth*R_earth*mu/r5;
        C2 = one- five*r_z*r_z/R2;
        DX[3] = DX[3] + X[0]*C1*C2;
        DX[4] = DX[4] + X[1]*C1*C2;
        DX[5] = DX[5] + X[2]*C1*(C2+two);
      }
    else{
        /*...Aspheric gravity computation*/
        Rij = r_x*r_x + r_y*r_y;
        SRij = sqrt(Rij);
        /*...Compute Normalized legendre polynomials*/
        legendre(n,sphi,P);
        /*...Compute Cos[m],Sin[m] and Tan[m]*/
        trigfunc(m,Lat,Lon,CN,SN,TN);
        /*...Compute partials*/
        C1 = R_earth/radius;
        C2 = C1;
        dPhi_dr = zero;
        dPhi_dphi = zero;
        dPhi_dlambda = zero;
        for(int i = 2; i<=n ; i++){
            idr = zero;
            idphi = zero;
            idlambda = zero;
```

```
            in = i+1;
            for(int j = 0; j <=i ; j++){
                im = j+1;
                C3 = C[i][j]*CN[j] + S[i][j]*SN[j];
                idr = idr + C3*P[i][j];
                if(im <= i){idphi = idphi + C3*(P[i][im]-TN[j]*P[i][j]);}
                if(im > i){idphi = idphi - C3*TN[j]*P[i][j];}
                if(j != 0){idlambda = idlambda + j*(S[i][j]*CN[j] -
                        C[i][j]*SN[j])*P[i][j];}
            }
            C2 = C2*C1;
            dPhi_dr = dPhi_dr + C2*in*idr;
            dPhi_dphi = dPhi_dphi + C2*idphi;
            dPhi_dlambda = dPhi_dlambda + C2*idlambda;
        }
        C3 = mu/radius;
        dPhi_dr = - dPhi_dr* C3/radius;
        dPhi_dphi = dPhi_dphi*C3;
        dPhi_dlambda = dPhi_dlambda*C3;
        Dr = dPhi_dr/radius;
        Dp = r_z/(R2*SRij)*dPhi_dphi;
        Dl = dPhi_dlambda/ Rij;
        DX[3] = DX[3] + (Dr - Dp)*r_x - (Dl*r_y);
        DX[4] = DX[4] + (Dr - Dp)*r_y + (Dl*r_x);
        DX[5] = DX[5] + Dr*r_z + SRij*dPhi_dphi/R2;
    }
}
if(MOON == 1){
    /*...Compute derivative due to the Moon*/
    /*...Compute Moon's position*/
    Moon(JD,rad_moon,r_moon);
```

```
    rm3 = rad_moon*rad_moon*rad_moon;
    vminus(r_sat,r_moon,r_satmoon);
    rsm = r_satmoon[0]*r_satmoon[0] + r_satmoon[1]*r_satmoon[1]
          + r_satmoon[2]*r_satmoon[2];
    rsm = sqrt(rsm);
    rsm3 = rsm*rsm*rsm;
    for(int i = 0; i<=2; i++){
        DX[i+3] = DX[i+3] - mu_moon*(r_satmoon[i]/rsm3 + r_moon[i]/rm3);}
}
if(SUN ==  1){
    /*...Compute derivative due to the Sun*/
    /*...Compute Sun's position*/
    Sun(JD,rad_sun,r_sun,su);
    rs3 = rad_sun*rad_sun*rad_sun;
    vminus(r_sat,r_sun,r_satsun);
    rss = r_satsun[0]*r_satsun[0] + r_satsun[1]*r_satsun[1]
          + r_satsun[2]*r_satsun[2];
    rss = sqrt(rss);
    rss3 = rss*rss*rss;
    for(int i = 0; i<=2; i++){
        DX[i+3] = DX[i+3] - mu_sun*(r_satsun[i]/rss3 + r_sun[i]/rs3);}
}
if(SRP == 1){
    /*...Compute derivative due to SRP*/
    PCSR = PSR*CR;
    ratm = R_earth + 90000.;
    mag_rsun = r_sun[0]*r_sun[0] + r_sun[1]*r_sun[1] + r_sun[2]*r_sun[2];
    mag_rsun = sqrt(mag_rsun);
    for(int i = 0; i<=2 ; i++){
        rs_unit[i] = r_sun[i]/mag_rsun;
        rss_unit[i]   = r_satsun[i]/rss;
```

```
    }
    /*...check for shadow condition*/
    rdrs = rs_unit[0]*X[0] + rs_unit[1]*X[1] + rs_unit[2]*X[2];
    theta = acos(rdrs/radius);
    if(rdrs >= 0.0){
        for(int i= 0; i<=2; i++){
            DX[i+3] = DX[i+3] - PCSR*(Area/mass)*rss_unit[i];}
    }
    else if(radius*sin(theta)< ratm){
        //Satellite in Earth's shadow
        for(int i = 0; i<=2; i++){
            DX[i+3] = DX[i+3] + 0;
        }
    }
}
    if(ADRAG == 1){
    /*...Compute derivative due to Drag*/
    omega = rate * DtR;
    /*...Compute atmospheric density*/
    altitude = sa[2];
    if(DENSMOD == 1){
        rho = Exp_model(altitude);
    }
    if(DENSMOD == 2){
        open_sfdf();
        if(int(t)%10800 == 0){
          sfdj70 (mjd, t, sf);
        }
        close_sfdf();
        jsmade(altitude, sf, te, an, &wmol, &rho);
    }
```

```
      /*...Compute the velocity relative to the rotating atmosphere*/
      v_rel[0] = X[3] + omega*X[1];
      v_rel[1] = X[4] - omega*X[0];
      v_rel[2] = X[5];
      vr2 = v_rel[0]*v_rel[0] + v_rel[1]*v_rel[1] + v_rel[2]*v_rel[2];
      vr_mag = sqrt(vr2);
      //Note: vr2 * v_rel(unit vector) = vr_mag*v_rel
      for(int i=0; i<=2 ; i++){
           DX[i+3] = DX[i+3] - half*rho*(C_D*Area/mass)*vr_mag*v_rel[i];
      }
  }
  /*...Compute Two-Body derivatives*/
  for(int i = 0; i<=2; i++){
      DX[i+3] = DX[i+3] - mu*X[i]/r_cubed;
  }
}

/*************************************************************************
 Name: RK4.cpp
 Description: This routine uses the Classical Runge Kutta method
              to approximate the solution of the differential
              equation y'=f(x,y) with the initial condition y = y[0] at
              x = x0.
 Input(s) ______________________________________________________________
       X[]       = 6-D current cartesian state X,Y,Z,XD,YD,ZD,(M,M/SEC)
       h         = initial stepsize
       x0        = current time
 Output(s) _____________________________________________________________
        *xout    = pointer to the updated solutions of diff eqns
                   after each step size.
 Calls: DXDT
```

```
 Date Last Modified: 24 July 2013
 Author: Tshilande T
 References:
             W.H. Press et.al "Numerical Recipes in C++ - The Art of
             Scientific Computing",Second Edition,2002.
*************************************************************************/
void RK4(void (*derivs)(double,double[],double[]),int n,double X[],
                                        double *xout,double t, double h)
 {
   int i;
   double hh = h*half,
          th = t+hh;
   double y[n], k1[n], k2[n], k3[n], k4[n];

  (*derivs)(t,X,k1); //first step
  for (i=0; i < n; i++){
   y[i] = X[i] + hh * k1[i];}

  (*derivs)(th,y, k2); //second step
  for (i=0; i < n; i++){
   y[i] = X[i] + hh*k2[i];}

  (*derivs)(th,y,k3); //third step
  for (i=0; i < n; i++){
   y[i] = X[i] + h*k3[i];}

  (*derivs)(t+h,y,k4); //fourth step
  for (i = 0; i < n; i++){
   *(xout+i) = X[i] + h*(k1[i] + two * k2[i] + two * k3[i] + k4[i]) / six;}
  return;
 }
```

```
/*************************************************************************
 Name: RK78.cpp
 Description: This routine uses Fehlberg's embedded 7th and 8th order
              method to approximate the solution of the differential
              equation y'=f(x,y) with the initial condition y = y[0] at
              x = x0.
 Input(s) ____________________________________________________________
       Y[]                = initial conditions at time x0
       h                  = initial stepsize
       x0                 = initial time
 Output(s) ___________________________________________________________
        *err              = pointer to the error in diff eqns
        *yout             = pointer to the updated solutions of diff eqns
                            after each step size.
 Date Last Modified: 24 July 2013
 Author: Tshilande T
 References: http://www.mymathlib.com/diffeq/embedded_runge_kutta/

             W.H. Press et.al "Numerical Recipes in C++ - The Art of
             Scientific Computing",Second Edition,2002.
*************************************************************************/
#include "RKF78.h"
void Runge_Kutta78(void (*f)(double,double[],double*),int n, double Y[],
double *err,double *yout,double x0,double h) {
   int i;
   double k1[n], k2[n], k3[n], k4[n], k5[n], k6[n], k7[n], k8[n], k9[n], k10[n],
   k11[n], k12[n], k13[n];
   double ytemp[n];
   double h2_7 = a2 * h;
```

```
(*f)(x0, Y,k1); //first step
for(i=0; i<n; i++){
 ytemp[i] = Y[i] + h2_7 * k1[i];}
(*f)(x0+h2_7,ytemp,k2); //second step
for(i=0; i<n; i++){
 ytemp[i] = Y[i] + h * ( b31*k1[i] + b32*k2[i]);}
(*f)(x0+a3*h,ytemp,k3); //third step
for(i=0; i<n; i++){
 ytemp[i] = Y[i] +  h * ( b41*k1[i] + b43*k3[i]);}
(*f)(x0+a4*h,ytemp,k4); //fourth step
for(i=0; i<n; i++){
 ytemp[i] = Y[i] + h * ( b51*k1[i] + b53*k3[i] + b54*k4[i]);}
(*f)(x0+a5*h,ytemp,k5); //fifth step
for(i=0; i<n; i++){
 ytemp[i] = Y[i] + h * ( b61*k1[i] + b64*k4[i] + b65*k5[i]);}
(*f)(x0+a6*h,ytemp,k6); //sixth step
for(i=0; i<n; i++){
 ytemp[i] = Y[i] + h * ( b71*k1[i] + b74*k4[i] + b75*k5[i]
            + b76*k6[i]);}
(*f)(x0+a7*h,ytemp,k7); //seventh step
for(i=0; i<n; i++){
 ytemp[i] = Y[i] + h * ( b81*k1[i] + b85*k5[i] + b86*k6[i]
            + b87*k7[i]);}
(*f)(x0+a8*h,ytemp,k8); //eighth step
for(i=0; i<n; i++){
 ytemp[i] = Y[i] + h * ( b91*k1[i] + b94*k4[i] + b95*k5[i]
            + b96*k6[i] + b97*k7[i] + b98*k8[i]);}
(*f)(x0+a9*h,ytemp,k9); //nineth step
for(i=0; i<n; i++){
 ytemp[i] = Y[i] + h * ( b10_1*k1[i] + b10_4*k4[i] + b10_5*k5[i]
            + b10_6*k6[i] + b10_7*k7[i] + b10_8*k8[i]
```

```
                + b10_9*k9[i] );}
   (*f)(x0+a10*h,ytemp,k10); //tenth step
   for(i=0; i<n; i++){
    ytemp[i] = Y[i] + h * ( b11_1*k1[i] + b11_4*k4[i] + b11_5*k5[i]
              + b11_6*k6[i] + b11_7*k7[i] + b11_8*k8[i] + b11_9*k9[i]
              + b11_10 *k10[i]);}
   (*f)(x0+h,ytemp,k11); //eleventh step
   for(i=0; i<n; i++){
    ytemp[i] = Y[i] + h * ( b12_1*k1[i] + b12_6*k6[i] + b12_7*k7[i]
               + b12_8*k8[i] + b12_9*k9[i] + b12_10 * k10[i]);}
   (*f)(x0,ytemp,k12); //twelveth step
   for(i=0; i<n; i++){
    ytemp[i] = Y[i] + h * ( b13_1*k1[i] + b13_4*k4[i] + b13_5*k5[i]
               + b13_6*k6[i] + b13_7*k7[i] + b13_8*k8[i] + b13_9*k9[i]
               + b13_10*k10[i] + k12[i]);}
   (*f)(x0+h,ytemp,k13); //thirteenth step
   for(i=0; i<n; i++){
     *(yout+i) = Y[i] +  h * (c_1_11 * (k1[i] + k11[i])  + c6 * k6[i]
                 + c_7_8 * (k7[i] + k8[i]) + c_9_10 * (k9[i] + k10[i]) );}
   for(i=0; i<n; i++){
     *(err+i) = err_factor * (k1[i] + k11[i] - k12[i] - k13[i]);}

   return;
}

/**************************************************************************
 Name: Embedded_Fehlberg_7_8.cpp
 Description: This function solves the differential equation y'=f(x,y)
              with the initial condition y(x) = y[0].
 Input(s) ______________________________________________________________
        *f      = Pointer to the differential equation DXDT
```

```
        Y[]     = initial state vector at x0
        x       = The initial time (sec).
        h       = Initial step size.
       xmax     = The final time (sec).
       tolerance= The tolerance of Y(xmax), i.e. a solution is sought
                  so that the relative error < tolerance.
 Output(s) ____________________________________________________________
       Y[]     = State vector from time x to xmax
 Date Last Modified: 24 July 2013
 Author: Tshilande T
 References: http://www.mymathlib.com/diffeq/embedded_runge_kutta/
*****************************************************************************/

void Embedded_Fehlberg_7_8( void (*f)(double, double[],double*),int n,
 double Y[],double x,double h, double xmax, double tolerance ){
   static const double err_exponent = 1.0 / 7.0;
   double scale,errmax;
   double err[n],y[n];
   double yy;
   int i;
   int last_interval = 0;
   // Insure that the step size h is not larger than the length of the //
   // integration interval.                                            //
   if (h > (xmax - x) ) { h = xmax - x; last_interval = 1;}
   // Redefine the error tolerance to an error tolerance per unit    //
   // length of the integration interval.                              //
   tolerance /= (xmax - x);
   // Integrate the diff eq y'=f(x,y) from x=x to x=xmax trying to  //
   // maintain an error less than tolerance * (xmax-x) using an     //
   // initial step size of h and initial value: y = y[0]            //
   while ( x < xmax ) {
```

```
      scale = 1.0;
      Runge_Kutta78(f,n,Y,err,y, x, h);
      yy = 0.0;
      for(i=0; i<n; i++){yy = max(yy,fabs(Y[i]));}
      yy = (yy == 0.0) ? tolerance : yy;
      errmax = 0.0;
      for(i=0; i<n; i++){errmax = max(errmax,fabs(err[i]));}
      if (errmax == 0.0) { scale = MAX_SCALE_FACTOR;}
      else{scale = 0.8 * pow( tolerance * yy /  errmax , err_exponent );}
      scale = min( max(scale,MIN_SCALE_FACTOR), MAX_SCALE_FACTOR);
      if ( errmax < ( tolerance * yy ) ){
          for(i=0; i<n;i++){Y[i] = y[i];}
          x += h;
         }
      h *= scale;
      if (  x + h > xmax ) { last_interval = 1; h = xmax - x; }
   }
return;
}

/*****************************************************************************
* Date last modified: 05 September 2012
* Name: trigfunc
* Purpose: Computes the trigonometric functions recursively.
* Input(s):
*      m       = Order of spherical harmonics
*      lat     = Geocentric latitude
*      lon     = Geographic longitude
* Output(s):
*      CN     = M-D array of Cos values
*      SN     = M-D array of Sin values
```

```
*      TN     = M-D array of Tan values
* Author: Tshilande T
* References: B. Opperman 2003,  SUBROUTINE ANGLES(M,A,B,CN,SN,TN)
**********************************************************************/


void trigfunc(int m,double lat,double lon,double *CN, double *SN,
              double *TN){
  //if(m ==0)
  *CN = one;
  *SN = zero;
  *TN = zero;
  //if(m == 1)
  *(CN+1) = cos(lon);
  *(SN+1) = sin(lon);
  *(TN+1) = tan(lat);
  for(int i=2; i <= m ; i++)
    {
     *(CN+i) = two**(CN+1)**((CN+i)-1) - *((CN+i)-2);
     *(SN+i) = two**(CN+1)**((SN+i)-1) - *((SN+i)-2);
     *(TN+i) = *(TN+1) + *((TN+i)-1);
    }
  return;
}

/**********************************************************************
* Date last modified: 19 June 2012                                    *
* Name: legendre                                                      *
* Purpose: Computes legendre and associated legendre functions up     *
*          to degree n and order m.                                   *
* Input(s):                                                           *
```

```
*      n      = Degree of spherical harmonics                          *
*      m      = Order of spherical harmonics                           *
*      x      = Sin(latitude)                                          *
* Output(s):                                                           *
*      p      = 2-D legendre functions                                 *
* Author: Tshilande T                                                  *
* Reference: Opperman 2003                                             *
***********************************************************************/
void legendre(int n,double x, double p[][61])
{
  double y;
  y =sqrt(one - x*x);
  p[0][0] = one;
  p[1][0] = x;
  p[1][1] = y;
  for(int i = 2; i<=n ; i++)
   {
    for(int j = 0; j <=i ; j++)
      {
       if(j == 0)
        {
         p[i][0] = ((2*i - 1)*x*p[i-1][0] - (i -1)*p[i-2][0])/i;
        }
       else if(i > j && j != 0)
        {
         p[i][j] = (2*i - 1)*y* p[i-1][j-1];
         if(i-2 >= j)
          {
           p[i][j] += p[i-2][j];
          }
        }
```

```
       else if(i == j && j != 0)
        {
         p[i][i] = (2*i - 1)*y*p[i-1][i-1];
        }
      }
    }
  return;
}

/*************************************************************
* Name Sun.cpp
* Description: computes geocentric solar ephemeris
* Input:
*     JD = Julian day
* Output:
*     rad_sun = Sun's distance magnitude (m)
*     r_sun[3]= Sun's position vector (m)
*     su[0]   = right ascension of the Sun(radians)
*               (0 <= su[0] <= 2pi)
*     su[1]   = declination of the sun(radians)
*               (-pi/2 <= su[1] <= pi/2)
* Author: Tshilande T
* Date last modified: 23 September 2013
* Reference: Meeus 1991 and Vallado 1997
*************************************************************/
void Sun(double JD,double &rad_sun,double r_sun[],double su[])
 {
  double T_TDB,L_sun,M_sun,C,lambda_sun,e,ecc,v,isu;
  /*...Compute Julian centuries*/
  T_TDB = (JD - 2451545.0)/36525.0;
  /*...Compute the Sun's mean anomaly*/
```

```
M_sun = 357.52911 + 35999.05029*T_TDB - 0.0001537*T_TDB*T_TDB;
M_sun = quadrant(M_sun);
/*...Compute the Mean geometric longitude of the sun*/
L_sun = 280.46646 + 36000.76983*T_TDB + 0.0003032*T_TDB*T_TDB;
L_sun = quadrant(L_sun);
/*...Compute the centre*/
C = (1.914602-0.004817*T_TDB-0.000014*T_TDB*T_TDB)*sin(M_sun*PI/180.0)
    +(0.019993-0.000101*T_TDB)*sin(two*M_sun*PI/180.0) +
    0.000289*sin(three*M_sun*PI/180.0);
/*...Compute true geometric longitude*/
lambda_sun = L_sun + C;
/*...Compute the mean obliquity of the ecliptic*/
e = 23.439291 - 0.0130042*T_TDB - 1.64e-07*T_TDB*T_TDB
    + 5.04e-07*T_TDB*T_TDB*T_TDB;
/*...Compute the Eccentricty of Earth's orbit*/
ecc = 0.016708634 - 0.000042037*T_TDB - 0.0000001267*T_TDB*T_TDB;
/*...Compute Sun's true anomaly*/
v = M_sun + C;
/*...Compute radial distance from Earth to the Sun*/
rad_sun = 1.000001018*(1-ecc*ecc)/(1-ecc*cos(v*PI/180.0)); //in AU
/*...Compute position of the sun in ECI
r_sun[0] = rad_sun*cos(lambda_sun*PI/180.0);
r_sun[1] = rad_sun*cos(e*PI/180.0)*sin(lambda_sun*PI/180.0);
r_sun[2] = rad_sun*sin(e*PI/180.0)*sin(lambda_sun*PI/180.0);

rad_sun = rad_sun*AUtokm*kmtom; //in meters
for(int i=0; i<=2 ; i++){
    r_sun[i] = r_sun[i]*AUtokm*kmtom;
}
/*...Compute the right ascension and declination*/
su[0] = atan2(cos(e*PI/180.0)*sin(lambda_sun*PI/180.0),
```

```
          cos(lambda_sun*PI/180.));
  isu = su[0];
  su[0] = QuadRad(isu);
  su[1] = asin(sin(e*PI/180.)*sin(lambda_sun*PI/180.));
  return;
 }

/*************************************************************
  Name Moon.cpp
  Description: computes geocentric lunar ephemeris

  Input:
      JD = Julian day

  Output:
      rad_moon = Moon's distance magnitude (m)
      r_moon[3]= Moon's position vector (m)

  Date last modified: 06 June 2013
  Author: Tshilande T
  Reference: Meeus 1991 and Vallado 1997
*************************************************************/
void Moon(double JD,double &rad_moon,double r_moon[])
 {
   double T_TDB,M_moon,lambda_moon,uM_moon,D_sun,
          lambda_ecliptic,phi_ecliptic,e,parallax;
  /*...Compute Julian centuries*/
  T_TDB = (JD - 2451545.0)/36525.0;
  /*...Compute Moon's mean anomaly(k=360degrees)*/
  M_moon = 134.9629814 + (1325.0*degrees+198.8673981)*T_TDB
          + 0.0086972*T_TDB*T_TDB + 1.778e-05*T_TDB*T_TDB*T_TDB;
```

```
/*...Compute Moon's longitude*/
lambda_moon = 218.32 + 481267.8813*T_TDB;
/*...Compute Moon's mean argument of latitude*/
uM_moon = 93.27191030 + (1342.0*degrees+82.0175381)*T_TDB
          - 0.0036825*T_TDB*T_TDB + 3.06e-06*T_TDB*T_TDB*T_TDB;
/*...Compute the mean elongation of the Sun*/
D_sun = 297.8503631 + (1236.0*degrees+307.111480)*T_TDB
        - 0.00191417*T_TDB*T_TDB + 5.28e-6*T_TDB*T_TDB*T_TDB;
/*...Compute the ecliptic longitude*/
lambda_ecliptic = lambda_moon + 6.29*sin(M_moon*PI/180.0) -
                  1.27*sin((M_moon-two*D_sun)*PI/180.0)
                  +0.66*sin(two*D_sun*PI/180.0)
                  + 0.21*sin(two*M_moon*PI/180.0)
                  -0.19*sin(M_moon*PI/180.0)
                  - 0.11*sin(two*uM_moon*PI/180.0);
lambda_ecliptic = quadrant(lambda_ecliptic);
/*...Compute the ecliptic latitude*/
phi_ecliptic = 5.13*sin(uM_moon*PI/180.0)
               + 0.28*sin((M_moon + uM_moon)*PI/180.0)
               -0.28*sin((uM_moon - M_moon)*PI/180.0)
               - 0.17*sin((uM_moon - two*D_sun)*PI/180.0);
/*...Compute the mean obliquity of the ecliptic*/
e = 23.439291 - 0.0130042*T_TDB - 1.64e-07*T_TDB*T_TDB
    + 5.04e-07*T_TDB*T_TDB*T_TDB;
/*...Compute parallax*/
parallax = 0.9508 + 0.0518*cos(M_moon*PI/180.0)
           +0.0095*cos((M_moon - two*D_sun)*PI/180.0)  +
           0.0078*cos(two*D_sun*PI/180.0)
           +0.0028*cos(two*M_moon*PI/180.0);
/*...Compute radial distance from Earth to the Moon(in Earth radii)*/
rad_moon = one/sin(parallax*PI/180.0); //in ER
```

```
  r_moon[0] = rad_moon*cos(phi_ecliptic*PI/180.0)
              *cos(lambda_ecliptic*PI/180.0);
  r_moon[1] = rad_moon*(cos(e*PI/180.0)*cos(phi_ecliptic*PI/180.0)
              *sin(lambda_ecliptic*PI/180.0)
              - sin(e*PI/180.0)*sin(phi_ecliptic*PI/180.0));
  r_moon[2] = rad_moon*(sin(e*PI/180.0)*cos(phi_ecliptic*PI/180.0)
              *sin(lambda_ecliptic*PI/180.0)
              + cos(e*PI/180.0)*sin(phi_ecliptic*PI/180.0));
  rad_moon = rad_moon*R_earth; //in meters
  r_moon[0] = r_moon[0]*R_earth;
  r_moon[1] = r_moon[1]*R_earth;
  r_moon[2] = r_moon[2]*R_earth;
  return;
 }

/*************************************************************
 Name: oe2eci.cpp
 Description: Transforms classical orbital elements
              to ECI.
 Input(s) ____________________________________________________
         oe[]     = Classical orbital elements
 Output(s) ___________________________________________________
         sv[]      = ECI state vector X,Y,Z,XD,YD,ZD,(m,m/sec)
 Date Last Modified: 17 September 2013
 Author: Tshilande T
 References: Chobotov 2002: pp 62-65
*************************************************************/
void oe2eci(double oe[], double sv[]){
    double p,R;
    double a,e,i,w,raan,nu;
    double csr,snr,csw,snw,csi,sni,mup;
```

```
double r00,r01,r02,r10,r11,r12,r20,r21,r22;
double r_PQW[3];
a = oe[0];
e = oe[1];
i = oe[2];
raan = oe[3];
w = oe[4];
nu = oe[5];
p = a*(1.0- e*e);
mup = sqrt(mu/p);
R = p/(1.0 + e*cos(nu));
//position in perifocal system(PQW)
r_PQW[0] = R*cos(nu);
r_PQW[1] = R*sin(nu);
r_PQW[2] = 0.0;
csr = cos(raan);
snr = sin(raan);
csw = cos(w);
snw = sin(w);
csi = cos(i);
sni = sin(i);
//Transformation matrix
r00 = csr*csw-snr*snw*csi;
r01 = -csr*snw-snr*csw*csi;
r02 = snr*sni;
r10 = snr*csw + csr*snw*csi;
r11 = -snr*snw + csr*csw*csi;
r12 = -csr*sni;
r20 = snw*sni;
r21 = csw*sni;
r22 = csi;
```

```
    //transformation of position from PQW to IJK
    sv[0] = r00*r_PQW[0]+ r01*r_PQW[1] + r02*r_PQW[2];
    sv[1] = r10*r_PQW[0]+ r11*r_PQW[1] + r12*r_PQW[2];
    sv[2] = r20*r_PQW[0]+ r21*r_PQW[1] + r22*r_PQW[2];
    //velocity in perifocal system (PQW)
    r_PQW[0] = -mup*sin(nu);
    r_PQW[1] = mup*(e+cos(nu));
    r_PQW[2] = 0.0;
    //transformation of velocity from PQW to IJK
    sv[3] = r00*r_PQW[0]+ r01*r_PQW[1] + r02*r_PQW[2];
    sv[4] = r10*r_PQW[0]+ r11*r_PQW[1] + r12*r_PQW[2];
    sv[5] = r20*r_PQW[0]+ r21*r_PQW[1] + r22*r_PQW[2];
    return;
}

/*************************************************************
 Name: ECI_ECEF.cpp
 Description: Transforms ECI state vector to ECEF then computes
              Geographic longitude, latitude and altitude.
 Input(s) ____________________________________________________
          JD       = Current Julian date
          R[]      = ECI state vector X,Y,Z,XD,YD,ZD,(m,m/sec)
 Output(s) ___________________________________________________
          *sa      = Pointer to an array containing:
                     Longitude (rad)
                     Latitude (rad)
                     Altitude (m)
 Date Last Modified: 05 March 2014
 Author: Tshilande T
*************************************************************/
void ECI_ECEF(double JD, double R[],double *sa)
```

```
{
    double gast,Range,Radius,sphi,x,y,z;
    double R_ECEF[3];
    //Compute GAST
    gast = app_sidereal_time(JD);
    gast = gast*DtR;
    //Rotate ECI vector to ECEF frame
    R_ECEF[0] = cos(gast)*R[0] + sin(gast)*R[1];
    R_ECEF[1] = -sin(gast)*R[0] + cos(gast)*R[1];
    R_ECEF[2] = R[2];
    //calculate geocentric lon, lat in radians
    x= R_ECEF[0]; y= R_ECEF[1]; z= R_ECEF[2];
    Range= x*x+ y*y + z*z;
    Radius = sqrt(Range);
    *(sa+0) = atan2(y,x);//GeoLon
    sphi = z/Radius;
    *(sa+1) = asin(sphi);//GeoLat
    //Altitude
    //*(sa+2) = Radius - R_earth;
    *(sa+2) = Radius -
sqrt(R_earth*R_earth*(1.-eccent*eccent)/(1.-pow(eccent*cos(*(sa+1)),2)));
  return;
}

/*************************************************************
 Name: JulianDate.cpp
 Description: Computes current julian date given year, month,
              day and universal time in seconds
 Input(s) ____________________________________________________
          UT      = Current Julian time (sec)
  day     = day of the month.
```

```
  month   = month of the year.
  year    = year
 Output(s) _____________________________________________________
          JD      = Current Julian date
 Date Last Modified: 13 February 2014
 Author: Tshilande T
*****************************************************************/
double JulianDate(int year, int month, int day, double UT){
    double JD,C1,C2,C3;
    C1 = 367. * year;
    C2 = int((7*(year+int((month+9)/12)))*0.25);
    C3 = int(275*month/9);
    UT = UT/3600.;
    JD = (C1-C2+C3)+ day + 1721013.5 + UT/24.;
    return JD;
}

int djm (int day, int month, int year)
/*
Purpose:
The function djm furnishes the modified julian
date with reference to the day, month and year,
input real parameters at zero hours of the day.
Input:
day     day of the month.
month   month of the year.
year    year
Output:
djm     modified julian date in days, referred to
                1950.0.
Remarks:
```

```
        This function is optimized for the period between
        the years 1900-2100.
Author:
        Valdemir Carrara        july 2005     (C version)
*/
{
      return 367*year + day - 712269 + (int)floor(275*month/9.)
     - (int)floor(7*(year + floor((month + 9)/12.))/4.);
} // djm
```

Bibliography

Auret, J. 2012. Design of an aerodynamic attitude control system for a CubeSat. Master's thesis, University of Stellenbosch.

Bate, R. R., Mueller, D. D. and White, J. E. 1971. *Fundamentals of Astrodynamics.*

Brouwer, D. 1959. Solution of the problem of artificial satellite theory without drag. *The Astronomical Journal*, 64(1274): 378–397.

Cash, J. and Karp, A. H. 1990. A variable order runge-kutta method for initial value problems with rapidly varying right-hand sides. *ACM Transactions on Mathematical Software*, 16(3): 201–222.

Catsoulis, J. 2005. *Designing Embedded Hardware.* Second edn, O'Reilly.

Chaisson, E. and McMillan, S. 2008. *Astronomy Today.* 6th edn, Pearson Education (US).

Chao, B. F. 2006. Earth's oblateness and its temporal variations. *Comptes Rendus Geoscience*, 338(14–15): 1123–1129.

Chobotov, V. A. 2002. *Orbital Mechanics.* Third edn, Reston, Virginia, AIAA.

Clarke, A. C. 1945. Extra-terrestrial relays: Can rocket stations give world-wide radio coverage? *Wireless World*, 11(10): 305–308.

Curtis, H. D. 2010. *Orbital Mechanics for Engineering Students.* Second edn, Elsevier.

Du Toit, D. N. 1997. Low Earth Orbit Satellite Constellation Control Using Atmosphere Drag. PhD thesis, University of Stellenbosch.

Escobal, P. R. 1985. *Methods of Orbit Determination*. Malabar, Florida, Krieger Publishing Company.

Fukushima, T. 1997. Vector integration of dynamical motions by the picard-chebyshev method. *The Astronomical Journal*, 113(6): 2325–2328.

Gaposchkin, E. and Coster, A. 1988. Analysis of satellite drag. *The Lincoln Laboratory*, 1(2): 203–224.

Ilyas, D. 2011. Orbital propagation and formation flying of cubesats within QB50 constellation. Master's thesis, Lulea University of Technology.

King-Hele, D. 1987. *Satellite Orbits in an Atmosphere: Theory and Applications*. Blackie.

Kozai, Y. 1962. Mean values of cosine functions in elliptic motion. *The Astronomical Journal*, 67(5): 311–312.

Liu, J. 1974. Satellite motion about an oblate earth. *AIAA Journal*, 12(11): 1511–1516.

Lumbwe, L. T. 2013. Development of an onboard computer(OBC) for a cubesat. Master's thesis, Cape Peninsula University of Technology.

Maini, A. K. and Agrawal, V. 2007. *Satellite Technology, Principles and Applications*. John Wiley and Sons Ltd.

Meeus, J. 1991. *Astronomical Algorithms*. Second edn, Richmond, Virginia 23235, Willmann-Bell.

Montenbruck, O. and Gill, E. 2001. *Satellite Orbits: Models,Methods, and Applications*. Springer.

Mostert, S. and Koekemoer, J.-A. 1997. The science and engineering payloads and experiments on SUNSAT. *Acta Astronautica*, 41(4-10): 401–411.

Opperman, B. D. 2003. Precision propagation and orbit decay prediction of low earth orbit satellites. Master's thesis, University of Stellenbosch.

Panajaya, F. M., Zee, R. E., Thomsen, P. L., Blanke, M., Wisniewski, R., Franklin, L. and Puig-Suari, J. 2003. An affordable, low-risk approach to launching research spacecraft as tertiary payloads, 17th Annual AIAA/USU Conference on Small Satellites.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. and Flannery, B. P. 2002. *Numerical Recipes in C++ : The Art of Scientific Computing*. Second edn.

Rim, H. J. and Schutz, B. E. 2002. PRECISION ORBIT DETERMINATION (POD), Technical report, The University of Texas, Austin.

San-Juan, J. F., Lara, M., Lopez, R., Lopez, L. M., Folcik, Z. J. and Cefola, P. J. 2011. Using the DSST semi-analytic orbit propagator package via the nondy webtools/astro webtools open science environment. *IAC*, 11(B5.2.9): 1–8.

Sellers, J. J., Astore, W. J., Giffen, R. B. and Larson, W. J. 2004. *Understanding Space: An introduction to Astronautics*. McGraw-Hill.

Steyn, W. 2008. An attitude control system for SumbandilaSat - an earth observation satellite, ESA 4S Symposium.

Vallado, D. A. 1997. *Fundamentals of Astrodynamics and Applications*. McGraw-Hill.