


11-2022

ORBIT PROPAGATION AND DETERMINATION ALGORITHMS FOR SATELLITE GROUND STATIONS

Shamma Esmaeel Jamali

Follow this and additional works at: https://scholarworks.uaeu.ac.ae/all_theses

 Part of the [Astrophysics and Astronomy Commons](#), [Other Physical Sciences and Mathematics Commons](#), and the [Other Physics Commons](#)



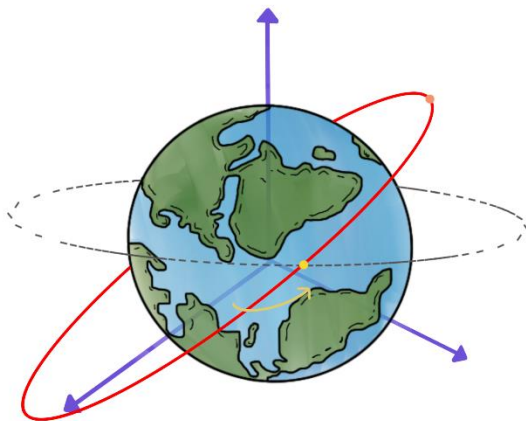
MASTER THESIS NO. 2022: 81

College of Science

Department of Physics

**ORBIT PROPAGATION AND DETERMINATION
ALGORITHMS FOR SATELLITE GROUND STATIONS**

Shamma Esmaeel Jamali



November 2022

United Arab Emirates University

College of Science

Department of Physics

ORBIT PROPAGATION AND DETERMINATION ALGORITHMS
FOR SATELLITE GROUND STATIONS

Shamma Esmaeel Jamali

This thesis is submitted in partial fulfilment of the requirements for the degree of Master
of Science in Space Science

November 2022

United Arab Emirates University Master Thesis
2022: 81

Orbit Propagation and Determination Algorithms for Ground Stations

Cover: Satellites Orbit

(Photo: By Shamma Esmaeel Jamali)

© 2022 Shamma Esmaeel Jamali, Al Ain, UAE

All Rights Reserved

Print: University Print Service, UAEU 2022

Declaration of Original Work

I, Shamma Esmaeel Jamali, the undersigned, a graduate student at the United Arab Emirates University (UAEU), and the author of this thesis entitled “*Orbit Propagation and Determination Algorithms for Satellite Ground Stations*”, hereby, solemnly declare that this is the original research work done by me under the supervision of Prof. Mohammad Mehedy Masud, in the College of Science at UAEU. This work has not previously formed the basis for the award of any academic degree, diploma or a similar title at this or any other university. Any materials borrowed from other sources (whether published or unpublished) and relied upon or included in my thesis have been properly cited and acknowledged in accordance with appropriate academic conventions. I further declare that there is no potential conflict of interest with respect to the research, data collection, authorship, presentation and/or publication of this thesis.

Student's Signature: _____



Date: 28/November /2022

Advisory Committee

1) Advisor: Mohammad Mehedy Masud

Title: Associate Professor

Department of Information Systems and Security

College of Information Technology

2) Co-advisor: David Gil

Title: Software Engineer

Department of National Space Science & Tech. Center

Approval of the Master Thesis

This Master Thesis is approved by the following Examining Committee Members:

- 1) Advisor (Committee Chair): Mohammad Mehedy Masud

Title: Associate Professor

Department of Information Systems and Security

College of Information Technology

Signature  Date 05 December 2022

- 2) Member: Mohammad Hayajneh

Title: Associate Professor

Department of Computer and Network Engineering

College of Information Technology

Signature 
Mohammad Hayajneh Date 07 December 2022

- 3) Member (External Examiner): Sean Shan Min Swei

Title: Professor

Department of Aerospace Engineering

Institution: Khalifa University, UAE

Signature  Date 07 December 2022

This Master Thesis is accepted by:

Dean of the College of Science: Professor Maamar Benkraouda

Signature Maamar Benkraouda

Date Jan. 5, 2023

Dean of the College of Graduate Studies: Professor Ali Al-Marzouqi

Signature _____

Date _____

Abstract

The satellite orbital parameters are essential for satellite operations. With these parameters, it is possible to estimate the satellite position in the recent past and near future, which is essential to effectively plan satellite operations and associate satellite telemetry with geographical locations.

However, for small or medium satellite operators who do not possess the infrastructure required to track their satellites, the problem of determining the satellite orbit is problematic. To access the orbit for their satellites, these organizations have to rely on third parties such as Celestrak. These entities provide the service free of charge but do not provide orbital parameters with the required frequency. Furthermore, another problem may arise during the mission's early phases. Suppose the satellite is launched together with a number of other satellites, as is often done for small satellites. In that case, it is also not known in the first days or weeks of the mission which orbital parameters are from which satellite launched in the group. This project aims to address the problem of orbital parameter determination by using GPS data, Kalman filters and AI (genetic algorithm).

Keywords: Genetic Algorithm, GPS, Kalman Filters, Orbit Determination Algorithms, Orbit Propagation, Orbital Parameters, Satellite.

Title and Abstract (in Arabic)

خوارزميات لانتشار ولتحديد المدار، للمحطات الأرضية للأقمار الصناعي

الملخص

تعتبر العناصر المدارية للقمر الصناعي ضرورية لعمليات الأقمار الصناعية، ومن خلال هذه العناصر، من الممكن تقدير موقع القمر الصناعي في الماضي والمستقبل القريب وهو أمر ضروري للتخطيط الفعال لعمليات الأقمار الصناعية وربط القياس عن بعد بالأقمار الصناعية بالمواقع الجغرافية.

ومع ذلك، بالنسبة للأقمار الصناعية الصغيرة / المتوسطة الذين لا يمتلكون البنية التحتية اللازمة لتتبع أقمارهم الصناعية، فإن مشكلة تحديد مدار القمر الصناعي هي مشكلة. للوصول إلى هذا من أجل أقمارها الصناعية، يتعين على المنظمات الاعتماد على أطراف ثالثة مثل "Celestrak" الذين على الرغم من أنهم يقدمون الخدمة مجاناً، إلا أنهم لا يوفران المعلومات المدارية بالتردد المطلوب، خاصة بالنسبة للأقمار الصناعية غير المهمة وأثناء المراحل المبكرة للمهمة التي قد تنشأ فيها مشكلة أخرى: إذا تم إطلاق القمر الصناعي مع عدد من الأقمار الصناعية الأخرى - كما هو الحال في كثير من الأحيان مع الأقمار الصناعية الصغيرة - فإنه لا يُعرف أيضاً الأيام أو الأسابيع الأولى من المهمة، أي العناصر المدارية التي نشأت منها إطلاق الأقمار الصناعية في المجموعة يهدف هذا المشروع إلى معالجة مشكلة تحديد المعلومات المدارية باستخدام بيانات نظام تحديد المواقع ومرشحات كالمان والذكاء الاصطناعي (الخوارزمية الجينية).

مفاهيم البحث الرئيسية: الخوارزمية الجينية، نظام تحديد المواقع، مرشحات كالمان، خوارزميات تحديد المدار، انتشار المدار، المعلومات المدارية، الأقمار الصناعية.

Acknowledgements

I would like to thank my committee for their guidance, support, and assistance throughout my preparation of this thesis, especially my advisor Dr. Mohammad Mehedy Masud and Eng. David Gil for their guidance and advice to make this project possible.

Special thanks go to Dr. Abdul-Halim Jallad and NSSTC for giving me this opportunity to work in such a project.

Dedication

To the people who have supported me...

Table of Contents

Title.....	i
Declaration of Original Work.....	iii
Advisory Committee.....	iv
Approval of the Master Thesis	v
Abstract.....	vii
Title and Abstract (in Arabic).....	viii
Acknowledgements.....	ix
Dedication.....	x
Table of Contents.....	xi
List of Tables	xiii
List of Figures.....	xiv
List of Abbreviations	xvi
Chapter 1: Introduction.....	1
1.1 Overview	1
1.2 Statement of the Problem	1
1.3 Research Objectives	1
1.4 Satellite.....	2
1.5 GNSSaS.....	3
1.6 Classical Kepler Orbital Parameters	3
1.6.1 Semi-Major Axis.....	4
1.6.2 Eccentricity	5
1.6.3 True Anomaly	7
1.6.4 Inclination	8
1.6.5 Right Ascending of the Ascending Node (RAAN)	10
1.6.6 Argument of Perigee	11
1.7 Two-Line Element (TLE).....	12
1.7.1 Julian Day with Fraction.....	15
1.7.2 Ephemeris Type	16
1.7.3 Mean Motion.....	17
1.7.4 B-STAR Drag (B*).....	18
1.7.5 Mean Anomaly.....	18

Chapter 2: Methods.....	19
2.1 Overview	19
2.2 Global Navigation Satellite System	19
2.3 Orbit Determination Problem.....	22
2.4 Extended Kalman Filter Algorithm.....	24
2.5 Unscented Kalman Filter Algorithm	26
2.6 Previous Studies on the Orbit Determination Problem	28
2.7 Genetic Algorithm.....	29
Chapter 3: Results and Discussions.....	31
3.1 Overview	31
3.2 Data Source	31
3.3 Satellite Dynamics Model	34
3.4 Extended Kalman Filter (EKF)	35
3.5 Unscented Kalman Filter (UKF).....	39
3.6 Genetic Algorithm (GA)	42
Chapter 4: Experiments, Results, Discussion, Recommendation and Conclusion	50
4.1 Results and Discussion.....	50
4.1.1 Experimental Setup.....	50
4.1.2 Part A	50
4.1.3 Part B	63
4.1.4 Part C	70
4.2 Recommendation	73
4.3 Conclusion.....	74
References.....	75
Appendices	79
Appendix A	79
Appendix B	82
Appendix C	84
Appendix D	95
Appendix E.....	116

List of Tables

Table 1: Satellite Sizes.....	2
Table 2: Type of Orbits Based on Semi-Major Axis.....	5
Table 3: Type of Orbits Based on Eccentricity	6
Table 4: Type of Orbits and Their Inclination.....	9
Table 5: TLE "line 1" Description.....	13
Table 6: TLE "line 2" Description.....	15
Table 7: Ephemeris Type Value and Simplified Perturbations Model.....	16
Table 8: Initial Position and Velocity Vector [MEZNSAT]	33
Table 9: Initial Orbital Parameters [MEZNSAT].....	33
Table 10: Initial Position and Velocity Vector [EOS-01]	34
Table 11: Initial Orbital Parameters [EOS-01].....	34
Table 12: System Hardware Configurations	50
Table 13: GA results Comparison	66
Table 14: GA results Comparison – with actual TLE values.....	67
Table 15: GA Results Comparison - EOS-01.....	70
Table 16: Actual TLE Vs Algorithm Best TLE	71

List of Figures

Figure 1: Semi-Major Axis.....	4
Figure 2: True Anomaly	7
Figure 3: Inclination	8
Figure 4: Right Ascending of The Ascending Node	11
Figure 5: Argument of Perigee	12
Figure 6: TLE Format.....	12
Figure 7: Extended Kalman Filter Flowchart.....	25
Figure 8: STK Steps.....	32
Figure 9: Extended Kalman Filter (EKF) Flowchart.....	36
Figure 10: Unscented Kalman Filter (UKF) Flowchart	40
Figure 11: Genetic Algorithm Flowchart	43
Figure 12: Python Code - Generating Initial Population.....	44
Figure 13: Example of Chromosome.....	45
Figure 14: Python Code - Convert TLE to Position and Velocity Vector.....	46
Figure 15: Type of Crossovers	47
Figure 16: Example of Crossover	48
Figure 17: Type of Mutation	49
Figure 18: Example of Mutation	49
Figure 19: Extended Kalman Filter Results Vs Actual Results – MEZNSAT (X-Axis).....	51
Figure 20: Extended Kalman Filter Results Vs Actual Results – MEZNSAT (Y-Axis).....	52
Figure 21: Extended Kalman Filter Results Vs Actual Results – MEZNSAT (Z-Axis)	53
Figure 22: EKF RMS – MEZNSAT.....	54
Figure 23: Extended Kalman Filter Result (Orbital Parameters) – MEZNSAT (Part 1)	55
Figure 24: Extended Kalman Filter Result (Orbital Parameters) – MEZNSAT (Part 2)	56
Figure 25: Unscented Kalman Filter Results Vs Actual Results (X-axis).....	57
Figure 26: Unscented Kalman Filter Results Vs Actual Results (Y-axis).....	58
Figure 27: Unscented Kalman Filter Results Vs Actual Results (Z-axis)	59
Figure 28: UKF – RMS	60
Figure 29: Unscented Kalman Filter Result (Orbital Parameters – Part 1).....	61

Figure 30: Unscented Kalman Filter Result (Orbital Parameters – Part 2)	62
Figure 31: GA Result with one hundred Alterations	64
Figure 32: GA Result with 1000 Alterations	65
Figure 33: GA results Comparison – with actual TLE values (Part 1)	68
Figure 34: GA results Comparison – with actual TLE values (Part 2)	69
Figure 35: Actual TLE Vs Algorithm Best TLE	72
Figure 36: Algorithms Run Time	73

List of Abbreviations

EKF	Extended Kalman Filter
GA	Genetic Algorithm
GEO	Geostationary Earth Orbit
GNSS	Global Navigation Satellite Systems
LEO	Low Earth Orbit
MEO	Medium Earth Orbit
NORAD	North American Aerospace Defense Command
NSSTC	National Space Science and Technology Center
RAAN	Right Ascension of The Ascending Node
RF	Radio Frequency
SATCAT	Satellite Catalogue Number
TLE	Two-Line Element
UKF	Unscented Kalman Filter

Chapter 1: Introduction

1.1 Overview

This thesis work investigates the utilization of the extended Kalman filter (EKF), unscented Kalman filter (UKF), and artificial intelligence algorithm such as genetic algorithm (GA) to address the orbit determination and orbit propagation problem of small or medium satellites using a GPS-based navigation system. The GPS navigation system uses Kalman filters to improve the position and velocity accuracy. In this thesis, the onboard orbit determination algorithm for the spacecraft is developed using three different techniques, namely, the EKF, UKF and GA. The EKF uses a state transition Jacobian matrix to determine the covariance. In contrast, the UKF adopts the second-order Taylor series expansion through finite sigma points to determine the covariance and mean. The GA adopts the principles of natural selection and Genetic to generate possible solutions to the orbit propagation problem and determine the best solution (TLE).

1.2 Statement of the Problem

Third-party companies like Celestrak provide data of the satellite orbital parameters and other data to help track the satellites. However, they do not provide more frequent data for small or medium satellites. This thesis aims to find the best algorithms to provide the needed information to track the satellite without depending on third parties by utilizing the Kalman filter algorithm and artificial intelligence algorithm.

1.3 Research Objectives

The research objectives are as follows:

- I. Implementing an orbit propagation algorithm that generates entries in a relational database, which are then retrieved by another device to visualize geo-referenced data.
- II. Implement orbit determination algorithms that use on-board data from the satellite (namely from the GPS receiver) with the purpose of producing orbital parameters

and therefore end or reduce the reliance on third parties and enable a smooth and efficient operation of satellites from the early days of the mission

1.4 Satellite

A satellite is an object that orbits another object, such as the moon orbiting the earth. Moon is a natural satellite. Alternatively, an artificial or man-made satellite is orbiting the earth at a different altitude. The altitude is described as the difference between the earth’s surface and the object that is orbiting the earth. Artificial satellites are mainly used for communication and navigation systems and to collect data to discover the earth, other planets, or objects in the universe. Their vast range of size and weights allow them to have different functions, as shown in Table 1.

Table 1: Satellite Sizes

Category	Weight [kg]	Orbit	Mission/ application
Large Satellites	>1000	GEO*	- Weather Data
			- Broadcast TV
			- communication
			- Early warning
Medium Satellites	500—1000	LEO**, MEO*** and GEO*	- Nuclear detection
			- GPS and navigation application
Small Satellites	100—500	LEO	- communication
	10—100	LEO	- Remote sensing
	1—10	LEO	- Space shuttle
	<1	LEO	

* Geostationary Earth Orbit: Altitude of 36,000 km
 ** Medium Earth Orbit: Altitude between 5,000—20,000 km
 *** Low Earth Orbit: Altitude between 500—1200 km

Satellites can have regular and irregular shapes. Nanosatellites, called CubeSats, are satellites with a cube shape with standard dimensions in a unit (U). A 1 U CubeSat has a configuration of 10 cm width, 10 cm length and 10 cm height, with a weight of 1 kg. Other nanosatellites have standard unit sizes of 1.5 U, 2 U, 3 U, 6 U, and 12 U.

1.5 GNSSaS

GNSSaS is a 6 U CubeSat by The National Space Science and Technology Center (NSSTC) in the United Arab Emirates University (UAEU). The CubeSat will be launched into LEO at an altitude of 500 — 550 km. With a mass of less than 12 kg and dimensions of 22.63 cm x 34.05 cm x 10 cm. The CubeSat is built to study various Global Navigation Satellite Systems (GNSS) — is a network of satellites that provide time and orbital information from space, used to determine the location, position and navigational measurement — augmentation techniques by transmitting augmentation signals from LEO. It will validate algorithms to verify high precision. It will perform remote sensing to track the ionosphere effects on RF signals.

1.6 Classical Kepler Orbital Parameters

Apart from using the altitude to describe an orbit, the Spacecraft orbit can also be described using the six classical Kepler orbital parameters. The Kepler Orbital Parameters are six unique orbital elements that describe a spacecraft's orbit. They are categorized as either dimensional or orientational elements. On the one hand, dimensional elements describe the orbit type, size, and shape. They include the semi-major axis, eccentricity, and true anomaly.

Conversely, the orientational elements describe the spacecraft's orbit orientation. They include the inclination, right ascension of the ascending node (RAAN), and argument of perigee.

1.6.1 Semi-Major Axis

The Semi-major axis (a) is one-half of the distance across the long axis. It extends from the center through a focus to the edge of the ellipse a in the Figure 1.

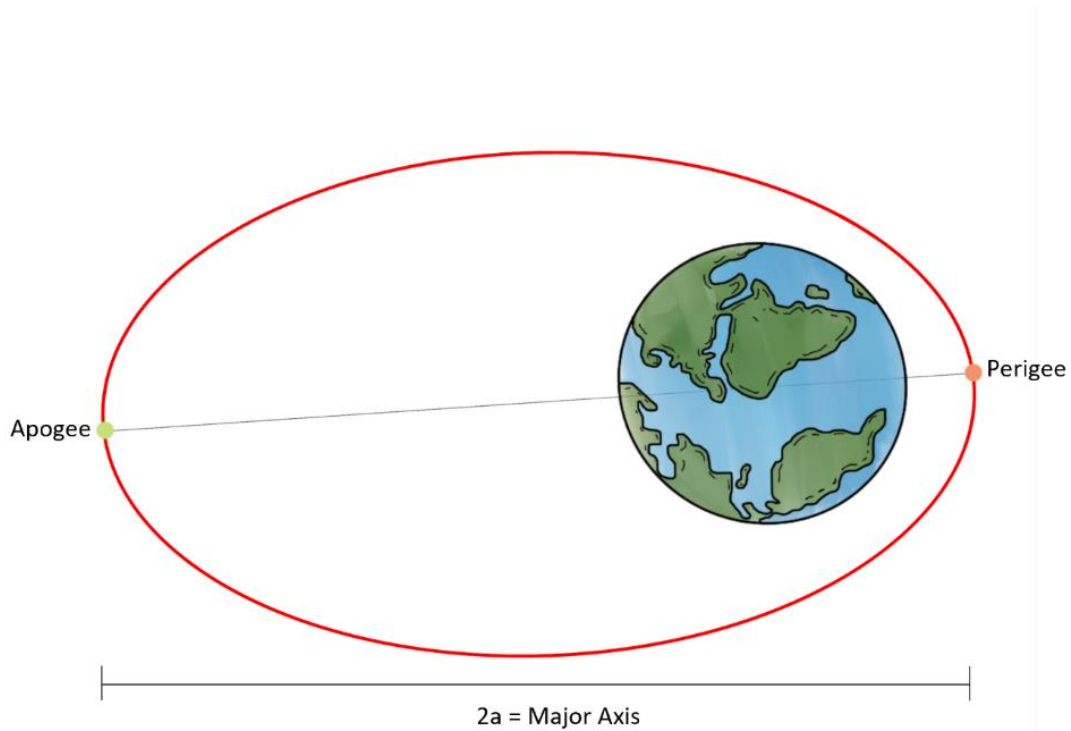


Figure 1: Semi-Major Axis

It can also be a tool to determine the time period of revolution made by a spacecraft's orbit. Semi-major axis is determined by the following equation:


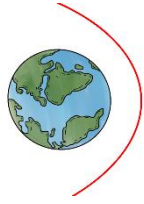
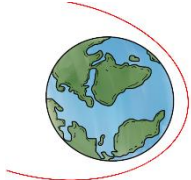
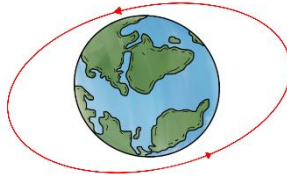
$$a = -\frac{\mu}{2\varepsilon} \quad (1)$$

Where μ is the standard earth gravitational constant ($\mu = 3.986004418 \times 10^5 \text{ km}^3\text{s}$), the specific energy of the orbiting body (ε) given by Equation (2):

$$\varepsilon = \frac{V^2}{2} - \frac{\mu}{R} \quad (2)$$

The semi-major axis must be greater than the orbital radius of the satellite. Otherwise, the satellite will crash into the planet. This parameter can provide information about the orbit's size and shape, as shown in Table 2.

Table 2: Type of Orbits Based on Semi-Major Axis

Semi-major axis value	Orbit type	Figure
Semi-major axis = attitude	Circular orbit	
Negative value	Hy-parabola orbit	
Infinite	parabola orbit	
Other value	Ellipse orbit	


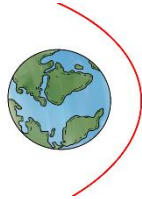
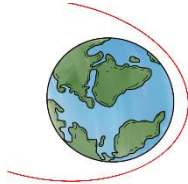
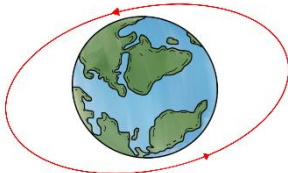
1.6.2 Eccentricity

The eccentricity (e) parameter indicates the deviation of the orbital shape from a perfect circle. It is utilized in fixing the shape of the satellite's orbit. In other words, the eccentricity determines the shape of the orbit (see Table 3) by considering the ratio of between the distance between the two foci and the length of the major axis as seen in Equations 3 and 4.

$$e = \frac{2c}{2a} \quad (3)$$

$$\mathbf{e} = \frac{1}{\mu} \left[\left(V^2 - \frac{\mu}{R} \right) \mathbf{R} - (\mathbf{R} \cdot \mathbf{V}) \mathbf{V} \right] \quad (4)$$

Table 3: Type of Orbits Based on Eccentricity

Eccentricity value	Orbit type	Figure
$e = 0$	Circular orbit	
$e > 1$	Hy-parabola orbit	
$e = 1$	parabola orbit	
$0 < e < 1$	Ellipse orbit	

1.6.3 True Anomaly

True anomaly (v) is an angle between the perigee and the satellite position vector, as reflected in Figure 2.

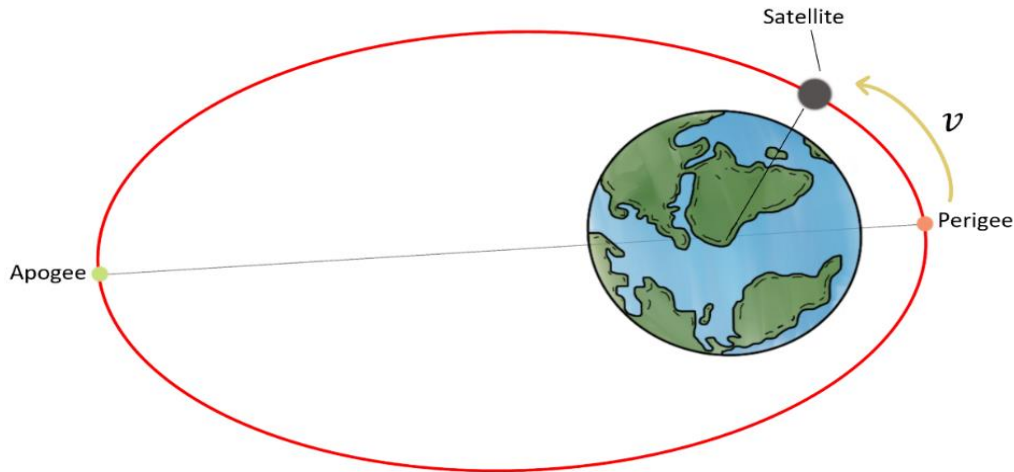


Figure 2: True Anomaly

This parameter provides the ground station with the knowledge of the satellite's location, making it the only parameter that changes with time. True anomaly is introduced by:

$$v = \cos^{-1} \left(\frac{\mathbf{e} \cdot \mathbf{R}}{eR} \right) \quad (5)$$

Suppose the value of the dot product of the position and velocity vectors is less than one. In that case, the value of the true Anomaly is between 180 and 360 degrees. However, if the result of the dot product is more significant than one, then the value of true Anomaly is between 0 and 180 degrees. In the case of zero, the true Anomaly will depend on the value R at the perigee — a point in the orbit where the satellite is nearest the earth — and apogee, a point in the orbit where the satellite is furthest from the earth, radius. If $R = R(\text{perigee})$, then true Anomaly is equal to 0 degrees, but if $R = R(\text{apogee})$,

then true Anomaly is equal to 180 degrees. In the case of a circular orbit ($e = 0$) a true anomaly is not defined due to the absence of the perigee point.

1.6.4 Inclination

Figure 3 shows the angle between the reference and the orbital planes called the inclination (i).

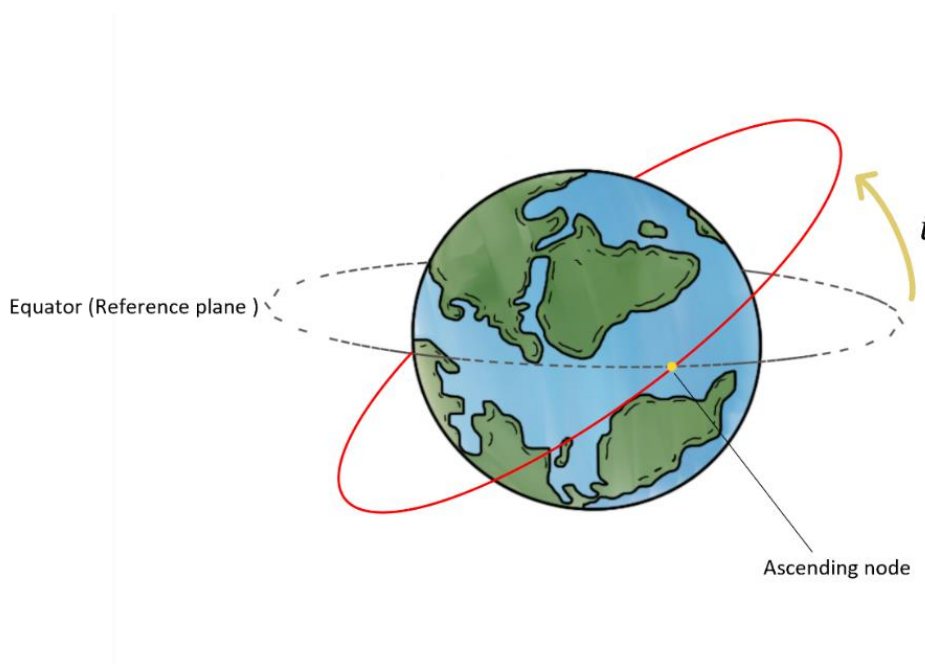


Figure 3: Inclination

The inclination is one of the hardest parameters to calculate mathematically. It can be calculated by determining the angle between two vectors. One of the vectors is perpendicular to the orbital plane (The specific angular momentum vector (\mathbf{h})) and one is perpendicular to the fundamental plane, as seen in the formula:

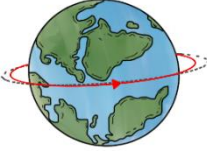
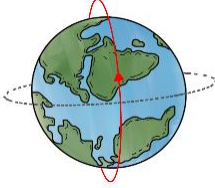
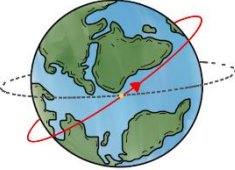
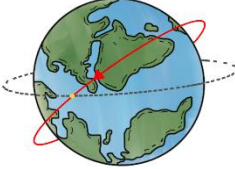
$$i = \cos^{-1} \left(\frac{\mathbf{h} \cdot \hat{\mathbf{K}}}{hk} \right) = \cos^{-1} \left(\frac{h_k}{h} \right) \quad (6)$$

In which, the specific angular moment vector (\mathbf{h}) is given by:

$$\mathbf{h} = \mathbf{R} \times \mathbf{V} \quad (7)$$

$\hat{\mathbf{K}}$ is a unit vector that points from the earth's centre to the north pole equal to $\hat{\mathbf{K}} = 0i + 0j + 1k$, h_k is the k^{th} Component of the specific angular moment vector. The inclination has a range of values between 0 and 180 degrees, and can provide different knowledge of the orbit, as seen in Table 4.

Table 4: Type of Orbits and Their Inclination

Inclination value	Orbit type	Figure
$i = 0^\circ \text{ or } 180^\circ$	Equatorial orbit	
$i = 90^\circ$	Polar orbit	
$0^\circ < i < 90^\circ$	Direct or Prograde orbit	
$90^\circ < i < 180^\circ$	Indirect or Retrograde orbit	

1.6.5 Right Ascending of the Ascending Node (RAAN)

The ascending node is the point where the satellite crosses the equatorial plane when moving from the southern hemisphere to the northern hemisphere. Hence, the right ascending of the ascending node (Ω), is the angle between the line of vernal or equinox and the ascending node towards the east direction on an equatorial plane and is used to describe the orbital orientation of the orbit plane with the respect to the principal direction $\hat{\mathbf{I}}$ (see Figure 4). RAAN can be determined using the following equation:

$$\Omega = \cos^{-1} \left(\frac{\hat{\mathbf{I}} \cdot \mathbf{n}}{In} \right) = \cos^{-1} \left(\frac{n_i}{h} \right) \quad (8)$$

Where, the ascending node vector (\mathbf{n}) is a node or a point where the orbital plate and the earth equatorial plate meets when the satellite move from south to north and it is given by:

$$\mathbf{n} = \hat{\mathbf{I}} \times \mathbf{n} \quad (9)$$

$\hat{\mathbf{I}}$ is a unit vector that points from the center of the earth to the vernal equinox direction — the line from the earth to the sun on the first day of spring, March 21 — equal to $\hat{\mathbf{I}} = 1i + 0j + 0k$, n_k is the i^{th} Component of ascending node vector. If n_j if less than zero, then RAAN value is between 180 and 360 degrees ($\Omega = 360 - \Omega$). However, if n_j greater or equal zero the RAAN is between 0 and 180 degrees.

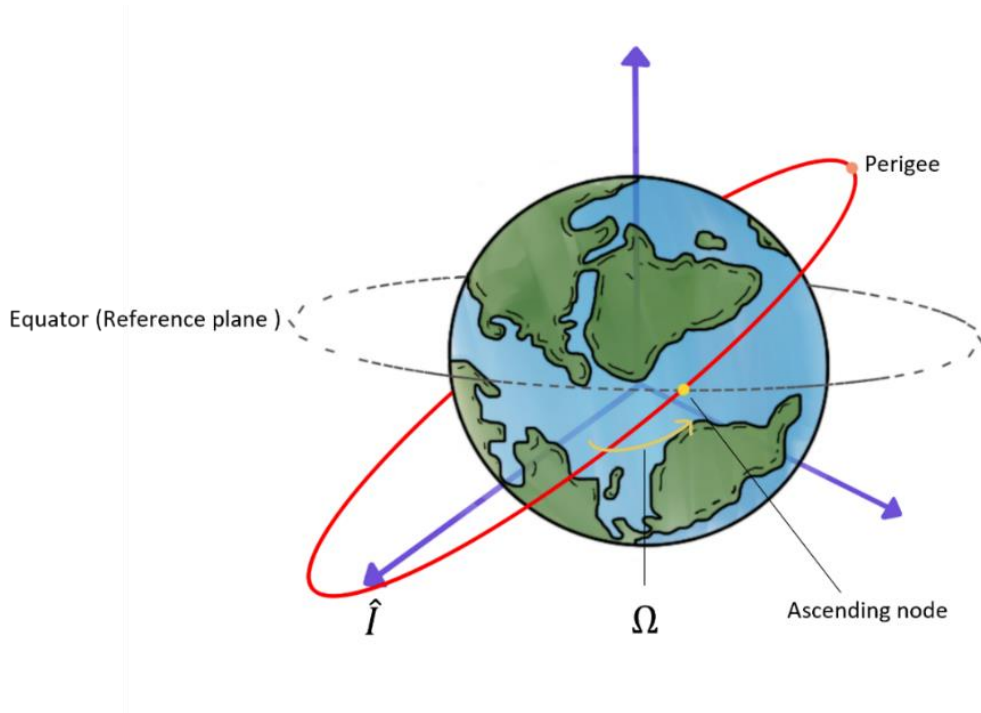


Figure 4: Right Ascending of The Ascending Node

1.6.6 Argument of Perigee

Argument of Perigee (ω) is the angle between the ascending node and perigee (see Figure 5). It is used to show the orientation of the orbit and located perigee points. It is calculated using the formula below:

$$\omega = \cos^{-1} \left(\frac{\mathbf{n} \cdot \mathbf{e}}{ne} \right) \quad (10)$$

If the ascending node and perigee are located at the same point, the argument of perigee is zero degrees. Besides, in circular orbits where the eccentricity is zero, the perigee is not defined, which results in the argument of perigee being undefined.

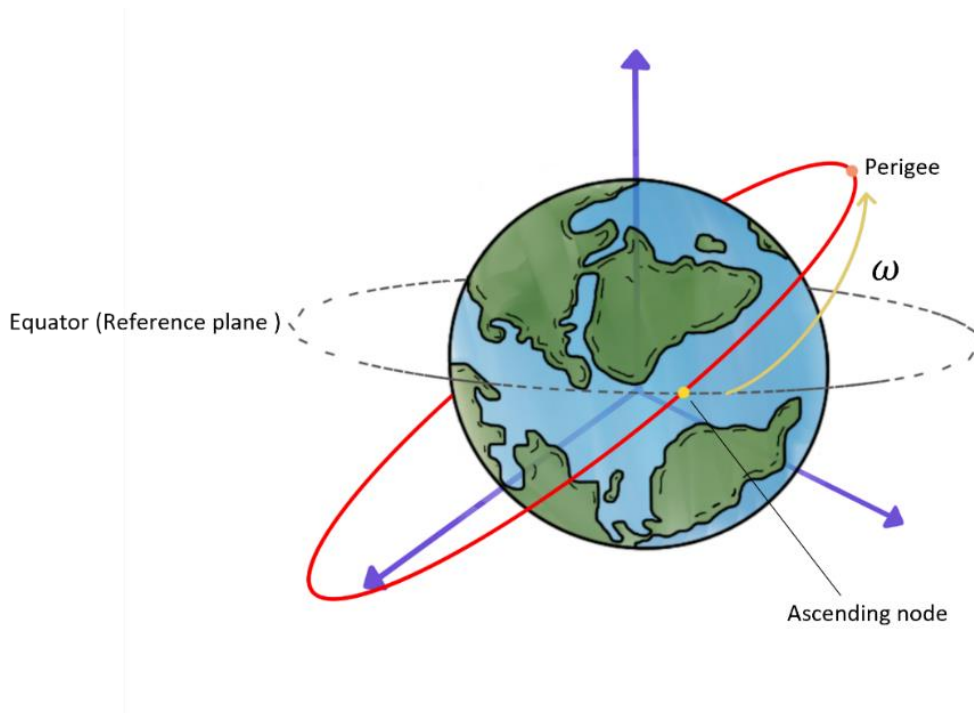


Figure 5: Argument of Perigee

1.7 Two-Line Element (TLE)

The two-Line Elements (TLE) summarizes the orbital parameters into two lines of information. These two lines are three lines called “line 0”, “line 1”, and “line 2”. Each line contains a maximum of sixty-nine alphanumeric characters in the following format:

```

AAAAAAAAAAAAAAAAAAAAAAAAAAAA
1 NNNNNU NNNNNAAA NNNNN.NNNNNNNN +.NNNNNNNNN +NNNNN-N +NNNNN-N N NNNNN
2 NNNNN NNN.NNNN NNN.NNNN NNNNNNNN NNN.NNNN NNN.NNNN NN.NNNNNNNNNNNNNNN

```

Figure 6: TLE Format

Columns with the letter 'N' can sometimes have any number 0 – 9 or space, whereas columns with the letter 'A' can have any character's A – Z. Columns with a '+' or '-' can have either a plus, minus, or a space (T.S, 2020).

“Line 0” comprise the satellite name in NORAD SATCAT, which are about 24 characters (T.S, 2020). Table 5 – 6 indicates the parameters in “line 1”, and “line 2”.

Table 5: TLE "line 1" Description

Column	Content	Format	Description
01	Line number of element	1	Can be 1 or 2
03 – 07	Satellite number/NORAD catalogue number	NNNNN	a unique identifier assigned by NORAD for each earth-orbiting artificial satellite in their SATCAT
08	Classification	U	It can be ‘U’ for unclassified data (available for the public) Or ‘S’ for Secret data
10 – 11	International Designator (Last two digits of launch year)	NN	
12 – 14	International Designator (Launch number of the year)	NNN	
15 – 17	International Designator (Piece of the launch)	AAA	This is useful in case of multiple launches Example ‘A’ indicates that this was the first satellite launched
19 – 20	Epoch Year (Last two digits of year)	NN	

Table 5: TLE "line 1" Description (continued)

Column	Content	Format	Description
21 – 23	Epoch (Day of the year and fractional portion of the day)	NNN.NNNNNNNN	Julian Day with Fraction
34 – 43	First Time Derivative of the Mean Motion [revolutions per day ²]	+/- .NNNNNNNN	represents the first derivative of the mean motion divided by two
45 – 52	Second Time Derivative of Mean Motion (decimal point assumed) [revolutions per day ³]	+/-NNNNN-N	represents the second derivative of the mean motion divided by six
54 – 61	BSTAR drag term (decimal point assumed)	NNNNN-N	
63	Ephemeris type	N	NORAD uses SGP4/SDP4 orbital model depend on whether the satellite is near-Earth* or deep-space**
65 – 68	Element number	NNNN	
69	Checksum (Modulo 10) (Letters, blanks, periods, plus signs = 0; minus signs = 1)	N	

Note. From T.S, K. (2020, July 1). *CelesTrak: NORAD Two-Line Element Set Format*. CelesTrak. <https://celestrak.org/NORAD/documentation/tle-fmt.php>

* Satellite with orbital period less than 225 minutes (Hoots & Ronald, 1988).
 ** Satellite with orbital period greater than or equal 225 minutes (Hoots & Ronald, 1988).

Table 6: TLE "line 2" Description

Column	Content	Format
01	Line number of element	2
03 – 07	Satellite number/NORAD catalogue number	NNNNN
09 – 16	Inclination [degrees]	NNN.NNNN
18 – 25	RAAN [degrees]	NNN.NNNN
27 – 33	Eccentricity (decimal point assumed)	NNNNNNN
35 – 42	Argument of Perigee [degrees]	NNN.NNNN
44 – 51	Mean Anomaly [degrees]	NNN.NNNN
53 – 63	Mean Motion [degrees]	NN.NNNNN NNN
64 – 68	Revolution number at epoch * [Revs]	NNNNNNN
69	Checksum (Modulo 10) (Letters, blanks, periods, plus signs = 0; minus signs = 1)	N

Note. From T.S, K. (2020, July 1). *CelesTrak: NORAD Two-Line Element Set Format*. CelesTrak. <https://celestrak.org/NORAD/documentation/tle-fmt.php>

* Counter to how many orbits the satellite has orbited from the launched time. It begins from the ascending node. (The first revolution equals 0)

1.7.1 Julian Day with Fraction

“An astronomical Julian day number is a count of astronomical nychthemérons from the astronomical nychthemeron which began at noon GMT on - 4712-01-01 JC” (Meyer, 2016). In the TLE, dates are calculated by adding all the full days to the elapsed hours, minutes, and seconds in a decimal format. The conversion from the Julian day to datetime can be done first by taking the whole number and look it up in the Julian Date Calendar table (see Appendix A) to get the day and the month. Then take the fractional part of the number and apply the Equations (11 – 15) to get the time in hh:mm:ss format.

$$0.\text{NNNNNNNN} \times 86400 = f \quad (11)$$

$$\frac{f}{3600} = hh.yyy \quad (12)$$

Where, hh is the hours.

$$f - (hh \times 3600) = f1 \quad (13)$$

$$\frac{f1}{60} = mm.yyy \quad (14)$$

Where, mm is the minutes.

$$f1 - (mm \times 60) = ss.yyy \quad (15)$$

Where, ss is the seconds.

1.7.2 Ephemeris Type

Ephemeris type is the Simplified perturbations model that has been used to generate the data. Table 7 represent the Ephemeris type values of different simplified perturbations models.

Table 7: Ephemeris Type Value and Simplified Perturbations Model

Ephemeris type value	simplified perturbations model
0	SGP4/SDP4
1	SGP
2	SGP4
3	SDP4
4	SGP8
5	SDP8

Simplified perturbations models are five mathematical models. Each model consists of a set of models used to predict the position and velocity vector of an object in orbit relative to the Earth-Centered Inertial coordinate system (ECI) (Ma et al., 2016). Space agencies mainly use these models to track satellites. The first Model is the SGP, which is used for near-Earth satellites. The SGP4 model is used for near-Earth satellites. The SDP4 model is an extension of the SGP4 model but is used for deep-space satellites. The SGP8 model is used for near-Earth satellites. The SDP8 model is an extension of SGP8, used for deep-space satellites. All the models except SGP have the same gravitational and atmospheric model (Hoots & Roehrich, 1980).

1.7.3 Mean Motion

The mean of motion (n) is the number of revolutions that the satellite complete within 24 hours, that can be compute using Kepler's third law — satellites with larger orbit move slower than satellites with smaller orbit — the squares of orbital periods of the satellite is proportional to the cubes of semi-major axes of their orbit.

$$n = 2\pi \sqrt{\frac{\mu}{R^3}} \quad (16)$$

The first derivative of motion shows how fast the value changes. If the first derivative of motion equals a negative value, the motion currently decreases with time.

$$\dot{n} = -\frac{2\pi}{p^2} \quad (17)$$

The Second derivative of motion shows how fast the first derivative of motion changes.

$$\ddot{n} = -\frac{2\pi}{p^3} \quad (18)$$

1.7.4 B-STAR Drag (B^*)

The B^* drag is the drag term used in modelling the aerodynamic drag of a satellite in the SGP4 model and it is related to the ballistic coefficient as the equation shows:

$$B^* = \frac{C_d \rho_0 A}{2m} \quad (19)$$

Where, C_d is the ballistic coefficient or the drag coefficient, a term used to describe/measure the ability of a body to resist air mid-flight. “ C_d depends on the temperature and composition of the surrounding atmosphere, surface properties of the satellite including its temperature, surface geometry, and orientation” (Aghav & Gangal, 2014), ρ_0 is the atmospheric density, A is the cross-sectional area of satellite and m is the mass of the satellite.

1.7.5 Mean Anomaly

The true anomaly measures the actual angle, the Mean anomaly (M) measure the angle between the periapsis point and the imagined position of an object for the same elapsed. The anomaly can be computed using Kepler's second law, satellite in elliptical orbits move slowly in their orbit when they are far to the earth than when they are close to the earth, in the given equation (Kidder, 2003):

$$M = E - e \sin (E) \quad (20)$$

Where, E is eccentricity anomaly.

Chapter 2: Methods

2.1 Overview

This literature review is composed of the review of the legacy and state-of-the-art techniques and algorithms for orbital parameter estimation from satellite GPS data. We focus our review on the three main algorithms, the unscented Kalman Filter and extended Kalman and genetic algorithm. The unscented Kalman filter accurately determines the posterior mean and covariance by using the second order Taylor series expansion through the sampled sigma points that are propagated by utilizing the true non-linear system. In contrast, the extended Kalman filter uses the linearized state transition matrix to determine the covariance. The genetic algorithm is based on the natural selection and natural genetic to generate possible solutions.

2.2 Global Navigation Satellite System

An extensive study has been carried on out to expand our knowledge of satellite position and velocity vectors in the domain of TLE and use genetic algorithm training to predict the satellite's future TLE, and using an extended, unscented Kalman filter to predict the satellite's future position and velocity vector. The strategy for its completion would consider understanding the functionality of global navigation system. The aforementioned schemes' implementation of electronic signals from astronomical location and space-time estimations to support position-dependent or time-related applications highlights the significance of modulation information relaying structures for accuracy requirements within a dynamic environment (Enriquez-Caldera, 2013). This result is based on the necessity of ensuring accuracy in measurements that relate to the dispersion of electromagnetic waves to reduce the likelihood of mistakes of significant magnitude. Satellite dynamic signals also identify the type of data deserving of reflecting a comparable level of complexity. The versatility of global navigation satellite systems from the standpoint of fostering performance and comprehensive applications creates the conditions for dynamically requesting their deployment for optimization (Ge et al., 2022). With this insight, it becomes clear how GPS data (position and velocity vector) are typically used in the field of orbital parameters.

Studies have shown that the real-time accuracy of the global positioning satellite system and its practice are both precise. In this literature review, Ge et al. (2022) study was used to explore the preciseness factor from the development and prospects point of view caused by the improvement of the global navigation satellite systems. The conclusions of this paper are based on a thorough examination of the state of development in this area of interest, with a focus on various aspects, including the advantages and difficulties of precise orbit, the convergence of precise point positioning time, and the approximation of a rotation parameter in a global ionosphere model. These developments are related to possibilities for quicker time convergence following improved satellite navigation signals, high speed paired with geometric configurations that allow for huge volume changes quickly, and accuracy due to constellation optimization. Since duplicating server- and user-end success is essential for real-time precise positioning services, exceptional setup, operational, and positioning principles are sought (Li et al., 2019). These features are linked to the conclusion that LEO-enhanced global navigation satellite systems enhance the real-time application of accurate point positioning duration using systematic research and numerical analyses (observation simulation and observation models). Overall, the studies have notable facts that guide the exploration of the association between global navigation satellite systems and orbit determination apparent in pursuing the most excellent satellite orbits.

Global navigation satellite systems are applied in orbit determination to achieve possible dynamic orbit solutions and precision requirements. For the characteristic of the dynamic orbit solutions, the Astronomical Institute of the University of Bern (AIUB) compares operations of precise orbit determination affiliated to the performance of GPS and the outcome of different low earth orbit satellites over the last two decades (the 2000s and 2010s). The finding linked to focusing on the empirical components to eliminate modelling deficiencies of satellite formations mentions orbit solutions as the ranging approximation of 20% improvement as a consequence reflected as a potential benefit of instituting a combination of best-suited satellite orbits (Dai et al., 2019). It is noted from this interpretation that precise orbit determination is constructed by employing a strategy in which the advanced model's adopted features enhance the probability of reaching an ideal geometric positioning. However, this effect is not

reached straightforwardly owing to the attribute of the low earth orbit varying even within a short orbit arc. The solution that manifests is a reduced-dynamic orbit as leverage between kinematic and dynamic solutions. Including additional empirical parameters (pseudo-stochastic parameters) reduces model complexity for an assurance of stable satellite trajectory. The mathematical model describing the evident motion in which reduced-dynamic orbit is achieved is:

$$\ddot{\mathbf{r}} = -GM \frac{\mathbf{r}}{r^3} + \mathbf{f}(t, \mathbf{r}, \dot{\mathbf{r}}, Q_1, \dots, Q_d, P_1, \dots, P_s) \quad (21)$$

The equation is a reference frame representing reduced-dynamic orbit motion within a phase that has already attained initial conditions of gravitational constant times mass, shown in Q and empirical parameters (Q_1, \dots, Q_d) to leverage deficiencies in the force model. Pseudo-stochastic in the form of P_1, \dots, P_s are parameters to leverage force model deficiencies. This equation could be interpreted to indicate the periodic and/or once-per-arc constant accelerations in directions of radial, along-track, and cross-track provide high-quality orbit solutions because of the demonstrated capability of compensating satellite dynamics uncertainties causing constrained pseudo stochastic parameters (Mao et al., 2020). Addressing offsetting problems related to elements that hinder a dynamic orbit solution is fundamental to eliminating deficiencies that limit the efficient working of satellite modelling.

Precision requirement as the portrayal of the global navigation satellite systems practice in orbit determination is the intention to predict maneuvers and restrict accompanying precise point positioning. Clock-Constrained Reverse Precise Point Positioning (RPPO), an approach toward this course, involves detecting the maneuvering time of GPS satellites depending on different events (Dai et al., 2019). The justification for maneuvering is its effectiveness in reducing the impact of the global navigation satellite system and promising positioning performance. This claim is a definite reflection of results from an exploration that encompasses constructing a predicting model to analyze clock prediction-related errors during maneuvers (Dai et al., 2019). A proposed method with "three steps: the dynamic POD, clock modelling and prediction, and clock-constrained RPPP POD" provides a simplified and well-explained reverse precise point positioning strategy that equated to precise orbit determination (Dai et al.,

2019). This citation is essential in operating global navigation satellite systems for orbit determination.

The advantage of global navigation satellite systems working within the orbit determination context is replicated in the delivered opportunity to constrain inherent offsets in the prediction model. In turn, the effect is improved orbit solutions regarding maneuvers. This claim is an obvious conclusion made in a study that proposes the method that eliminates satellite biases with the outcome revealing accuracy and convergence enhancing ambiguity resolution. Relating to the satellite stations dependent on precise positioning and correcting clock errors, global navigation satellite systems are excellent at collecting the necessary data (carrier-phase and code observations) (Zhao et al., 2021). The notion of estimation is introduced here in the background of the argument that considers satellite orbit goal and data tracking as inseparable components. In a paper that suggests settings facilitating promoting attained desired orbit determination to offset constraints, accuracy of GPS cycle operation characterized by on/off intervals is considered as a compelling state, justifying the creation of an advanced structure (automatic fault management) to curtail deficit attributed to detection limitation, which assures inbuilt accommodation features are operative in a well-defined manner (Bolandi et al., 2013). Awareness of the satellite features seems evident in these findings. Notwithstanding current knowledge regarding the satellite position and velocity vector, orbit determination persists issues.

2.3 Orbit Determination Problem

Findings from current research provide theoretical consideration for reasons that attract attention to the orbit determination problem. A quantitative analysis study enhances awareness about solving the orbit determination problem to ascertain that observation time influences Two-Line Elements depending on accuracy (Ardaens & Gaias, 2019).

This exploration comprises the execution of the least-squares test corresponding analytically to varying inter-satellite distances of collinear solutions classification apparent from the perspective of linear theory. With the computed numerical analysis of the relative motion parameterization, the interpretation in this experiment is that the

insights about the relative orbital elements (mean relative longitude) prove eligibility concerning the suitable onboard execution method, which reflects accuracy in the view of the observation and visibility time range—the composition of the formation and the manifestation of the implementation of maneuvers. Previous studies confirm that the GPS-related estimation issue requires computations to solve complex errors (Chiaradia et al., 2000). The guiding factor for such analyses is to enhance predicting accuracy. Similarly, a research paper concluded that the accuracy of electric thrust calibration, which includes the accuracy of self-positioning data at the 10 m level, has been upgraded and that the transfer orbit, which entails a prediction accuracy of 14 days, is excellent compared to 1 km. This declaration is related to the analysis of simulation data to examine the various factors that influence the geostationary orbit satellites (Lu et al., 2022). However, how accurate GPS accuracy is still to be seen. Precisely estimating longitude has remained an unsolved calculation problem. Chiaradia et al. (2000) made this claim about continuous human participation in accurate time measurements, for example, astronomical observations, to approximate the position of moon positioning transit and positioning.

This undertaking remained ineffective due to related complex errors, which necessitated a different approach to determining the longitude coordinate system. In another investigation confirming the orbit determination pattern, the explanation is that the context in which the variables are estimated relative to GPS is the specification of satellite trajectory to process the dataset on pseudo-range measurement (Pardal et al., 2013). This description identifies real-time and realization of systems to assist computational implementation complexity attributable to aspects of accuracy, not an error in evaluating relevant conditions associated with velocities and acceleration. A solution to inadequacy in measurement has led to identifying relevant techniques intended to improve orbit determination. The rationale for such consideration is based on the requirement that precise satellite orbit is characterized by high precision and that this condition must replicate a navigation system operational to estimate the position, velocity, and time of the GPS receiver (Eliasson, 2014). Corresponding to this clarification, algorithm systems formulated to yield results that mirror orbit determination enriched in accuracy are extended Kalman filter and unscented Kalman

systems.

2.4 Extended Kalman Filter Algorithm

The extended Kalman filter emerges as a non-linear version with peculiar aspects of instant measurement updating. In contrast, the distinctive characteristic that dominates the utilization of the Kalman filter is linearity equated to the dynamical system and feedback model to confirm its perfection in its real-time use. In their attempt to accurately separate the dynamics relative to the difficulty in modelling artificial satellites, Pardal et al. (2013) make this claim, signifying generalized efficiency in utilizing extended Kalman filters in settings denoting such context. The success of this algorithm in steering the updated reference trajectory while accounting for current estimates is what gives it its core practicality. Based on this account, Chiaradia et al. (2000) affirm that the specific update orientation related to the extended Kalman filter is time-update and measurement-update cycles. These approximation equations are examined in detail to ascertain their magnitude in predicting the accuracy of errors.

To start with updating the time, the classification refers to prediction equations formulated to estimate the current nature of a problem and error covariance. The intent is to provide *priori* approximations for the expected next phase (Nath et al., 2020). Mathematical models to prove this clarification are provided (see Figure 9). Compared to these estimates, the measurement update equations symbolizing incorporated feedback after initial calculations denote improved *posteriori* evaluation. Corresponding equations comprising predication state, update the state, and a block representation of Kalman filter is presented (see Figure 6). The interpretation regarding the prior estimate is that the algorithm involved here is setting of predicting context while the update calculations are presumed corrector evaluations. Mechanism signified is that algorithm linking to this nature mirrors predictor-corrector procedure aimed at solving numerical measurement data.

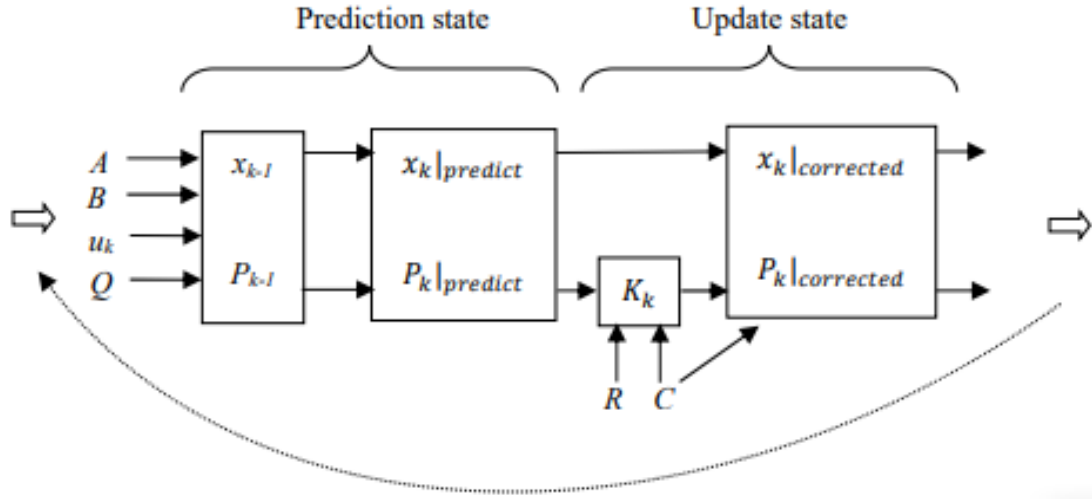


Figure 7: Extended Kalman Filter Flowchart

Prediction State:

$$x_k|_{predict} = Ax_{k-1} + Bu_k \quad (22)$$

$$P_k|_{predict} = AP_{k-1}A' + Q \quad (23)$$

Update State:

$$K_k = \frac{P_k|_{predict}C'}{CP_k|_{predict}C' + R} \quad (24)$$

$$x_k|_{corrected} = x_k|_{predict} + K_k(y_k - Cx_k|_{predict}) \quad (25)$$

$$P_k|_{corrected} = (I - K_kC)P_k|_{predict} \quad (26)$$

Equations (22 – 26) for the prediction and update states for the Kalman filter as embodied in these simplified mathematical models indicate measurement error realization from the initial state to the next phases. The outcome is dependent on the consideration of adequate uncertainty. The assumed expectation of this injection is elevated reliability of such measurement process from *priori* to *posteriori* calculation (Nath et al., 2020). The explanation of the extended Kalman filter based on the mathematical modelling and diagrammatic illustration sets the background to evaluate the use of this algorithm is orbit determination as presented in literature findings.

Underlining extended Kalman filter algorithm usage is the inbuilt nature of the non-linear dynamical and non-linear measurement systems of orbit determination problem. These elements confirm the conformity of the extended Kalman filter process as an estimating model with a real-time feature. According to Chiaradia et al. (2000), this procedure consists of obtaining parameters' values to measure orbiting satellites through observation criteria. A paper that models the observability criteria equated to non-linear condition for relative orbital shows that differential geometric method is a sufficient condition to elaborate upon dynamics of orbital determination given that assumed observability measure establishes magnitude to which it is difficult or straightforward when it comes to estimating quantitative measurements pertaining to orbit determination (Kaufman et al., 2016). These accounts are connected to the extended Kalman filter. Its practice entailed numerically proving non-linear orbits observability, especially concerning angles-only measurement, and filtered solution accuracy. Affirming significance allied to observability analysis, evidence pinpoints optimal orbit determination as a solution that emphasizes measurements and dynamics considering orbit configurations (unobservable). Thus, applying additional improvement methods, especially for cooperative optical orbit determination, has been cited as essential (Luo et al., 2022). While this inference draws from context relative to analysis of orbit elements to substantiate observability criteria, accuracy improvement owing to the extended Kalman filter procedure could be examined further to ascertain its completeness in orbit determination.

2.5 Unscented Kalman Filter Algorithm

The outstanding prediction performance of this approach is credited with the GPS and orbit determination inclination toward the use of the unscented category Kalman filter format. Real-time satellite demands the dynamic model effects and accounting complexity that an estimating technique may provide. The unscented Kalman filter maintains a competitive position in orbit determination for non-linear observations by emphasizing convergence speed, divergence occurrence, and statistical features (weaknesses and strengths) criteria (Pardal et al., 2013). It is a revelation that supports how non-linear contexts that call for updating frequently rely on approximation systems that keep the orbit station system's magnitude of mistakes to a minimum. That way, the

mechanisms of the unscented Kalman filter algorithm are presented in the steps relating to adequate prediction of the status of a system comparative to its associated covariance focusing on noise process effects, estimation of projected observations and related residual innovation matrix accounting effects of observation noise, and approximation of cross correlation matrix (Pardal et al., 2013). Fundamental indication from these phases concerning this process's mechanisms, the algorithm emphasizes the dynamics and structure of a system for generating a new state. Other characteristics of the unscented Kalman filter technique explain whether the update is an orientation that replicates an enhanced performance in the realm of the navigation system.

The debate about adopting an unscented Kalman filter algorithm for the global positioning system is relative to reliability and accuracy parameters. The basic consideration for this system is that its implementation in navigation satellite systems underlines superior performance to unintegrated structures described as insecurely coupled or firmly coupled (Sun et al., 2020). The most peculiar aspect that influences the practicality of this procedure is its integration effect as a technique that transforms estimation within a non-linear system. This alteration is apparent in the joint filtering approach connected to a state vector state characterized by states and parameters and a simultaneous approach comprising state and parameter approximation within which parameter estimator is stated filter input (Onat, 2019). Fundamental to these operations equated to unscented Kalman filter, its performance for the dimension of non-linear measurements could be established as reliable not only as a progression of other categories of Kalm filter, such as extended Kalman filter, but also its elevated feature of exceptional performance. According to Onat (2019), an exact guarantee of unscented or its different adaptations, such as the joint unscented Kalman filtering system, decreases computational complexity management since it allows for convergence parameter estimation and detaches parameters from the state vector. Overall, the usefulness of this algorithm for non-linear dynamic systems is affirmed. Nevertheless, what remains to be verified regarding its effectiveness connects to the realm of orbit propagation and determination.

2.6 Previous Studies on the Orbit Determination Problem

Many research materials investigate real-time orbit determination using the GPS navigation solution for onboard processing because of its capability to yield autonomous satellite operation and navigation.

Chiaradia et al. (2000) utilize the extended Kalman filter estimation model to determine the onboard orbit determination algorithm by applying the 30 second step-size propagation and an analytical approach to compute the state transition matrix. Their research findings indicate that an accuracy of about 20 m RMS in position and 0.1 m/s RMS in velocity were obtained for the Topex/Poseidon satellite using the extended Kalman filter estimation method. Gill et al. (2004) conducted similar studies to Chiaradia et al. (2000) that involved a GPS-based navigation system using the extended Kalman Filter for small satellites. The findings indicated that a position accuracy of 25-30 m can be obtained for the X-SAT satellite. Yoon et al. (2000) also conducted similar studies to Chiaradia et al. (2000) They indicated that combining the extended Kalman filter and GPS navigation solutions yielded a position accuracy of 15.8 m and velocity accuracy of 0.0141 m/s RMS. Yoon et al. (2000) also indicated that the observations provided real-time orbit ephemeris without abnormal excursions. However, the findings from Chiaradia et al. (2000) indicate that, for real-time processing, the inclusion of high order terms of geo-potential may result in unnecessary computational burden without any significant increase in the position and velocity accuracies.

Although most previous studies have utilized the extended Kalman filter as the baseline estimation algorithm, modelling errors occur because of the methodology. According to Choi et al. (2010), the extended Kalman Filter utilizes a state distribution approximated by a Gaussian random variable, which may introduce significant errors in the posterior mean and covariance. Besides, the actual GPS navigation system has model uncertainties that cannot be expressed by the linear state-space model utilized by the extended Kalman filter, which increases the modelling errors. Julier & Uhlmann (1997) utilize the unscented Kalman filter, which uses a non-linear distribution method and the finite number of sigma points to propagate the probability of a state distribution through the non-linear dynamics of the system to determine the real-time positional and velocity

accuracies of the spacecraft orbit. The findings of the research from Julier & Uhlmann (1997) indicate that similar results are obtained when utilizing the unscented Kalman filter for short measurement sampling periods as the extended Kalman filter. However, the unscented Kalman filter yielded more stable and robust convergence for long measurement periods when compared to the extended Kalman filter. Choi et al. (2010) also utilized the unscented Kalman filter for an onboard orbit determination algorithm and obtained similar results to those acquired by Julier & Uhlmann (1997). Thus, the findings from previous studies on the extended Kalman filter and unscented Kalman filter prompt the need to investigate the estimation method that will yield better position and velocity accuracies in the orbit determination problem.

2.7 Genetic Algorithm

A Genetic algorithm (GA) is considered a type of optimization algorithm. It is a powerful heuristic search algorithm created by J. Holland in the early 1970s, and he based his algorithm on the mechanics of natural selection and natural Genetic (Benhachmi et al., 2001; Brown & Sumichrast, 2005). The genetic algorithm provides a new and improved solution that improves over time. It uses a population of solutions instead of one and combines parts of reasonable solutions to achieve better solutions (Brown & Sumichrast, 2005). The GA can be utilized in transportation, DNA analysis, and aircraft design applications.

Hinagawa et al. (2014) describe solving the problem using a genetic algorithm based on using two different ways, the Lambert's solver and the Battin's methods for the GEO satellite. Usual optical observations, such as the methods of Laplace, Gauss, or Escobal, do not give the desired results for determining the orbit and its propagation. The velocity vector was obtained from Lambert's solvers, while the position vector was from the two line-of-sight vectors. The fitness function was obtained by finding the inverse of the sum of errors for the right ascension (α) and the declination (δ) as represented in Equation 27 (Hinagawa et al., 2014).

$$fitness\ function = \sum_{i=2}^{N-1} \frac{1}{\sqrt{(\alpha_{mi} - \alpha_{ei})^2 + (\delta_{mi} - \delta_{ei})^2}} \quad (27)$$

Where, α_{mi} and δ_{mi} are the mean measured right ascension and declination, α_{ei} and δ_{ei}

are the estimated right ascension and declination.

In an article by Ansalone & Curti (2013), scientists used the genetic algorithm for a short-term observation to determine orbital parameters for LEO satellites to solve the Lambert's problem. "The Lambert's boundary value problem consists in finding the orbit of a generic object between two positions at two different times" (Ansalone & Curti, 2013). The initial population was fixed with a 1000 chromosome. The fitness function is obtained by computing the product of the dot product of an actual and estimated measurement at different a time, as shown in Equation 28.

$$fitness\ function = \prod_{i=2}^{n-1} (\hat{L}_m(t_i) \cdot \hat{L}_e(t_i)) \quad (28)$$

Where, $\hat{L}_m(t_i)$ is the actual measurement of angular noiseless and $\hat{L}_e(t_i)$ is the estimated measurement of angular noise.

Unlike the published articles in this paper, the genetic algorithm is much simpler. The fitness function is obtained using the GPS data – position and velocity vector – without the need to utilize external methods and solvers in the algorithm. Also, the GA generate the TLE, not only the orbital parameters. This paper compares the GA with the Kalman filter algorithms (EKF, UKF) to find the best algorithm to compute the TLE.

Chapter 3: Results and Discussions

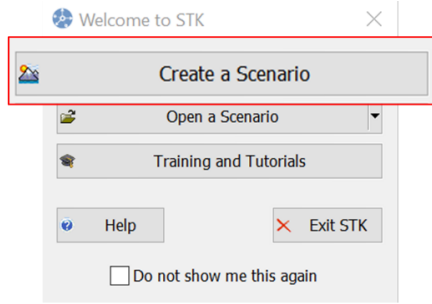
3.1 Overview

The thesis is divided into two parts. The first part is dedicated in estimating the position and velocity vector over a specific time and generating TLE's during the early days of mission using two orbital determination algorithms, namely, the Extended Kalman Filter (EKF) and the Unscented Kalman Filter (UKF) with the use of Kepler orbital parameter formulas (Equation 1 – 10) to determine the TLE parameters instead of using third parties. This part of the project mostly revolves around finding what is the best of the previous work in this topic. The second part is dedicated to utilizing one of the artificial intelligence algorithms called the genetic algorithm, to find the best TLE. To the best of our knowledge, this is the first work to use GA for TLE computation using GPS data.

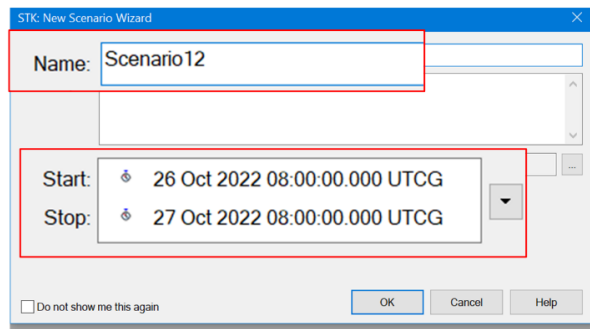
The next subsection (3.2) describes the data, and the source of data that are utilized in three algorithms. The following subsection (3.3) provides the dynamic satellite model used in the orbit determination algorithms. The next three subsections explain the Extended and Unscented Kalman Filter algorithms, and the genetic algorithm, respectively. Appendices C and D show the implementation of mathematical equation for the EKF and UKF to Python code. The genetic algorithm implementation is reported in Appendix E.

3.2 Data Source

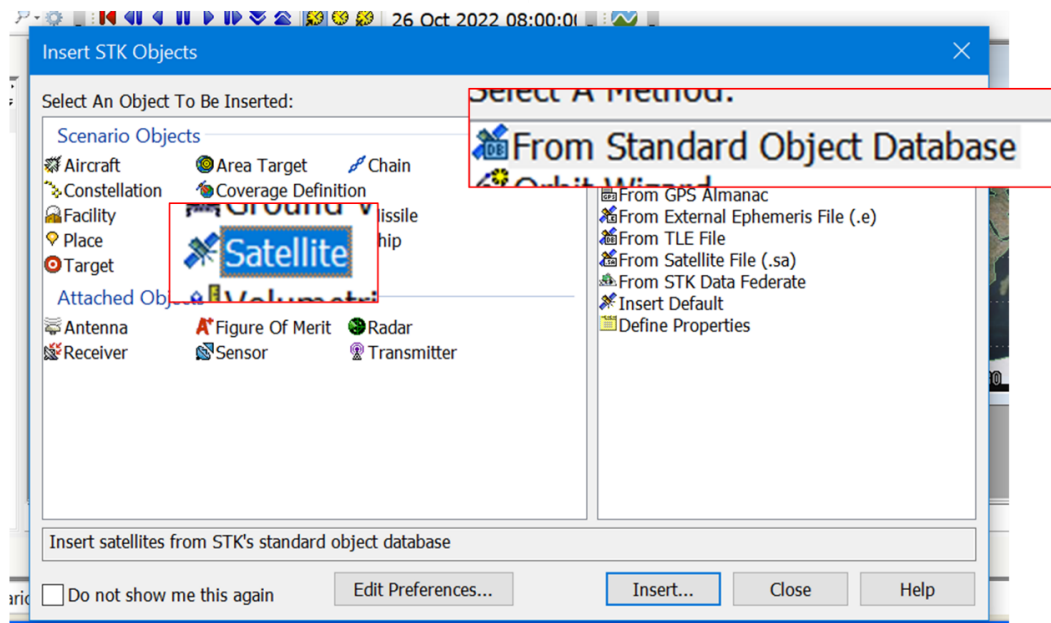
The data used in this paper was obtained only from Systems Tool Kit (STK). This software provides an accurate physics-based modelling environment that helps engineers to perform analysis and simulation for realistic mission platforms and payloads. Two different satellites were used to evaluate the performance of the algorithm, small and medium satellites. Both satellites' measurements were taken at 10-second intervals over 24-hour simulations in the ECI coordinate frame. Figure 8 shows the steps of using and performing the simulation using STK.



Step 1: Create a Scenario



Step 2: Name the Scenario and Select the Start and Stop Time of the Scenario



Step 3: Add an object (Satellite) From the Database, by Searching the Name or the NORAD Catalog Number to Find a Specific Satellite


Step 4: Click the Report & Graph Manager () icon or extend the Analysis menu and select Report & Graph Manager.

Figure 8: STK Steps

Table 8 provides the initial position and velocity vector from STK utilized in the orbit determination algorithm for MEZNSAT, a 3U CubeSat with a mass of 3 kg launched at an altitude of 500 km. While Table 9 provides orbital parameters from STK from the same satellite.

Table 8: Initial Position and Velocity Vector [MEZNSAT]

Nr	Epoch Time [s]	Initial position components [km]			Initial velocity components [km/s]		
		x	y	z	x	y	z
1	1601251200	-1461.24	-1674.24	-6567.87	-6.6604	-2.8625	2.2016
2	1601251210	-1527.76	-1702.76	-6545.46	-6.6427	-2.8424	2.2800
3	1601251220	-1594.09	-1731.09	-6522.27	-6.6241	-2.8219	2.3578
4	1601251230	-1660.24	-1759.2	-6498.31	-6.6047	-2.8012	2.4355

Table 9: Initial Orbital Parameters [MEZNSAT]

Nr	Epoch Time [s]	Inclination [degree]	RAAN [degree]	Argument of Perigee [degree]	Mean Anomaly [degree]	Eccentricity	Semi- major Axis [km]
1	1601251200	97.623	205.586	147.044	139.934	0.001927	6923.411
2	1601251210	97.623	205.586	147.94	139.663	0.00193	6923.527
3	1601251220	97.623	205.587	148.838	139.389	0.001933	6923.647
4	1601251230	97.623	205.587	149.738	139.115	0.001936	6923.771

Table 10 provides the initial position and velocity vector from STK utilized in the orbit determination algorithm for a medium satellite with a mass of 615 kg, launched at an altitude of 555 km, known as RISAT-2BR2 or EOS-01, with a NORAD number of 46905. Table 11 provides orbital parameters from STK.

Table 10: Initial Position and Velocity Vector [EOS-01]

Nr	Epoch Time [s]	Initial position components [km]			Initial velocity components [km/s]		
		x	y	z	x	y	z
1	1632816000	-1844	-5252.72	-4171.19	7.1359	-2.51602	0.017213
2	1632816010	-1772.53	-5277.57	-4170.77	7.1573	-2.45374	0.066685
3	1632816020	-1700.85	-5301.79	-4169.86	7.1779	-2.39117	0.11615
4	1632816030	-1628.98	-5325.39	-4168.45	7.1975	-2.32832	0.165601

Table 11: Initial Orbital Parameters [EOS-01]

Nr	Epoch Time [s]	Inclination [degree]	RAAN [degree]	Argument of Perigee [degree]	Mean Anomaly [degree]	Eccentricity	Semi- major Axis [km]
1	1632816000	36.843	340.404	72.178	198.023	0.000889	6950.433
2	1632816010	36.843	340.403	72.247	198.578	0.000889	6950.435
3	1632816020	36.843	340.401	72.311	199.139	0.000888	6950.439
4	1632816030	36.844	340.400	72.369	199.706	0.000887	6950.444

3.3 Satellite Dynamics Model

Kalman Filter algorithms are designed and created for each system differently and specifically by creating a dynamic model for the system. Therefore, the dynamic model is essential to accomplish the orbital determination processes using the GPS data with an Extended Kalman Filter or Unscented Kalman Filter. The model was determined by utilizing the force method model — a dynamical method of analysis based on Newton’s law — on satellite at a specific altitude. The satellite is subjected to different forces such as gravitational, solar radiation pressure and atmospheric drag force. In this paper, the gravitational and the atmospheric drag force are used to determine the dynamic model.

As in Equation (21), The total acceleration of the satellite is equal to Newton’s

acceleration (gravitational force) and the acceleration perturbation due to forces. The following equation gives the atmospheric drag acceleration (\mathbf{a}_{drag}).

$$\mathbf{a}_{drag} = -\frac{1}{2} \frac{C_d A}{m} \rho v_{rel}^2 \frac{\mathbf{v}_{rel}}{|\mathbf{v}_{rel}|} \quad (29)$$

Where, ρ is the atmospheric density (see Appendix B).

The acceleration is represented in the Earth-Centered Inertial (ECI) coordinate system; the same coordinate system NORAD and the simplified perturbations models uses to generate the TLEs. Satellites are objects that revolve around the center of the earth. For that reason, the ECI coordinate system was chosen because it is a coordinate system with an origin in the center of the earth. Also, It is fixed in space relative to the stars (T.S, 2020). The acceleration can be determined using Equation 30.

$$\dot{\mathbf{r}} = -GM \frac{\mathbf{r}}{r^3} + \mathbf{a}_{drag} \quad (30)$$

During the simulation, the zonal perturbation J_2 is considered. The acceleration is represented in the following equations (Aghav & Gangal, 2014):

$$\begin{aligned} \ddot{x} &= -GM \frac{x}{r^3} \left[1 + J_2 \frac{3}{2} \left(\frac{R_e}{r} \right)^2 \left(1 - 5 \frac{z^2}{r^2} \right) \right] + a_{drag_x} \\ \ddot{y} &= -GM \frac{y}{r^3} \left[1 + J_2 \frac{3}{2} \left(\frac{R_e}{r} \right)^2 \left(1 - 5 \frac{z^2}{r^2} \right) \right] + a_{drag_y} \\ \ddot{z} &= -GM \frac{z}{r^3} \left[1 + J_2 \frac{3}{2} \left(\frac{R_e}{r} \right)^2 \left(3 - 5 \frac{z^2}{r^2} \right) \right] + a_{drag_z} \end{aligned} \quad (31)$$

Where, R_e is the earth radius equal to 6,371 km, J_2 equal to 0.001082.

3.4 Extended Kalman Filter (EKF)

“Kalman filter addresses the general problem of trying to estimate the state of a discrete-time controlled process that is governed by a linear stochastic difference equation” (Welch & Bishop, 2006). Satellites of orbit do not exhibit linear behavior. The extended Kalman filter (EKF) is the nonlinear version of the Kalman filter that linearizes the current mean and covariance (Welch & Bishop, 2006). It does not require extra

power from the satellite to be performed. However, it requires an initial state vector $\hat{\mathbf{x}}_k$ of the satellite dynamics. The state vector $\hat{\mathbf{x}}_k$ contains the position and the velocity vector in the Earth Centered Inertial (ECI) coordinate frame of the satellite on the 3-axis, and it is given by (Aghav & Gangal, 2014; Amaral et al., 2007):

$$\hat{\mathbf{x}}_k = [x \quad y \quad z \quad v_x \quad v_y \quad v_z]^T \quad (32)$$

Also, requires a diagonal initial state error covariance matrix (P_k) given by:

$$; \quad P_{k+1} = \text{diag}[\sigma_x^2 \quad \sigma_y^2 \quad \sigma_z^2 \quad \sigma_{v_x}^2 \quad \sigma_{v_y}^2 \quad \sigma_{v_z}^2]_{6 \times 6} \quad (33)$$

The standard deviations are assumed to be 10 m for the position and 0.1 m/s for the velocity (Aghav & Gangal, 2014). The EKF consists of two main phases (see Figure 9) utilized to estimate the next position and velocity of the satellite.

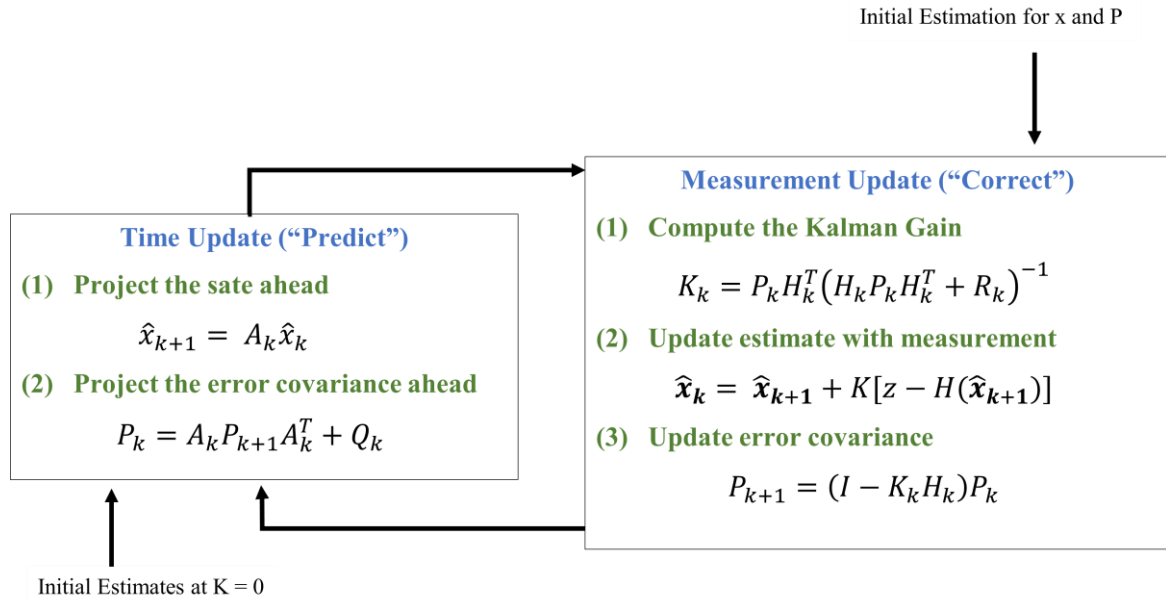


Figure 9: Extended Kalman Filter (EKF) Flowchart. *Note.* From Welch, G., & Bishop, G. (2006). *An Introduction to the Kalman Filter*. 16.

The first phase is the prediction phase which predicts the system state, that define by the given differential equation (Aghav & Gangal, 2014):

$$\hat{\mathbf{x}}_{k+1} = A_k \hat{\mathbf{x}}_k \quad (34)$$

$$P_k = A_k P_{k+1} A_k^T + Q_k \quad (35)$$

Where, A_k is the state transition matrix, is an $n \times n$ matrix, given by (Aghav & Gangal, 2014; Amaral et al., 2007; Lam et al., 2010):

$$A_k = \left[I + \begin{bmatrix} 0_{3 \times 3} & -\Delta t \times I_{3 \times 3} \\ J_{3 \times 3} & 0_{3 \times 3} \end{bmatrix} \right]^T \quad (36)$$

Where, $J_{3 \times 3}$ is the Jacobian coefficient matrix can be determined by taking the partial derivative of Equation (30) without taking on consideration the zonal perturbation J_2 and given by:

$$J_{3 \times 3} = \begin{bmatrix} 3\mu x^2 r^{-5} - \mu r^{-r} & 3\mu y x r^{-5} & 3\mu z x r^{-5} \\ 3\mu y x r^{-5} & 3\mu y^2 r^{-5} - \mu r^{-r} & 3\mu z y r^{-5} \\ 3\mu z x r^{-5} & 3\mu z y r^{-5} & 3\mu x^2 r^{-5} - \mu r^{-r} \end{bmatrix} \quad (37)$$

Where, $r = \sqrt{x^2 + y^2 + z^2}$ is the orbit radius magnitude. However, if the zonal perturbation J_2 is included in, the Jacobian coefficient matrix is equal to:

$$J_{3 \times 3} = \begin{bmatrix} \frac{\partial a_x}{\partial x} & \frac{\partial a_x}{\partial y} & \frac{\partial a_x}{\partial z} \\ \frac{\partial a_y}{\partial x} & \frac{\partial a_y}{\partial y} & \frac{\partial a_y}{\partial z} \\ \frac{\partial a_z}{\partial x} & \frac{\partial a_z}{\partial y} & \frac{\partial a_z}{\partial z} \end{bmatrix} \quad (38)$$

Where,

$$\frac{\partial a_x}{\partial x} = \frac{3\mu x^2}{r^5} - \frac{\mu}{r^3} + \frac{15\mu J_2 R_e^2 z^2}{2r^7} - \frac{3\mu J_2 R_e^2}{2r^5} + \frac{15\mu J_2 R_e^2 x^2}{2r^7} - \frac{105\mu J_2 R_e^2 x^2 z^2}{2r^9};$$

$$\frac{\partial a_y}{\partial x} = \frac{3\mu xy}{r^5} + \frac{15\mu J_2 R_e^2 xy}{2r^7} - \frac{105\mu J_2 R_e^2 xyz^2}{2r^9};$$

$$\frac{\partial a_z}{\partial x} = \frac{30\mu J_2 R_e^2 zx}{2r^7} + \frac{3\mu xz}{r^5} + \frac{15\mu J_2 R_e^2 xz}{2r^7} - \frac{105\mu J_2 R_e^2 xz^3}{2r^9};$$

$$\frac{\partial a_x}{\partial y} = \frac{3\mu xy}{r^5} + \frac{15\mu J_2 R_e^2 xy}{2r^7} - \frac{105\mu J_2 R_e^2 xyz^2}{2r^9};$$

$$\frac{\partial a_y}{\partial y} = \frac{3\mu y^2}{r^5} - \frac{\mu}{r^3} + \frac{15\mu J_2 R_e^2 z^2}{2r^7} - \frac{3\mu J_2 R_e^2}{2r^5} + \frac{15\mu J_2 R_e^2 y^2}{2r^7} - \frac{105\mu J_2 R_e^2 y^2 z^2}{2r^9};$$

$$\frac{\partial a_z}{\partial y} = \frac{30\mu J_2 R_e^2 zy}{2r^7} + \frac{3\mu yz}{r^5} + \frac{15\mu J_2 R_e^2 yz}{2r^7} - \frac{105\mu J_2 R_e^2 yz^3}{2r^9};$$

$$\frac{\partial a_x}{\partial z} = \frac{3\mu xz}{r^5} + \frac{45\mu J_2 R_e^2 xz}{2r^7} - \frac{105\mu J_2 R_e^2 xz^3}{2r^9};$$

$$\frac{\partial a_y}{\partial z} = \frac{3\mu yz}{r^5} + \frac{45\mu J_2 R_e^2 yz}{2r^7} - \frac{105\mu J_2 R_e^2 yz^3}{2r^9};$$

$$\frac{\partial a_z}{\partial z} = \frac{30\mu J_2 R_e^2 z}{2r^7} - \frac{9\mu J_2 R_e^2}{2r^7} - \frac{\mu}{r^3} + \frac{3\mu z^2}{r^5} + \frac{45\mu J_2 R_e^2 z^2}{2r^7} - \frac{105\mu J_2 R_e^2 z^4}{2r^9}$$

The second phase, the system compares the prediction value with the measured value from the sensors (GPS data). This second phase start with defining the Kalman gain (K).

$$K_k = P_k H_k^T (H_k P_k H_k^T + R_k)^{-1} \quad (39)$$

Where, R_k is the measurement error covariance matrix, Q_k is the process error covariance matrix. Both parameters have an essential role in improving the estimation effect, due to their effects on the Kalman gain. Q_k is 6x6 covariance matrix given by Equation 40.

$$Q_k = \begin{bmatrix} \text{var}(x) & \text{cov}(x,y) & \text{cov}(x,z) & \text{cov}(x,v_x) & \text{cov}(x,v_y) & \text{cov}(x,v_z) \\ \text{cov}(y,x) & \text{var}(y) & \text{cov}(y,z) & \text{cov}(y,v_x) & \text{cov}(y,v_y) & \text{cov}(y,v_z) \\ \text{cov}(z,x) & \text{cov}(z,y) & \text{var}(z) & \text{cov}(z,v_x) & \text{cov}(z,v_y) & \text{cov}(z,v_z) \\ \text{cov}(v_x,x) & \text{cov}(v_x,y) & \text{cov}(v_x,z) & \text{var}(v_x) & \text{cov}(v_x,v_y) & \text{cov}(v_x,v_z) \\ \text{cov}(v_y,x) & \text{cov}(v_y,y) & \text{cov}(v_y,z) & \text{cov}(v_y,v_x) & \text{var}(v_y) & \text{cov}(v_y,v_z) \\ \text{cov}(v_z,x) & \text{cov}(v_z,y) & \text{cov}(v_z,z) & \text{cov}(v_z,v_x) & \text{cov}(v_z,v_y) & \text{var}(v_z) \end{bmatrix} \quad (40)$$

R_k is a diagonal matrix, and it is constant given by Equation 40, the initial P_{k+1} is equal to (Aghav & Gangal, 2014; Amaral et al., 2007).

$$R_k = \text{diag}[\sigma_x^2 \quad \sigma_y^2 \quad \sigma_z^2 \quad \sigma_{v_x}^2 \quad \sigma_{v_y}^2 \quad \sigma_{v_z}^2]_{6 \times 6} \quad (41)$$

Where σ_p is the standard deviation of position equal to 10 m, and σ_v is the standard

deviation of velocity equal to 0.1 m/s. Equations 40 – 41 were chosen after testing multiple, Q_k and, R_k to find best matrix that provides the best estimation results. The Kalman Gain used to determine the updated $\hat{\mathbf{x}}_k$ and P_k with use of Equation (42 – 43).

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_{k+1} + K[z - H(\hat{\mathbf{x}}_{k+1})] \quad (42)$$

$$P_{k+1} = (I - K_k H_k)P_k \quad (43)$$

H_k is 6x6 observation matrix given by the following equation (Lam et al., 2010):

$$H = \frac{\partial h}{\partial r} = \begin{bmatrix} \frac{\partial h_1}{\partial x} & \frac{\partial h_1}{\partial y} & \frac{\partial h_1}{\partial z} & 0 & 0 & 0 \\ \frac{\partial h_2}{\partial x} & \frac{\partial h_2}{\partial y} & 0 & 0 & 0 & 0 \\ \frac{\partial h_3}{\partial x} & \frac{\partial h_3}{\partial y} & \frac{\partial h_3}{\partial z} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (44)$$

Where,

$$\frac{\partial h_1}{\partial x} = \frac{x}{r}; \quad \frac{\partial h_1}{\partial y} = \frac{y}{r}; \quad \frac{\partial h_1}{\partial z} = \frac{z}{r}; \quad \frac{\partial h_2}{\partial x} = \frac{y}{x^2 + y^2}; \quad \frac{\partial h_2}{\partial y} = \frac{x}{x^2 + y^2};$$

$$\frac{\partial h_3}{\partial x} = \frac{2xz}{(x^2 + y^2)r}; \quad \frac{\partial h_3}{\partial y} = \frac{2yz}{(x^2 + y^2)r};$$

$$\frac{\partial h_3}{\partial z} = \frac{(x^2 + y^2 + z^2) - 2z^2}{(x^2 + y^2)(x^2 + y^2 + z^2)^2}$$

3.5 Unscented Kalman Filter (UKF)

The Unscented Kalman Filter is an improvement of the EKF. “Instead of linearizing using Jacobian matrices, the UKF using a deterministic sampling approach to capture the mean and covariance estimates with a minimal set of sample points” (LaViola, 2003). As the EKF the UKF requires and initial state vector (\mathbf{x}) and initial covariance matrix (P_o).

$$P_o = (x_0 - \bar{x}_0)(x_0 - \bar{x}_0)^T \quad (45)$$

Where, \bar{x}_0 is the mean of the initial states.

Figure 9 shows the main two phases of the UKF, they are the prediction and the correction phase.

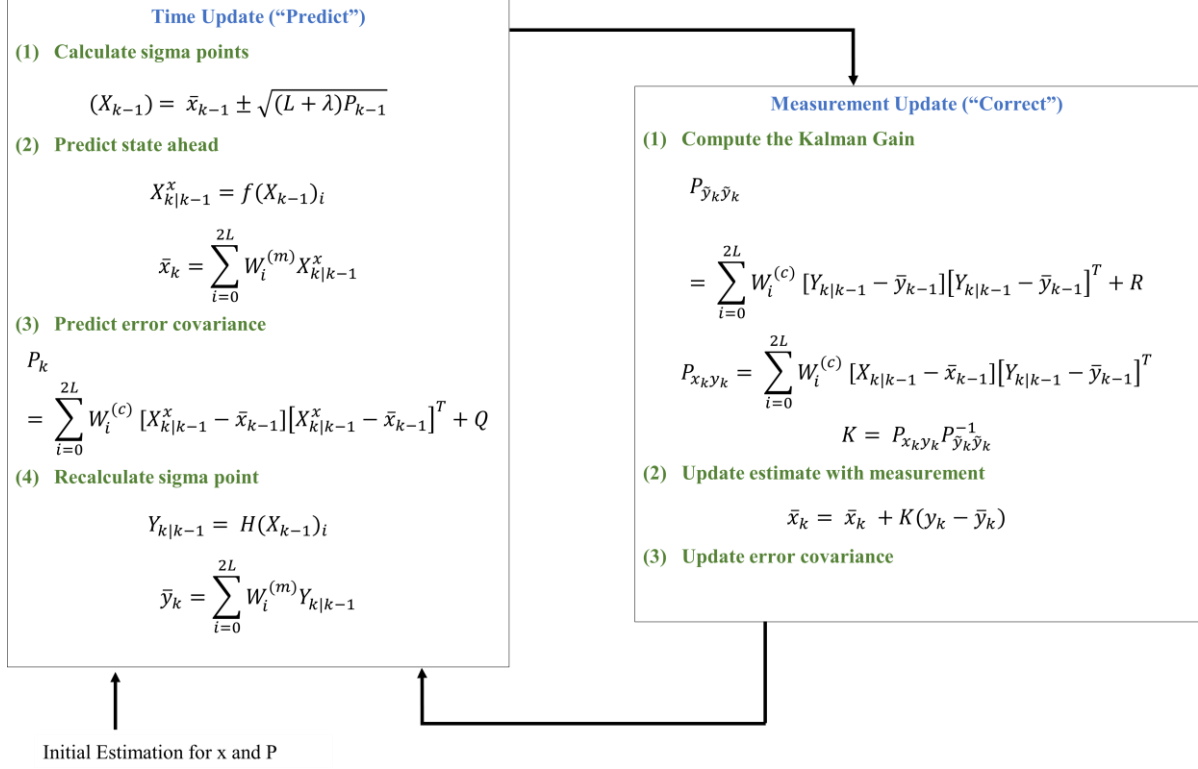


Figure 10: Unscented Kalman Filter (UKF) Flowchart

The prediction phase begins with computing the sigma point matrix (X_{k-1}) that can be computed by:

$$(X_{k-1})_0 = \bar{x}_{k-1}$$

$$(X_{k-1})_i = \bar{x}_{k-1} + \sqrt{(L + \lambda)P_{k-1}}$$

$$(X_{k-1})_{2i} = \bar{x}_{k-1} - \sqrt{(L + \lambda)P_{k-1}}$$
(46)

Where, L is sigma counter, λ is a scale parameter equal to:

$$\lambda = \alpha^2(L + \kappa) - L$$
(47)

Where, α is scaling parameter set to determine the spread of the sigma points ($1 \times 10^{-4} \leq \alpha \leq 1$), and it is set to small positive value. κ is secondary scaling parameter that provide an extra freedom ($\kappa = 3 - L$) (Wan & Van Der Merwe, 2000). Once the sigma matrix is computed the prediction process start with predicting the *priori* state ahead using the following equations:

$$X_{k|k-1}^x = f(X_{k-1})_i$$

$$\bar{x}_k = \sum_{i=0}^{2L} W_i^{(m)} X_{k|k-1}^x \quad (48)$$

Where, f is a differential equation defined in section 3.1. $W_i^{(m)}$ is the weight of the sigma defined by:

$$W_0^{(m)} = \frac{\lambda}{L + \lambda}$$

$$W_i^{(m)} = \frac{\lambda}{2(L + \lambda)} \quad (49)$$

Finally, the *priori* error covariance matrix in calculated using the following equation:

$$P_k = \sum_{i=0}^{2L} W_i^{(c)} [X_{k|k-1}^x - \bar{x}_{k-1}] [X_{k|k-1}^x - \bar{x}_{k-1}]^T + Q \quad (50)$$

Where, Q is defined in the EKF, and $W_i^{(c)}$ is defined by:

$$W_0^{(c)} = \frac{\lambda}{L + \lambda} + (1 - \alpha^2 + \beta)$$

$$W_i^{(c)} = \frac{\lambda}{2(L + \lambda)} \quad (51)$$

Where, β equal to 2, and it is a “parameter used to incorporate any prior knowledge about the distribution of \mathbf{x} ”(LaViola, 2003; Wan & Van Der Merwe, 2000). Before moving to the second phase, X_{k-1} must be transform though measuring function and create new sigma points defined by (LaViola, 2003):

$$Y_{k|k-1} = H(X_{k-1})_i$$

$$\bar{y}_k = \sum_{i=0}^{2L} W_i^{(m)} Y_{k|k-1} \quad (52)$$

Where H is the same function that has been defined in the EKF. With the new sigma points the correction phase starts with determining the Kalman gain by:

$$K = P_{x_k y_k} P_{\tilde{y}_k \tilde{y}_k}^{-1} \quad (53)$$

Where,

$$P_{\tilde{y}_k \tilde{y}_k} = \sum_{i=0}^{2L} W_i^{(c)} [Y_{k|k-1} - \bar{y}_{k-1}] [Y_{k|k-1} - \bar{y}_{k-1}]^T + R \quad (54)$$

$$P_{x_k y_k} = \sum_{i=0}^{2L} W_i^{(c)} [X_{k|k-1} - \bar{x}_{k-1}] [Y_{k|k-1} - \bar{y}_{k-1}]^T$$

Finally, the *posteriori* state vector and *posteriori* estimate of the error covariance by:

$$\bar{x}_k = \bar{x}_k + K(y_k - \bar{y}_k) \quad (55)$$

$$P_k = P_k - K P_{\tilde{y}_k \tilde{y}_k} K^T \quad (56)$$

3.6 Genetic Algorithm (GA)

A genetic algorithm (GA) is a probabilistic search algorithm used to solve optimization problem and in modelling where randomness is involved (Kumar et al., 2010). Figure 11 shows the main operations of the genetic algorithm they are selection, crossover, and mutation, starting with generating the initial population, a set of chromosomes. A chromosome is one of the populations made up of genes and is considered as a solution. In this paper, an initial of variation of number of initial chromosomes is generated by the following Python code (Shown in Figure 11). Where each chromosome represents a TLE. The gene is the parameter of the TLE, and the population is a subset of the TLEs (the possible solutions).

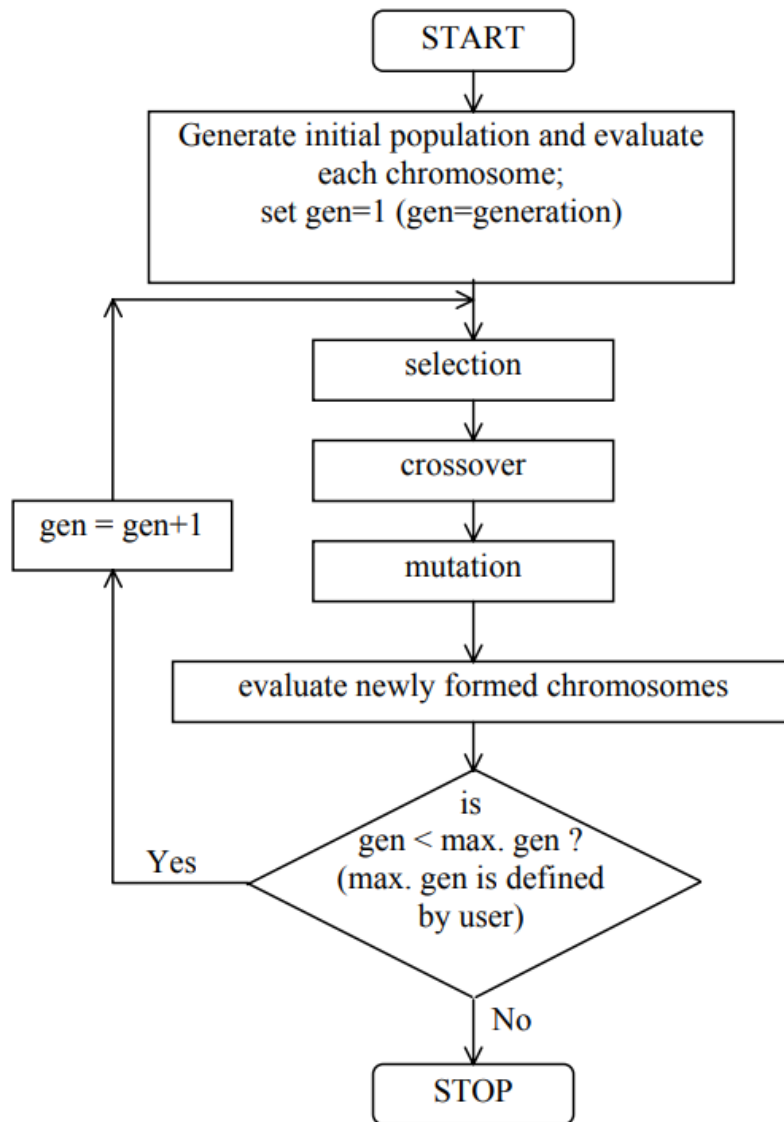


Figure 11: Genetic Algorithm Flowchart. *Note.* From Moazami Goodarzi, H., & Kazemi, M. H. (2018). An optimal autonomous microgrid cluster based on distributed generation droop parameter optimization and renewable energy sources using an improved grey wolf optimizer. *Engineering Optimization*, 50(5), 819–839. <https://doi.org/10.1080/0305215X.2017.1355970>

As illustrated in Figure 12, lines 23 – 30 show that the genes of the TLE is randomly generated using the defined function “`def get_random_number (low, high)`”, which is a simple function that only returns the random value. These values were generated in a constant and uniform range that can be defined by a low and high value using NumPy functions “`np.random.uniform (low, high)`” (see Figure 10, lines 5 – 7).

Also, with the defined function “**def** change_formatting (parameter, length)”. This function is used to have a specific number of needed digits based on Tables 6 and 5.

```

1  import numpy as np
2  population = []
3  # define the population size
4  n_pop = 1000
5  def get_random_number(low, high):
6
7      return np.random.uniform(low, high)
8  def change_formatting(parameter, length):
9      if len(str(parameter)) > length:
10         parameter = str(parameter)[:length]
11     elif len(str(parameter)) < length:
12         rem = length - len(str(parameter))
13         if "." in str(parameter):
14             parameter = str(parameter) + rem*"0"
15         else:
16             parameter = str(parameter) + "." + (rem-1)*"0"
17     return parameter
18
19 def genetic_algorithm(n_pop):
20     population = []
21     # initial population of random bitstring
22     for index in range(0, n_pop):
23         e1 = float (get_random_number(0,0.2))
24         e = change_formatting(split at decimal(e1),7)
25         i = change_formatting(float(get_random_number(90, 100)),8)
26         MA = change_formatting(float(get_random_number(0,360)),8)
27         RAAN = change_formatting(float(get_random_number(0, 360)),8)
28         AG = change_formatting(float(get_random_number(0,360)),8)
29         Mean_montion =
30 change_formatting(float(get_random_number(15,16)),11)
31
32         line_0 = 'SatelliteName'
33         line_1_str = f'1 {SatelliteNumber}U {InternationalDesignator}
34 {YY}{julian date} {FDoM} {SDoM} {B Drag} {Ephemeris type} {CheckNumber}'
35         line_2_str = f'2 {SatelliteNumber} {i} {RAAN} {e} {AG} {MA}
36 {Mean_montion} {RevNum}'
37
38         tle_output = f''{line_0}\n{line_1_str}\n{line_2_str}''
39
40         population.append(tle_output)
41

```

Figure 12: Python Code - Generating Initial Population

After generating the genes, the chromosomes are generated by creating the TLE. the TLE is split into three lines, as shown in Figure 12, lines 32 – 36. Line 37 combines the three lines to create the final TLE creating a chromosome. Figure 13 show an example of a chromosome.

```
SatelliteName
1 25544U 98067A 21271.50000000 1.4e-07 00000-0 67960-4 0 5293
2 25544 38.41893 267.3984 1623853 75.10171 170.5851 13.16578995 346978
```

Figure 13: Example of Chromosome

Then TLE saved in a list called population. The above steps were in a for-loop, as shown in line 22, Figure 12. The for-loop range between zero and “n_pop.” “n_pop” is the number of chromosomes in the initial population.

After generating the initial population, the evaluation process takes place. The evaluation process is the most important part of the genetic algorithm. The evaluation process of the individual chromosome is done by computing a fitness function or the objective function (Equation 57) that provides a fitness score for each chromosome. The lower the fitness score the better.

$$fitness\ function = \sqrt{(r_{x_A} - r_{x_G})^2 + (r_{y_A} - r_{y_G})^2 + (r_{z_A} - r_{z_G})^2} \quad (57)$$

Where, r_{x_A} is the average actual position on x-axis, r_{y_A} is the average actual position on y-axis and r_{z_A} are the average actual position on z-axis. While r_{x_G} , r_{y_G} and r_{z_G} are the generated position vector component in x, y, and z frame by utilizing the following Python code (see Figure 14) to convert the initial population (TLE) to position and velocity vector using the SGP4 propagator. By first importing the “sgp4.api” library. Then using “Satrec.twoline2rv” function to combine line 1 and line 2 into parameter called “satellite” as shown in line 52. In line 53, the position vector (r) in km, the velocity vector (v) in km and (e) non-zero error code equal to “satellite.sgp4(jd, fr)”. Where “.sgp4” is the type of propagator used to convert the TLE to position and velocity vector in the ECI coordinate frame. “jd” is the whole number of the Julian Date, and “fr”

is the fraction of the Julian Date.

```
50 from sgp4.api import Satrec
51
52     satellite = Satrec.twoline2rv(line_1_str, line_2_str)
53     e, r, v = satellite.sgp4(jd, fr)
54
```

Figure 14: Python Code - Convert TLE to Position and Velocity Vector

After computing the fitness of each chromosome in a population, the selection process occurs. In this phase, the operator will select a number of chromosomes for the next phase which is the crossover.

There are three types of crossovers (see Figure 15): One-point crossover, multi-point crossover and uniform crossover. One point crossover has only one crossover — point a point on both pairs of chromosomes that were picked/generated randomly — where the genes of the two chromosomes swapped between the two chromosomes. Multi-point crossover is like the One-point crossover, but rather than having one crossover point, we have multiple crossover points. The uniform crossover is where the chromosomes are not divided into segments and each gene is treated separately. Each chromosome then has a 0.5 gene flip probability to decide whether they are included in the offspring. The uniform crossover can involve biasing the flip probability to one parent so that the child has more genetic material from the selected parent.

One-point Crossover



Multi-point Crossover



Uniform Crossover



Figure 15: Type of Crossovers

Uniform crossover is performed because it is based on probability and is randomly generated. Each time the crossover is performed, different genes are exchanged between the chromosomes. Figure 16 shows an example of a Uniform crossover performed with a crossover rate of 0.9.

```

-----
Before Crossover

SatelliteName
1 25544U 98067A 21271.50000000 1.4e-07 00000-0 67960-4 0 5293
2 25544 34.30424 308.8966 3086408 69.19105 312.6056 14.88345325 346978

SatelliteName
1 25544U 98067A 21271.50000000 1.4e-07 00000-0 67960-4 0 5293
2 25544 31.67484 243.7989 2773106 80.36055 344.8216 14.86102049 346978

After Crossover

SatelliteName
1 25544U 98067A 21271.50000000 1.4e-07 00000-0 67960-4 0 5293
2 25544 34.30424 243.7989 2773106 69.19105 344.8216 14.86102049 346978

SatelliteName
1 25544U 98067A 21271.50000000 1.4e-07 00000-0 67960-4 0 5293
2 25544 31.67484 308.8966 3086408 80.36055 312.6056 14.88345325 346978

```

Figure 16: Example of Crossover

Before evaluating the new chromosome, the mutation phase takes place. The crossover process can generate errors. The mutation process is necessary to fix these errors by flipping between the gene in the same chromosome. As illustrated in Figure 17, it can be done by flipping between genes next to each other. This type is known as Random Resetting. Alternatively, flipping between a random gene in the same chromosome and this type of mutation is called Swap Mutation. Rather than swapping only one gene in the chromosome, we swipe a set of genes in the chromosome. This type is called Scramble Mutation. The last type is the Inversion Mutation, where we select a set of genes in the chromosome and invert the entire gene.

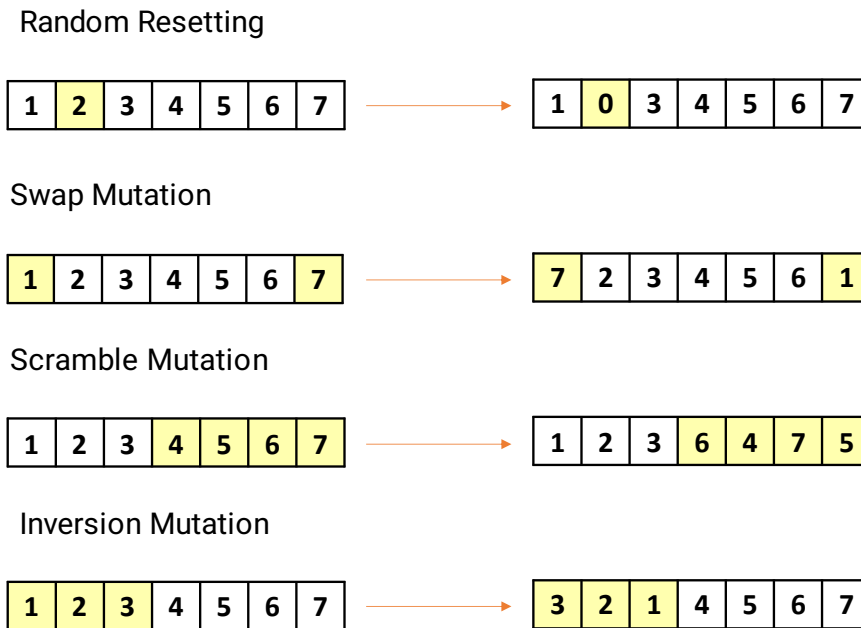


Figure 17: Type of Mutation

The random resetting mutation was used during the experiment, as shown in Figure 18. The type was chosen because of the simplicity of being implemented using Python.

```

Before Mutation
SatelliteName
1 25544U 98067A 21271.50000000 1.4e-07 00000-0 67960-4 0 5293
2 25544 31.67484 308.8966 3086408 80.36055 312.6056 14.88345325 346978

After Mutation
SatelliteName
1 25544U 98067A 21271.50000000 1.4e-07 00000-0 67960-4 0 5293
2 25544 31.67484 309.8966 3086408 80.36055 313.6056 15.88345325 346978

```

Figure 18: Example of Mutation

Chapter 4: Experiments, Results, Discussion, Recommendation and Conclusion

4.1 Results and Discussion

This section is divided into four different subsections. The first subsection provides information about the experimental setup. The Kalman filter results for the two satellites are provided in the second subsection. The third subsection shows the GA results. While in the final subsection shows a comparison between the Kalman filter and GA results.

4.1.1 Experimental Setup

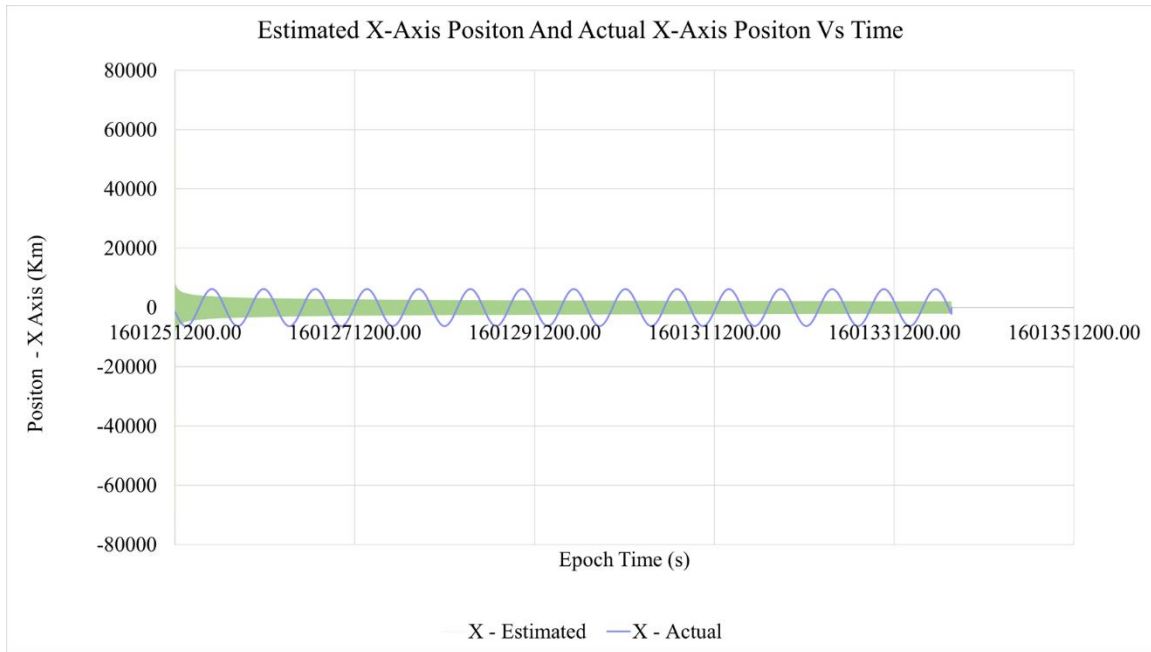
All three algorithms were implemented using Python code. The code was run using IDLE Python version 3.10 software. IDLE is an Integrated Development and Learning Environment with 100% pure python coding. The software is installed on a windows device with the following configuration:

Table 12: System Hardware Configurations

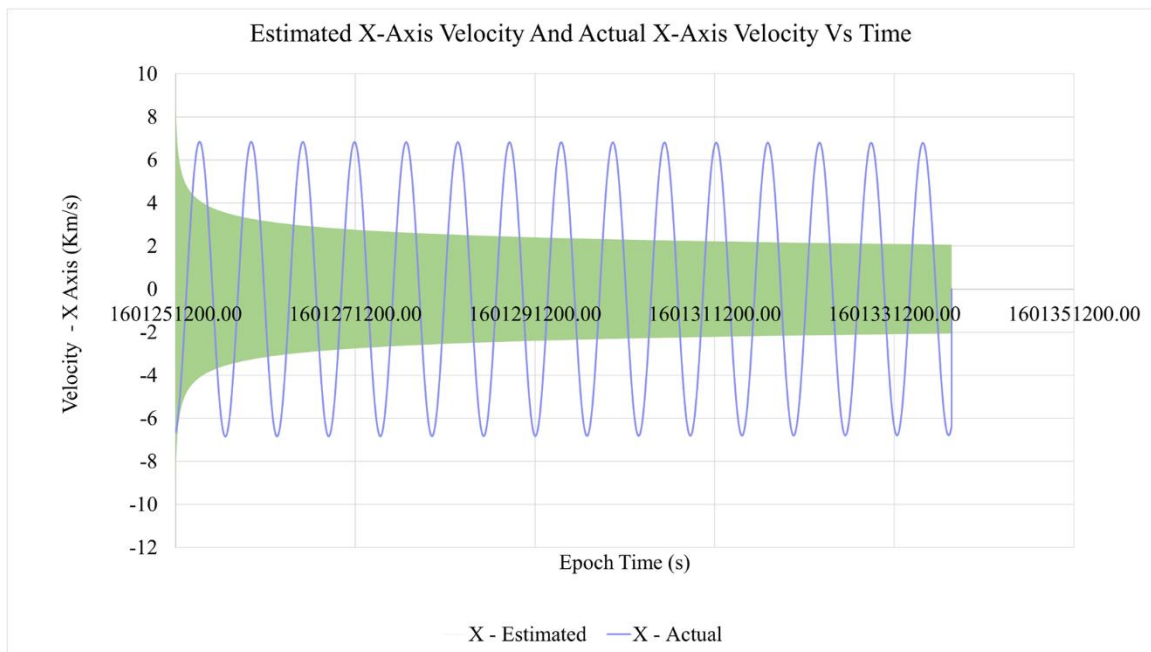
Hardware name	Specific Model
System Type	64-bit operating system, x64-based processor
Processor	Intel® Core™ i5-7300U CPU @ 2.60GHz 2.71 GHz
Memory	4.00 GB RAM
Storage	Solid state drive (SSD) options: 128GB
System	Windows 10 Pro, version 21H2

4.1.2 Part A

Figures 19 – 24 provide the EKF results for a MEZNSAT. Figures 19 – 21 shows the result of the Extended Kalman Filter estimations of the position and the velocity vector for the next 24 hours with a period of 10s. The simulation was done using Python, and zonal perturbation was included. The runtime was 151 s.

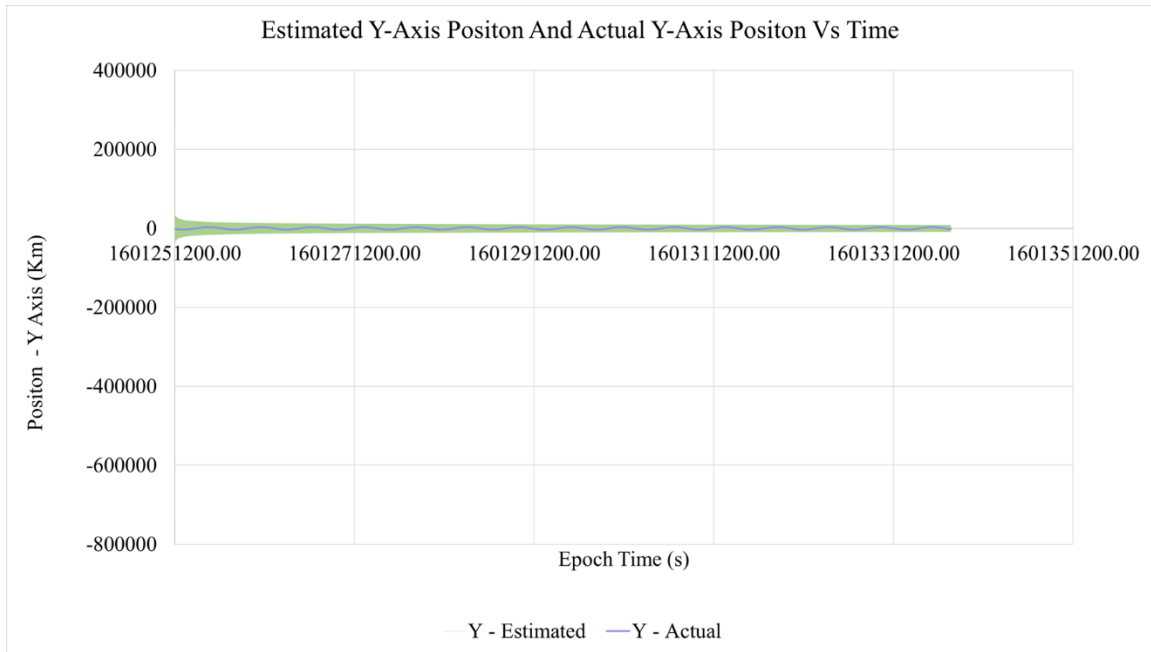


(a) Extended Kalman Position Filter Results Vs Actual Position Results (X-axis) – MEZNSAT

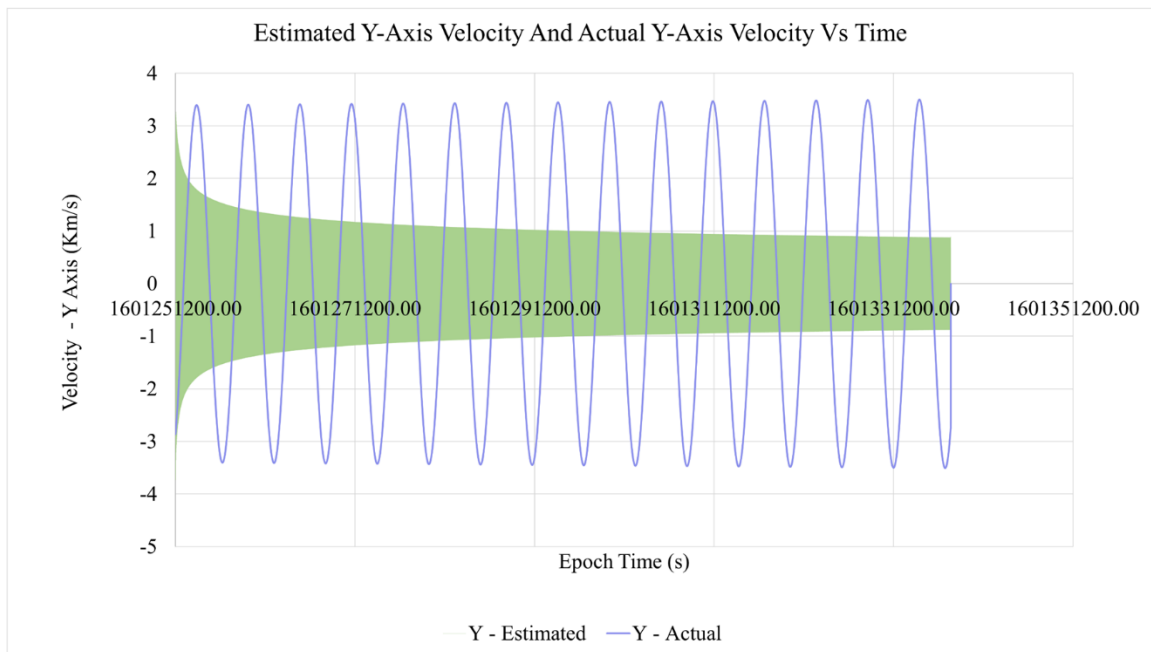


(b) Extended Kalman Filter Results Vs Velocity Actual Velocity Results (X-axis) – MEZNSAT

Figure 19: Extended Kalman Filter Results Vs Actual Results – MEZNSAT (X-Axis)

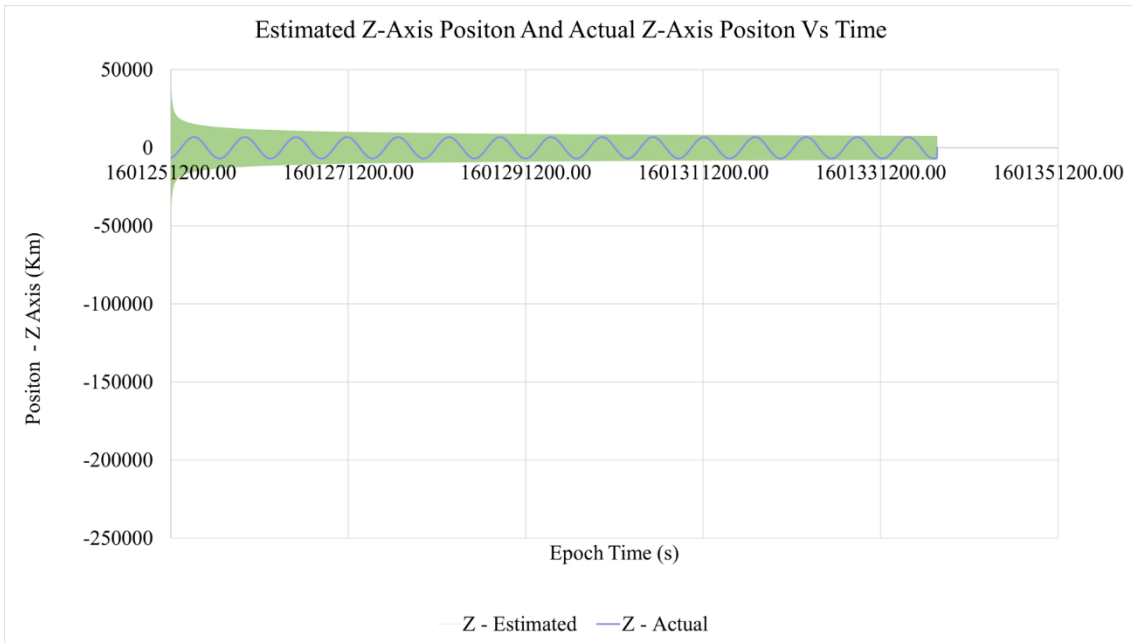


(a) Extended Kalman Position Filter Results Vs Actual Position Results (Y-axis) – MEZNSAT

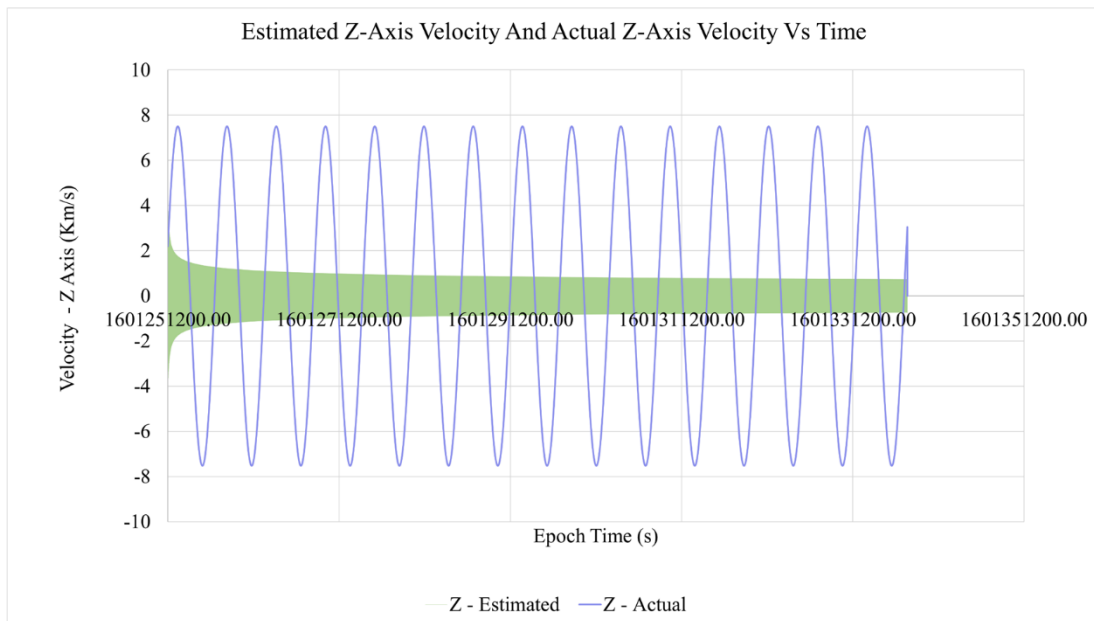


(b) Extended Kalman Filter Results Vs Velocity Actual Velocity Results (Y-axis) – MEZNSAT

Figure 20: Extended Kalman Filter Results Vs Actual Results – MEZNSAT (Y-Axis)



(a) Extended Kalman Position Filter Results Vs Actual Position Results (Z-axis) – MEZNSAT



(b) Extended Kalman Filter Results Vs Velocity Actual Velocity Results (Z-axis) – MEZNSAT

Figure 21: Extended Kalman Filter Results Vs Actual Results – MEZNSAT (Z-Axis)

As presented in Figure 22, the EKF started with high RMSs. Then start decreasing. With a position accuracy of 29.39 km and velocity accuracy of 0.030 km/s RMS.

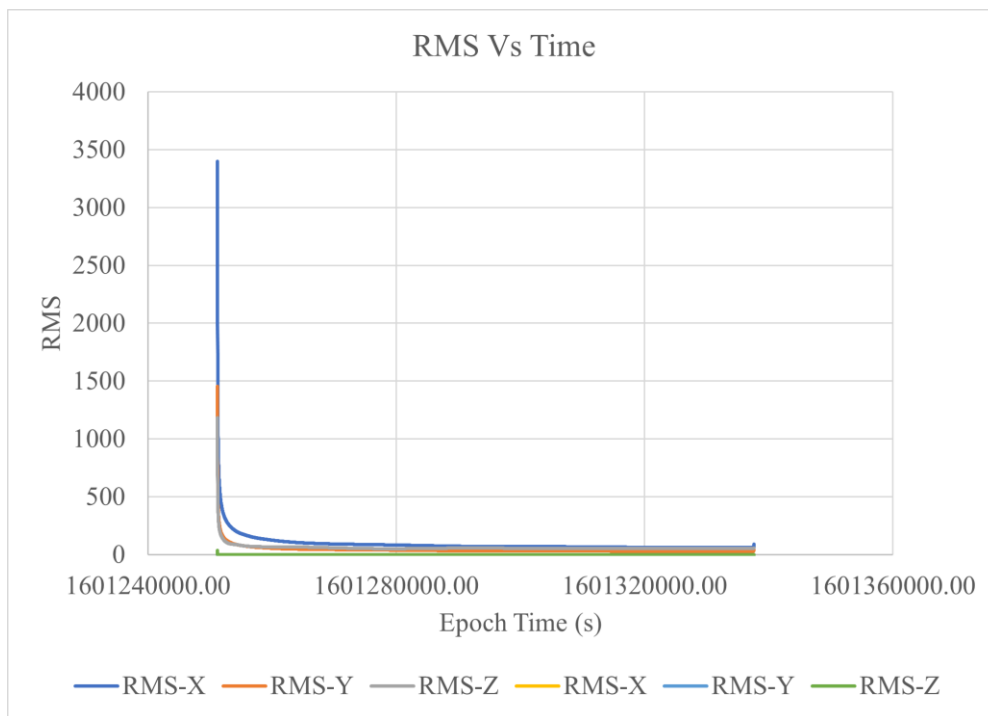
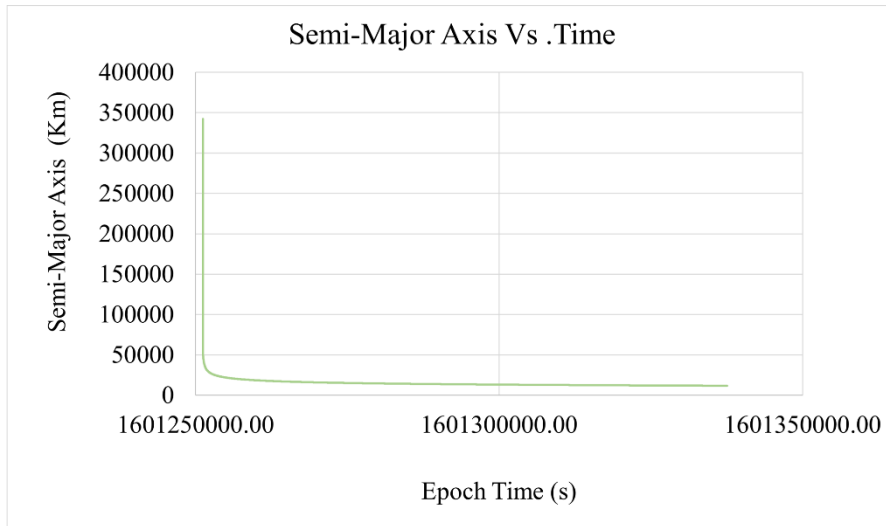
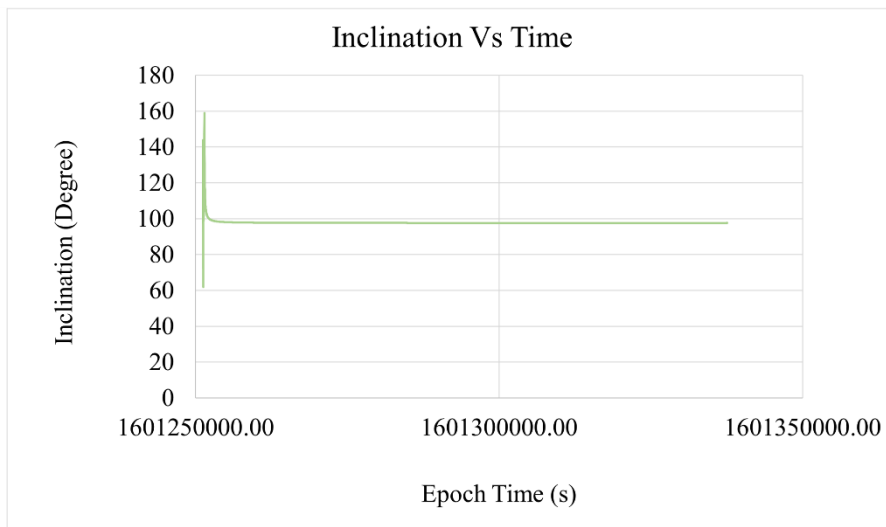


Figure 22: EKF RMS – MEZNSAT

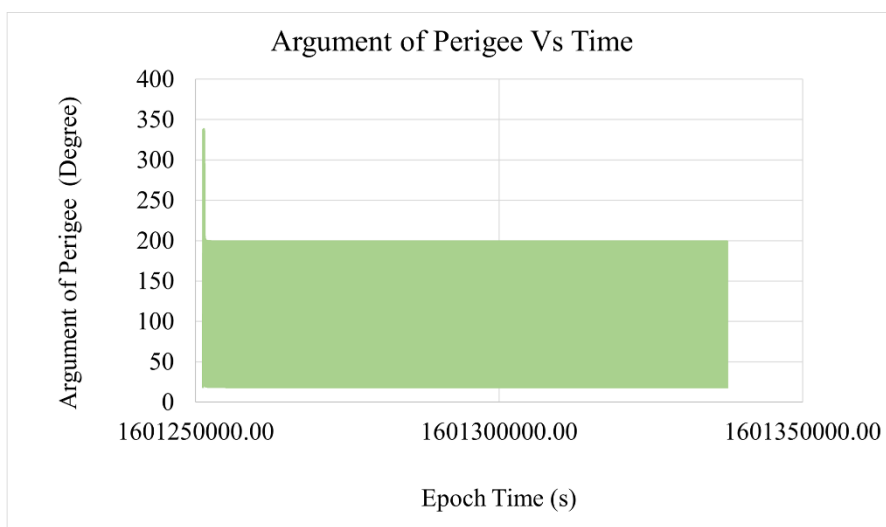
Figures 23 and 24 represent the estimated orbital parameter semi-major axis, eccentricity, inclination, RAAN, the argument of perigee and true anomaly using equations in chapter 1. As illustrated in Figures 23 and 24, there are no changes in the orbital parameter except at the beginning of the EKF, where the RMS values were high, the argument of perigee, and the true anomaly.



(a) Semi-major Axis

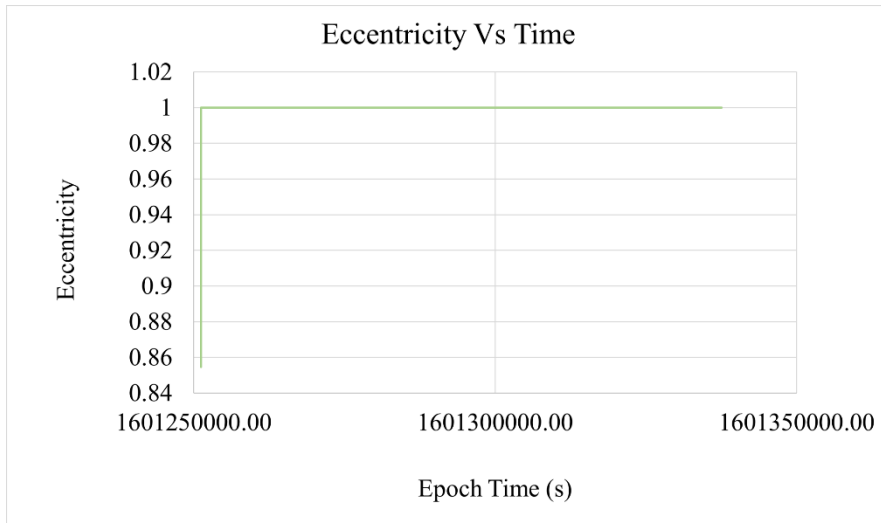


(b) Inclination

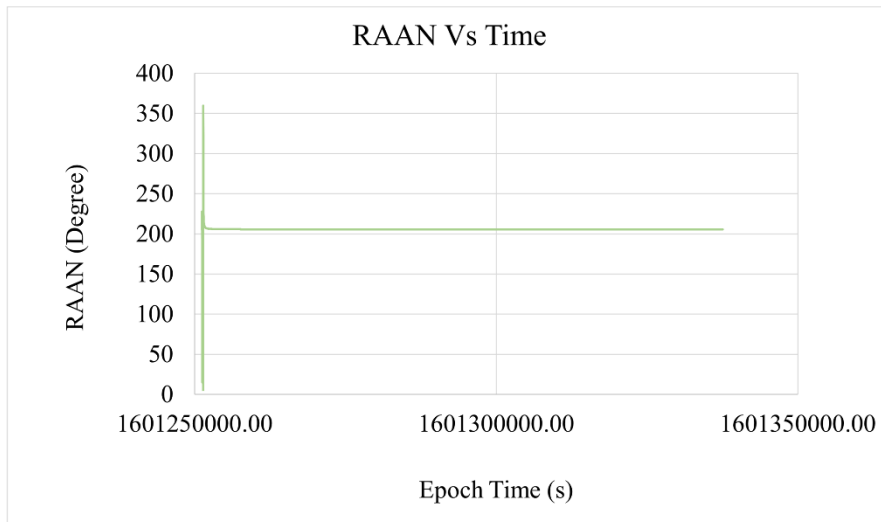


(c) Argument of Perigee

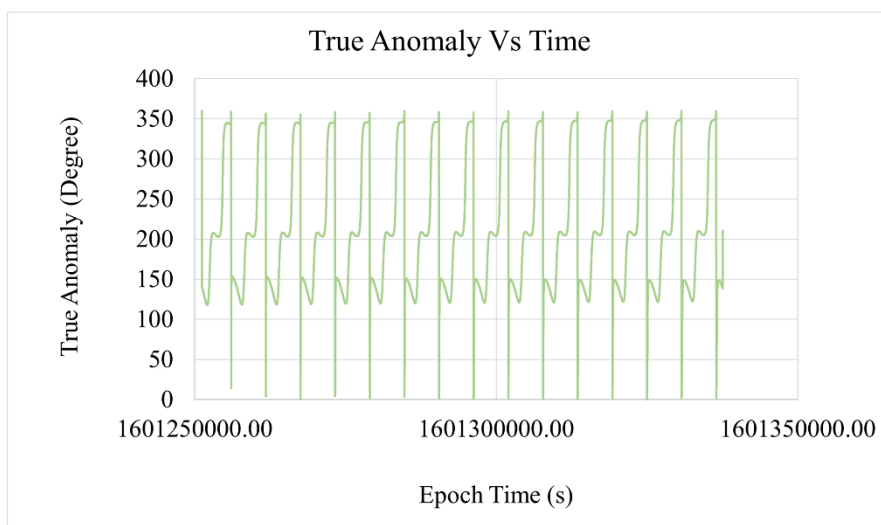
Figure 23: Extended Kalman Filter Result (Orbital Parameters) – MEZNSAT (Part 1)



(a) Eccentricity



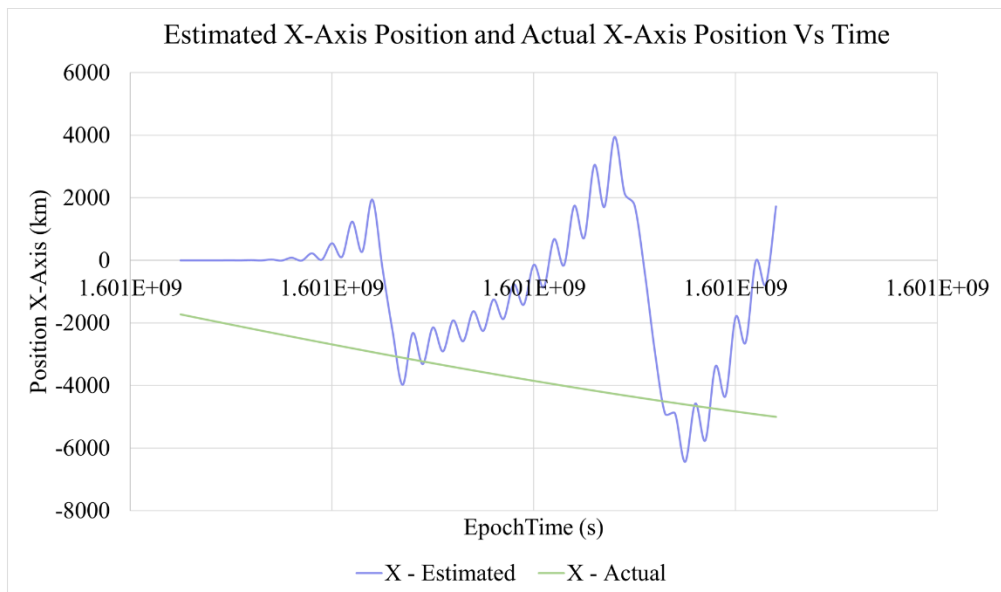
(b) RAAN



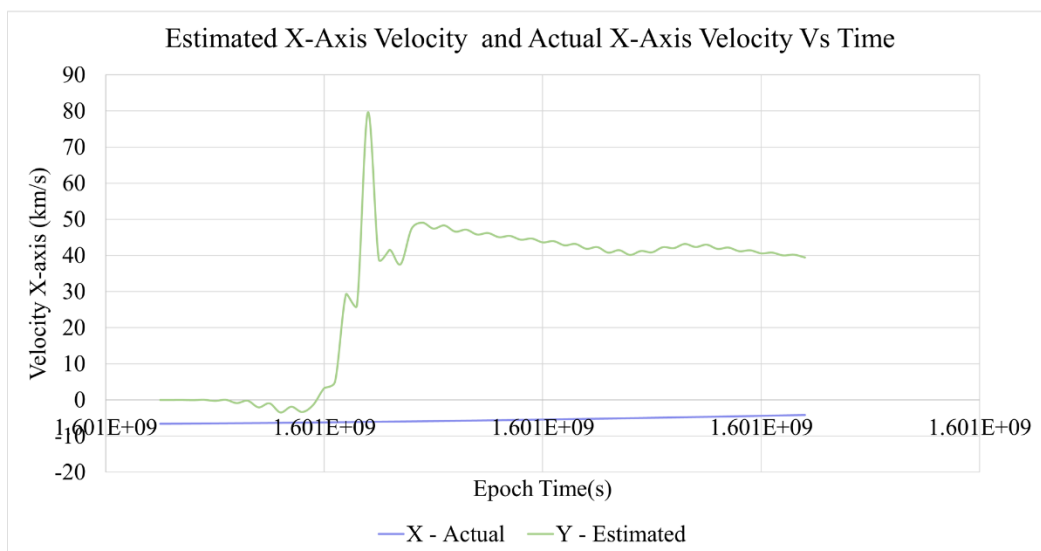
(c) True Anomaly

Figure 24: Extended Kalman Filter Result (Orbital Parameters) – MEZNSAT (Part 2)

Unlike the EKF, The UKF is a better for short period estimation. Figures 25 - 27 represent the UKF estimation results for the position and velocity vector for α (scaling parameter) equal to 0.01.

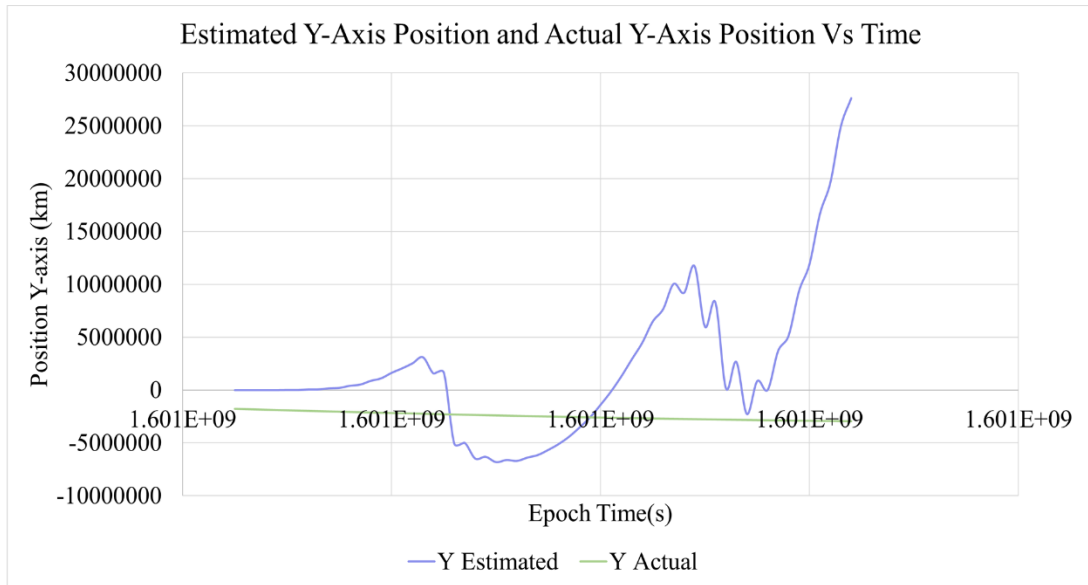


(a) Unscented Kalman Position Filter Results Vs Actual Position Results (X-axis) – MEZNSAT

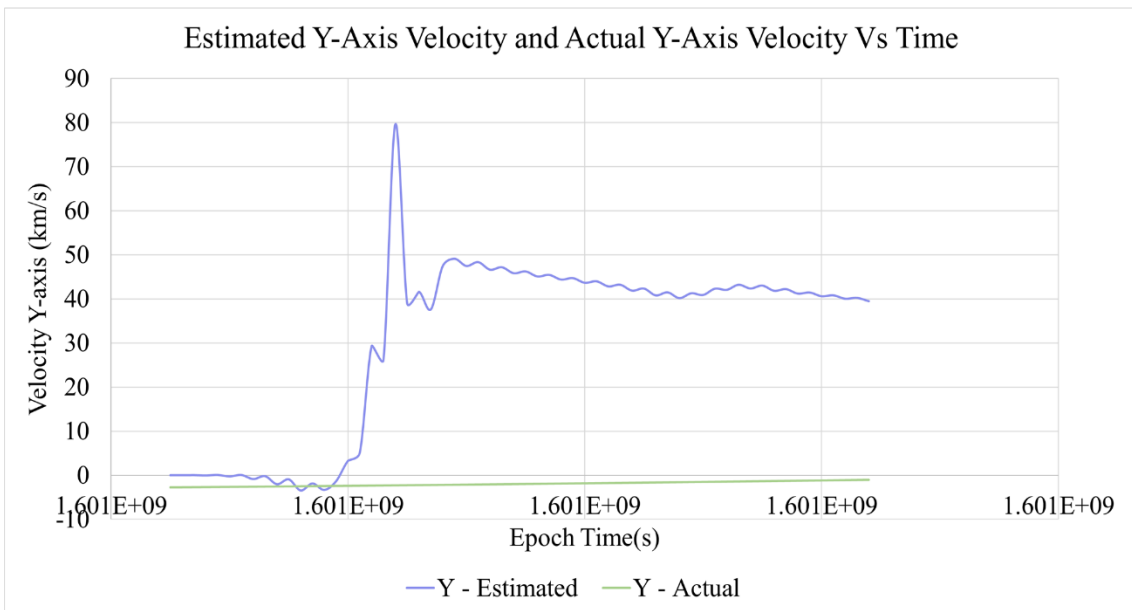


(b) Unscented Kalman Filter Results Vs Velocity Actual Velocity Results (X-axis) – MEZNSAT

Figure 25: Unscented Kalman Filter Results Vs Actual Results (X-axis)

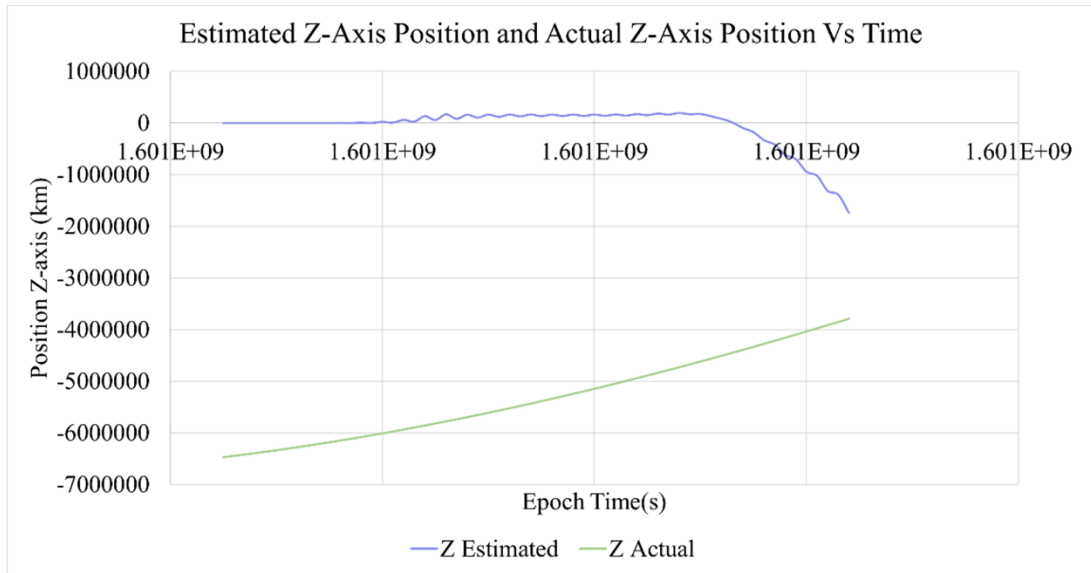


(a) Unscented Kalman Position Filter Results Vs Actual Position Results (Y-axis) – MEZNSAT

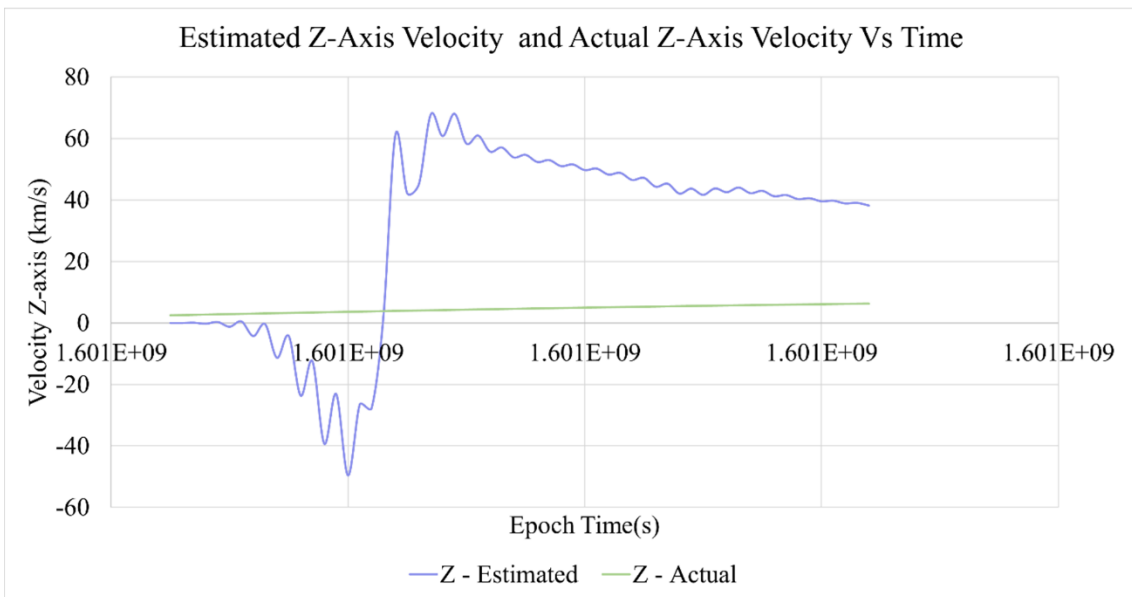


(b) Unscented Kalman Filter Results Vs Velocity Actual Velocity Results (Y-axis) – MEZNSAT

Figure 26: Unscented Kalman Filter Results Vs Actual Results (Y-axis)



(a) Unscented Kalman Position Filter Results Vs Actual Position Results (Z-axis) – MEZNSAT



(b) Unscented Kalman Filter Results Vs Velocity Actual Velocity Results (Z-axis) – MEZNSAT

Figure 27: Unscented Kalman Filter Results Vs Actual Results (Z-axis)

Unlike the EKF the UKF results start from a smaller value then increasing to the approximately the actual values, and after reaching the actual value the estimation values keep increasing because the UKF depends on adding the previous data to the new one. The UKF has a much lower and unacceptable accuracy with an average accuracy of 20%. Figure 28 show that the RMS keeps increasing by time.

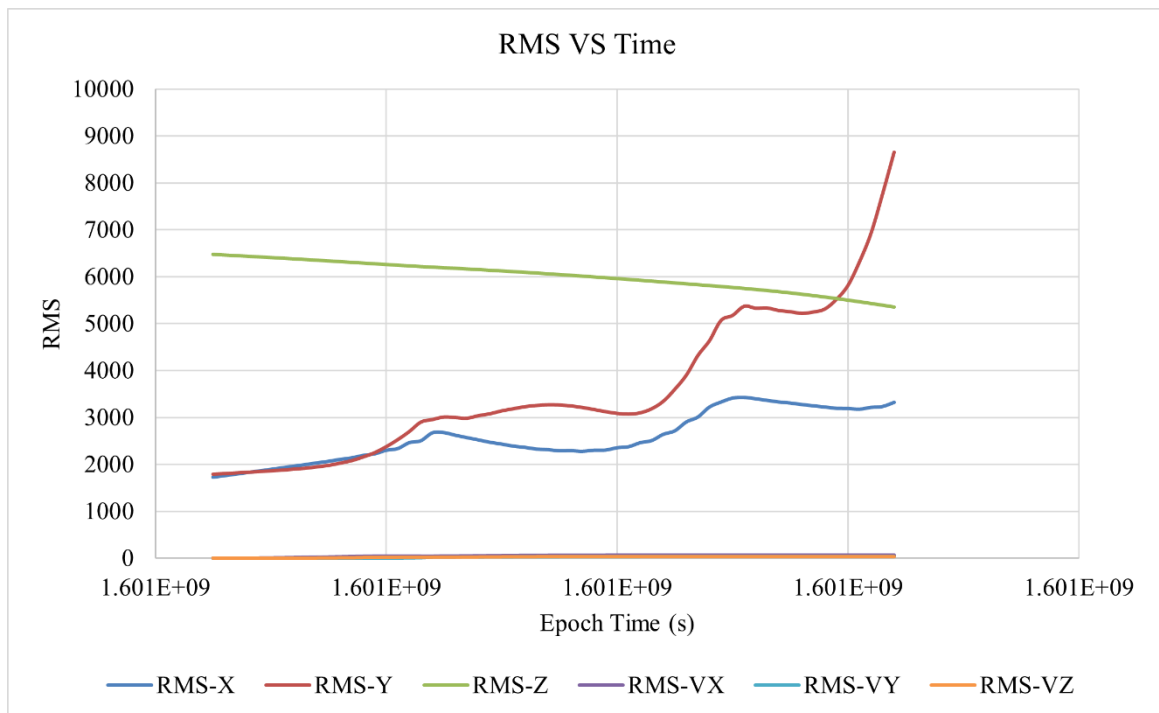
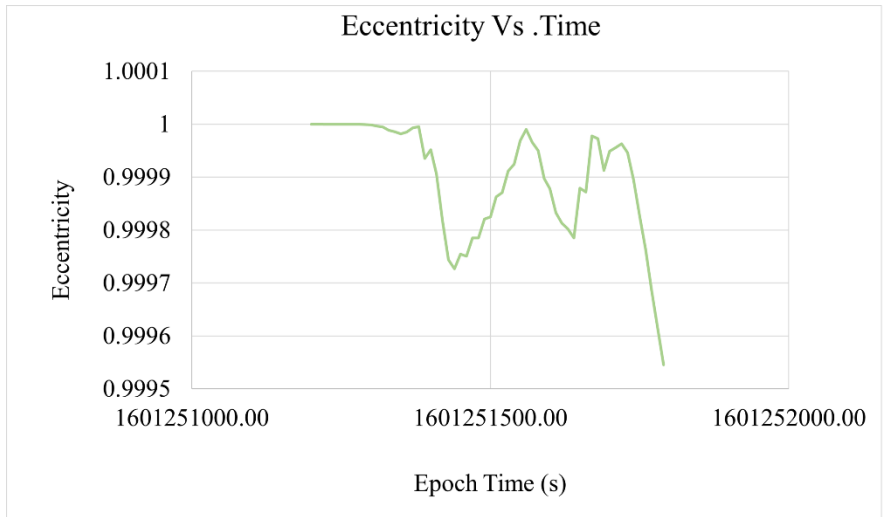
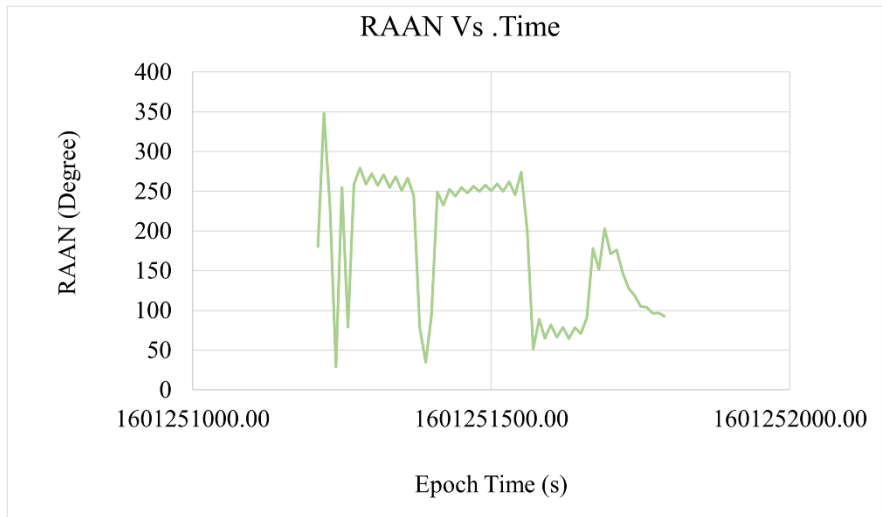


Figure 28: UKF – RMS

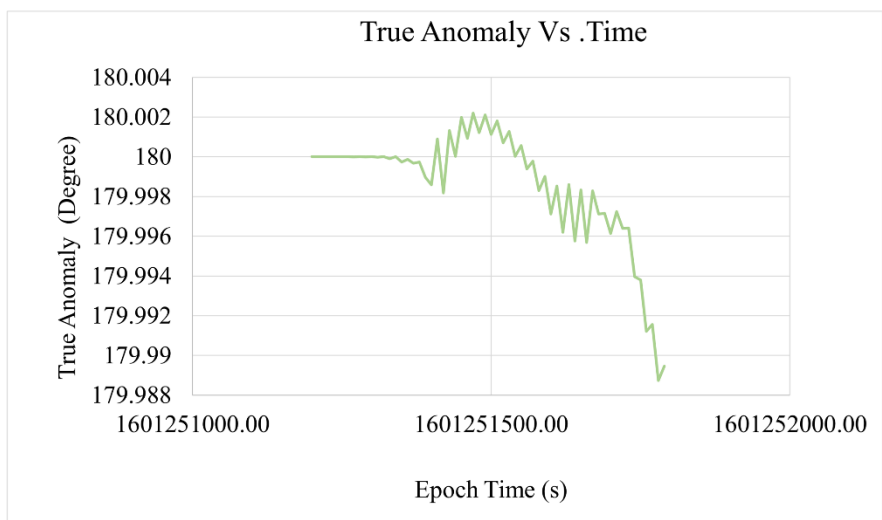
Figures 29 – 30 represent the calculated orbital parameter semi-major axis, eccentricity, inclination, RAAN, the argument of perigee and true anomaly using the UKF results. Unlike the EKF the orbital parameters were mostly stable. The UKF are unstable, with reasonable changing.



(a) Eccentricity

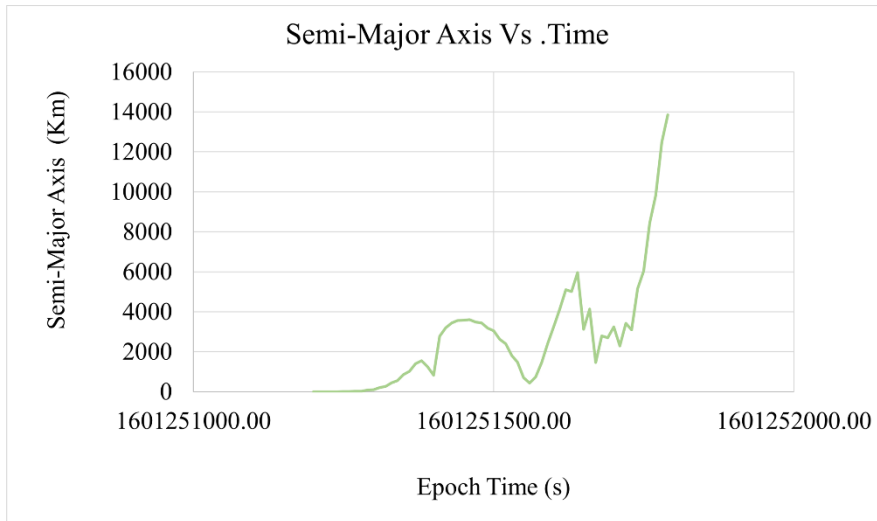


(b) RAAN

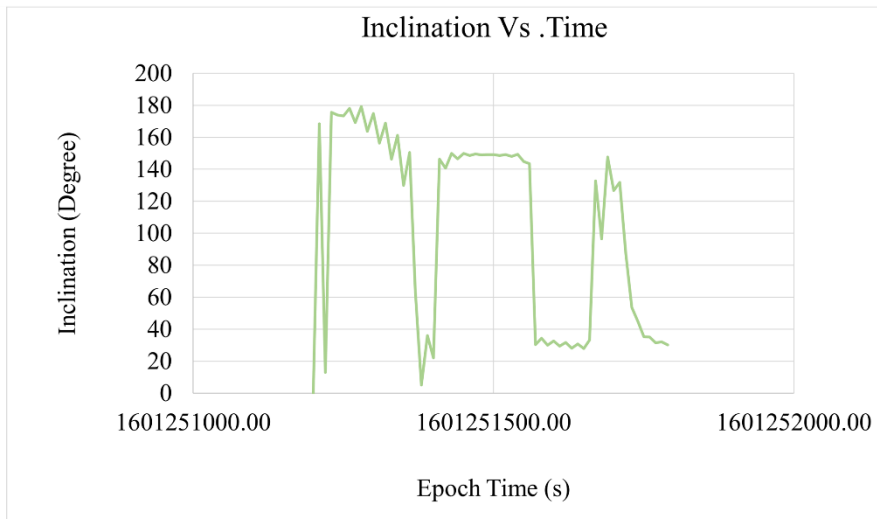


(c) True Anomaly

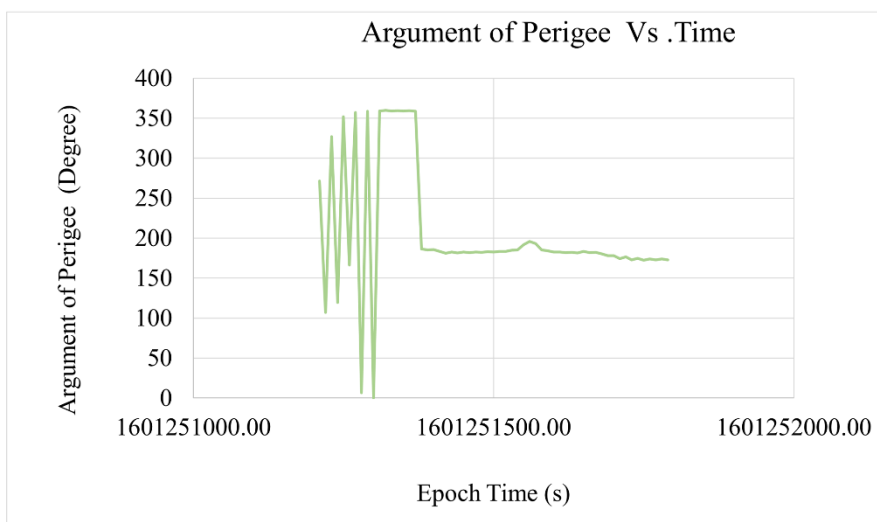
Figure 29: Unscented Kalman Filter Result (Orbital Parameters – Part 1)



(a) Semi-major Axis



(b) Inclination



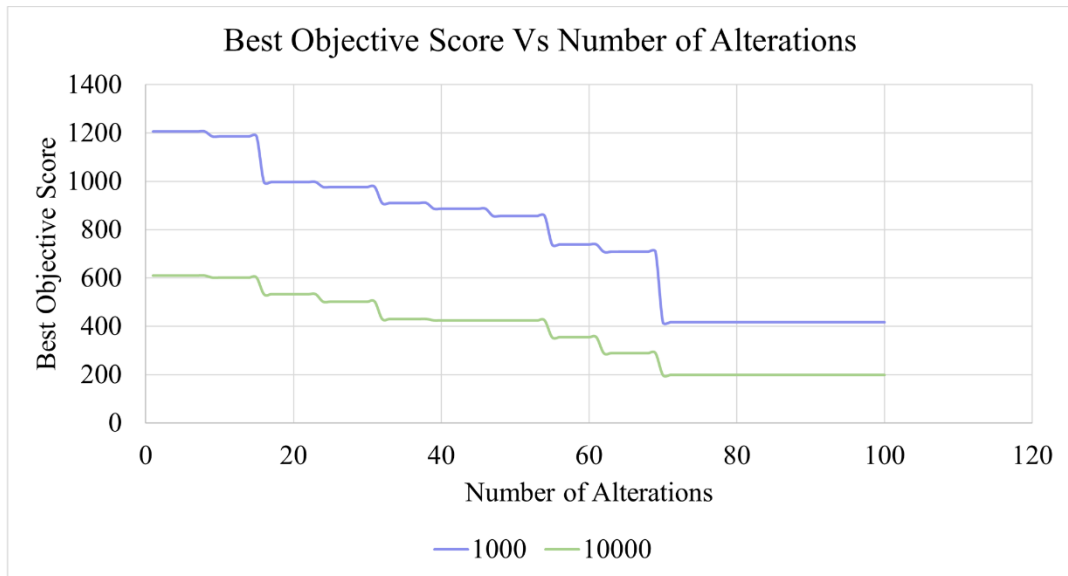
(c) Argument of Perigee

Figure 30: Unscented Kalman Filter Result (Orbital Parameters – Part 2)

4.1.3 Part B

The results show the first run of the GA under four different conditions. (a) The first condition has one thousand chromosomes and iterates over one hundred generations. Only 10% of the best chromosomes were selected for the crossover, and 0.1% of the best ones were selected for mutation. (b) The second condition has ten thousand chromosomes and iterates over one hundred generations, with only 1% of the best chromosomes selected for the crossover, and 0.01% of the best ones were selected for mutation. (c) The third condition has one thousand chromosomes and iterates over one thousand generations. Only 10% of the best chromosomes were selected for the crossover, and 0.1% of the best ones were selected for mutation. (d) The final condition has ten thousand chromosomes iterating over one thousand generations. Only 1% of the best chromosomes were selected for the crossover, and 0.01% of the best ones were selected for mutation. A uniform crossover rate of 0.9 and random resetting mutation is used in all the conditions. MEZNSAT data was utilized to find which of the four conditions is the best.

Figure 31 shows the first run of GA for the first two conditions (a) and (b). As illustrated in Figure 30a, the objective function score decreases with the new generation to get the best results. The run with the ten thousand initial population had a better score than the one thousand, with an objective function score of 199.422.



(a) Best Objective function score in Each Alteration

```

SatelliteName
1 25544U 98067A 20272.16666667 1.4e-07 00000-0 67960-4 0 5293
2 25544 99.03605 279.8936 1551641 358.4051 246.0540 15.31104327 346978
Done!
>>>

```

(b) Final TLE (initial population: 1000, number of alterations:100)

```

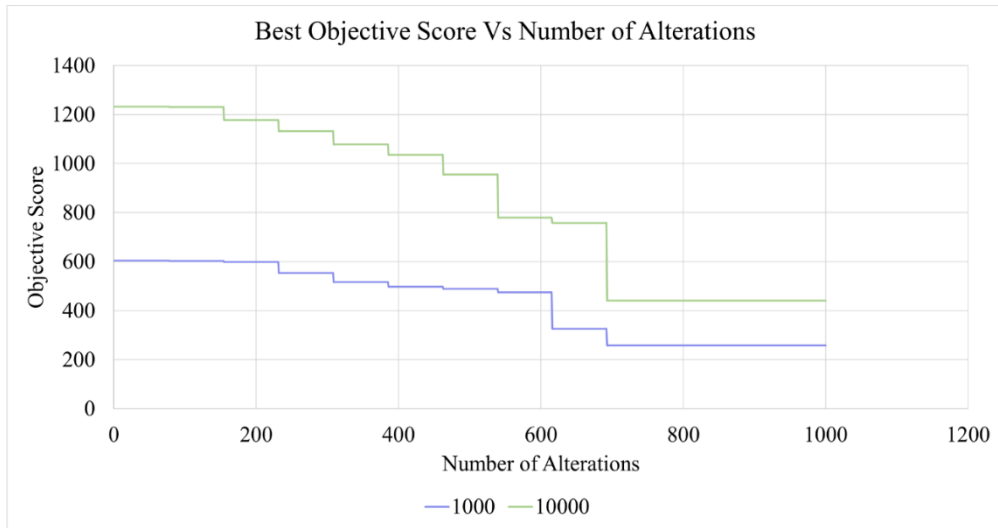
SatelliteName
1 25544U 98067A 20272.16666667 1.4e-07 00000-0 67960-4 0 5293
2 25544 91.15563 150.3381 1929557 265.1945 112.1862 15.77265452 346978
Done!
>>>

```

(c) Final TLE (initial population: 10000, number of alterations:100)

Figure 31: GA Result with one hundred Alterations

Figure 32 shows the first run of GA for conditions (c) and (d). Unlike the first two conditions, the condition with the higher initial population had a better score, with a score of 257.903. As shown in Figures 31 and 32 a, the conditions with a lower initial objective score got the lowest/best results.



(a) Best Objective function score in Each Alteration

```

SatelliteName
1 25544U 98067A 20272.16666667 1.4e-07 00000-0 67960-4 0 5293
2 25544 99.47170 306.5077 1332457 200.5814 238.0533 15.29321406 346978
Done

```

(b) Final TLE (initial population: 1000, number of alterations:1000)

```

SatelliteName
1 25544U 98067A 20272.16666667 1.4e-07 00000-0 67960-4 0 5293
2 25544 97.70946 338.7206 1603407 165.7043 45.75825 15.95225643 346978
Done!
>>>

```

(c) Final TLE (initial population: 10000, number of alterations:1000)

Figure 32: GA Result with 1000 Alterations

Table 13 compares the running time and the best score between the four conditions. Condition (b) has the best objective score, were condition (d) has the worst score of 440.639. However, condition (d) has the lowest maximum score of 36590.490. Condition (c) has the slowest running time with 630 s (10.5 minutes) were condition (a) has the fastest run time with approximately 14 s. Based on the results, increasing the number of alterations and the number of the initial population affect the running period.

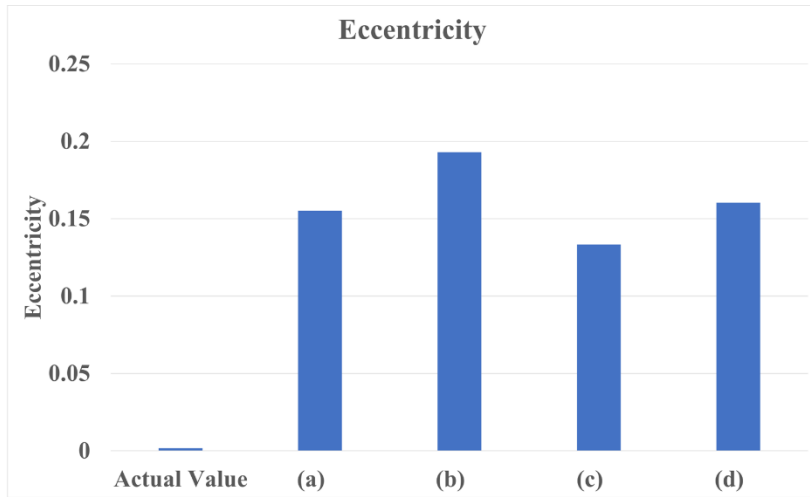
Table 13: GA results Comparison

	(a)	(b)	(c)	(d)
Runtime (s)	13.462	87.200	630.000	133.914
Minimum (best) Fitness Score	417.333	199.442	257.903	440.639
Maximum (worst) Fitness Score	54715.860	363737.6	80299.53	36590.490
Average Fitness Score	9343.426	9633.369	9432.5	9640.857

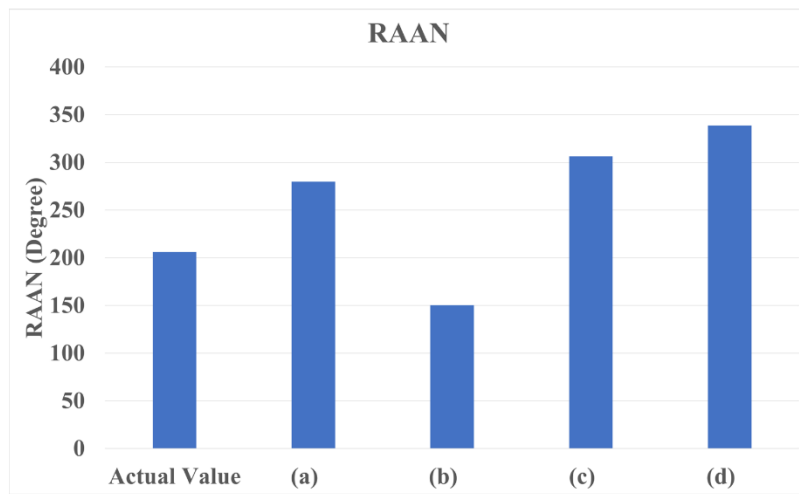
Table 14 and Figures 33 and 34, shows the difference between the generated and actual TLE values—the table highlights where the minimum difference was in each column. Of the four conditions, condition (b) got more minimum values and generally has the lowest differences. The table results confirm that the chosen objective function is correct.

Table 14: GA results Comparison – with actual TLE values

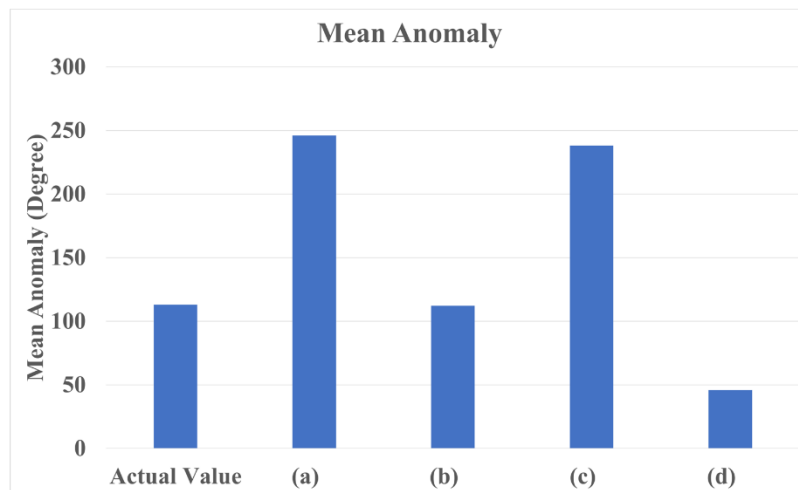
	Inclination [Degree]	RAAN [Degree]	Eccentricity	Argument of Perigee [Degree]	Mean Anomaly [Degree]	Mean Motion [Degree]
Actual Value	97.7327	206.065 3	0.0016388	317.8187	113.1421	15.0659
(a)	99.03605	279.893 6	0.1551641	258.4051	246.0540	15.3110 4
Difference	-1.30335	-73.8283	-0.15353	59.4136	-132.912	-0.2451
Actual Value	97.7327	206.065 3	0.0016388	317.8187	113.1421	15.0659
(b)	91.15563	150.338 1	0.1929557	265.1945	112.1862	15.7727
Difference	6.57707	55.7272	-0.19132	52.6242	0.9559	-0.70671
Actual Value	97.7327	206.065 3	0.0016388	317.8187	113.1421	15.0659
(c)	99.47170	306.507 7	0.1332457	200.5814	238.0533	15.2932
Difference	-1.739	-100.442	-0.13361	117.2373	-124.911	-0.22727
Actual Value	97.7327	206.065 3	0.0016388	317.8187	113.1421	15.0659
(d)	97.70946	338.720 6	0.160347	165.7043	45.75825	15.9523
Difference	0.02324	-132.655	-0.15871	152.1144	67.38385	-0.88631



(a) Eccentricity

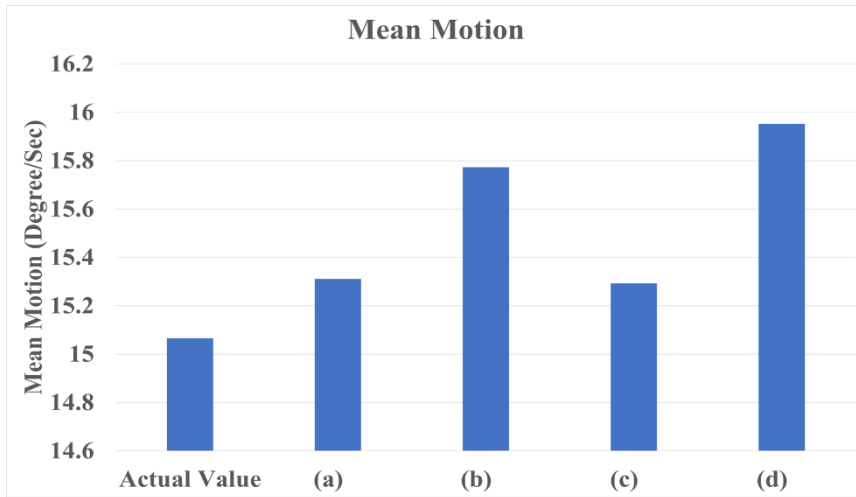


(b) RAAN

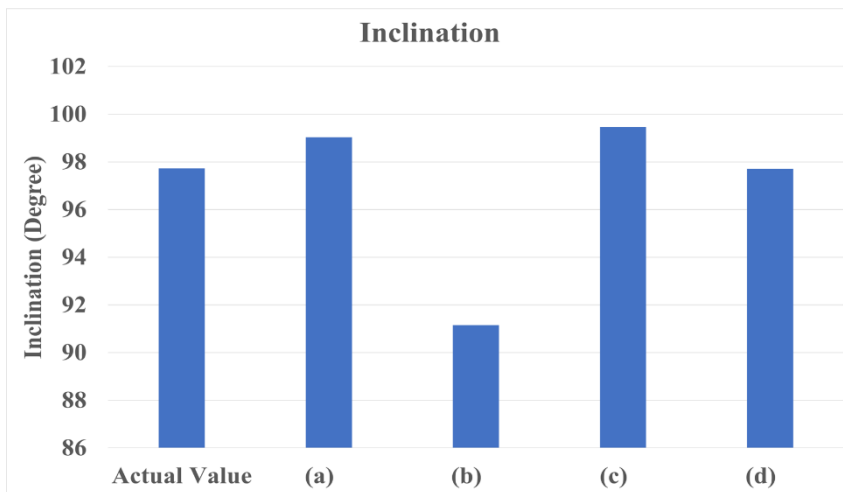


(c) Mean Anomaly

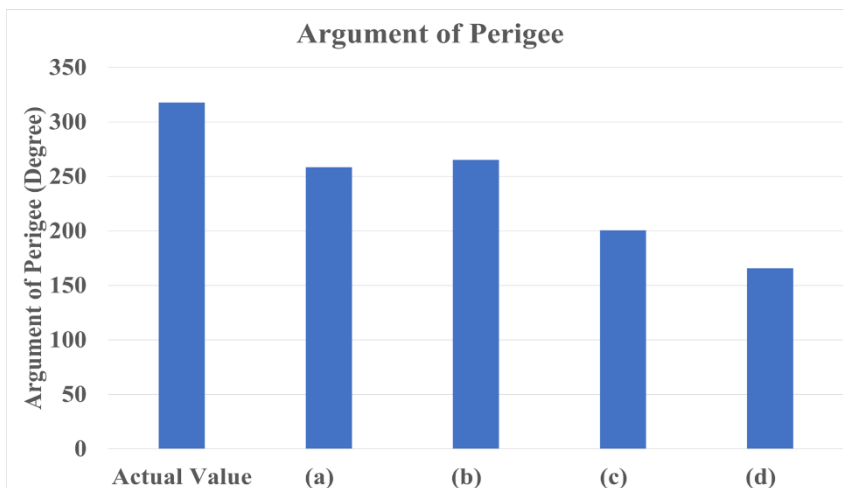
Figure 33: GA results Comparison – with actual TLE values (Part 1)



(a) Mean Motion



(b) Inclination



(c) Argument of Perigee

Figure 34: GA results Comparison – with actual TLE values (Part 2)

Table 15 shows the GA results for EOS-01. The GA was performed three times using the best-found settings (condition b) to get the best result. Due to mysterious reasons the runtime is not constant. The final run has the best objective function score of 229.5723.

Table 15: GA Results Comparison - EOS-01

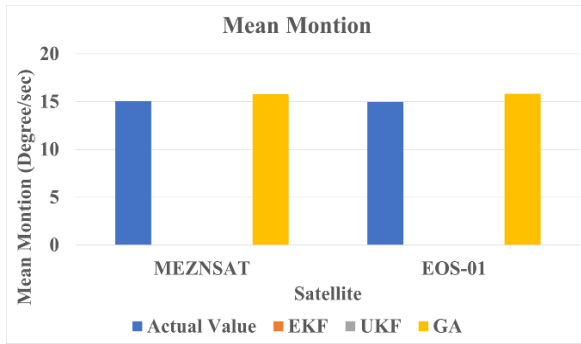
	First run	Second Run	Third Run
Runtime (s)	127.085	265.764	105.851
Minimum (best) Fitness Score	338.473	273.6379	229.5723
Maximum (worst) Fitness Score	27320.9	67486.09	278160.9
Average Fitness Score	9933.519	10525.97	9851.712

4.1.4 Part C

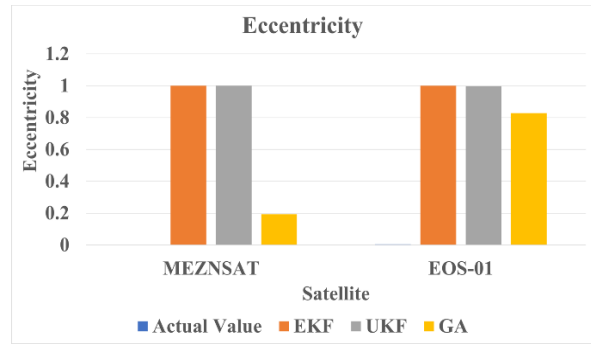
Table 16 and Figure 35 compare the best of the three algorithms used to compute the TLE for MEZNSAT and EOS-01 with the actual TLE. Table 16 determined the difference by detecting the algorithm result from the actual value. For MEZNSAT, the average EKF TLE was compared with the GA condition (b) and actual value. While for EOS-01, the average EKF TLE was compared with the GA third run and actual value. Even though the EKF could estimate the satellites' next position and velocity vector, the GA showed a better result in calculating the TLE for both satellites. While the UKF showed the worst results for both satellites. Unlike the GA, the EKF and the UKF requires more parameters and knowing the satellite. The EKF for EOS-01 showed the worst results due to the lack of knowledge and estimating some parameters, such as the cross-sectional area.

Table 16: Actual TLE Vs Algorithm Best TLE

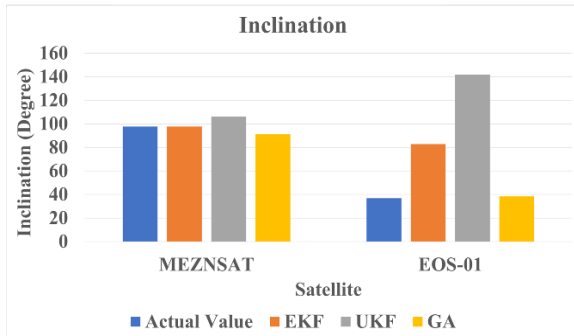
	Inclination [Degree]	RAAN [Degree]	Eccentricity	Argument of Perigee [Degree]	Mean Anomaly [Degree]	Mean Motion [Degree]
MEZNSAT						
Actual Value	97.7327	206.0653	0.0016388	317.8187	113.1421	15.0659
EKF	97.6250	205.5875	0.999983	199.0020	138.2730	-
Difference	0.1077	0.4778	0.998344	118.817	25.1309	-
Actual Value	97.7327	206.0653	0.0016388	317.8187	113.1421	15.0659
UKF	106.17	181.3872	0.9999	206.8678	179.9986	-
Difference	-8.4373	24.6781	-0.99826	110.9509	-66.8565	-
Actual Value	97.7327	206.0653	0.0016388	317.8187	113.1421	15.0659
GA (b)	91.15563	150.3381	0.1929557	265.1945	112.1862	15.7726
Difference	6.57707	55.7272	-0.19132	52.6242	0.9559	-0.70671
EOS-01						
Actual Value	36.9035	333.7447	0.004696	69.9588	290.1640	14.9809
EKF	82.81379	322.0986	0.99999	38.12821	180.1381	-
Difference	-45.9103	11.6461	-0.99529	31.83059	110.0259	-
Actual Value	36.9035	333.7447	0.004696	69.9588	290.1640	14.9809
UKF	141.8172	261.7364	0.996778	66.955	179.9899	-
Difference	-104.914	72.0083	-0.99208	3.0038	110.1741	-
Actual Value	36.9035	333.7447	0.004696	69.9588	290.1640	14.9809
GA (3 rd Run)	38.37146	231.9948	0.8287462	108.0261	304.5818	15.8318
Difference	-1.46796	101.7499	-0.82405	-38.0673	-14.4178	-0.85098



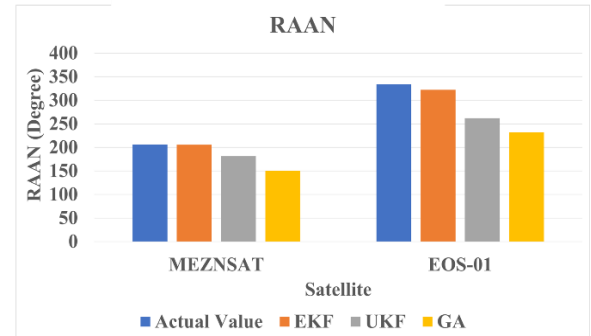
(a) Mean Motion



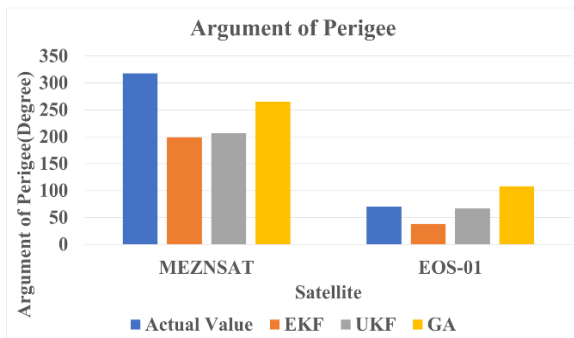
(b) Eccentricity



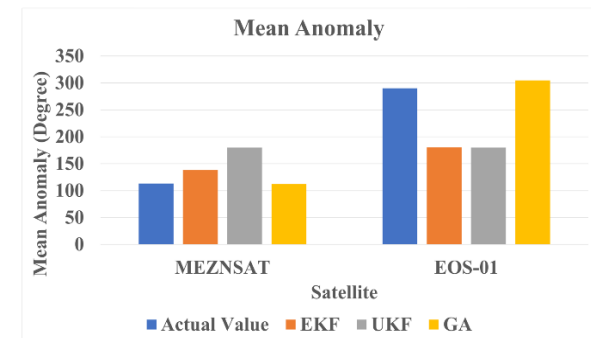
(c) Inclination



(d) RAAN



(e) Argument of Perigee



(f) Mean Anomaly

Figure 35: Actual TLE Vs Algorithm Best TLE

Figure 36 show a comparison between the EKF, UKF, and GA run time. The GA is much faster than the EKF, but it provides only one TLE, where the EKF provide more than 8000 TLE. Overall, the UKF is the fastest but unlike the EKF it only provided 60 inaccurate TLE.

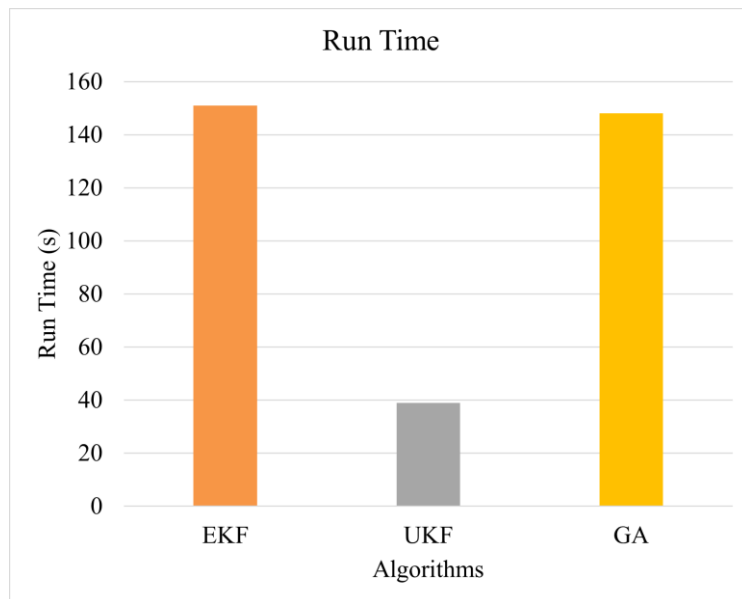


Figure 36: Algorithms Run Time

4.2 Recommendation

Here are some recommendations to improve the algorithms results:

- Considering other forces applied to the satellite, such as gravitational and solar radiation pressure, while designing the dynamic model will help improving the Extended Kalman filter and the Unscented Kalman Filter results.
- Using different scaling parameters might improve the UKF results.
- All the publication about UKF, they used the same state transition matrix and the same observation matrix that has been used in the EKF, maybe using different matrix will help improving the UKF results.
- Using SGP4/SDP4 rather than Kepler equation to compute the TLE.
- Using the Kalman filter to estimate the TLE rather than the position and velocity vector.
- Generating more initial population will help improve the GA results.
- Narrow the random value range for the initial population genes to get better results.
- Increase the percentage of the used chromosomes in the crossover process and in the

mutation, process will help improving the GA results.

- Change the type of crossover and mutation might help in improving the GA results.
- Change the uniform crossover rate might help improving GA results.
- Changing the fitness function that is used in GA can help in improving the prediction of the algorithms.
- Changing the fitness function to be able to predict a new TLE based on time.
- Using the GA to estimate the next position and velocity vector of satellite like the Kalman filters.

4.3 Conclusion

This thesis investigates the uses of the extended Kalman filter, unscented Kalman filter, and genetic algorithm to solve the orbit determination and orbit propagation of small/medium satellites in space using a GPS-based navigation system. In this paper, the extended Kalman filter and unscented Kalman filter algorithm were utilized to estimate the next position and velocity. The GA was utilized to find the best TLE.

From the results of this paper, it can be concluded that EKF achieved the goal of estimating the next position and velocity vector and showed a better result than the UKF. The UKF showed the worst results, with an accuracy of approximately 20%. However, according to Choi et al. (2010), the UKF were 20% better than the EKF, with an average RMS of 10.135 m and 0.0244 m/s. Pardal et al. (2013) show that the EKF and The UKF have a similar performance with an average RMS of 30 m and 25 m. GA showed an excellent result in finding the best TLE, better results than the Kalman filter algorithms only if the fitness score/objective score is less than two hundred.

In conclusion, in the case of only updating the TLE, the GA is a better choice than the Kalman filter algorithms because the genetic algorithm requires one parameter and showed a better result with an approximate average accuracy of 88%. However, in the case of updating the position and velocity vector, the Kalman filter algorithms are a better choice than the GA but, between the two filters the EKF is a much better than the UKF.

References

- Aghav, S., & Gangal, S. (2014). Simplified Orbit Determination Algorithm for Low Earth Orbit Satellites Using Spaceborne GPS Navigation Sensor. *Artificial Satellites*, 49, 81-99. <https://doi.org/10.2478/arsa-2014-0007>
- Amaral, J., Kuga, H., & Souza, M. (2007). *Real Time Multisatellite Orbit Determination for Constellation Maintenance*. Accessed on October 10, 2021, from https://www.researchgate.net/publication/350823755_REAL_TIME_MULTISATELLITE_ORBIT_DETERMINATION_FOR_CONSTELLATION_MAINTENANCE
- Ansalone, L., & Curti, F. (2013). A genetic algorithm for Initial Orbit Determination from a too short arc optical observation. *Advances in Space Research*, 52(3), 477–489. <https://doi.org/10.1016/j.asr.2013.04.004>
- Ardaens, J.-S., & Gaias, G. (2019). A numerical approach to the problem of angles-only initial relative orbit determination in low earth orbit. *Advances in Space Research*, 63(12), 3884–3899. <https://doi.org/10.1016/j.asr.2019.03.001>
- Benhachmi, M., Ouazar, D., Naji, A., Cheng, A., & el Harrouni, K. (2001). *Optimal Management in Saltwater-Intruded Coastal Aquifers By Simple Genetic Algorithm*. Accessed on October 10, 2021, from https://www.researchgate.net/publication/254667749_Optimal_Management_in_Saltwater-Intruded_Coastal_Aquifers_By_Simple_Genetic_Algorithm
- Bolandi, H., Larki, M. H. A., Abedi, M., & Esmailzade, M. (2013). GPS based onboard orbit determination system providing fault management features for a LEO satellite. *Journal of Navigation*, 66(4), 539–559. <https://doi.org/10.1017/S0373463313000179>
- Brown, E. C., & Sumichrast, R. T. (2005). Evaluating performance advantages of grouping genetic algorithms. *Engineering Applications of Artificial Intelligence*, 18(1), 1–12. <https://doi.org/10.1016/j.engappai.2004.08.024>
- Chiaradia, A. P., Gill, E., Montenbruck, O., Kuga, H., & Prado, A. (2000). *Algorithms for On-Board Orbit Determination using GPS OBODE-GPS*. Accessed on October 10, 2021, from https://www.researchgate.net/publication/224781954_Algorithms_for_On-Board_Orbit_Determination_using_GPS_OBODE-GPS
- Choi, E.-J., Yoon, J.-C., Lee, B.-S., Park, S.-Y., & Choi, K.-H. (2010). Onboard orbit determination using GPS observations based on the unscented Kalman filter. *Advances in Space Research*, 46(11), 1440–1450. <https://doi.org/10.1016/j.asr.2010.07.022>
- Dai, X., Lou, Y., Dai, Z., Hu, C., Peng, Y., Jing, Q., & Shi, C. (2019). Precise Orbit Determination for GNSS Maneuvering Satellite with the Constraint of a Predicted Clock. *Remote Sensing*, 11, 1949-1964. <https://doi.org/10.3390/rs11161949>

- Eliasson, M. (2014). *A Kalman filter approach to reduce position error for pedestrian applications in areas of bad GPS reception*. Accessed on October 10, 2021, from <https://www.diva-portal.org/smash/get/diva2:743775/FULLTEXT01.pdf>
- Enriquez-Caldera, R. (2013). Global Navigation Satellite Systems: Orbital Parameters, Time and Space Reference Systems and Signal Structures. In *Handbook of Satellite Applications* (pp. 573–601). https://doi.org/10.1007/978-1-4419-7671-0_93
- Ge, H., Li, B., Jia, S., Nie, L., Wu, T., Yang, Z., Shang, J., Zheng, Y., & Ge, M. (2022). LEO Enhanced Global Navigation Satellite System (LeGNSS): Progress, opportunities, and challenges. *Geo-Spatial Information Science*, 25(1), 1–13. <https://doi.org/10.1080/10095020.2021.1978277>
- Gill, E., Montenbruck, O., Arichandran, K., Tan, S. H., & Bretschneider, T. (2004). *High-Precision Onboard Orbit Determination for Small Satellites – The Gps-Based Xns On X-Sat*. Accessed on October 10, 2021, from https://www.researchgate.net/publication/234269518_High-precision_onboard_orbit_determination_for_small_satellites_-_The_GPS-based_XNS_on_X-SAT
- Hinagawa, H., Yamaoka, H., & Hanada, T. (2014). Orbit determination by genetic algorithm and application to GEO observation. *Advances in Space Research*, 53(3), 532–542. <https://doi.org/10.1016/j.asr.2013.11.051>
- Hoots, F. R., & Roehrich, R. L. (1980). *Models for Propagation of NORAD Element Sets*: Defense Technical Information Center. 1-100. <https://doi.org/10.21236/ADA093554>
- Julier, S. J., & Uhlmann, J. K. (1997). *A New Extension of the Kalman Filter to Nonlinear Systems*. Accessed on October 10, 2021, from https://www.cs.unc.edu/~welch/kalman/media/pdf/Julier1997_SPIE_KF.pdf
- Kaufman, E., Lovell, T. A., & Lee, T. (2016). Nonlinear Observability for Relative Orbit Determination with Angles-Only Measurements. *The Journal of the Astronautical Sciences*, 63(1), 60–80. <https://doi.org/10.1007/s40295-015-0082-9>
- Kidder, S. Q. (2003). SATELLITES | Orbits. In J. R. Holton (Ed.), *Encyclopedia of Atmospheric Sciences* (pp. 2024–2038). Academic Press. <https://doi.org/10.1016/B0-12-227090-8/00362-6>
- Kumar, M., Husain, M., Upreti, N., & Gupta, D. (2010). Genetic Algorithm: Review and Application. *SSRN Electronic Journal*. 451-454. <https://doi.org/10.2139/ssrn.3529843>
- Lam, Q., Junker, D., Anhalt, D., & Vallado, D. (2010). *Analysis of an Extended Kalman Filter Based Orbit Determination System*. 1-7. <https://doi.org/10.2514/6.2010-7600>

- LaViola, J. J. (2003). A comparison of unscented and extended Kalman filtering for estimating quaternion motion. *Proceedings of the 2003 American Control Conference, 2003.*, 3, 2435–2440. <https://doi.org/10.1109/ACC.2003.1243440>
- Li, B., Ge, H., Ge, M., Nie, L., Shen, Y., & Schuh, H. (2019). LEO enhanced Global Navigation Satellite System (LeGNSS) for real-time precise positioning services. *Advances in Space Research*, 63(1), 73–93. <https://doi.org/10.1016/j.asr.2018.08.017>
- Lu, W., Wang, H., Wu, G., & Huang, Y. (2022). Orbit Determination for All-Electric GEO Satellites Based on Space-Borne GNSS Measurements. *Remote Sensing*, 14(11), Article 11, 2627-2629. <https://doi.org/10.3390/rs14112627>
- Luo, Y., Qin, T., & Zhou, X. (2022). Observability Analysis and Improvement Approach for Cooperative Optical Orbit Determination. *Aerospace*, 9(3), Article 3, 166-167. <https://doi.org/10.3390/aerospace9030166>
- Ma, L., Xu, X., & Pang, F. (2016). Accuracy Assessment of Geostationary-Earth-Orbit with Simplified Perturbations Models. *Artificial Satellites*, 51(2), 55–59. <https://doi.org/10.1515/arsa-2016-0005>
- Mao, X., Arnold, D., Girardin, V., Villiger, A., & Jäggi, A. (2020). Dynamic GPS-based LEO orbit determination with 1 cm precision using the Bernese GNSS Software. *Advances in Space Research*, 67, 788-792. <https://doi.org/10.1016/j.asr.2020.10.012>
- Meyer, P. J. (2016). *Julian Day Numbers*. Accessed on October 10, 2021, from https://www.academia.edu/28437708/Julian_Day_Numbers
- Nath, U. M., Dey, C., & Mudi, R. K. (2020). Designing of dynamic Kalman filter for prediction of mean arterial blood pressure. *Procedia Computer Science*, 167, 2478–2485. <https://doi.org/10.1016/j.procs.2020.03.300>
- Onat, A. (2019). A Novel and Computationally Efficient Joint Unscented Kalman Filtering Scheme for Parameter Estimation of a Class of Nonlinear Systems. *IEEE Access*, 7, 31634–31655. <https://doi.org/10.1109/ACCESS.2019.2902368>
- Pardal, P. C. P. M., Kuga, H. K., & De Moraes, R. V. (2013). Analyzing the Unscented Kalman Filter Robustness for Orbit Determination through Global Positioning System Signals. *Journal of Aerospace Technology and Management*, 5(4), 395–408. <https://doi.org/10.5028/jatm.v5i4.252>
- Sun, J., Tao, L., Niu, Z., & Zhu, B. (2020). An Improved Adaptive Unscented Kalman Filter With Application in the Deeply Integrated BDS/INS Navigation System. *IEEE Access*, 8, 95321–95332. <https://doi.org/10.1109/ACCESS.2020.2995746>
- T.S, K. (2020, July 1). *CelesTrak: NORAD Two-Line Element Set Format*. CelesTrak. Accessed on October 10, 2021, from <https://celestrak.org/NORAD/documentation/tle-fmt.php>

- Wan, E. A., & Van Der Merwe, R. (2000). The unscented Kalman filter for nonlinear estimation. *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*, 153–158. <https://doi.org/10.1109/ASSPCC.2000.882463>
- Welch, G., & Bishop, G. (2006). *An Introduction to the Kalman Filter*. Accessed on October 10, 2021, from https://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf
- Yoon, J.-C., Lee, B.-S., & Choi, K.-H. (2000). Spacecraft orbit determination using GPS navigation solutions. *Aerospace Science and Technology*, 4(3), 215–221. [https://doi.org/10.1016/S1270-9638\(00\)00130-9](https://doi.org/10.1016/S1270-9638(00)00130-9)
- Zhao, Q., Guo, J., Liu, S., Tao, J., Hu, Z., & Chen, G. (2021). A variant of raw observation approach for BDS/GNSS precise point positioning with fast integer ambiguity resolution. *Satellite Navigation*, 2, 2-22. <https://doi.org/10.1186/s43020-021-00059-7>

Appendices

Appendix A

Julian Day Calendar

This appendix consists of the Julian date calendars in tables 17 and 18. The Julian date calendars are a tool that is used as a first step to convert the time in the TLE to Date Time.

Table 17: Julian Date Calendar

Day	JAN.	FEB.	MAR.	APR.	MAY	JUN.	JUL.	AUG.	SEP.	OCT.	NOV.	DEC.
1	1	32	60	91	121	152	182	213	244	274	305	335
2	2	33	61	92	122	153	183	214	245	275	306	336
3	3	34	62	93	123	154	184	215	246	276	307	337
4	4	35	63	94	124	155	185	216	247	277	308	338
5	5	36	64	95	125	156	186	217	248	278	309	339
6	6	37	65	96	126	157	187	218	249	279	310	340
7	7	38	66	97	127	158	188	219	250	280	311	341
8	8	39	67	98	128	159	189	220	251	281	312	342
9	9	40	68	99	129	160	190	221	252	282	313	343
10	10	41	69	100	130	161	191	222	253	283	314	344
11	11	42	70	101	131	162	192	223	254	284	315	345
12	12	43	71	102	132	163	193	224	255	285	316	346
13	13	44	72	103	133	164	194	225	256	286	317	347
14	14	45	73	104	134	165	195	226	257	287	318	348
15	15	46	74	105	135	166	196	227	258	288	319	349
16	16	47	75	106	136	167	197	228	259	289	320	350
17	17	48	76	107	137	168	198	229	260	290	321	351
18	18	49	77	108	138	169	199	230	261	291	322	352

Table 17: Julian Date Calendar (continued)

Day	JAN.	FEB.	MAR.	APR.	MAY	JUN.	JUL.	AUG.	SEP.	OCT.	NOV.	DEC.
19	19	50	78	109	139	170	200	231	262	292	323	353
20	20	51	79	110	140	171	201	232	263	293	324	354
21	21	52	80	111	141	172	202	233	264	294	325	355
22	22	53	81	112	142	173	203	234	265	295	326	356
23	23	54	82	113	143	174	204	235	266	296	327	357
24	24	55	83	114	144	175	205	236	267	297	328	358
25	25	56	84	115	145	176	206	237	268	298	329	359
26	26	57	85	116	146	177	207	238	269	299	330	360
27	27	58	86	117	147	178	208	239	270	300	331	361
28	28	59	87	118	148	179	209	240	271	301	332	362
29	29		88	119	149	180	210	241	272	302	333	363
30	30		89	120	150	181	211	242	273	303	334	364
31	31		90		151		212	243		304		365

Table 18: Julian Date Calendar (For leap year only)

Day	JAN.	FEB.	MAR.	APR.	MAY	JUN.	JUL.	AUG.	SEP.	OCT.	NOV.	DEC.
1	1	32	61	92	122	153	183	214	245	275	306	336
2	2	33	62	93	123	154	184	215	246	276	307	337
3	3	34	63	94	124	155	185	216	247	277	308	338
4	4	35	64	95	125	156	186	217	248	278	309	339
5	5	36	65	96	126	157	187	218	249	279	310	340
6	6	37	66	97	127	158	188	219	250	280	311	341
7	7	38	67	98	128	159	189	220	251	281	312	342
8	8	39	68	99	129	160	190	221	252	282	313	343
9	9	40	69	100	130	161	191	222	253	283	314	344
10	10	41	70	101	131	162	192	223	254	284	315	345
11	11	42	71	102	132	163	193	224	255	285	316	346

Table 18: Julian Date Calendar (For leap year only) (continued)

Day	JAN.	FEB.	MAR.	APR.	MAY	JUN.	JUL.	AUG.	SEP.	OCT.	NOV.	DEC.
12	12	43	72	103	133	164	194	225	256	286	317	347
13	13	44	73	104	134	165	195	226	257	287	318	348
14	14	45	74	105	135	166	196	227	258	288	319	349
15	15	46	75	106	136	167	197	228	259	289	320	350
16	16	47	76	107	137	168	198	229	260	290	321	351
17	17	48	77	108	138	169	199	230	261	291	322	352
18	18	49	78	109	139	170	200	231	262	292	323	353
19	19	50	79	110	140	171	201	232	263	293	324	354
20	20	51	80	111	141	172	202	233	264	294	325	355
21	21	52	81	112	142	173	203	234	265	295	326	356
22	22	53	82	113	143	174	204	235	266	296	327	357
23	23	54	83	114	144	175	205	236	267	297	328	358
24	24	55	84	115	145	176	206	237	268	298	329	359
25	25	56	85	116	146	177	207	238	269	299	330	360
26	26	57	86	117	147	178	208	239	270	300	331	361
27	27	58	87	118	148	179	209	240	271	301	332	362
28	28	59	88	119	149	180	210	241	272	302	333	363
29	29	60	89	120	150	181	211	242	273	303	334	364
30	30		90	121	151	182	212	243	274	304	335	365
31	31		91		152		213	244		305		366

Appendix B

Atmosphere Density

This appendix consists of the atmospheric density at different altitudes. The atmospheric density decreases with increasing altitude, the value of the density is presented in the table below:

Table 19: Nominal Density with the Respect of Altitude

Altitude measured from Earth Surface [km]	Base Altitude [km]	Nominal Density [kg/m ³]	Scale Height at base altitude [km]
0 – 25	0	1.23	7.25
25 – 30	25	3.899×10^{-2}	6.35
30 – 40	30	1.774×10^{-2}	6.68
40 – 50	40	3.972×10^{-3}	7.55
50 – 60	50	1.057×10^{-3}	8.38
60 – 70	60	3.206×10^{-4}	7.71
70 – 80	70	8.770×10^{-5}	6.55
80 – 90	80	1.905×10^{-5}	5.80
90 – 100	90	3.396×10^{-6}	5.38
100 – 110	100	5.297×10^{-7}	5.88
110 – 120	110	9.661×10^{-8}	7.26
120 – 130	120	2.438×10^{-8}	9.47
130 – 140	130	8.484×10^{-9}	12.64

Table 19: Nominal Density with the Respect of Altitude (Continued)

Altitude measured from Earth Surface [km]	Base Altitude [km]	Nominal Density [kg/m ³]	Scale Height at base altitude [km]
140 – 150	140	3.845×10^{-9}	16.15
150 – 180	150	2.070×10^{-9}	22.52
180 – 200	180	5.464×10^{-10}	29.74
200 – 250	200	2.789×10^{-10}	37.11
250 – 300	250	7.248×10^{-11}	45.55
300 – 350	300	2.418×10^{-11}	53.63
350 – 400	350	9.518×10^{-12}	53.30
400 – 450	400	3.725×10^{-12}	58.52
450 – 500	450	1.585×10^{-12}	60.83
500 – 600	500	6.967×10^{-13}	63.82
600 – 700	600	1.454×10^{-13}	71.84
700 – 800	700	3.614×10^{-14}	88.67
800 – 900	800	1.170×10^{-14}	124.64
900 – 1000	900	5.245×10^{-15}	181.05
1000	1000	3.019×10^{-15}	268.00

Note. From Aghav, S., & Gangal, S. (2014). Simplified Orbit Determination Algorithm for Low Earth Orbit Satellites Using Spaceborne GPS Navigation Sensor. *Artificial Satellites*, 49.
<https://doi.org/10.2478/arsa-2014-0007>

Appendix C

EKF Python Code

This appendix consists of the Python code of the EKF.

```
from mpl_toolkits import mplot3d
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pymap3d as pm
import sys
import math
import pylab
import cv2
import xlrtd
import openpyxl
import datetime
from numpy.linalg import inv
from openpyxl import load_workbook

#-----Plot Parameters-----
x = [];
y = [];
z = [];
vx = [];
vy = [];
vz = [];

errx=[];
erry=[];
errz=[];
errvx=[];
errvy=[];
errvz=[];

px = [];
py = [];
pz = [];
pvx = [];
pvy = [];
pvz = [];
ptime = [];

Semi = [];
Ecc = [];
Incli = [];
LoTAN = [];
AoP = [];
TRA = [];

ex = [];
ey = [];
ez = [];
evx = [];
```

```

evy = [];
evz = [];
timer = [];

pex = [];
pey = [];
pez = [];
pevx = [];
pevy = [];
pevz = [];

#-----General Parameters-----
I = np.diag([1,1,1,1,1])
meo = float(3.986004418 *10**(14)) #m3/s-2
Altitude = 500000 #m
Period = 86400#s
Time = 10 #s
SD_PV = 10 #m
SD_VV = 0.01 #m/s
c = float (299792458) #m/s
J2 = 0.001082
Re = float(6371000) #m
A_X = 0.01 #m^2
A_Y = 0.03 #m^2
A_Z = 0.03 #m^2
Cd = 2.2
rho = float(((6.967*10**(-13))*(np.exp(-(500000-500000)/63830))))
AVE = float (7.27*10**(-5))
mass = 3#kg
f = 1
#-----GPS Data-----

GPS_DATA = load_workbook(r'C:\Users\pinkw\OneDrive\Desktop\GPS.xlsx', data_only=True)
GPS_DATA_sheet1 = GPS_DATA['Sheet2']
all_rows = list(GPS_DATA_sheet1.rows)
ws=GPS_DATA.active
column_j = ws['A']

PVi_0 = float((ws.cell(101,2).value)*1000)
PVj_0 = float((ws.cell(101,3).value)*1000)
PVk_0 = float((ws.cell(101,4).value)*1000)
VVi_0 = float((ws.cell(101,5).value)*1000)
VVj_0 = float((ws.cell(101,6).value)*1000)
VVk_0 = float((ws.cell(101,7).value)*1000)
time_0 = int(ws.cell(101,1).value)

x.append(PVi_0)
y.append(PVj_0)
z.append(PVk_0)
vx.append(VVi_0)
vy.append(VVj_0)
vz.append(VVk_0)

time0 = datetime.datetime.fromtimestamp(time_0)

PVi_1 = float((ws.cell(102,2).value)*1000)
PVj_1 = float((ws.cell(102,3).value)*1000)

```

```

PVk_1 = float((ws.cell(102,4).value)*1000)
VVi_1 = float((ws.cell(102,5).value)*1000)
VVj_1 = float((ws.cell(102,6).value)*1000)
VVk_1 = float((ws.cell(102,7).value)*1000)
time_1 = time_0+10#(ws.cell(3,1).value)

x.append(PVi_1)
y.append(PVj_1)
z.append(PVk_1)
vx.append(VVi_1)
vy.append(VVj_1)
vz.append(VVk_1)

```

```
time1 = datetime.datetime.fromtimestamp( time_1 )
```

```

PVi_2 = float((ws.cell(103,2).value)*1000)
PVj_2 = float((ws.cell(103,3).value)*1000)
PVk_2 = float((ws.cell(103,4).value)*1000)
VVi_2 = float((ws.cell(103,5).value)*1000)
VVj_2 = float((ws.cell(103,6).value)*1000)
VVk_2 = float((ws.cell(103,7).value)*1000)
time_2 = time_1+10#(ws.cell(4,1).value)

```

```

x.append(PVi_2)
y.append(PVj_2)
z.append(PVk_2)
vx.append(VVi_2)
vy.append(VVj_2)
vz.append(VVk_2)

```

```
time2 = datetime.datetime.fromtimestamp( time_2 )
```

```

PVi_3 = float((ws.cell(104,2).value)*1000)
PVj_3 = float((ws.cell(104,3).value)*1000)
PVk_3 = float((ws.cell(104,4).value)*1000)
VVi_3 = float((ws.cell(104,5).value)*1000)
VVj_3 = float((ws.cell(104,6).value)*1000)
VVk_3 = float((ws.cell(104,7).value)*1000)
time_3 = time_2+10#(ws.cell(5,1).value)
time_epoch = time_3

```

```

x.append(PVi_3)
y.append(PVj_3)
z.append(PVk_3)
vx.append(VVi_3)
vy.append(VVj_3)
vz.append(VVk_3)

```

```
time3 = datetime.datetime.fromtimestamp( time_3 )
```

```
#-----Orbital Parameters Function-----
```

```
def orbital_parameters( PVi,PVj,PVk,VVi,VVj,VVk):
```

```
    ''' Position vector magnitude '''
```

```
    PV = np.array([PVi,PVj, PVk])
```

```
    magnitude_PV = np.linalg.norm(PV)
```



```

''' Velocity vector magnitude '''
VV = np.array([VVi,VVj, VVk])
magnitude_VV = np.linalg.norm(VV)

''' PV.VV ( dot product ) '''
dotproduct_PV_VV = (PVi*VVi)+(PVj*VVj) + (PVk *VVk)

''' PVxVV ( cross product )'''
cPV_VVi = PVj*VVk - PVk*VVj
cPV_VVj = PVk*VVi - PVi*VVk
cPV_VVk = PVi*VVj - PVj*VVi
Crossproduct_PV_VV = [cPV_VVi ,cPV_VVj ,cPV_VVk]
''' Specific Angular momentum (h)'''
h = Crossproduct_PV_VV
''' VXh'''
cVhi = float(VVj*h[2] - VVk* h[1])
cVhj = float(VVk*h[0] - VVi* h[2])
cVhk = float(VVi*h[1] - VVj* h[0])

Crossproduct_VV_h = [cVhi ,cVhj ,cVhk]

''' Specific mechanical energy (ME) '''
ME = ((magnitude_VV ** 2)*0.5)-((meo)/magnitude_PV)
if ME > 0:
    ''' Semi-major axis (a) '''
    a = (meo/(2*ME))
    Semi.append(a)
    print ('Semi-major axis :', a , 'm')
elif ME < 0:
    ''' Semi-major axis (a) '''
    a = - (meo/(2*ME))
    Semi.append(a)
    print ('Semi-major axis :', a , 'm')

''' Eccentricity vector (eV) '''
s1 = ((1/meo)*Crossproduct_VV_h[0])
s2 = ((1/meo)*Crossproduct_VV_h[1])
s3 = ((1/meo)*Crossproduct_VV_h[2])
eV1 = [s1,s2,s3]
eV =eV1-(PV/magnitude_PV)
''' Eccentricity (e) '''
e = np.absolute(np.linalg.norm(eV))
Ecc.append(e)
print ('Eccentricity :',e)

''' inclination (i) '''
i= math.degrees (math.acos (cPV_VVk/np.linalg.norm(h)))
Incli.append(i)
print ('inclinationum :',i,'degree')

''' ascending node vector (ANV) = k(0,0,1)xh(vector) '''
K = [0,0,1]
chKi = K[1]*cPV_VVk - K[2]*cPV_VVj
chKj = K[2]*cPV_VVi - K[0]*cPV_VVk
chKk = K[0]*cPV_VVj - K[1]*cPV_VVi
ANV = [chKi,chKj,chKk]

```

```

AN = np.linalg.norm(ANV)

''' RAAN '''
RAAN= math.degrees (math.acos (chKi/AN))
if chKj >= 0 :
    RAAN = RAAN
    LoTAN.append(RAAN)
    print ('RAAN :',RAAN,'degree')
elif chKj < 0 :
    RAAN = 360 - RAAN
    LoTAN.append(RAAN)
    print ('RAAN :',RAAN,'degree')

''' Argument of Perigee (AG) '''
dotproduct_ANV_eV = ((chKi*eV[0]) + (chKj*eV[1]) + (chKk*eV[2]))
AG = math.degrees (math.acos (dotproduct_ANV_eV/(AN*e)))
if eV[2] >= 0 :
    AG = AG
    AoP.append(AG)
    print ('Argument of Perigee :',AG,'degree')
elif eV[2] < 0 :
    AG = 360 - AG
    AoP.append(AG)
    print ('Argument of Perigee :',AG,'degree')

''' True Anomaly (TN) '''
dotproduct_eV_PV = (PVi*eV[0]) + (PVj*eV[1]) + (PVk*eV[2])
ta = dotproduct_eV_PV/(magnitude_PV*e)
if ta <= -1 or ta >=1:
    ta = math.trunc(dotproduct_eV_PV/(magnitude_PV*e))
TA = np.degrees (np.arccos ((ta)))

if dotproduct_PV_VV > 0 :
    TA = TA
    TRA.append(TA)
    print ('True Anomaly :',TA,'degree')
elif dotproduct_PV_VV < 0 :
    TA = 360 - TA
    TRA.append(TA)
    print ('True Anomaly :',TA,'degree')

''' mean motion (n) '''
n0 = 2*math.pi*math.sqrt(((magnitude_PV)**3)/(3.985*10**5))
n = 86400/n0
print ('mean motion',n,'rev/day')
print ('mean motion',(n*7.27221*10**(-5)), 'rad/s')

return (a,e,i,RAAN,AG,TA,n)

```

#-----Extended Kalman Filter -----

''' first time'''

''' Estimated Process Error Covariance '''

cov_m = np.stack((x,y,z,vx,vy,vz), axis=0)

cov_f = np.cov(cov_m)

Q = cov_f

''' Convergence Matrix '''

```
Pkk_1 = np.array([[SD_PV**(2),0,0,0,0],
                  [0,SD_PV**(2),0,0,0],
                  [0,0,SD_PV**(2),0,0],
                  [0,0,0,SD_VV**(2),0],
                  [0,0,0,0,SD_VV**(2),0],
                  [0,0,0,0,SD_VV**(2)]]])
```

```
time_diff1 = np.absolute(np.subtract(time_3 ,time_0))#s
time_diff2 = np.absolute(np.subtract(time_3 , time_1))#s
time_diff3 = np.absolute(np.subtract(time_3 , time_2))#s
```

while Time <= Period:

#-----Extended Kalman Filter Parametera-----

#print ('-----')

''' The initial State vector '''

```
X = np.array([[PVi_3],
              [PVj_3],
              [PVk_3],
              [VVi_3],
              [VVj_3],
              [VVk_3]])
```

''' Position vector magnitude '''

```
PV_3 = np.array([PVi_3,PVj_3, PVk_3])
magnitude_PV_3 = float(np.linalg.norm(PV_3))
```

''' Velocity vector magnitude '''

```
VV_3 = np.array([VVi_3,VVj_3, VVk_3])
magnitude_VV_3 = float( np.linalg.norm(VV_3))
```

''' The State Transition Matrix (J2) '''

```
r3 = float (np.power(magnitude_PV_3,3))
r5 = float (np.power(magnitude_PV_3,5))
r7 = float (np.power(magnitude_PV_3,7))
r9 = float (np.power(magnitude_PV_3,9))
Re2 = float (np.power(Re,2))
PV3 = float (np.power(PVk_3,3))
PV4 = float (np.power(PVk_3,4))
```

```
A41 = ((3*meo*np.power(PVi_3,2))/r5)-(meo/r3)+((15*meo*J2*Re2*(np.power(PVk_3,2)))/(2*r7))-
((3*meo*J2*Re2)/(2*r5))+((15*meo*J2*Re2*(np.power(PVi_3,2)))/(2*r7))-
((105*meo*J2*Re2*np.power(PVk_3,2)*np.power(PVi_3,2))/(2*r9))
```

```
A51 = ((3*meo*PVi_3*PVj_3)/r5)+((15*meo*J2*Re2*PVi_3*PVj_3)/(2*r7))-
((105*meo*J2*Re2*np.power(PVk_3,2)*PVi_3*PVj_3)/(2*r9))
```

A61 =

```
((30*meo*J2*Re2*PVk_3*PVi_3)/(2*r7))+((3*meo*PVi_3*PVk_3)/r5)+((15*meo*J2*Re2*np.power(PVi_3,2))/(
2*r7))-((105*meo*J2*Re2*np.power(PVk_3,3)*PVi_3)/(2*r9))
```

```
A42 = ((3*meo*PVi_3*PVj_3)/r5)+((15*meo*J2*Re2*PVi_3*PVj_3)/(2*r7))-
((105*meo*J2*Re2*np.power(PVk_3,2)*PVi_3*PVj_3)/(2*r9))
```

```
A52 = ((3*meo*np.power(PVj_3,2))/r5)-(meo/r3)+((15*meo*J2*Re2*np.power(PVk_3,2))/(2*r7))-
((3*meo*J2*Re2)/(2*r5))+((15*meo*J2*Re2*np.power(PVj_3,2))/(2*r7))-
((105*meo*J2*Re2*np.power(PVk_3,2)*np.power(PVj_3,2))/(2*r9))
```

A62 =

```
((30*meo*J2*Re2*PVk_3*PVj_3)/(2*r7))+((3*meo*PVj_3*PVk_3)/r5)+((15*meo*J2*Re2*np.power(PVj_3,2))/(
2*r7))-((105*meo*J2*Re2*np.power(PVk_3,3)*PVj_3)/(2*r9))
```

```
A43 = ((3*meo*PVi_3*PVk_3)/r5)+((45*meo*J2*Re2*PVi_3*PVk_3)/(2*r7))-
((105*meo*J2*Re2*PV3*PVi_3)/(2*r9))
```

```
A53 = ((3*meo*PVj_3*PVk_3)/r5)+((45*meo*J2*Re2*PVj_3*PVk_3)/(2*r7))-
((105*meo*J2*Re2*PV3*PVj_3)/(2*r9))
```

```

A63 = (-meo/r3)-
((9*J2*meo*Re2)/(2*r5))+((30*meo*J2*Re2)*PVk_3/(2*r7))+((3*meo*np.power(PVk_3,2))/r5)+((45*meo*J2*Re2*np.power(PVk_3,2))/(2*r7))-((105*J2*meo*Re2*PV4)/(2*r9))

```

```

AS_1 = np.array([[0,0,0,10,0,0],
                 [0,0,0,10,0],
                 [0,0,0,0,10],
                 [A41,A42,A43,0,0,0],
                 [A51,A52,A53,0,0,0],
                 [A61,A62,A63,0,0,0]])

```

```

AS_2 = I + AS_1
A = np.linalg.inv(AS_2)
A_T = np.transpose(A)

```

''' Estimated Measurement Error Covariance'''

```

R = np.array([[SD_PV**(2),0,0,0,0,0],
              [0,SD_PV**(2),0,0,0,0],
              [0,0,SD_PV**(2),0,0,0],
              [0,0,0,SD_VV**(2),0,0],
              [0,0,0,0,SD_VV**(2),0],
              [0,0,0,0,0,SD_VV**(2)]])

```

''' Observation matrix '''

```

HK_11 = PVi_3/(magnitude_PV_3)
HK_12 = PVj_3/(magnitude_PV_3)
HK_13 = PVk_3/(magnitude_PV_3)
HK_21 = PVj_3/((PVj_3**(2)+(PVi_3**(2)))
HK_22 = PVi_3/((PVj_3**(2)+(PVi_3**(2)))
HK_31 = (2*PVi_3*PVk_3)/(((PVj_3**(2)+(PVi_3**(2)))*magnitude_PV_3)
HK_32 = (2*PVj_3*PVk_3)/(((PVj_3**(2)+(PVi_3**(2)))*magnitude_PV_3)
HK_33 = (((PVi_3**(2)+(PVj_3**(2)+(PVk_3**(2)))-
(2*PVk_3**(2)))/(((PVi_3**(2)+(PVj_3**(2)+(PVk_3**(2)))*(2))*((PVj_3**(2)+(PVi_3**(2))))))

```

```

HK = np.array([[HK_11,HK_12,HK_13,0,0,0],
               [HK_21,HK_22,0,0,0,0],
               [HK_31,HK_32,HK_33,0,0,0],
               [0,0,0,0,0,0],
               [0,0,0,0,0,0],
               [0,0,0,0,0,0]])

```

```

HK_T = np.transpose(HK)

```

#----- Apply EKF first -----

''' New Estimate'''

```

Xt = np.matmul(A,X)
Pk_s1 = np.matmul(A,Pkk_1)
Pk_s2 = np.matmul(Pk_s1,A_T)
Pk = Pk_s2+Q
HXt = np.matmul(HK,X)

```

```

Vrel_x = VVi_3 + AVE*PVj_3
Vrel_y = VVj_3 - AVE*PVi_3

```

```

Vrel_z = VVk_3
Vrel = np.array([Vrel_x,Vrel_y, Vrel_z])
magnitude_Vrel = np.linalg.norm(Vrel)

Z =[[0],
     [0],
     [0],
     [(-0.5)*((Cd*A_X)/(mass))*rho*(np.power(Vrel_x ,2))*(Vrel_x /magnitude_Vrel)],
     [(-0.5)*((Cd*A_Y)/(mass))*rho*(np.power(Vrel_y ,2))*(Vrel_y /magnitude_Vrel)],
     [(-0.5)*((Cd*A_Z)/(mass))*rho*(np.power(Vrel_z ,2))*(Vrel_z /magnitude_Vrel)]]

RPK = np.subtract(R,Pkk_1)
# K = ((Pk)*HK_T)*(HK*Pk*HK_T + R)**(-1)
K_S1 = np.matmul(HK,Pk)
K_S2 = np.matmul(K_S1,HK_T)
K_S3 = np.add(K_S2, R)
K_S4 = np.linalg.inv(K_S3)
K_S5 = np.matmul( Pk,HK_T)
K = np.matmul(K_S5,K_S4)
K_T = np.transpose(K)
#new estimation
Y = Xt + np.matmul(K,np.subtract(Z,HXt))
Pkk_1 = np.matmul(I-(np.matmul(K,HK)),Pk)
print ('x: ',Y[0,0],'(m)')
print ('y: ',Y[1,0],'(m)')
print ('z: ',Y[2,0],'(m)')
print ('vx: ',Y[3,0],'(m/s)')
print ('vy: ',Y[4,0],'(m/s)')
print ('vz: ',Y[5,0],'(m/s)')
Position = np.sqrt((np.power(Y[0,0],2)+ np.power(Y[1,0],2)+ np.power(Y[2,0],2)))
Velocity = np.sqrt((np.power(Y[3,0],2)+ np.power(Y[4,0],2)+ np.power(Y[5,0],2)))
print ('R: ', Position, 'm')
print ('V: ', Velocity, 'm/s')
orbital_parameters(Y[0,0],Y[1,0],Y[2,0],Y[3,0],Y[4,0],Y[5,0]);

time_diff20 = time_diff2
time_diff2 = time_diff1
time_diff3 = time_diff20
time_diff1 = 10 #s
PVi_10 = PVi_1
PVj_10 = PVj_1
PVk_10 = PVk_1
VVi_10 = VVi_1
VVj_10 = VVj_1
VVk_10 = VVk_1
PVi_20 = PVi_2
PVj_20 = PVj_2
PVk_20 = PVk_2
VVi_20 = VVi_2
VVj_20 = VVj_2
VVk_20 = VVk_2
PVi_30 = PVi_3
PVj_30 = PVj_3
PVk_30 = PVk_3
VVi_30 = VVi_3
VVj_30 = VVj_3
VVk_30 = VVk_3

```

```

PVi_0 = PVi_10
PVj_0 = PVj_10
PVk_0 = PVk_10
VVi_0 = VVi_10
VVj_0 = VVj_10
VVk_0 = VVk_10
PVi_1 = PVi_20
PVj_1 = PVj_20
PVk_1 = PVk_20
VVi_1 = VVi_20
VVj_1 = VVj_20
VVk_1 = VVk_20
PVi_2 = PVi_30
PVj_2 = PVj_30
PVk_2 = PVk_30
VVi_2 = VVi_30
VVj_2 = VVj_30
VVk_2 = VVk_30
PVi_3 = Y[0,0]
PVj_3 = Y[1,0]
PVk_3 = Y[2,0]
VVi_3 = Y[3,0]
VVj_3 = Y[4,0]
VVk_3 = Y[5,0]
x.append(PVi_3)
y.append(PVj_3)
z.append(PVk_3)
vx.append(VVi_3)
vy.append(VVj_3)
vz.append(VVk_3)
px.append(PVi_3)
py.append(PVj_3)
pz.append(PVk_3)
pvx.append(VVi_3)
pvy.append(VVj_3)
pvz.append(VVk_3)

cov_m = np.stack((x,y,z,vx,vy,vz), axis=0)
cov_f = np.cov(cov_m)
Q = cov_f

```

''' for the plot we need to get the parameter from the excel file and compare it with the estimated values '''

for cell **in** column_j:

```

if cell.value == time_epoch:
    rowss = [cell.coordinate[1],cell.coordinate[2],cell.coordinate[3]]
    rowV=int("".join(map(str, rowss)))
    PVi_r = float((ws.cell(rowV+1,2).value)*1000)
    PVj_r = float((ws.cell(rowV+1,3).value)*1000)
    PVk_r = float((ws.cell(rowV+1,4).value)*1000)
    VVi_r = float((ws.cell(rowV+1,5).value)*1000)
    VVj_r = float((ws.cell(rowV+1,6).value)*1000)
    VVk_r = float((ws.cell(rowV+1,7).value)*1000)
    time_r = float(ws.cell(rowV+1,1).value)
    ex.append(PVi_r)
    ey.append(PVj_r)
    ez.append(PVk_r)

```

```

evx.append(VVi_r)
evy.append(VVj_r)
evz.append(VVk_r)
timer.append(time_r)

errorx = errx.append((np.power(PVi_r - Y[0,0],2)))
errorx1 = np.sqrt(np.sum(errx)/f)
pex1 = pex.append(errorx1)

error = erry.append((np.power(PVj_r - Y[1,0],2)))
error1 = np.sqrt(np.sum(erry)/f)
pey1 = pey.append(error1)

errorz = errz.append((np.power(PVk_r - Y[2,0],2)))
errorz1 = np.sqrt(np.sum(errz)/f)
pez1 = pez.append(errorz1)

errorvx = errvx.append((np.power(VVi_r - Y[3,0],2)))
errorvx1 = np.sqrt(np.sum(errvx)/f)
pevx1 = pevx.append(errorvx1)

errorvy = errvy.append((np.power(VVj_r - Y[4,0],2)))
errorvy1 = np.sqrt(np.sum(errvy)/f)
pevy1 = pevy.append(errorvy1)

errorvz = errvz.append((np.power(VVk_r - Y[5,0],2)))
errorvz1 = np.sqrt(np.sum(errvz)/f)
pevz1 = pevz.append(errorvz1)

```

```
f = f+1
```

```

time_3 = time_3 + 10
time_epoch = time_3
ptime.append(time_epoch)

```

```

time0 = time1
time1 = time2
time2 = time3
time3 = datetime.datetime.fromtimestamp( time_3 )

```

```

print ('GPS EPOCH',time_epoch )
Time = Time +10

```

```
'''RMS PLOT'''
```

```
''' Save to excel file'''
```

```

col1 = "X"
col2 = "Y"
col3 = "Z"
col4 = "VX"
col5 = "VY"
col6 = "VZ"
data = pd.DataFrame({col1:px,col2:py,col3:pz,col4:pvx,col5:pyy,col6:pvz})

```

```
data.to_excel('EKF_RESULT1.xlsx', sheet_name='sheet1', index=False)
```

```
print('DOONE')
```


Appendix D

UKF Python Code

This appendix consists of the Python code of the UKF.

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import pymap3d as pm
import sys
import math
import pylab
import cv2
import xlrd
import openpyxl
import datetime
import xlswriter
from numpy.linalg import inv
from openpyxl import load_workbook

#-----Plot Parameters-----
Xk11 = [];
Xk12 = [];
Xk13 = [];
Xk14 = [];
Xk15 = [];
Xk16 = [];

Xk21 = [];
Xk22 = [];
Xk23 = [];
Xk24 = [];
Xk25 = [];
Xk26 = [];

Xk31 = [];
Xk32 = [];
Xk33 = [];
Xk34 = [];
Xk35 = [];
Xk36 = [];

Xk41 = [];
Xk42 = [];
Xk43 = [];
Xk44 = [];
Xk45 = [];
Xk46 = [];

Xk51 = [];
Xk52 = [];
Xk53 = [];
Xk54 = [];
Xk55 = [];
```

Xk56 = [];

Xk61 = [];

Xk62 = [];

Xk63 = [];

Xk64 = [];

Xk65 = [];

Xk66 = [];

Pvv11 = [];

Pvv12 = [];

Pvv13 = [];

Pvv14 = [];

Pvv15 = [];

Pvv16 = [];

Pvv21 = [];

Pvv22 = [];

Pvv23 = [];

Pvv24 = [];

Pvv25 = [];

Pvv26 = [];

Pvv31 = [];

Pvv32 = [];

Pvv33 = [];

Pvv34 = [];

Pvv35 = [];

Pvv36 = [];

Pvv41 = [];

Pvv42 = [];

Pvv43 = [];

Pvv44 = [];

Pvv45 = [];

Pvv46 = [];

Pvv51 = [];

Pvv52 = [];

Pvv53 = [];

Pvv54 = [];

Pvv55 = [];

Pvv56 = [];

Pvv61 = [];

Pvv62 = [];

Pvv63 = [];

Pvv64 = [];

Pvv65 = [];

Pvv66 = [];

Yk11 = [];

Yk12 = [];

Yk13 = [];

Yk14 = [];

Yk15 = [];

Yk16 = [];

Yk21 = [];
Yk22 = [];
Yk23 = [];
Yk24 = [];
Yk25 = [];
Yk26 = [];

Yk31 = [];
Yk32 = [];
Yk33 = [];
Yk34 = [];
Yk35 = [];
Yk36 = [];

Yk41 = [];
Yk42 = [];
Yk43 = [];
Yk44 = [];
Yk45 = [];
Yk46 = [];

Yk51 = [];
Yk52 = [];
Yk53 = [];
Yk54 = [];
Yk55 = [];
Yk56 = [];

Yk61 = [];
Yk62 = [];
Yk63 = [];
Yk64 = [];
Yk65 = [];
Yk66 = [];

Pxy11 = [];
Pxy12 = [];
Pxy13 = [];
Pxy14 = [];
Pxy15 = [];
Pxy16 = [];

Pxy21 = [];
Pxy22 = [];
Pxy23 = [];
Pxy24 = [];
Pxy25 = [];
Pxy26 = [];

Pxy31 = [];
Pxy32 = [];
Pxy33 = [];
Pxy34 = [];
Pxy35 = [];
Pxy36 = [];

Pxy41 = [];
Pxy42 = [];
Pxy43 = [];
Pxy44 = [];
Pxy45 = [];
Pxy46 = [];

Pxy51 = [];
Pxy52 = [];
Pxy53 = [];
Pxy54 = [];
Pxy55 = [];
Pxy56 = [];

Pxy61 = [];
Pxy62 = [];
Pxy63 = [];
Pxy64 = [];
Pxy65 = [];
Pxy66 = [];

X11 = [];
X12 = [];
X13 = [];
X14 = [];
X15 = [];
X16 = [];

X21 = [];
X22 = [];
X23 = [];
X24 = [];
X25 = [];
X26 = [];

X31 = [];
X32 = [];
X33 = [];
X34 = [];
X35 = [];
X36 = [];

X41 = [];
X42 = [];
X43 = [];
X44 = [];
X45 = [];
X46 = [];

X51 = [];
X52 = [];
X53 = [];
X54 = [];
X55 = [];
X56 = [];

X61 = [];
X62 = [];
X63 = [];
X64 = [];
X65 = [];
X66 = [];

Pk11 = [];
Pk12 = [];
Pk13 = [];
Pk14 = [];
Pk15 = [];
Pk16 = [];

Pk21 = [];
Pk22 = [];
Pk23 = [];
Pk24 = [];
Pk25 = [];
Pk26 = [];

Pk31 = [];
Pk32 = [];
Pk33 = [];
Pk34 = [];
Pk35 = [];
Pk36 = [];

Pk41 = [];
Pk42 = [];
Pk43 = [];
Pk44 = [];
Pk45 = [];
Pk46 = [];

Pk51 = [];
Pk52 = [];
Pk53 = [];
Pk54 = [];
Pk55 = [];
Pk56 = [];

Pk61 = [];
Pk62 = [];
Pk63 = [];
Pk64 = [];
Pk65 = [];
Pk66 = [];

x = [];
y = [];
z = [];
vx = [];
vy = [];
vz = [];

```
Semi = [];  
Ecc = [];  
Incli = [];  
LoTAN = [];  
AoP = [];  
TRA = [];
```

```
ex = [];  
ey = [];  
ez = [];  
evx = [];  
evy = [];  
evz = [];  
timer = [];
```

```
px = [];  
py = [];  
pz = [];  
pvx = [];  
pvy = [];  
pvz = [];  
ptime = [];
```

```
errx=[];  
erry=[];  
errz=[];  
errvx=[];  
errvy=[];  
errvz=[];
```

```
pex = [];  
pey = [];  
pez = [];  
pevx = [];  
pevy = [];  
pevz = [];
```

```
#-----General Parameters-----
```

```
II = np.diag([1,1,1,1,1])  
meo = float(3.986004418 *10**(14)) #m3/s-2  
Altitude = 500000 #m  
Period = 86400#s  
Time = 10 #s  
SD_PV = 10 #m  
SD_VV = 0.01 #m/s  
c = float (299792458) #m/s  
J2 = 0.001082  
Re = float(6371000) #m  
A_X = 0.02 #m^2  
A_Y = 0.06 #m^2  
A_Z = 0.03 #m^2  
Cd = 2.2  
rho = float((6.967*10**(-13))*(np.exp(-(500000-500000)/63830)))  
AVE = float (7.27*10**(-5))  
mass = 8#kg  
L = 0#sigme counter
```

```

zero_m = np.array([[0,0,0,0,0],
                  [0,0,0,0,0],
                  [0,0,0,0,0],
                  [0,0,0,0,0],
                  [0,0,0,0,0]])
''' Estimated Measurement Error Covariance'''
R = np.array([[SD_PV**(2),0,0,0,0],
              [0,SD_PV**(2),0,0,0],
              [0,0,SD_PV**(2),0,0],
              [0,0,0,SD_VV**(2),0],
              [0,0,0,0,SD_VV**(2),0],
              [0,0,0,0,SD_VV**(2)]]])
f = 1
#-----GPS Data-----

GPS_DATA = load_workbook(r'C:\Users\pinkw\OneDrive\Desktop\GPS.xlsx', data_only=True)
GPS_DATA_sheet1 = GPS_DATA[Sheet2]
all_rows = list(GPS_DATA_sheet1.rows)
ws=GPS_DATA.active
column_j = ws['A']
PVi_0 = float((ws.cell(102,2).value)*1000)
PVj_0 = float((ws.cell(102,3).value)*1000)
PVk_0 = float((ws.cell(102,4).value)*1000)
VVi_0 = float((ws.cell(102,5).value)*1000)
VVj_0 = float((ws.cell(102,6).value)*1000)
VVk_0 = float((ws.cell(102,7).value)*1000)
time_0 = (ws.cell(102,1).value)

x.append(PVi_0)
y.append(PVj_0)
z.append(PVk_0)
vx.append(VVi_0)
vy.append(VVj_0)
vz.append(VVk_0)

time0 = datetime.datetime.fromtimestamp(time_0)

PVi_1 = float((ws.cell(103,2).value)*1000)
PVj_1 = float((ws.cell(103,3).value)*1000)
PVk_1 = float((ws.cell(103,4).value)*1000)
VVi_1 = float((ws.cell(103,5).value)*1000)
VVj_1 = float((ws.cell(103,6).value)*1000)
VVk_1 = float((ws.cell(103,7).value)*1000)
time_1 = (ws.cell(103,1).value)

x.append(PVi_1)
y.append(PVj_1)
z.append(PVk_1)
vx.append(VVi_1)
vy.append(VVj_1)
vz.append(VVk_1)

time1 = datetime.datetime.fromtimestamp( time_1 )

```

```

PVi_2 = float((ws.cell(104,2).value)*1000)
PVj_2 = float((ws.cell(104,3).value)*1000)
PVk_2 = float((ws.cell(104,4).value)*1000)
VVi_2 = float((ws.cell(104,5).value)*1000)
VVj_2 = float((ws.cell(104,6).value)*1000)
VVk_2 = float((ws.cell(104,7).value)*1000)
time_2 = (ws.cell(104,1).value)

x.append(PVi_2)
y.append(PVj_2)
z.append(PVk_2)
vx.append(VVi_2)
vy.append(VVj_2)
vz.append(VVk_2)

time2 = datetime.datetime.fromtimestamp( time_2 )

```

```

PVi_3 = float((ws.cell(105,2).value)*1000)
PVj_3 = float((ws.cell(105,3).value)*1000)
PVk_3 = float((ws.cell(105,4).value)*1000)
VVi_3 = float((ws.cell(105,5).value)*1000)
VVj_3 = float((ws.cell(105,6).value)*1000)
VVk_3 = float((ws.cell(105,7).value)*1000)
time_3 = (ws.cell(105,1).value)
time_epoch = time_3

```

```

x.append(PVi_3)
y.append(PVj_3)
z.append(PVk_3)
vx.append(VVi_3)
vy.append(VVj_3)
vz.append(VVk_3)

```

```

time3 = datetime.datetime.fromtimestamp( time_3 )
time_epoch = time_3

```

#-----Orbital Parameters Function-----

```

def orbital_parameters( PVi,PVj,PVk,VVi,VVj,VVk):

    ''' Position vector magnitude '''
    PV = np.array([PVi,PVj, PVk])
    magnitude_PV = np.linalg.norm(PV)

    ''' Velocity vector magnitude '''
    VV = np.array([VVi,VVj, VVk])
    magnitude_VV = np.linalg.norm(VV)

    ''' PV.VV ( dot product ) '''
    dotproduct_PV_VV = (PVi*VVi) + (PVj*VVj) + (PVk *VVk)

    ''' PVxVV ( cross product )'''
    cPV_VVi = PVj*VVk - PVk*VVj
    cPV_VVj = PVk*VVi - PVi*VVk
    cPV_VVk = PVi*VVj - PVj*VVi
    Crossproduct_PV_VV = [cPV_VVi ,cPV_VVj ,cPV_VVk]
    ''' Specific Angular momentum (h)'''
    h = Crossproduct_PV_VV
    ''' VXh'''

```



```

cVhi = float(VVj*h[2] - VVk* h[1])
cVhj = float(VVk*h[0] - VVi* h[2])
cVhk = float(VVi*h[1] - VVj* h[0])

Crossproduct_VV_h = [cVhi ,cVhj ,cVhk]

''' Specific mechanical energy (ME) '''
ME = ((magnitude_VV ** 2)*0.5)-((meo)/magnitude_PV)
if ME > 0:
    ''' Semi-major axis (a) '''
    a = (meo/(2*ME))
    Semi.append(a)
    #print ('Semi-major axis :', a , 'm')
elif ME < 0:
    ''' Semi-major axis (a) '''
    a = - (meo/(2*ME))
    Semi.append(a)
    #print ('Semi-major axis :', a , 'm')

''' Eccentricity vector (eV) '''

s1 = ((1/meo)*Crossproduct_VV_h[0])
s2 = ((1/meo)*Crossproduct_VV_h[1])
s3 = ((1/meo)*Crossproduct_VV_h[2])
eV1 = [s1,s2,s3]
eV =eV1-(PV/magnitude_PV)
''' Eccentricity (e) '''
e = np.absolute(np.linalg.norm(eV))
Ecc.append(e)
#print ('Eccentricity :',e)

''' inclination (i) '''
i= math.degrees (math.acos (cPV_VVk/np.linalg.norm(h)))
Incli.append(i)
#print ('inclinationum :',i, 'degree')

''' ascending node vector (ANV) = k(0,0,1)xh(vector) '''
K = [0,0,1]
chKi = K[1]*cPV_VVk - K[2]*cPV_VVj
chKj = K[2]*cPV_VVi - K[0]*cPV_VVk
chKk = K[0]*cPV_VVj - K[1]*cPV_VVi
ANV = [chKi,chKj,chKk]
AN = np.linalg.norm(ANV)

''' RAAN '''
RAAN= math.degrees (math.acos (chKi/AN))
if chKj >= 0 :
    RAAN = RAAN
    LoTAN.append(RAAN)
    #print ('RAAN :',RAAN,'degree')
elif chKj < 0 :
    RAAN = 360 - RAAN
    LoTAN.append(RAAN)
    #print ('RAAN :',RAAN,'degree')

''' Argument of Perigee (AG) '''
dotproduct_ANV_eV = ((chKi*eV[0]) + (chKj*eV[1]) + (chKk*eV[2]))

```

```

AG = math.degrees (math.acos (dotproduct_ANV_eV/(AN*e)))
if eV[2] >= 0 :
    AG = AG
    AoP.append(AG)
    #print ('Argument of Perigee :',AG,'degree')
elif eV[2] < 0 :
    AG = 360 - AG
    AoP.append(AG)
    #print ('Argument of Perigee :',AG,'degree')

''' True Anomaly (TN) '''
dotproduct_eV_PV = (PVi*eV[0]) + (PVj*eV[1]) + (PVk*eV[2])
ta = dotproduct_eV_PV/(magnitudo_PV*e)
if ta <= -1 or ta >= 1:
    ta = math.trunc(dotproduct_eV_PV/(magnitudo_PV*e))
TA = np.degrees (np.arccos ((ta)))

if dotproduct_PV_VV > 0 :
    TA = TA
    TRA.append(TA)
    #print ('True Anomaly :',TA,'degree')
elif dotproduct_PV_VV < 0 :
    TA = 360 - TA
    TRA.append(TA)
    #print ('True Anomaly :',TA,'degree')

''' mean motion (n) '''
n0 = 2*math.pi*math.sqrt(((magnitudo_PV)**3)/(3.985*10**5))
n = 86400/n0
#print ('mean motion',n,'rev/day')
#print ('mean motion',(n*7.27221*10**(-5)), 'rad/s')

return (a,e,i,RAAN,AG,TA,n)

```

#-----UKF-----

```
def f_cv(PVi_3,PVj_3,PVk_3,VVi_3,VVj_3,VVk_3):
```

```
''' Position vector magnitude '''
```

```
PV_3 = np.array([PVi_3,PVj_3, PVk_3])
magnitudo_PV_3 = float(np.linalg.norm(PV_3))
```

```
''' Velocity vector magnitude '''
```

```
VV_3 = np.array([VVi_3,VVj_3, VVk_3])
magnitudo_VV_3 = float( np.linalg.norm(VV_3))
```

```
''' The State Transition Matrix '''
```

```
r3 = float (np.power(magnitudo_PV_3,3))
r5 = float (np.power(magnitudo_PV_3,5))
r7 = float (np.power(magnitudo_PV_3,7))
r9 = float (np.power(magnitudo_PV_3,9))
Re2 = float (np.power(Re,2))
PV2 = float (np.power(PVk_3,2))
PV3 = float (np.power(PVk_3,3))
PV4 = float (np.power(PVk_3,4))
```

```

A41 = ((3*meo*np.power(PVi_3,2))/r5)-(meo/r3)+((15*meo*J2*Re2*(np.power(PVk_3,2)))/(2*r7))-
((3*meo*J2*Re2)/(2*r5))+((15*meo*J2*Re2*(np.power(PVi_3,2)))/(2*r7))-
((105*meo*J2*Re2*np.power(PVk_3,2)*np.power(PVi_3,2))/(2*r9))
A51 = ((3*meo*PVi_3*PVj_3)/r5)+((15*meo*J2*Re2*PVi_3*PVj_3)/(2*r7))-
((105*meo*J2*Re2*np.power(PVk_3,2)*PVi_3*PVj_3)/(2*r9))
A61 =
((30*meo*J2*Re2*PVk_3*PVi_3)/(2*r7))+((3*meo*PVi_3*PVk_3)/r5)+((15*meo*J2*Re2*np.power(PVi_3,2))/(
2*r7))-((105*meo*J2*Re2*np.power(PVk_3,3)*PVi_3)/(2*r9))
A42 = ((3*meo*PVi_3*PVj_3)/r5)+((15*meo*J2*Re2*PVi_3*PVj_3)/(2*r7))-
((105*meo*J2*Re2*np.power(PVk_3,2)*PVi_3*PVj_3)/(2*r9))
A52 = ((3*meo*np.power(PVj_3,2))/r5)-(meo/r3)+((15*meo*J2*Re2*np.power(PVk_3,2))/(2*r7))-
((3*meo*J2*Re2)/(2*r5))+((15*meo*J2*Re2*np.power(PVj_3,2))/(2*r7))-
((105*meo*J2*Re2*np.power(PVk_3,2)*np.power(PVj_3,2))/(2*r9))
A62 =
((30*meo*J2*Re2*PVk_3*PVj_3)/(2*r7))+((3*meo*PVj_3*PVk_3)/r5)+((15*meo*J2*Re2*np.power(PVj_3,2))/(
2*r7))-((105*meo*J2*Re2*np.power(PVk_3,3)*PVj_3)/(2*r9))
A43 = ((3*meo*PVi_3*PVk_3)/r5)+((45*meo*J2*Re2*PVi_3*PVk_3)/(2*r7))-
((105*meo*J2*Re2*PV3*PVi_3)/(2*r9))
A53 = ((3*meo*PVj_3*PVk_3)/r5)+((45*meo*J2*Re2*PVj_3*PVk_3)/(2*r7))-
((105*meo*J2*Re2*PV3*PVj_3)/(2*r9))
A63 = (-meo/r3)-
((9*J2*meo*Re2)/(2*r5))+((30*meo*J2*Re2)*PVk_3/(2*r7))+((3*meo*np.power(PVk_3,2))/r5)+((45*meo*J2*R
e2*np.power(PVk_3,2))/(2*r7))-((105*J2*meo*Re2*PV4)/(2*r9))

```

```

AS_1 = np.array([[0,0,0,-10,0,0],
                 [0,0,0,0,-10,0],
                 [0,0,0,0,0,-10],
                 [A41,A51,A61,0,0,0],
                 [A42,A52,A62,0,0,0],
                 [A43,A53,A63,0,0,0]])

```

```

AS_2 = II + AS_1
A = np.transpose(AS_2)
A_T = np.transpose(A)

```

return A

```
def h_cv(PVi_3,PVj_3,PVk_3,VVi_3,VVj_3,VVk_3):
```

''' Position vector magnitude '''

```

PV_3 = np.array([PVi_3,PVj_3, PVk_3])
magnitude_PV_3 = float(np.linalg.norm(PV_3))

```

''' Velocity vector magnitude '''

```

VV_3 = np.array([VVi_3,VVj_3, VVk_3])
magnitude_VV_3 = float( np.linalg.norm(VV_3))

```

```

HK_11 = PVi_3/(magnitude_PV_3)
HK_12 = PVj_3/(magnitude_PV_3)
HK_13 = PVk_3/(magnitude_PV_3)
HK_21 = PVj_3/((PVj_3**2)+(PVi_3**2))
HK_22 = PVi_3/((PVj_3**2)+(PVi_3**2))
HK_31 = (2*PVi_3*PVk_3)/(((PVj_3**2)+(PVi_3**2))*magnitude_PV_3)
HK_32 = (2*PVj_3*PVk_3)/(((PVj_3**2)+(PVi_3**2))*magnitude_PV_3)
HK_33 = (((PVi_3**2)+(PVj_3**2)+(PVk_3**2))-
(2*PVk_3**2)/(((PVi_3**2)+(PVj_3**2)+(PVk_3**2))**2))*((PVj_3**2)+(PVi_3**2))

```

```

HK = np.array([[HK_11, HK_12, HK_13, 0, 0, 0],
               [HK_21, HK_22, 0, 0, 0, 0],
               [HK_31, HK_32, HK_33, 0, 0, 0],
               [0, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 0, 0],
               [0, 0, 0, 0, 0, 0]])

```

```

return HK

```

```

'''initial parameters'''

```

```

X = np.array([[PVi_3, 0, 0, 0, 0, 0],
              [PVj_3, 0, 0, 0, 0, 0],
              [PVk_3, 0, 0, 0, 0, 0],
              [VVi_3, 0, 0, 0, 0, 0],
              [VVj_3, 0, 0, 0, 0, 0],
              [VVk_3, 0, 0, 0, 0, 0]])

```

```

X11.append(X[0,0])
X12.append(X[0,1])
X13.append(X[0,2])
X14.append(X[0,3])
X15.append(X[0,4])
X16.append(X[0,5])

```

```

X21.append(X[1,0])
X22.append(X[1,1])
X23.append(X[1,2])
X24.append(X[1,3])
X25.append(X[1,4])
X26.append(X[1,5])

```

```

X31.append(X[3,0])
X32.append(X[2,1])
X33.append(X[2,2])
X34.append(X[2,3])
X35.append(X[2,4])
X36.append(X[2,5])

```

```

X41.append(X[3,0])
X42.append(X[3,1])
X43.append(X[3,2])
X44.append(X[3,3])
X45.append(X[3,4])
X46.append(X[3,5])

```

```

X51.append(X[4,0])
X52.append(X[4,1])
X53.append(X[4,2])
X54.append(X[4,3])
X55.append(X[4,4])
X56.append(X[4,5])

```

```

X61.append(X[5,0])
X62.append(X[5,1])
X63.append(X[5,2])
X64.append(X[5,3])

```

```

X65.append(X[5,4])
X66.append(X[5,5])

X_Mean = np.array([[np.mean(x),np.mean(X12),np.mean(X13),np.mean(X14),np.mean(X15),np.mean(X16)],
                    [np.mean(y),np.mean(X22),np.mean(X23),np.mean(X24),np.mean(X25),np.mean(X26)],
                    [np.mean(z),np.mean(X32),np.mean(X33),np.mean(X34),np.mean(X35),np.mean(X36)],
                    [np.mean(vx),np.mean(X42),np.mean(X43),np.mean(X44),np.mean(X45),np.mean(X46)],
                    [np.mean(vy),np.mean(X52),np.mean(X53),np.mean(X54),np.mean(X55),np.mean(X56)],
                    [np.mean(vz),np.mean(X62),np.mean(X63),np.mean(X64),np.mean(X65),np.mean(X66)]])

PK_1_S_1 = np.subtract(X,X_Mean)
PK_1_S_2 = np.transpose(PK_1_S_1)
PK_1 = np.matmul(PK_1_S_1,PK_1_S_2)

#----- START-----
while Time <= Period :

    X_Mean = np.array([[np.mean(x),np.mean(X12),np.mean(X13),np.mean(X14),np.mean(X15),np.mean(X16)],
                        [np.mean(y),np.mean(X22),np.mean(X23),np.mean(X24),np.mean(X25),np.mean(X26)],
                        [np.mean(z),np.mean(X32),np.mean(X33),np.mean(X34),np.mean(X35),np.mean(X36)],
                        [np.mean(vx),np.mean(X42),np.mean(X43),np.mean(X44),np.mean(X45),np.mean(X46)],
                        [np.mean(vy),np.mean(X52),np.mean(X53),np.mean(X54),np.mean(X55),np.mean(X56)],
                        [np.mean(vz),np.mean(X62),np.mean(X63),np.mean(X64),np.mean(X65),np.mean(X66)]])

    #sigma Point
    kk = 3-L
    Alpha_1 = float(0.03)# 10^(-4) -> 1
    Lambda_1 = (Alpha_1**2)*(L+kk)-L
    Beta_1 = 2

    if L == 0:
        X_sigma = X_Mean
        W_m = float((Lambda_1)/(L+Lambda_1))
        W_c = float((Lambda_1/(L+Lambda_1))+(1-np.power(Alpha_1,2)+Beta_1))

    elif (L % 2) == 0:
        X_sigma_S1 = (L + Lambda_1)*P_k
        X_sigma_S2 = np.sqrt(np.absolute(X_sigma_S1))
        X_sigma = X_Mean - X_sigma_S2
        W_m = float(1/(2*(L+Lambda_1)))
        W_c = float(1/(2*(L+Lambda_1)))
    else:
        X_sigma_S1 = (L + Lambda_1)*P_k
        X_sigma_S2 = np.sqrt(np.absolute(X_sigma_S1))
        X_sigma = X_Mean + X_sigma_S2
        W_m = float(1/(2*(L+Lambda_1)))
        W_c = float(1/(2*(L+Lambda_1)))

    #predict state ahead
    ''' step 1'''
    X_sigma_k = (f_cv(X_sigma[0,0],X_sigma[1,1],X_sigma[2,2],X_sigma[3,3],X_sigma[4,4],X_sigma[5,5]))

    '''step 2'''
    Xk = W_m*X_sigma_k

```

```

Xk11.append(Xk[0,0])
Xk12.append(Xk[0,1])
Xk13.append(Xk[0,2])
Xk14.append(Xk[0,3])
Xk15.append(Xk[0,4])
Xk16.append(Xk[0,5])
Xk21.append(Xk[1,0])
Xk22.append(Xk[1,1])
Xk23.append(Xk[1,2])
Xk24.append(Xk[1,3])
Xk25.append(Xk[1,4])
Xk26.append(Xk[1,5])
Xk31.append(Xk[2,0])
Xk32.append(Xk[2,1])
Xk33.append(Xk[2,2])
Xk34.append(Xk[2,3])
Xk35.append(Xk[2,4])
Xk36.append(Xk[2,5])
Xk41.append(Xk[3,0])
Xk42.append(Xk[3,1])
Xk43.append(Xk[3,2])
Xk44.append(Xk[3,3])
Xk45.append(Xk[3,4])
Xk46.append(Xk[3,5])
Xk51.append(Xk[4,0])
Xk52.append(Xk[4,1])
Xk53.append(Xk[4,2])
Xk54.append(Xk[4,3])
Xk55.append(Xk[4,4])
Xk56.append(Xk[4,5])
Xk61.append(Xk[5,0])
Xk62.append(Xk[5,1])
Xk63.append(Xk[5,2])
Xk64.append(Xk[5,3])
Xk65.append(Xk[5,4])
Xk66.append(Xk[5,5])

X_k = np.array([[np.sum(Xk11),np.sum(Xk12),np.sum(Xk13),np.sum(Xk14),np.sum(Xk15),np.sum(Xk16)],
                [np.sum(Xk21),np.sum(Xk22),np.sum(Xk23),np.sum(Xk24),np.sum(Xk25),np.sum(Xk26)],
                [np.sum(Xk31),np.sum(Xk32),np.sum(Xk33),np.sum(Xk34),np.sum(Xk35),np.sum(Xk36)],
                [np.sum(Xk41),np.sum(Xk42),np.sum(Xk43),np.sum(Xk44),np.sum(Xk45),np.sum(Xk46)],
                [np.sum(Xk51),np.sum(Xk52),np.sum(Xk53),np.sum(Xk54),np.sum(Xk55),np.sum(Xk56)],
                [np.sum(Xk61),np.sum(Xk62),np.sum(Xk63),np.sum(Xk64),np.sum(Xk65),np.sum(Xk66)]])
'''step 3'''
cov_m = np.stack((x,y,z,vx,vy,vz), axis=0)
cov_f = np.cov(cov_m)
Q = cov_f

Pk_s1 = X_sigma_k - X_k
Pk_s2 = np.transpose(Pk_s1)
Pk_s3 = np.matmul(Pk_s1,Pk_s2)
Pk_s4 = W_c*Pk_s3
Pk = Pk_s4 + Q

Pk11.append(Pk[0,0])
Pk12.append(Pk[0,1])
Pk13.append(Pk[0,2])

```

```

Pk14.append(Pk[0,3])
Pk15.append(Pk[0,4])
Pk16.append(Pk[0,5])
Pk21.append(Pk[1,0])
Pk22.append(Pk[1,1])
Pk23.append(Pk[1,2])
Pk24.append(Pk[1,3])
Pk25.append(Pk[1,4])
Pk26.append(Pk[1,5])
Pk31.append(Pk[2,0])
Pk32.append(Pk[2,1])
Pk33.append(Pk[2,2])
Pk34.append(Pk[2,3])
Pk35.append(Pk[2,4])
Pk36.append(Pk[2,5])
Pk41.append(Pk[3,0])
Pk42.append(Pk[3,1])
Pk43.append(Pk[3,2])
Pk44.append(Pk[3,3])
Pk45.append(Pk[3,4])
Pk46.append(Pk[3,5])
Pk51.append(Pk[4,0])
Pk52.append(Pk[4,1])
Pk53.append(Pk[4,2])
Pk54.append(Pk[4,3])
Pk55.append(Pk[4,4])
Pk56.append(Pk[4,5])
Pk61.append(Pk[5,0])
Pk62.append(Pk[5,1])
Pk63.append(Pk[5,2])
Pk64.append(Pk[5,3])
Pk65.append(Pk[5,4])
Pk66.append(Pk[5,5])
P_k = np.array([[np.sum(Pk11),np.sum(Pk12),np.sum(Pk13),np.sum(Pk14),np.sum(Pk15),np.sum(Pk16)],
                [np.sum(Pk21),np.sum(Pk22),np.sum(Pk23),np.sum(Pk24),np.sum(Pk25),np.sum(Pk26)],
                [np.sum(Pk31),np.sum(Pk32),np.sum(Pk33),np.sum(Pk34),np.sum(Pk35),np.sum(Pk36)],
                [np.sum(Pk41),np.sum(Pk42),np.sum(Pk43),np.sum(Pk44),np.sum(Pk45),np.sum(Pk46)],
                [np.sum(Pk51),np.sum(Pk52),np.sum(Pk53),np.sum(Pk54),np.sum(Pk55),np.sum(Pk56)],
                [np.sum(Pk61),np.sum(Pk62),np.sum(Pk63),np.sum(Pk64),np.sum(Pk65),np.sum(Pk66)]])
if L == 0:
    X_sigma = X_Mean
elif (L % 2) == 0:
    X_sigma_S1 = (L + Lambda_1)*P_k
    X_sigma_S2 = np.sqrt(np.absolute(X_sigma_S1))
    X_sigma = X_Mean - X_sigma_S2
else:
    X_sigma_S1 = (L + Lambda_1)*P_k
    X_sigma_S2 = np.sqrt(np.absolute(X_sigma_S1))
    X_sigma = X_Mean + X_sigma_S2

#Compute Kalman Gain
'''Step 1'''
Y_sigma = (h_cv(X_sigma[0,0],X_sigma[1,1],X_sigma[2,2],X_sigma[3,3],X_sigma[4,4],X_sigma[5,5]))
Yk = W_m*Y_sigma
Yk11.append(Yk[0,0])
Yk12.append(Yk[0,1])

```

```

Yk13.append(Yk[0,2])
Yk14.append(Yk[0,3])
Yk15.append(Yk[0,4])
Yk16.append(Yk[0,5])
Yk21.append(Yk[1,0])
Yk22.append(Yk[1,1])
Yk23.append(Yk[1,2])
Yk24.append(Yk[1,3])
Yk25.append(Yk[1,4])
Yk26.append(Yk[1,5])
Yk31.append(Yk[2,0])
Yk32.append(Yk[2,1])
Yk33.append(Yk[2,2])
Yk34.append(Yk[2,3])
Yk35.append(Yk[2,4])
Yk36.append(Yk[2,5])
Yk41.append(Yk[3,0])
Yk42.append(Yk[3,1])
Yk43.append(Yk[3,2])
Yk44.append(Yk[3,3])
Yk45.append(Yk[3,4])
Yk46.append(Yk[3,5])
Yk51.append(Yk[4,0])
Yk52.append(Yk[4,1])
Yk53.append(Yk[4,2])
Yk54.append(Yk[4,3])
Yk55.append(Yk[4,4])
Yk56.append(Yk[4,5])
Yk61.append(Yk[5,0])
Yk62.append(Yk[5,1])
Yk63.append(Yk[5,2])
Yk64.append(Yk[5,3])
Yk65.append(Yk[5,4])
Yk66.append(Yk[5,5])

```

```

Y_k = np.array([[np.sum(Yk11),np.sum(Yk12),np.sum(Yk13),np.sum(Yk14),np.sum(Yk15),np.sum(Yk16)],
               [np.sum(Yk21),np.sum(Yk22),np.sum(Yk23),np.sum(Yk24),np.sum(Yk25),np.sum(Yk26)],
               [np.sum(Yk31),np.sum(Yk32),np.sum(Yk33),np.sum(Yk34),np.sum(Yk35),np.sum(Yk36)],
               [np.sum(Yk41),np.sum(Yk42),np.sum(Yk43),np.sum(Yk44),np.sum(Yk45),np.sum(Yk46)],
               [np.sum(Yk51),np.sum(Yk52),np.sum(Yk53),np.sum(Yk54),np.sum(Yk55),np.sum(Yk56)],
               [np.sum(Yk61),np.sum(Yk62),np.sum(Yk63),np.sum(Yk64),np.sum(Yk65),np.sum(Yk66)]])

```

'''Step 2'''

```

#P_vv = W_c *(np.matmul((Y_sigma - Y_k),np.transpose(Y_sigma - Y_k)))+R

```

```

P_vv_s1 = Y_sigma - Y_k

```

```

P_vv_s2 = np.transpose(P_vv_s1)

```

```

P_vv_s3 = np.matmul(P_vv_s1,P_vv_s2)

```

```

P_vv_s4 = W_c*P_vv_s3

```

```

P_vv = P_vv_s4 + R

```

```

Pvv11.append(P_vv[0,0])

```

```

Pvv12.append(P_vv[0,1])

```

```

Pvv13.append(P_vv[0,2])

```

```

Pvv14.append(P_vv[0,3])

```

```

Pvv15.append(P_vv[0,4])

```

```

Pvv16.append(P_vv[0,5])

```

```

Pvv21.append(P_vv[1,0])

```



```

Pvv22.append(P_vv[1,1])
Pvv23.append(P_vv[1,2])
Pvv24.append(P_vv[1,3])
Pvv25.append(P_vv[1,4])
Pvv26.append(P_vv[1,5])

Pvv31.append(P_vv[2,0])
Pvv32.append(P_vv[2,1])
Pvv33.append(P_vv[2,2])
Pvv34.append(P_vv[2,3])
Pvv35.append(P_vv[2,4])
Pvv36.append(P_vv[2,5])

Pvv41.append(P_vv[3,0])
Pvv42.append(P_vv[3,1])
Pvv43.append(P_vv[3,2])
Pvv44.append(P_vv[3,3])
Pvv45.append(P_vv[3,4])
Pvv46.append(P_vv[3,5])

Pvv51.append(P_vv[4,0])
Pvv52.append(P_vv[4,1])
Pvv53.append(P_vv[4,2])
Pvv54.append(P_vv[4,3])
Pvv55.append(P_vv[4,4])
Pvv56.append(P_vv[4,5])

Pvv61.append(P_vv[5,0])
Pvv62.append(P_vv[5,1])
Pvv63.append(P_vv[5,2])
Pvv64.append(P_vv[5,3])
Pvv65.append(P_vv[5,4])
Pvv66.append(P_vv[5,5])

Pvv =
np.array([[float(np.sum(Pvv11)),float(np.sum(Pvv12)),float(np.sum(Pvv13)),float(np.sum(Pvv14)),float(np.sum(Pv
v15)),float(np.sum(Pvv16))],

[float(np.sum(Pvv21)),float(np.sum(Pvv22)),float(np.sum(Pvv23)),float(np.sum(Pvv24)),float(np.sum(Pvv25)),floa
t(np.sum(Pvv26))],

[float(np.sum(Pvv31)),float(np.sum(Pvv32)),float(np.sum(Pvv33)),float(np.sum(Pvv34)),float(np.sum(Pvv35)),floa
t(np.sum(Pvv36))],

[float(np.sum(Pvv41)),float(np.sum(Pvv42)),float(np.sum(Pvv43)),float(np.sum(Pvv44)),float(np.sum(Pvv45)),floa
t(np.sum(Pvv46))],

[float(np.sum(Pvv51)),float(np.sum(Pvv52)),float(np.sum(Pvv53)),float(np.sum(Pvv54)),float(np.sum(Pvv55)),floa
t(np.sum(Pvv56))],

[float(np.sum(Pvv61)),float(np.sum(Pvv62)),float(np.sum(Pvv63)),float(np.sum(Pvv64)),float(np.sum(Pvv65)),floa
t(np.sum(Pvv66))]])

Pvv_invs = np.linalg.inv(Pvv)

'''Step 3'''
#P_xy = W_c *(np.matmul((X_sigma_k - X_k),(np.transpose(Y_sigma - Y_k))))

```

```

P_xy_s1 = X_sigma_k - X_k
P_xy_s2 = Y_sigma - Y_k
#P_xy_s3 = np.transpose(P_xy_s2)
P_xy_s4 = np.matmul(P_xy_s1,P_xy_s2)
P_xy = W_c*P_xy_s4

Pxy11.append(P_xy[0,0])
Pxy12.append(P_xy[0,1])
Pxy13.append(P_xy[0,2])
Pxy14.append(P_xy[0,3])
Pxy15.append(P_xy[0,4])
Pxy16.append(P_xy[0,5])

Pxy21.append(P_xy[1,0])
Pxy22.append(P_xy[1,1])
Pxy23.append(P_xy[1,2])
Pxy24.append(P_xy[1,3])
Pxy25.append(P_xy[1,4])
Pxy26.append(P_xy[1,5])

Pxy31.append(P_xy[2,0])
Pxy32.append(P_xy[2,1])
Pxy33.append(P_xy[2,2])
Pxy34.append(P_xy[2,3])
Pxy35.append(P_xy[2,4])
Pxy36.append(P_xy[2,5])

Pxy41.append(P_xy[3,0])
Pxy42.append(P_xy[3,1])
Pxy43.append(P_xy[3,2])
Pxy44.append(P_xy[3,3])
Pxy45.append(P_xy[3,4])
Pxy46.append(P_xy[3,5])

Pxy51.append(P_xy[4,0])
Pxy52.append(P_xy[4,1])
Pxy53.append(P_xy[4,2])
Pxy54.append(P_xy[4,3])
Pxy55.append(P_xy[4,4])
Pxy56.append(P_xy[4,5])

Pxy61.append(P_xy[5,0])
Pxy62.append(P_xy[5,1])
Pxy63.append(P_xy[5,2])
Pxy64.append(P_xy[5,3])
Pxy65.append(P_xy[5,4])
Pxy66.append(P_xy[5,5])

Pxy =
np.array([[float(np.sum(Pxy11)),float(np.sum(Pxy12)),float(np.sum(Pxy13)),float(np.sum(Pxy14)),float(np.sum(Pxy15)),float(np.sum(Pxy16))],
[ float(np.sum(Pxy21)),float(np.sum(Pxy22)),float(np.sum(Pxy23)),float(np.sum(Pxy24)),float(np.sum(Pxy25)),float(np.sum(Pxy26))],
[ float(np.sum(Pxy31)),float(np.sum(Pxy32)),float(np.sum(Pxy33)),float(np.sum(Pxy34)),float(np.sum(Pxy35)),float(np.sum(Pxy36))],

```

```
[float(np.sum(Pxy41)),float(np.sum(Pxy42)),float(np.sum(Pxy43)),float(np.sum(Pxy44)),float(np.sum(Pxy45)),float(np.sum(Pxy46))],
```

```
[float(np.sum(Pxy51)),float(np.sum(Pxy52)),float(np.sum(Pxy53)),float(np.sum(Pxy54)),float(np.sum(Pxy55)),float(np.sum(Pxy56))],
```

```
[float(np.sum(Pxy61)),float(np.sum(Pxy62)),float(np.sum(Pxy63)),float(np.sum(Pxy64)),float(np.sum(Pxy65)),float(np.sum(Pxy66))]]]
```

```
'''Step 4'''
```

```
K = np.matmul(Pxy,Pvv_invs)
```

```
#NEW
```

```
'''STEP 1'''
```

```
X_Mean = X_k +np.matmul(K,(Y_sigma - Y_k))
```

```
x.append(X_Mean[0,0])
```

```
y.append(X_Mean[0,1])
```

```
z.append(X_Mean[0,2])
```

```
vx.append(X_Mean[0,3])
```

```
vy.append(X_Mean[0,4])
```

```
vz.append(X_Mean[0,5])
```

```
px.append(X_Mean[0,0])
```

```
py.append(X_Mean[0,1])
```

```
pz.append(X_Mean[0,2])
```

```
pvx.append(X_Mean[0,3])
```

```
pvy.append(X_Mean[0,4])
```

```
pvz.append(X_Mean[0,5])
```

```
X11.append(X_Mean[0,0])
```

```
X12.append(X_Mean[0,1])
```

```
X13.append(X_Mean[0,2])
```

```
X14.append(X_Mean[0,3])
```

```
X15.append(X_Mean[0,4])
```

```
X16.append(X_Mean[0,5])
```

```
X21.append(X_Mean[1,0])
```

```
X22.append(X_Mean[1,1])
```

```
X23.append(X_Mean[1,2])
```

```
X24.append(X_Mean[1,3])
```

```
X25.append(X_Mean[1,4])
```

```
X26.append(X_Mean[1,5])
```

```
X31.append(X_Mean[3,0])
```

```
X32.append(X_Mean[2,1])
```

```
X33.append(X_Mean[2,2])
```

```
X34.append(X_Mean[2,3])
```

```
X35.append(X_Mean[2,4])
```

```
X36.append(X_Mean[2,5])
```

```
X41.append(X_Mean[3,0])
```

```
X42.append(X_Mean[3,1])
```

```
X43.append(X_Mean[3,2])
```

```
X44.append(X_Mean[3,3])
```

```
X45.append(X_Mean[3,4])
```

```
X46.append(X_Mean[3,5])
```

```
X51.append(X_Mean[4,0])
```

```
X52.append(X_Mean[4,1])
```

```
X53.append(X_Mean[4,2])
```

```
X54.append(X_Mean[4,3])
```

```
X55.append(X_Mean[4,4])
```

```
X56.append(X_Mean[4,5])
```

```
X61.append(X_Mean[5,0])
```

```
X62.append(X_Mean[5,1])
```

```
X63.append(X_Mean[5,2])
```

```
X64.append(X_Mean[5,3])
```

```
X65.append(X_Mean[5,4])
```

```
X66.append(X_Mean[5,5])
```

```
PK_1 = P_k - np.matmul(np.matmul(K,Pvv),np.transpose(K))
```

```
print ('x: ',X_Mean[0,0],'(m)')
```

```
print ('y: ',X_Mean[0,1],'(m)')
```

```
print ('z: ',X_Mean[0,2],'(m)')
```

```
print ('vx: ',X_Mean[0,3],'(m/s)')
```

```
print ('vy: ',X_Mean[0,4],'(m/s)')
```

```
print ('vz: ',X_Mean[0,5],'(m/s)')
```

```
orbital_parameters(X_Mean[0,0],X_Mean[0,1],X_Mean[0,2],X_Mean[0,3],X_Mean[0,4],X_Mean[0,5]);
```

```
for cell in column_j:
```

```
if cell.value == time_epoch:
```

```
rowss = [cell.coordinate[1],cell.coordinate[2],cell.coordinate[3]]
```

```
rowV=int(".".join(map(str, rowss)))
```

```
PVi_r = float((ws.cell(rowV+1,2).value)*1000)
```

```
PVj_r = float((ws.cell(rowV+1,3).value)*1000)
```

```
PVkj_r = float((ws.cell(rowV+1,4).value)*1000)
```

```
VVi_r = float((ws.cell(rowV+1,5).value)*1000)
```

```
VVj_r = float((ws.cell(rowV+1,6).value)*1000)
```

```
VVkj_r = float((ws.cell(rowV+1,7).value)*1000)
```

```
time_r = float(ws.cell(rowV+1,1).value)
```

```
ex.append(PVi_r)
```

```
ey.append(PVj_r)
```

```
ez.append(PVkj_r)
```

```
evx.append(VVi_r)
```

```
evy.append(VVj_r)
```

```
evz.append(VVkj_r)
```

```
timer.append(time_r)
```

```
errx.append((np.power(PVi_r - X_Mean[0,0],2)))
```

```
errorx1 = np.sqrt(np.sum(errx)/f)
```

```
pex.append(errorx1)
```

```
erry.append((np.power(PVj_r - X_Mean[0,1],2)))
```

```
error1 = np.sqrt(np.sum(erry)/f)
```

```
pey.append(error1)
```

```
errz.append((np.power(PVk_r-X_Mean[0,2],2)))
errorz1 = np.sqrt(np.sum(errz)/f)
pez.append(errorz1)
```

```
errvx.append((np.power(VVi_r-X_Mean[0,3],2)))
errorvx1 = np.sqrt(np.sum(errvx)/f)
pevx.append(errorvx1)
```

```
errvy.append((np.power(VVj_r-X_Mean[0,4],2)))
errorvy1 = np.sqrt(np.sum(errvy)/f)
pevy.append(errorvy1)
```

```
errvz.append((np.power(VVk_r-X_Mean[0,5],2)))
errorvz1 = np.sqrt(np.sum(errvz)/f)
pevz.append(errorvz1)
```

```
f= f+1
```

```
Time = Time +10
time_epoch = time_epoch+10
ptime.append(time_epoch)
L = L+1
```

```
print('Done')
```

Appendix E

GA Python Code

This appendix consists of the Python code of the GA

```
import numpy as np
import datetime
import math
import xlswriter
import pandas as pd
from sgp4.api import Satrec
from openpyxl import load_workbook
from numpy.random import randint
from numpy.random import rand
import time

start = time.time()
#-----General Parameters-----
rowV = 101
n_iter = 100
# bits
n_bits = 2
# define the population size
n_pop = 10000
Select_rate_cross = 1
Select_rate_mut = 0.01
# crossover rate
r_cross = 0.6
# mutation rate
r_mut = 1.0 / float(n_bits)

population = []
SA = []
Ecc = []
MAnomly = []
raan = []
ag = []
inc = []
Best_Score = []

rx = []
ry = []
rz = []
vx = []
vy = []
vz = []

position_error = []
position_error1 = []
selectionResults = []
final_error = []
JD = []
FR = []
#-----GPS Data-----
```

```

GPS_DATA = load_workbook(r'C:\Users\pinkw\OneDrive\Desktop\GPS2.xlsx', data_only=True)
GPS_DATA_sheet1 = GPS_DATA['Sheet1']
all_rows = list(GPS_DATA_sheet1.rows)
ws=GPS_DATA.active
column_j = ws['A']
#----- Generate Random Population-----
#get random value
def split_at_decimal(num):
    if num < 0.001:
        num = 0.0
    integer, decimal = (int(i) for i in str(num).split("."))
    return decimal

def get_random_number(low, high):

    return np.random.uniform(low, high)

def change_formatting(parameter, length):
    if len(str(parameter)) > length:
        parameter = str(parameter)[:length]
    elif len(str(parameter)) < length:
        rem = length - len(str(parameter))
        if "." in str(parameter):
            parameter = str(parameter) + rem*"0"
        else:
            parameter = str(parameter) + "." + (rem-1)*"0"
    return parameter

def get_julian_day (epoch_time):
    return (epoch_time/86400)+244058.5

def get_julian_date (epoch_time):
    date_time = datetime.datetime.fromtimestamp( epoch_time )
    dateFormatted = date_time.strftime("%y")
    day = int(date_time.strftime("%j"))
    hour = int(date_time.strftime("%H"))*3600
    minute = int(date_time.strftime("%M"))*60
    second = int(date_time.strftime("%S"))
    Time1 = (hour+minute+second )/86400
    Time = change_formatting(round(day+Time1,8),12)
    return Time

def year_detect (epoch_time):
    date_time = datetime.datetime.fromtimestamp( epoch_time )
    dateFormatted = date_time.strftime("%y")
    year = "{}".format(dateFormatted)
    return year

def get_fraction(num):
    integer, decimal = (int(i) for i in str(num).split("."))
    decimal = num - integer
    fr = 1 - decimal
    return fr
#-----GA-----

```

```

# tournament selection
def selection (position_error,population,Select_rate):

    Select_number = int(n_pop*(Select_rate/100))
    position_error = np.where(np.isnan(position_error1), 100000000000, position_error1)
    position_error_min = np.sort(position_error)
    position_error_max = np.flip(position_error_min)
    Best_Score.append(position_error_min [0])

    if Select_number <=1:
        loc = np.where(position_error == position_error_min [0])
        loc1 = int(loc[0][-1])
        selectionResults.append([population[loc1]])
        Best_Score.append(position_error_min [0])

    else:
        for i in range (0,Select_number):
            loc = np.where(position_error == position_error_min [i])
            Best_Score.append(position_error_min [i])
            loc1 = int(loc[0][-1])
            selectionResults.append([population[loc1]])

    return selectionResults

# crossover 2x parents to create x children
def crossover(selectionResults, r_cross, position_error,population,PVj,PVj,PVj):
    i = 0

    while i < (len(selectionResults)):
        if i == 100:
            break
        else:
            # children are copies of parents by default
            p1 = selectionResults[i]
            p2 = selectionResults[i+1]
            p1 = p1[0].split()
            p2 = p2[0].split()
            # select crossover point that is not on the end of the string
            P = np.random.rand(len(p1))
            for x in range (len(P)):
                if P[x]<r_cross:
                    if x > 12:
                        if x<= 17 :
                            temp = p1[x]
                            p1[x] = p2[x]
                            p2[x] = temp

            line_0 = p1[0]
            line_1_str = f'{p1[1]} {p1[2]} {p1[3]} {p1[4]} {p1[5]} {p1[6]} {p1[7]} {p1[8]} {p1[9]}'
            line_2_str = f'{p1[10]} {p1[11]} {p1[12]} {p1[13]} {p1[14]} {p1[15]} {p1[16]} {p1[17]} {p1[18]}'

            c1 = f'{line_0}\n{line_1_str}\n{line_2_str}'

```



```

line_0_1 = p2[0]
line_1_str_1 = f'{p2[1]} {p2[2]} {p2[3]} {p2[4]} {p2[5]} {p2[6]} {p2[7]} {p2[8]} {p2[9]}'
line_2_str_1 = f'{p2[10]} {p2[11]} {p2[12]} {p2[13]} {p2[14]} {p2[15]} {p2[16]} {p2[17]} {p2[18]}'

c2 = f"{line_0_1}\n{line_1_str_1}\n{line_2_str_1}"

position_error = np.where(np.isnan(position_error1), 100000000000, position_error1)
position_error_min = np.sort(position_error)
position_error_max = np.flip(position_error_min)
loc1 = np.where(position_error == position_error_max [i])
loc_1 = int(loc1[0][-1])
population[loc_1] = c1
loc2 = np.where(position_error == position_error_max [i+1])
loc_2 = int(loc2[0][-1])
population[loc_2] = c2

i = i + 2

```

```

for j in range(0,(len(position_error))):

```

```

    pop = population[j]

```

```

    if isinstance(pop, str):

```

```

        lis = pop.split("\n")

```

```

    else:

```

```

        lis = pop[0].split("\n")

```

```

    line_0 = lis[0]

```

```

    line_1 = lis[1]

```

```

    line_2 = lis[2]

```

```

    jd = JD[i]

```

```

    fr = FR[i]

```

```

    satellite = Satrec.twoline2rv(line_1,line_2)

```

```

    e, r, v = satellite.sgp4(jd, fr)

```

```

    rx = r[0]

```

```

    ry = r[1]

```

```

    rz = r[2]

```

```

    vx = v[0]

```

```

    vy = v[1]

```

```

    vz = v[2]

```

```

    error = float(np.sqrt(((PVi-rx)**2)+((PVj-ry)**2)+((PVk-rz)**2)))

```

```

    position_error[j] = error

```

```

    final_error.append(error)

```

```

return population

```

```

# mutation operator

```

```

def mutation(selectionResults, r_mut,population,position_error,PVi,PVj,PVk):

```

```

    for i in range(0,(len(selectionResults))):

```

```

# children are copies of parents by default
bitstring = selectionResults[i]
bitstring = bitstring[0].split()
bitstring[13] = float(bitstring[13])+1
bitstring[16] = float(bitstring[16])+1
line_0 = bitstring[0]
line_1_str = f'{bitstring[1]} {bitstring[2]} {bitstring[3]} {bitstring[4]} {bitstring[5]} {bitstring[6]}
{bitstring[7]} {bitstring[8]} {bitstring[9]}'
line_2_str = f'{bitstring[10]} {bitstring[11]} {bitstring[12]} {bitstring[13]} {bitstring[14]} {bitstring[15]}
{bitstring[16]} {bitstring[17]} {bitstring[18]}'
bitstring1 = f'{line_0}\n{line_1_str}\n{line_2_str}'

position_error = np.where(np.isnan(position_error1), 100000000000, position_error1)
position_error_min = np.sort(position_error)
position_error_max = np.flip(position_error_min)

loc1 = np.where(position_error == position_error_max [0])
loc_1 = int(loc1[0][-1])
population[loc_1] = bitstring1

for i in range(0,(len(position_error))):
    pop = population [i]
    if isinstance(pop, str):
        lis = pop.split("\n")
    else:
        lis = pop[0].split("\n")

    line_0 = lis[0]
    line_1 = lis[1]
    line_2 = lis[2]

    jd = JD[i]
    fr = FR[i]

    satellite = Satrec.twoline2rv(line_1,line_2)
    e, r, v = satellite.sgp4(jd, fr)

    rx = r[0]
    ry = r[1]
    rz = r[2]
    vx = v[0]
    vy = v[1]
    vz = v[2]

    error = float(np.sqrt(((PVi-rx)**2)+((PVj-ry)**2)+((PVk-rz)**2)))

    position_error[i] = error
    final_error.append(error)

return population

```

```

# genetic algorithm
def genetic_algorithm(rowV,n_pop,r_cross,r_mut,n_iter):
    population = []
    # initial population of random bitstring
    for index in range(0,n_pop):
        #Population initialization
        # altitude ( 500 - 550km)
        # semi major axis (6871 - 6921 km)
        # Eccentricity (0 - 1)
        # Inclination ( 0 - 180 degree)
        # true anomly ( 0 - 360 degree)
        # RAAN ( 0 - 360 degree)
        # argument of perigee ( 0 - 360 degree)
        # mean montion (0.168971 - 0.165335) rev /day

        SatelliteNumber = '25544'
        InternationalDesignator = '98067A'
        julian_date = get_julian_date(float(ws.cell(rowV,1).value))
        jd = get_julian_day(float(ws.cell(rowV,1).value))
        JD.append(jd)
        fr = get_fraction(jd)
        FR.append(fr)
        YY = year_detect(float(ws.cell(rowV,1).value))
        FDoM = 0.00000014

        '''
        FDoM1 = 0.00000014
        FDoM = '{0:.8f}'.format(FDoM1)
        '''

        SDoM = '00000-0'
        B_Drag = '67960-4'
        Ephemeris_type = 0
        CheckNumber = 5293
        RevNum = 346978
        a = float(get_random_number(6871,6921))
        e1 = float (get_random_number(float(0.0001),float(0.005)))
        e = change_formatting(split_at_decimal(e1),7)
        i = change_formatting(float(get_random_number(20, 40)),8)
        MA = change_formatting(float(get_random_number(300,360)),8)
        RAAN = change_formatting(float(get_random_number(0, 360)),8)
        AG = change_formatting(float(get_random_number(0,360)),8)
        Mean_montion = change_formatting(float(get_random_number(13,15)),11)

        SA.append(a)
        Ecc.append(e1)
        inc.append(i)
        raan.append(RAAN)
        ag.append(AG)
        MAnomly.append(MA)

        line_0 = 'SatelliteName'
        line_1_str = f'1 {SatelliteNumber}U {InternationalDesignator} {YY}{julian_date} {FDoM} {SDoM}
{B_Drag} {Ephemeris_type} {CheckNumber}'
        line_2_str = f'2 {SatelliteNumber} {i} {RAAN} {e} {AG} {MA} {Mean_montion} {RevNum}'

        tle_output = f'"{line_0}\n{line_1_str}\n{line_2_str}"'

```

```

population.append(tle_output)

PVi = float((ws.cell(rowV,2).value))
PVj = float((ws.cell(rowV,3).value))
PVk = float((ws.cell(rowV,4).value))
VVi = float((ws.cell(rowV,5).value))
VVj = float((ws.cell(rowV,6).value))
VVk = float((ws.cell(rowV,7).value))

satellite = Satrec.twoline2rv(line_1_str,line_2_str)
e, r, v = satellite.sgp4(jd, fr)
#print('r',r)
#print('v',v)
#print('e',float(e))
rx = r[0]
ry = r[1]
rz = r[2]
vx = v[0]
vy = v[1]
vz = v[2]
#objective function
error = float(np.sqrt(((PVi-rx)**2)+((PVj-ry)**2)+((PVk-rz)**2)))

final_error.append(error)
position_error1.append(error)
position_error = np.where(np.isnan(position_error1), 100000000000, position_error1)

for i in range (0,n_iter):

    selection(position_error1,population,Select_rate_cross)
    crossover(selectionResults, r_cross, position_error1,population,PVi,PVj,PVk)
    selectionResults.clear()
    selection(position_error1,population,Select_rate_mut)
    mutation(selectionResults, r_mut,population,position_error1,PVi,PVj,PVk)
    selectionResults.clear()

fittest_1 = min(position_error)
loc_1 = np.where(position_error == fittest_1)
loc1 = int(loc_1[0][-1])
print(population[loc1])
return

genetic_algorithm(rowV,n_pop,r_cross,r_mut,n_iter)

''' Save to excel file '''
workbook = xlswriter.Workbook('EKFRESULT.xlsx')
worksheet = workbook.add_worksheet()
col1 = "score"
col2 = " all the error"
data1 = pd.DataFrame({col1:Best_Score[:1048576]})
data = pd.DataFrame({col2:final_error[:1048576]})

```

```
data.to_excel('GA_position_error_1.xlsx', sheet_name='sheet1', index=False)
data1.to_excel('GA_BEST_Score_1.xlsx', sheet_name='sheet1', index=False)

print('Done')
print('\n')
print('\n')
end = time.time()
print("The time of execution of above program is :",
      (end-start) * 10**3, "ms")
```

UAEU

جامعة الإمارات العربية المتحدة
United Arab Emirates University



UAE UNIVERSITY MASTER THESIS NO. 2022:81

The aim of this thesis is to be able to determine and track the small or medium satellite orbit without depending on third parties by using the Extended and Unscented Kalman filter and genetic algorithm.

www.uaeu.ac.ae

Shamma Jamali received her Bachelor of Science in Mechanical Engineering from College of Science, American University of Ras Al Khaimah (AURAK), UAE.

Online publication of thesis:
<https://scholarworks.uaeu.ac.ae/etds/>

UAEU

عمادة المكتبات
Libraries Deanship

جامعة الإمارات العربية المتحدة
United Arab Emirates University

