UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de Formació Interdisciplinària Superior

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial, Aeroespacial i Audiovisual de Terrassa

Grau en Enginyeria en Tecnologies Aeroespacials
Grau en Enginyeria en Tecnologies Industrials

Study on the system architecture of the ONION project

ThalesAlenia Space
a Thales / Leonardo company

Author: Ivan Parrot Martínez
Supervisor: Gérald Emmanuel Garcia
Local tutor: Joaquín Hernández Ortega

Cannes

February 2019

Abstract

Earth Observation, is one of the multiple purpose a satellite can have. Either with a meteorological, surveying or imaging goal. Most of EO missions have been launched as a single satellite, whose orbit have been carefully designed to cover most of the Earth over their repeating cycle, which can last several days. A major drawback is the temporal resolution, depending on the satellite ground track, which could last from several hours to a few days. However, little efforts have been made so far to launch a constellation with the purpose of increasing such resolution.

A need to create such constellation arises when constant monitoring of a place requires a location to be revisited in shorter periods of time, in general, with the mission of monitoring disasters such as tsunamis or hurricanes. This makes a single satellite unsuitable, and only a group of satellites working together makes it possible.

In this project, a little approach to existing constellations is made, and a study of an architecture that would accomplish these objectives is developed. The algorithm used for this particular project can therefore be used with other missions with similar goals and requirements.

Keywords

Satellite constellation, Earth Observation, ocean monitoring, LEO, orbit, optimization.

Acknowledgements

I would like to thank the following people, who helped me during the developing of this project, in one way or another.

To my family, for morally supporting me, besides the distance.

To Thales Alenia Space, for providing me the possibility to develop this thesis during this six-month experience.

To CFIS, for getting me in touch with Thales Alenia Space.

To Gérald Garcia, for his support and constant comments about my work.

To the rest of colleagues I met, for making this stage much more pleasant.

Contents

List of Figures

List of Tables

Nomenclature

Acronyms

CNES	Centre National d'Études Spatiales
EO	Earth Observation
ESA	European Space Agency
FFSS	Fractionated and Federated Satellite Systems
GA	Genetic Algorithm
GEO	Geostationary Earth Orbit
GPS	Global Positioning System
HEO	Highly Elliptical Orbit
LEO	Low Earth Orbit
MATLAB	Matrix Laboratory
MEO	Medium Earth Orbit
ONION	Operational Network of Individual Observation Nodes
RGT	Repeating Ground Track
TAS	Thales Alenia Space

Roman symbols

a	Semi-major axis
e	Orbit excentricity
f	Fitness/objective function
F	Phasing parameter for Walker and Flower Constellations
g	Gravity acceleration
G	Universal gravitational constant
h	Altitude
i	Orbit inclination
N	Total number of satellites
P	Number of orbit planes
R_E	Earth radius
S	Number of satellites per plane
T	Period
w	Weight for objectives

Greek symbols

γ	Vernal equinox
ε	Satellite elevation
φ	Latitude
λ	Longitude
μ	Gravitational constant
ν	True anomaly
τ	RGT period ratio
ω	Perigee argument
Ω	Right ascension of ascending node

Physical constants

Symbol	**Name**	**Value**
μ_E	Earth gravitational parameter	$3.986 \times 10^{14}\,\mathrm{m^3\,s^{-2}}$
M_E	Mass of Earth	$5.98 \times 10^{24}\,\mathrm{kg}$
G	Gravitational constant	$6.67 \times 10^{-11}\,\mathrm{m^3\,kg^{-1}\,s^{-2}}$
R_E	Earth equatorial radius	$6\,378\,137\,\mathrm{m}$
c	Earth polar radius	$6\,356\,752\,\mathrm{m}$
ω_E	Earth rotation rate	$7.292\,115 \times 10^{-5}\,\mathrm{rad\,s^{-1}}$
J_2	Zonal coefficient	$1.081\,874 \times 10^{-3}$
-	Solar day	$86\,400\,\mathrm{s}$
-	Sidereal day	$86\,164.09\,\mathrm{s}$

Chapter 1

Introduction

Aim

Design the architecture of a mission related to the ONION project.

Scope

This thesis covers the following:

- Develop an algorithm to compute the appropriateness of a constellation for the mission.
- Study some analytical initial guesses.
- Run an optimizer algorithm and decide the final(s) option(s).

Background

This thesis has been developed in a six-month internship at the French company Thales Alenia Space. Some of the work will help contribute further related projects.

Disclaimer: sensitive information in this thesis has been modified and/or redacted.

Chapter 2

Description of the project

2.1 Requirements

- Coverage of at least 95 % of the global surface.
- Revisit time no superior than 2 hours.

2.2 Constraints

- Satellite elevation must be at least 60^o.
- LEO Constellation.

2.3 Objective

- Minimize the total cost of implementing the constellation.

2.4 State of the Art

Earth Observation satellites are not a new concept. The very first satellite ever launched, Sputnik 1, on October 4, 1957, although not essentially for observation, it was useful for collecting data of the ionosphere. This concept then developed into weather satellites, the first of which was launched on April 1, 1960 with the name TIROS (Television Infra-Red Observation Satellite).

Nowadays, some remarkable satellites in Earth Observation missions are the following. From ESA, the Sentinel series are being carried out, as part of the Copernicus Programme, consisting of a group of seven satellites successively deployed. They are not considered a constellation though, as each of them has a different goal (for instance, Sentinel-2 provides land imaging services, whereas Sentinel-3 provides ocean monitoring, and Sentinel-4, athmospheric data).

Another collaborative project is the A-train, a string of satellites located few minutes apart, whose collective data helps achieve high-resolution images.

All of the cited missions are on a Low Earth Orbit, about 700 or 800 km of altitude, and in a polar orbit, in order to reach global coverage. Most of them are also in Sun-synchronous orbits.

2.5 ONION

ONION (standing for Operational Network of Individual Observation Nodes) [1] is a collaborative project, coordinated by Thales Alenia Space. One of its main objectives is to analyse the emerging needs in the European Earth Observation market, and to identify and characterise solutions to meet these needs. A few examples of these needs are, land monitoring, atmospheric measurements, water quality, maritime surveillance, emergency management and security.

Space-based Earth Observation brings a lot of benefits when compared to ground based methods, such as the coverage, which is potentially global, and a better resolution. Part of the project is also to study the practicalities of Fractionated and Federated Satellite Systems (FFSS), which are a group of satellites, closely flying, but each one with and independent mission. This approach makes it easier to upgrade a specific part, without having to decouple the whole system, increases the responsiveness to a failure and the compatibility (exchange of resources) between systems. This comes with the cost of a more complex design.

2.6 Motivation

After the study of the Earth Observation needs in the market, three clear needs were found. Coordinated Observations, Low-latency observation data, and High-Resolution data.

This comes from the fact that EO satellites have a revisit time of several days. With a better temporal resolution, a new approach into disaster prevention, such as tsunamis or wildfires, could be possible by satellites. At the present time other methods are used. For instance, countries in areas of tsunami risks have monitoring bouys located at strategic spots. Fires have detection methods too, most of them requiring to install infrastructure at the very same forest. [2]

A clear inconvenient to this are the costs (both economical and environmental) it would require to build such infrastructures in every single place to be monitored, whereas global coverage would be achieved by *just* deploying a single satellite constellation.

Said constellation should have all its satellites synchronized with each other, in order to share the information they gather, and to optimally cover all Earth. Finally, high-resolution is a *must*, so that the images have as much quality as possible.

This thesis focuses on the work performed to design an architecture with the goal of reaching low-latency, near-global coverage, optimizing the cost.

Chapter 3

Main concepts

3.1 Orbit theory

In this section, some basic definitions of satellite orbits are going to be introduced in order to correctly understand the concepts that will later be exposed when dealing with the constellation orbits.

To understand the movement in space it is useful to take a look at Kepler's laws. [3]

3.1.1 Kepler's laws of planetary motion

Johannes Kepler was a German astronomyst, known for his laws of planetary motion. They are as follows:

1. *The orbit of a planet is an ellipse with the Sun at one of the two foci.*
2. *A line segment joining a planet and the Sun sweeps out equal areas during equal intervals of time.*
3. *The square of the orbital period of a planet is directly proportional to the cube of the semi-major axis of its orbit.*

The first law can be expanded to the orbit of a satellite with the Earth at one pf the two foci. It could also by represented by any conic section (ellipse, parabola, or hyperbola). However, only an elliptical orbit is of interest for Earth Observation missions, as the other two represent orbits escaping from Earth.

Figure 3.1 and table 3.1 show the main parameters of an ellipse and their relationships, they will be important when considering the shape of an orbit.

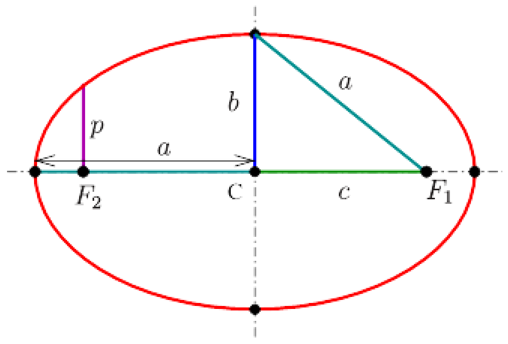


Figure 3.1: Shape parameters of an ellipse

Symbol	**Name**	**Definition**
a	Semi-major axis	One half of the major axis.
b	Semi-major axis	One half of the minor axis.
c	Linear eccentricity	Distance between the geometric center and one of the two foci.
e	Eccentricity	A measure of orbital deviation from a perfect circle.
F	Focus	Considering an orbit, the Earth is located in one of the two foci.
p	Semi-latus rectum	One half of the latus rectum (chord perpendicular to the foci).

Table 3.1: Shape parameters of an ellipse

The second orbit makes reference to the speed of the orbiting object, it makes clear that it goes faster the closer it is to the central object, and vice-versa. An equation can be obtained for the precise orbital speed of a body at any given point in its trajectory:

$$v = \sqrt{\mu\left(\frac{2}{d} - \frac{1}{a}\right)} \tag{3.1}$$

Where μ is the gravitational parameter, d is the distance from the satellite to the center of the central body, and a is the semi major axis of the ellipse. In case of circular orbits where $a = d =$ constant:

$$v = \sqrt{\frac{\mu}{a}} \tag{3.2}$$

The third law provides a relationship between the period and the semi-major axis. It was first found empirically, and later on developed into the actual formula:

$$T^2 = \frac{4\pi^2}{\mu}a^3 \tag{3.3}$$

This last relation is really useful when designing orbits whose period (or cycles of periods) matches the rotation of the Earth itself.

3.1.2 Orbit definition

There are many ways to define an orbit, however, one of the most used systems is the keplerian elements. In consists in an pseudo-inertial reference system called *geocentric-equatorial* frame. The XY planes coincides with the Equatorial plane, with the X axis pointing at the direction of the vernal equinox. The Z axis, by applying the right-hand rule, points at the north, and corresponds to the Earth rotation axis.

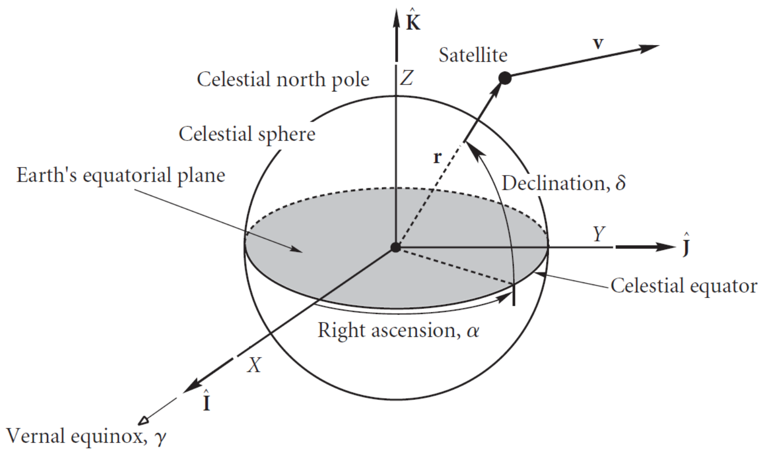


Figure 3.2: Geocentric Equatorial Frame [4]

Once this reference system is introduced, the keplerian elements are as follows:

- **Semi major axis (a):** It is related to the size of the orbit and it is defined by the sum of the apogee (furthest point from Earth) and the perigee (nearest point) divided by two.
- **Eccentricity (e):** It defines the shape of the orbit, $e = 0$ corresponding to a perfect circle, $0 < e < 1$ to an ellipse, $e = 1$ to a parabola, and $e > 1$ to a hyperbola.
- **Inclination (i):** It is the angle between the positive Z axis and the angular momentum vector, which is perpendicular to the orbital plane. For $0^o \leq i < 90^o$ the motion is prograde, and for $90^o < i \leq 180^o$ the motion is retrograde.
- **Right ascension of the ascending node (Ω):** It is the angle, in counter-clockwise direction, between the reference direction (vernal equinox) and the intersection of the orbital plane with the equatorial plane, at the moment it passes from the Souther to the Northern Hemisphere.
- **Perigee argument (ω):** It is the angle between the ascending node and the perigee. Measured in the orbit's motion direction.
- **True anomaly (ν):** Used to describe the satellite instantaneous position, it is the angle between the perigee and the satellite instantaneous position. Measured in the orbit motion direction.

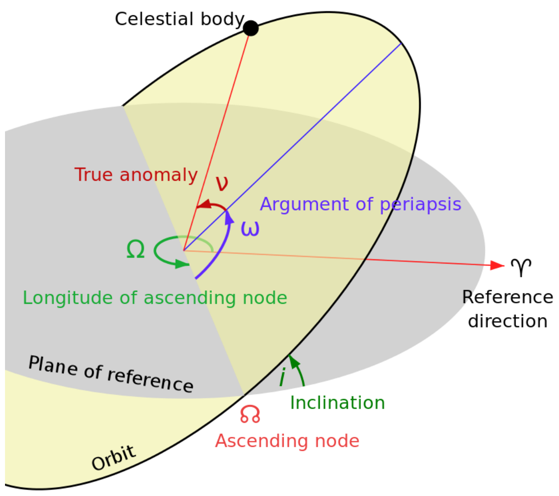


Figure 3.3: Graphical representation of the keplerian elements. [5]

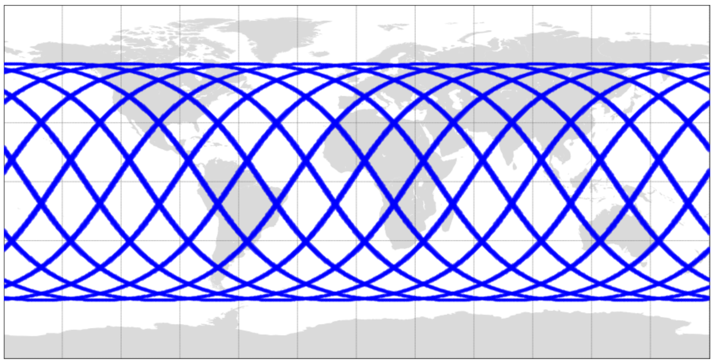
Another interesting parameter to take into account is the epoch, which is the point in time taken as a reference for these elements.

Specifically related to satellites, the following concepts are also introduced.

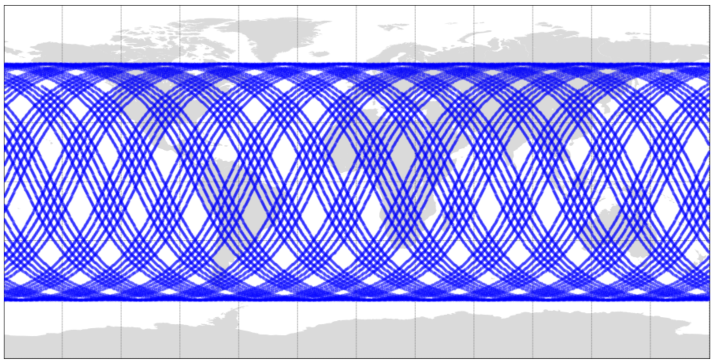
- **Ground track**: the direct projection of the satellite trajectory onto the Earth surface. It can also be understood as the set of points at which the satellite will pass directly overhead. It can repeat itself over a cycle, or taking different paths each time. At the first case, the number of revolutions it has to make around the Earth is defined as the Repeating Ground Track (RGT) ratio (see figure 3.4). [6]
- **Elevation angle** (ε): the minimum angle (measured from the horizon) at which, from a given point of earth, the satellite can be detected. And vice-versa, from the satellite, which points on Earth are covered in an instantaneous moment (see figure 3.5).
- **Footprint**: the total area that is covered by the satellite at a given moment.
- **Altitude (h):** the difference between the semi-major axis (a) and the Earth radius (R_E). More easy to conceptually understand than the semi-major axis, but source to a lot of confusions. Unless indicated otherwise, we are generally going to treat with altitude parameters. The altitude allows it to classify an orbit in three types (see table 3.2).

Name	**Abreviation**	**Altitude range [km]**
Low Earth Orbit	LEO	200 - 2000
Medium Earth Orbit	MEO	2000 - 35 785
Geostationary Orbit	GEO	35 786

Table 3.2: Classification of orbits in regards of the altitude



(a) Repeating Ground Track



(b) Non-Repeating Ground Track

Figure 3.4: Representation of different ground tracks:
(a) repeats exactly the same pattern after 12 revolutions, whereas (b) slowly shifts westwards.

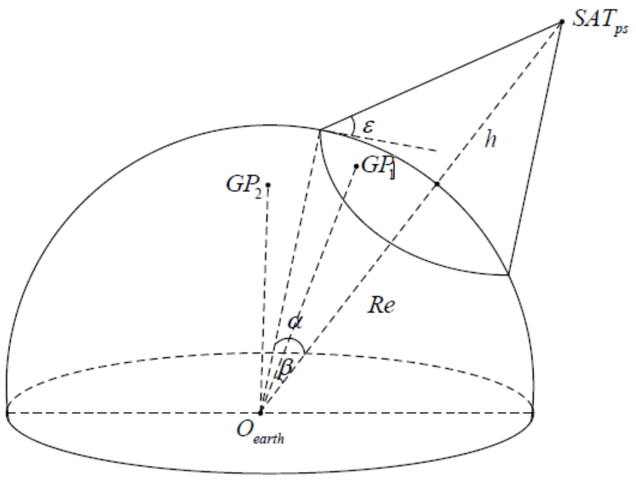


Figure 3.5: Example of elevation angle. GP_1 is covered by the satellite, but GP_2 is not. [7]

3.1.3 Perturbations

The previous formulation makes some assumptions that make it a bit different from reality. Some corrections should be made to take into account the effects due to more bodies attracting the satellite (Sun, Moon, etc.), the existence of other forces (atmospheric drag, radiation pressure, etc.), or the Earth not being a perfect sphere, among others. [8]

Perturbations is an obscure field, most of the models come from empirical observations. It is difficult to get an analytical form, and even so some empirical factors will appear. The following perturbation sources will be introduced at a minimal basis, more in a conceptual level than practical.

Athmosferic drag

If the orbit is low enough, the atmospheric drag will contribute highly in orbit decay, reducing its semi-major axis (and thus altitude) to the point that if it is not corrected, the satellite would reenter the atmosphere and its lifetime would cease.

The formulation of atmospheric drag equations is plagued with uncertainties of atmospheric fluctuations, the drag coefficient, different cross-sections afected (depending on the orientation of the satellite) and other parameters.

A fairly simple formulation will be given here, based on the aerodynamics theory. Drag, by definition, is opposite to the velocity of the vehicle. The atmospheric deceleration effect is as follows:

$$\dot{v} = -\frac{1}{2}C_D\frac{A}{m}\rho v^2 \tag{3.4}$$

Where C_D is the drag coefficient, A is the cross-section area of the satellite, normal to the direction of motion, m is the mass of the object, ρ is the air density at that altitude, and v is the orbital velocity relative to the air.

This equation complicates even more if one takes into account the theoretical density given by ISA (International Standard Atmosphere), the effects of Earth oblateness, lifting forces, etc.

In general, it is enough to overcome this forces by generating regularly a little thrust, in order to get the satellite back to orbit.

Radiation pressure

Radiation pressure, though small, can have considerable effect on satellites with a large area-mass ratio satellites. In the geocentric, equatorial coordinate system the accelerations caused by such perturbation are:

$$\ddot{x} = f \cos A_o \tag{3.5}$$

$$\ddot{y} = f \cos i \sin A_o \tag{3.6}$$

$$\ddot{z} = f \sin i \sin A_o \tag{3.7}$$

Where f is a shape parameter, i is Earth's axis inclination, and A_o is the mean right ascension of the sun during the perturbation.

Again, the formula is very ambiguous and depends a lot on empirical data.

Oblateness of the Earth

This is probably the most relevant perturbation phenomenon, for our purposes.

The Earth is not a perfectly spherical body but is bulged at the equator, flattened at the poles and is generally asymmetric. If its real shape is known, the gravity real effects can be found.

Some institutions like NASA or ESA, defined the geopotential model of the Earth by means of the Legendre polynomials, where the zero-degree polynomial would correspond to the perfect sphere and the higher-order polynomials, to its harmonics. Table 3.3 shows the most relevant coefficients for planet Earth, according to the Earth Gravitational Model 1996 (EGM96). [9]

J_n	**Value**
2	$1.082\,626\,683\,553\,15 \times 10^{-3}$
3	$-2.532\,656\,485\,332\,24 \times 10^{-6}$
4	$-1.619\,621\,591\,367 \times 10^{-6}$
5	$-2.272\,960\,828\,686\,98 \times 10^{-7}$
6	$5.406\,812\,391\,070\,85 \times 10^{-7}$

Table 3.3: Zonal coefficients, by EGM96 model

The second order coefficient is at least three orders of magnitude more relevant than all the others.

This oblateness creates a torque that affects the angular momentum of the orbit, causing it to gyroscopically precess, eventually moving the ascending node to the east or to the west. The equation describing the rate of precession caused by this bulge is given by:

$$\dot{\Omega} = -\frac{3}{2} J_2 \left(\frac{r_E}{p}\right)^2 \sqrt{\frac{\mu_E}{a^3}} \cos i \tag{3.8}$$

Where J_2 is the zonal coefficient recently mentioned, r_E and μ_E are Earth radius and gravitational parameter, respectively, a, i and Ω are orbital elements, and p is the semilatus rectum, with the expression $p = a(1 - e^2)$.

3.1.4 Sun-synchronous orbits

An interesting type of orbit is the sun-synchronous orbit, widely used in Earth Observation missions for its unique characteristics. [10] By making it precess exactly 360° each sidereal year [1], it maintains the same relationship with the Sun. The immediate consequence is that a satellite in a sun-synchronous orbit will pass over a point always at the same local mean solar time.

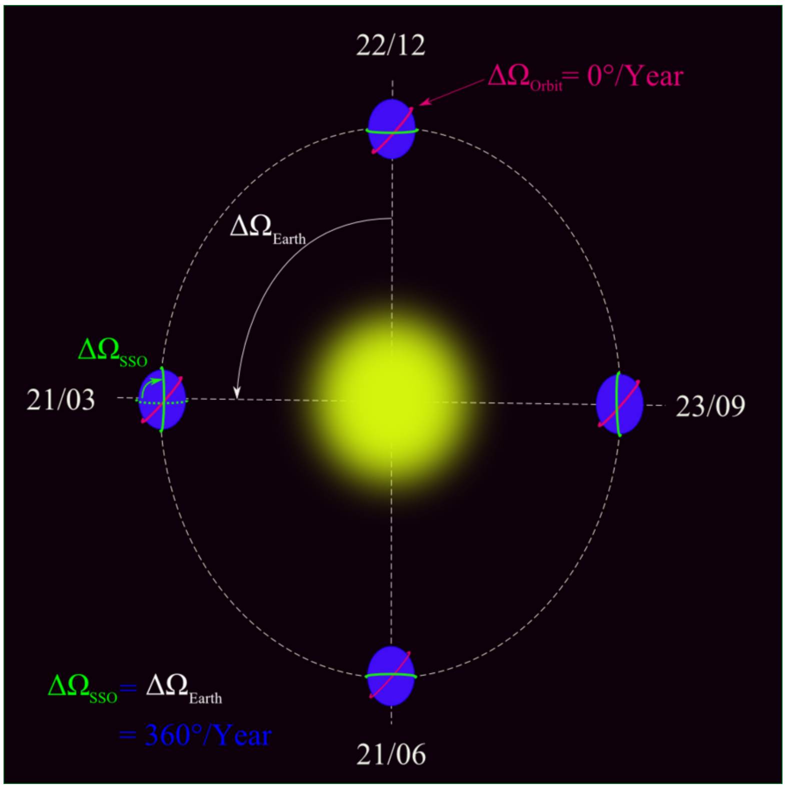

22/12
$\Delta\Omega_{\text{Orbit}} = 0°/\text{Year}$
$\Delta\Omega_{\text{Earth}}$
$\Delta\Omega_{\text{SSO}}$
21/03
23/09
$\Delta\Omega_{\text{SSO}} = \Delta\Omega_{\text{Earth}}$
$= 360°/\text{Year}$
21/06

Figure 3.6: Example of sun-synchronous orbit. The red orbit keeps the same orientation relative to the inertial frame. The green one keeps the same position relative to the Sun [11]

A sun-synchronous orbit can be designed by matching $\dot{\Omega}$ to the Earth translation rate:

$$\dot{\Omega} = \frac{360^{\circ}}{365.242\,199\,\text{days}} = 1.991\,063 \times 10^{-7}\,\text{rad}\,\text{s}^{-1} \tag{3.9}$$

Assuming that Low Earth Orbits are almost elliptical ($a \approx p$), taking (3.8) and rearranging:

$$\cos i = -\frac{\dot{\Omega}\, a^{7/2}}{3/2\; J_2\; r_E^2\, \sqrt{\mu_E}} \tag{3.10}$$

[1] 365.242 199 solar days

Where J_2, r_E and μ_E are Earth parameters, and $\dot{\Omega}$ is the desired precess ratio, leaving only the semi-major axis a as a variable. This pairs the orbit altitude and inclination in a unique way, making the inclination automatically determined by the altitude location. Figure 3.7 shows this pairing for the most typical LEO altitudes:

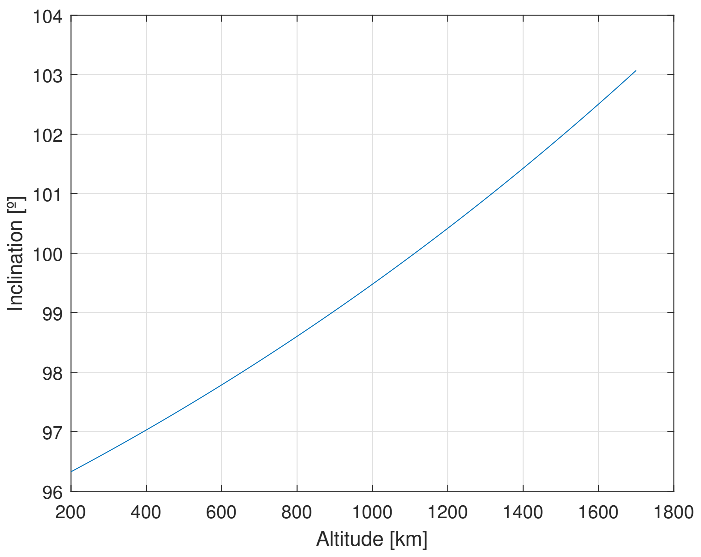


Figure 3.7: Inclination in function of altitude

Another handy conclusion is that the Earth rotation around the Sun and the orbital precession will cancel out, making it that the projection of any orbital node on the Earth will be at the same point after exactly 24 hours. In addition, if the satellite completes an integer number of revolutions in an integer number of days, the ground track will repeat a regular pattern, with each point having the same sunlight conditions than in the previous cycle.

However, one drawback that this type of orbits have is that they are in retrograde motion, which implies a costly change of plane.

3.2 Satellite constellations

A constellation can be defined as the set of satellites working together to provide a combined coverage area. There is no defined common process for constellation design since it is a process that varies significantly with the mission objectives. In general, it is desired to find the minimum number of satellites that accomplish the mission objectives. However, other parameters, such as the number of planes, or the inclination, may increase the implementing cost.

To make it easier to study them, we can divide them into constellation with homogeneous patterns, and constellations with heterogeneous patterns, the latter of which have a much higher variability. Homogeneous patterns include Walker-Delta, near-polar (Walker Star) constellations, and elliptical orbit constellations. Heterogeneous patterns are combinations or variations of homogeneous patterns. [12]

3.2.1 Geosynchronous constellations

As the name implies, it consists of satellites in a geosynchronous orbit, or Geostationary Earth Orbit (GEO), always facing the same point. It is achieved by taking an equatorial orbit, and choosing an altitude so that the satellite period around the Earth matches the duration of a sidereal day [2]. By taking the third law of Kepler (3.3):

$$a = \sqrt[3]{\frac{\mu}{4\pi^2}T^2} = 42\,164\,\text{km} \tag{3.11}$$

The altitude is:

$$h = a - R_E = 42\,164 - 6378 = 35\,786\,\text{km} \tag{3.12}$$

It only needs three satellites to cover almost the entire globe, and the fact that they appear stationary in relation to the ground makes it very useful when a specific area wants to be covered. The most known example is Meteosat.

However, they have a few cons which make them unsuitable for Earth Observation. The distance is too big to have good-quality images, and the polar regions cannot be covered well. Another problem is that there is a limited number of lots available en GEO, as they require a very specific orbit parameters.

3.2.2 Polar constellations

Also known as Walker-Star (due to the shape the ground tracks create at the poles). It solves one of the main problems of GEO. By using orbit inclinations of 90° (or near), one can make sure that the polar regions will be covered, and then use the "Streets of Coverage" method (figure 3.8) at the Equator to figure out the number of planes and satellites necessary to achieve global coverage.

Its strict definition requires dividing the Earth longitudinally into two imaginary hemispheres, where the satellites at one side would travel northwards and the satellites at the other side would travel southwards. However, any constellation consisting purely of polar or near polar orbits could be considered as such. [13]

[2]Full rotation of the Earth relative to the inertial frame instead of the Sun. Slightly shorter than the solar day.

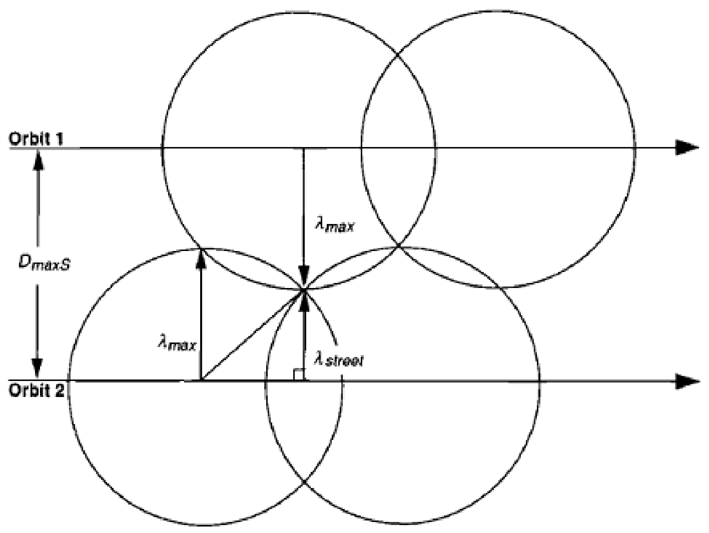


Figure 3.8: Simple example of applying the "Streets of Coverage" method. [14]

The main advantage of polar constellations is that a global coverage is guaranteed, given any altitude and elevation. A clear example is the Iridium constellation, whose inclination is around 86.4 degrees.

Some problems with these constellations are the following: the number of satellites is really big, accounting for a in-the-limit coverage for the low latitudes, while the polar latitudes are highly covered, producing a big mismatch among different areas, and the big inclination makes it very costly to reach the mentioned orbits, not forgetting the risk of collision at the poles, where all the orbits converge. [15] A solution proposed is to decrease the inclination, creating the Walker-Delta constellations.

3.2.3 Walker-Delta constellations

Deeply studied by John Walker, Walker-Delta constellations consists on symmetric quasi-circular orbits, sharing the same radius and inclination with the same number of satellites on every plane.

This high symmetry leads to the satellites taking similar patterns during each orbital period, and makes all the satellites undergo similar perturbation effects; which makes it more simple to study the different possible designs. John Walker introduced [16] the notation $i : T/P/F$ where each parameter represents:

- i: the constellation inclination
- T: the total number of satellites
- P: the total number of orbital planes
- F: the relative phasing between satellites in adjacent planes

With another parameter, the number of satellites per plane (S), being easily computable by the formula $S = T/P$.

F can take any integer value from 0 to $P-1$, and the exact phase difference between adjacent planes (expressed in degrees) is defined as $\varphi = F \cdot 360^o/T$. A more intuitive way to describe it is by considering $\Phi = 360^o/S$ as the phase difference between satellites in the *same* plane, making it $\varphi = \frac{F}{P}\Phi$, where F/P would take the values $F/P = \{0,\ 1/P,\ 2/P \ldots (P-2)/P,\ (P-1)/P\}$.

One of the most relevant examples of a constellation using the Walker notation is Galileo, [17] the European global navigation system, whose parameters are $56^o : 24/3/1$, this means, a total of 24 satellites distributed in three equidistant planes, with an inclination of 56 degrees, the phasing between contiguous planes is $\varphi = 1 \cdot 360^o/24 = 15^o$.

3.2.4 Elliptical constellations

All the previous were assuming circular orbits, which in general are more easy to study and to implement. However, highly elliptical orbits have advantages too. [18]

By taking advantage of the fact that the velocity at the apogee is much smaller than at the perigee, and by taking a high inclination, one can cover high-latitude areas for long periods of time, and with a number as little as three satellites, full-time coverage is reached. It is not surprising then, that a country with so many area in high latitudes as Russia was one of the first to implement it in a practical way, what is known as Molniya orbit, carefully designed so that the perturbations cancel each other, staying stationary (a *frozen* orbit). [19]

3.2.5 Heterogeneous systems

So far, all the previous examples consisted on all the satellites sharing the same shape elements and inclination, a part from following regular patterns. One can make heterogeneous systems by taking some homogeneous ones, and combining their advantages.

For instance, with a couple of geostationary satellites covering the equatorial regions and two sets of Molniya-orbiting satellites (one for each hemisphere) global coverage is reached.

Another studied case, is generating a Walker constellation but omitting some planes and instead covering them with one or more polar orbits. This approach is understood as a way to complement each other's flaws; while the first fails to reach the polar regions, the latter over-covers them.

It should be taken into consideration whether the different homogeneous patterns act in a totally independent way, or they need some sort of coordination. In the latter case, one of the main complication of heterogeneous constellation arise; perturbations, as little as they may be, will act differently in each element.

For instance, orbit decay caused by friction would affect equally all satellites in a Walker-Delta orbit, and the same correction -if any- should by applied, but if different altitudes and eccentricities are in play, this would completely desynchronize the constellation. Thus, a thorough perturbation study must be carried out when dealing with heterogeneity.

3.2.6 Real-world examples

Name	**Mission**	**Number of satellites**	**Orbit type**	**Inclination [º]**
GPS (USA)	Navigation	31	Circular MEO	56
GLONASS (Russia)	Navigation	24	Circular MEO	∼65
Galileo (EU)	Navigation	24	Circular MEO	55
Iridium	Communication	76	Polar LEO	86
Globalstar	Communication (TV)	48	Circular MEO	52
DMC	Monitoring	4	Sun-synchronous LEO	98.2
RapidEye	Earth Observation	5	Sun-synchronous LEO	97.8
A-Train	Earth Observation	4	Sun-synchronous LEO	98.1

Table 3.4: Real-world examples of deployed constellations

Table 3.4 shows known examples of constellations that are used nowadays. They can be classified in three general groups: navigation, communications an Earth Observation.

The three navigation systems listed are practically the same. They have global coverage and they provide navigation services, but even the orbital characteristics are similar. It is probably for political reasons that each country keeps its own navigation system, coincidentally the two big powers of the Cold War. Moreover, the European Union (and China) began developing its own system recently, and it is due to be fully functional next year.

Satellites for communications were among the first to be launched, and still it is one of the principal missions of a satellite. There is a lot of variety in its orbits, ranging from LEO, to even GEO for broadcasting satellites. They may have global or just regional coverage, depending on its purpose.

It is worth noting that most of the Earth Observation satellites are in a Sun-synchronous LEO. This is because of the requirements in terms of illumination conditions and ground resolution. In the other hand, the number of satellites is far lesser than in the other mentioned cases.

Knowing this information will be useful at the time of designing our constellation regarding the mission objectives, which in our case is Earth Observation.

3.2.7 Coverage

There has been plenty of research on Earth coverage analysis. From a methodological point of view they can be classified as numerical or analytical methods. The most commonly used method for satellite coverage is the grid-point method, which is going to be fully explained in Chapter 4.

There is no general way to compute a constellation coverage purely by analytical methods. Analytical methods are only used when dealing with a single satellite and regular target areas (e.g. a single point, a altitude or longitude line). Although they are exact over time for a single satellite (provided the orbital parameters are correct), they are not suitable for calculating the coverage capacity of satellite constellations for any specified ground areas. [20]

3.3 Delta-v

An interesting concept concerning space mechanics is the delta-v.

Orbital maneuvers are necessary to transfer a spacecraft from one orbit to another. They can completely change the orbit, or they can just adjust the orbit to the desired parameters. Nevertheless, the fuel necessary for these maneuvers is an important part regarding the whole of the mission than needs to be accounted.

A practical way to approach is to treat each maneuver one at a time, and consider it instantaneous. During that impulsive maneuver, the velocity vector of the spacecraft will have changed. By computing the increment in the velocity, and adding up the values of all the maneuvers, we can approximate the cost of putting that object in orbit.

$$\frac{\Delta m}{m} = 1 - e^{-\frac{\Delta v}{I_{sp} g_0}} \tag{3.13}$$

Where, I_{sp} the specific impulse of the rocket, and g_0 is the gravity acceleration at sea-level. A clear observation from (3.13) is that if the delta-v or the total mass increases, the fuel also increases (and thus, the total cost of the mission).

Due to the characteristics of our mission, we will first consider the fuel necessary to launch the satellite from the Earth to a LEO orbit, and then into the exact orbit of the constellation.

Launch into LEO

It is not clear the delta-v needed to put a satellite in orbit. A lot of factors have to be taken into consideration such as the launch location, the type of propeller, whether or not they have heat shields, but a good approximation would be 9.3-10 km/s. [21] We are going to do some small calculation to reach some values on our own.

Using the gravitational potential equation:

$$V = -\frac{\mu_E}{r} \tag{3.14}$$

Considering that the increment in gravitational potential equals the kinetic energy per unit mass, it is possible to compute the total delta-v required to reach that altitude:

$$\Delta V = -\frac{\mu_E}{r_E + h} - \left(-\frac{\mu_E}{r_E}\right) = \frac{1}{2}\Delta v^2 \tag{3.15}$$

Rearranging:

$$\Delta v_0 = \sqrt{2\frac{\mu_E h}{R(R+h)}} \tag{3.16}$$

And then, the orbital velocity to maintain a circular orbit at that altitude:

$$\Delta v_1 = \sqrt{\frac{\mu_E}{r_E + h}} \tag{3.17}$$

Taking (3.16) and (3.17), figure 3.9 shows the delta-v budget in terms of the desired altitude, within the LEO range:

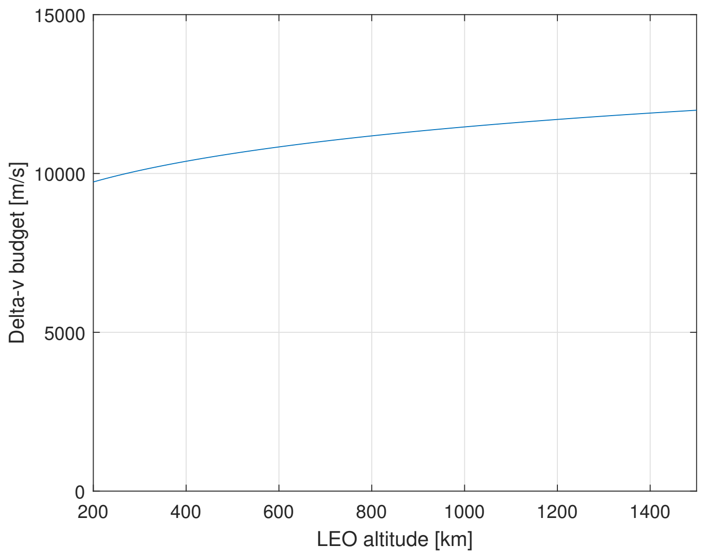


Figure 3.9: Delta-v budget in function of altitude

As we can see, the delta-v values from the lower altitudes fairly match what the literature says, whereas it gets slowly higher.

However, this a rough approach omitting a lot of factors. So, in order to ease the calculations, we are going to suppose that a launch into LEO (regardless of the altitude) requires a delta-v budget of $\Delta v = 10\,000\,\mathrm{m/s}$

Equatorial LEO into inclined LEO

Once LEO is reached, assuming the satellite was launched from Guiana Space Center (located at a latitude of 5°), the orbit will be nearly equatorial, and almost always a change in inclination will be necessary to reach the desired orbit.

The space mechanics theory provides the following equation for inclination changes, assuming circular orbits [4]:

$$\Delta v_2 = 2v\sin(\Delta i/2) \tag{3.18}$$

Where v is the same velocity as in (3.17).

The other parameters such as perigee argument, right ascension of ascending node, and true anomaly, are way less relevant in terms of delta-v, and depend more on the specific moment of performing such maneuver, so they could be considered more time-consuming than fuel-consuming.

To sum up, the delta-v budget for a whole constellation to be put into orbit can be simplified as:

$$\boxed{\Delta v_{total} = n\left(10\,000 + 2\sqrt{\frac{\mu_E}{r_E + h}}\sin(i/2)\right)} \tag{3.19}$$

Where n is the total number of satellites, μ_E and r_E are the Earth gravitational parameter and radius, respectively, and h and i are the orbit altitude and inclination, respectively.

Chapter 4

Setup

This chapter will explain the procedure to design the constellation. It was decided to create a program that would simulate the performance of each constellation, focusing on the main objectives, which are coverage and revisit time.

4.1 Software

Before taking any step, it was discussed which specific software would be more suitable. Some options were available:

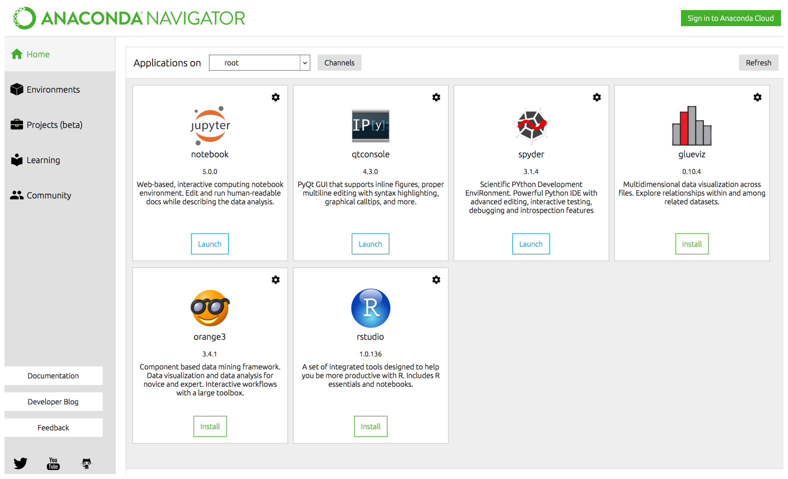
- **Python:** a programming language, available in open source. Its large library, providing tools to do a lot of different tasks, is one of its main strengths. It counts with a community that continuously provides with more libraries and packages. It also has compilers, such as Jython or JCC, that allow running other-language programs in Python.
- **MATLAB:** widely used in the engineering field, it is a numerical computing environment. Its main capability is to do operations with matrices, but also plottings of functions, implementation of algorithms and data analysis. A paid license is needed.
- **Systems Tool Kit (STK):** pysics-based software package. Formerly known as *Satelites Tool Kit*, it provides a great range of tools to solve problems related to space. A free version is available, but most complex modules are under pay.

Eventually, it was decided to use **Python** for the following reasons: it is a universally known computing language, it is easy to implement it in any computer and there is no copyright or licensing issues.

It should be noted that, although existing MATLAB licenses for students, the aim of the project was to develop a program available for non-students users. For instance, it has not been used at the development of the constellation itself, but for the plotting of some figures at the post-process.

To sum up, a program would be configured from scratch in the Python language, specifically using the **Anaconda** distribution. [22]

Anaconda, apart from being a Python distribution, includes an interface named Anaconda Navigator (figure 4.1), providing access to JupyterLab. The latter is a browser-based tool enabling interactive manipulation of the code, automatic highlight, indenting, and specially, the ability to execute the code by parts, and having the results associated to the part that generated them.



Web-based, interactive computing notebook environment. Edit and run human-readable docs while describing the data analysis.

PyQt GUI that supports inline figures, proper multiline editing with syntax highlighting, graphical calltips, and more.

Scientific PYthon Development EnviRonment. Powerful Python IDE with advanced editing, interactive testing, debugging and introspection features

Multidimensional data visualization across files. Explore relationships within and among related datasets.

Component based data mining framework. Data visualization and data analysis for novice and expert. Interactive workflows with a large toolbox.

A set of integrated tools designed to help you be more productive with R. Includes R essentials and notebooks.

Figure 4.1: Screenshot of Anaconda Navigator interface

Anaconda Prompt, on the other hand, is the terminal of Anaconda, where the Python scripts can be called and where new packages can be installed. One of those packages is Orekit.

4.1.1 Orekit

Orekit is a library originally developed for Java. It contains a low level space dynamics library, which provides the user with a big number of tools, useful for different purposes. [23]

It has been used specially by the Centre National d'Études Spatiales (CNES, the French space agency) for some critical missions, and by ISAE-SUPAERO, with academic goals.

Orekit includes the following packages, figure 4.2 shows the relationships between them:

- `attitudes`: provides classes to represent simple attitudes.
- `bodies`: provides an interface to representation of the position and geometry of space objects such as stars, planets or asteroids.
- `errors`: provides classes to generate and handle errors, based on the classical mechanism of exceptions.
- `estimation`: provides classes to manage orbit determination.

- `forces`: provides the interface for the force models to be used by a numerical propagator. It includes forces such as atmospheric drag, third body gravity force, radiation pressure, and more.
- `frames`: provides classes to handle frames and transforms between them. Includes for example, geocentric frames defined by the IERS Conventions. [24]
- `orbits`: provides classes to represent orbits in four different ways. It is the basis for all of the other space mechanics tools.
- `propagation`: provides tools to propagate orbital states with different methods, either numerically or analytically. Includes the `ElevationDetector` tool, which is going to be useful for our project.
- `tessellation`: provides tools to discretize geographical zones defined over an ellipsoid.
- `time`: independent package providing classes to handle epochs and time scales, and to compare instants.
- `tle`: provides classes to read, and extrapolate from, orbital elements in Two-Line Elements (TLE) format.
- `utils`: provides methods for managing mathematical or geometrical objects. Including different physical constants.

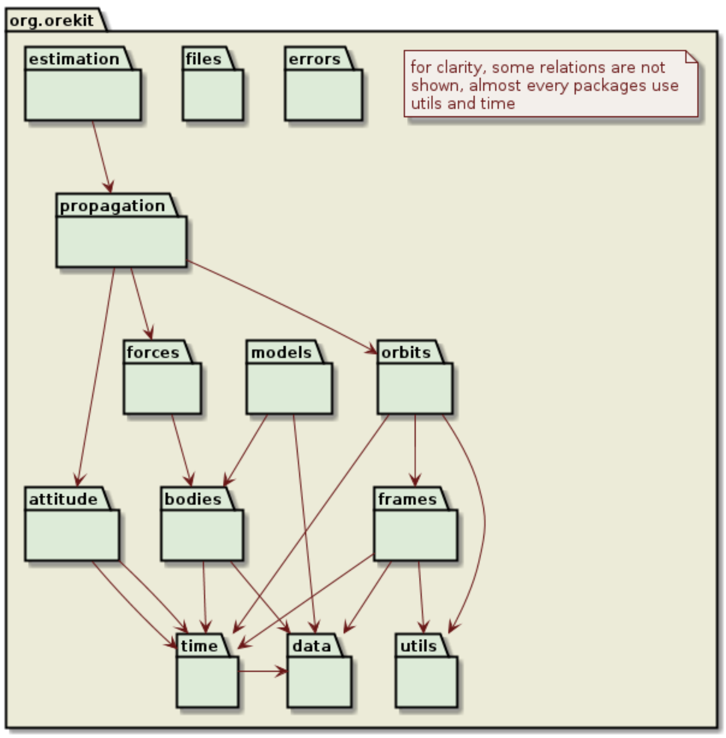
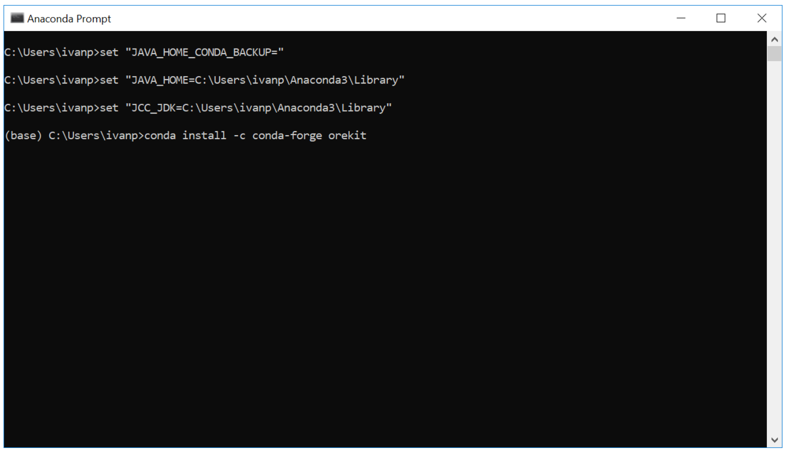


Figure 4.2: Orekit packages

4.1.2 Orekit Python wrappper

Due to the robustness of Orekit and its multiple applications, a Python wrapper was developed and put available to the public, so as to make it accessible for Python users. It consists on a Python-Java bridge, based on the compilator JCC. The commands are mainly the same, but adapted to the Python language. However, a little knowledge of Java is advised.

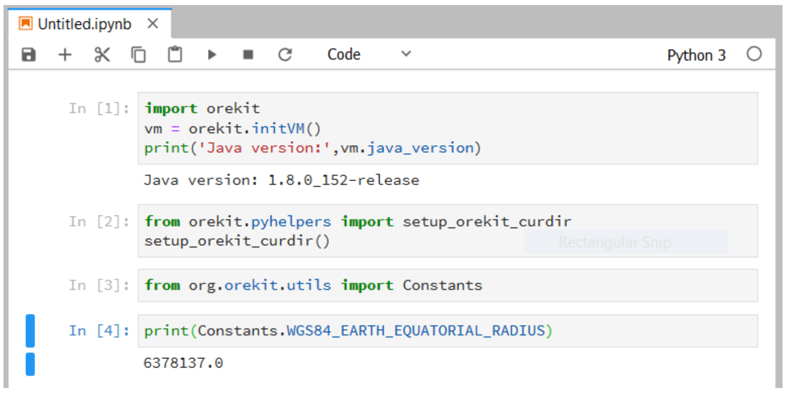
To install it into the Anaconda distribution, one has to go to the "Anaconda Promt" and then input the command `install -c conda-forge orekit`, as figure 4.3 shows:



```
C:\Users\ivanp>set "JAVA_HOME_CONDA_BACKUP="

C:\Users\ivanp>set "JAVA_HOME=C:\Users\ivanp\Anaconda3\Library"

C:\Users\ivanp>set "JCC_JDK=C:\Users\ivanp\Anaconda3\Library"

(base) C:\Users\ivanp>conda install -c conda-forge orekit
```

Figure 4.3: Installation of Orekit in the Anaconda distribution

A data file called `orekit-data.zip`, containing ephemerides and Earth parameters, has to be manually downloaded from its website, and it must be located at the same directory as the scripts are.



```
In [1]: import orekit
        vm = orekit.initVM()
        print('Java version:',vm.java_version)

        Java version: 1.8.0_152-release

In [2]: from orekit.pyhelpers import setup_orekit_curdir
        setup_orekit_curdir()

In [3]: from org.orekit.utils import Constants

In [4]: print(Constants.WGS84_EARTH_EQUATORIAL_RADIUS)

        6378137.0
```

Figure 4.4: JupyterLab interface, showing the first steps to import the Orekit packages

4.2 System model

As mentioned before, we are going to use the grid-point method to numerically simulate the constellation, and check if it meets the requirements. The ultimate goal of the model is to compute the revisit time around the Earth, given any constellation of satellites.

Earth is going to be divided into a discrete range of points, uniformly distributed along the latitude and longitude coordinates, it must be taken into consideration that the points will not be uniformly distributed considering the whole globe as a sphere (point concentration will increase at higher latitudes). The main concern with that is, in terms of coverage, high-latitude points will have a smaller area than equatorial ones, and a correction must be made.

Afterwards, by combining all the results, the global behaviour of the constellation can be studied.

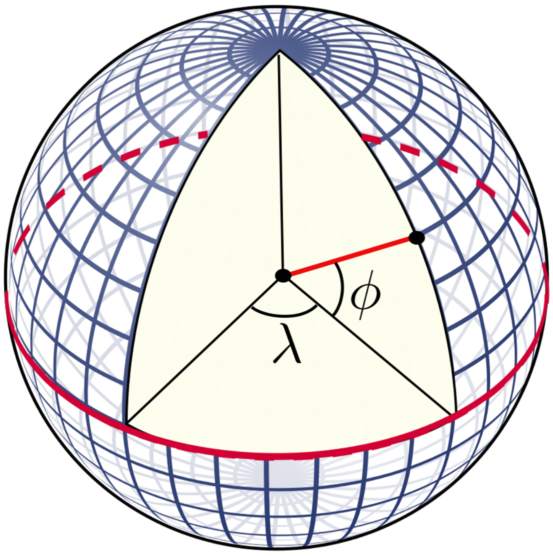


Figure 4.5: Graticule of the Earth, showing ϕ as the latitude and λ as the longitude. The grid resolution is 10 degrees for both magnitudes.

Once all the grid-points have been clearly defined, it needs to be solved how often they "see" a satellite.

Provided that the constellation data has been introduced (either as its Walker parameters, or as the keplerian elements for each satellite), all the satellite orbits are propagated one at a time, with a duration of several times the revisit time target, in order to have the points visited more than once, and achieve more accurate results.

During the propagation, it should be recorded each time a point is under the coverage of the satellite that is being propagated. This is done by adding an "event detector"; if said points are under the coverage of the satellite, the event is triggered, and its trigger-time is stored in a big list known as "logger", that stores all the visiting times for all the points in the grid.

Each time an orbit is propagated, the "logger" variable increases size. Once all the orbits are propagated, the values (visiting times) in each point are in the same order as the satellites were propagated, so they must be sorted in temporal order, in order for the algorithm to work.

The final step is to compute, for each point, the time difference (Δt) between each satellite visit (a.k.a. revisit time). Since the objective of the mission is to reach a minimal temporal resolution, the reference value will be the maximum temporal gap between revisits.

After running the subroutine, a 2-D variable `revTime`, showing the maximum revisit time for each grid-point, will be available. It will provide enough information to show the coverage rate graphically, and to compute a figure of merit [1].

Also, the delta-v budget required to implement that constellation is computed, using the formula from (3.19).

The following considerations will be taken:

- A point must be visited at least THREE times to be considered covered.
- If a point is not covered, it is given an arbitrary revisit time, which shall be longer than the propagation time, in order to avoid confusions

Summary

These are the main steps of the algorithm:

1. Data Input
2. Preliminary calculations. Define the orbit objects from the raw data.
3. Define a grid that will cover Earth.
4. Add an "event" detector to each point, which will trigger every time they "see" a satellite.
5. Propagate the orbits. Also, store the "events" in a big list.
6. Sort the events. Compute revisit time for each grid-point.
7. Plot (optional).
8. Compute the cost and the Figure of Merit.

The final code can be found at the Appendix A.

Inputs

`h`: Altitude above Earth surface (in meters).
`i_deg`: Inclination of the orbits (in degrees).
`P`: Number of uniformly distributed orbital planes.
`S`: Number of uniformly distributed satellites in each plane.

[1] A figure of merit is a quantity used to characterize the performance of a device, system or method, in order to compare it to its alternatives.

Parameters

They are not supposed to be changed during the optimization process, but depend highly on the characteristics of each mission. Thus, it is easy to change their value within the script itself.

`XX, YY`: The grid-points of the Earth. The resolution can be changed.
`prop_time`: Propagation time (in hours).
`elev_deg`: Minimum elevation from the horizon, for the satellite to be detected (in degrees).

Outputs

`cost`: Estimation of the cost it would require to put that constellation in orbit.
`revTime`: 2D array, representing all the grid-points, with its revisit time (in seconds).
`FoM`: Taking into account the previous outputs, a Figure of Merit can be computed.

Figure of Merit

After this considerations are taken, a way to define the appropriateness of the constellation is needed, in order to compare it with other ones. However, firstly a correction must be made.

Since Earth has an almost-spherical shape, the grid-points at higher latitudes will have a smaller actual area than points at the equator. Considering Δs the surface of each point, it can be computed multiplying the increment of longitude ($\Delta\lambda$) by the increment of latitude ($\Delta\phi$). The increment of latitude will remain constant, but as long as we reach higher latitudes, the increment of longitude will decrease:

$$\Delta s = \Delta\lambda \cdot \Delta\phi = \frac{\Delta\lambda_0}{\cos\phi}\Delta\phi \tag{4.1}$$

Where $\Delta\lambda_0$ is the increment of longitude at the Equator, which is a fixed value. Since all the data is going to be normalized at the end, we can discard the constant values and only keep the variable ones. Thus, when computing the different outcomes, each point is going to have a weighing factor related to its latitude, $k = \frac{1}{\cos\phi}$.

The output of the subroutine is the revisit time in every grid-point (and an arbitrarily big value if the point is not covered), the different schemes are proposed.

- **Scheme A:** Compute the mean revisit time for all Earth, including the points not covered, which take an arbitrary value. This is a fast way to study both the coverage and the revisit time. However, that arbitrary value is chosen tentatively, and may cause great distortions. It is advisable to use it only at the very beginning, when little of the constellation topology is known, as a fast way to check the different parameters.
- **Scheme B:** Now two values are going to be calculated. The coverage ratio, and the mean revisit time *just* for the covered points. It was not considered to assign weighing factors

to both of them, in order to reach a unique objective function, for the following reasons: the nature of those values is very different. The first takes values from 0 to 1 and has to be maximized, the other has a much wider range, and could even reach to erratic values if only a really small area is covered with a high temporal resolution.

- **Scheme C:** Just taking into account the cost, and minimize it under the constraints defined at Chapter 2.
- **Scheme D:** Once a reasonable constellation is reached, execute further optimization by appointing some "points of interest", which will be added weighing relevance at the final value. Focus only whether an area is covered under a certain revisit time. In this case, the objective is to reach the maximum coverage possible with the minimum number of satellites.

The previous schemes follow more or less the order that could be followed. The process begins with little or none information about how the final design should be, and it is used as a way to "tentatively" try different options.

Afterwards, a little more precision is required, and two independent values are returned, instead of just one. It would be a good idea to add some sort of limitation to the P, S parameters, or else they would grow into a extremely large swarm of satellites, that would surely comply with the requirements, but probably in a really inefficient way.

At the end, once most of the constellation is already determined, a little more optimization can be reached by just trying to minimize the cost, but still complying with the given constraints. This final step requires specific requirements, which sometimes may be too ambiguous (due to the lack of information given by the stakeholders). In those cases, the best practice is to study the Pareto-efficient configurations [2], taking the ambiguous requirements as variables. [25]

The last one is conceptually explained, as an idea for when a specific area requires more frequent monitoring. [26] However, as our initial objectives didn't require to pay more attention to any area over another, it is not going to be developed.

4.3 Initial guess

In order to improve the efficiency of the model, some analytical study was carried out, so as to begin the subroutine with suitable enough constellations.

Having orbits with Repeating Ground Tracks is highly advisable [27], which will make the simulation easier, because of the several regularities that will appear. It also makes it more simple to keep track of the satellites, as they always pass over the same spot once they completed their cycle. We will define n_s as the number of revolution the satellite does at each cycle, and n_E the number of revolutions of the Earth. Also, we will use the notation $\tau = n_s : n_E$ to identify them.

As mentioned in Chapter 3, most Earth Observation satellites follow a Sun-synchronous orbit in LEO, due to its unique and useful characteristics.

[2] In multi-objective optimization, a configuration is Pareto-efficient when none of its objective functions can be improved without worsening, at least, another one.

Let's begin considering those orbits whose ground track repeats after only one-day period, represented at table 4.1. The period is easily computed as $T = 86\,400\,\mathrm{s}/n_s$, and the altitude, by the third law of Kepler, $a = \sqrt[3]{\frac{\mu_E}{4\pi^2}T^2}$ and $h = a - r_E$.

Revs/day	**Period [s]**	**Altitude [km]**
11	*7854.55*	*2162.17*
12	7200.00	1680.86
13	6646.15	1262.09
14	6171.43	893.80
15	5760.00	566.90
16	5400.00	274.42
17	*5082.35*	*10.91*

Table 4.1: Orbit parameters for Sun-synchronous orbits, for $n_E = 1$

The first and last solutions are out of the LEO range, so they are discarded.

The solution for $n_s = 12$ is more or less the same as represented at figure 3.4 (a). It is obvious that its ground track is too wide to cover the whole Earth. Two approaches can be made to solve this problem. Adding more satellites to the constellation (thus creating more ground tracks) or analysing cases for $n_E = 2$ (i.e. the ground tracks repeat every two-day period).

Revs/2-days	**Period [s]**	**Altitude [km]**
24	*7200,00*	*1680,86*
25	6912,00	1464,49
26	*6646,15*	*1262,09*
27	6400,00	1072,26
28	*6171,43*	*893,80*
29	5958,62	725,65
30	*5760,00*	*566,90*
31	5574,19	416,73
32	*5400,00*	*274,42*

Table 4.2: Orbit parameters for Sun-synchronous orbits, for $n_E = 2$

It should be noted that the cases with an even value of n_s are degenerates from the first solutions, and don't really count as new configurations. However, there are four brand new solutions available.

If one continued to increase the number of cycle-days, one would encounter with more solutions, some of whom would be degenerates. For now, we are going to stay with the ones we have already found.

From figure 4.6 one can extract equation (4.2), which relates a satellite's coverage angle to its altitude and elevation:

$$\cos(\theta + \varepsilon) = \frac{\cos\varepsilon}{1 + h/R_E} \tag{4.2}$$

And with that value θ we will be able to approximate the number of different ground tracks needed to cover the whole Earth. A way to ensure that ground tracks are different is by putting the satellites at the same orbital plane. And to compute the number of satellites per plane, it will be enough to divide 360 degrees by two times theta times the number of revolutions around the Earth: $S = \frac{360^{\text{o}}}{2\theta n_s}$.

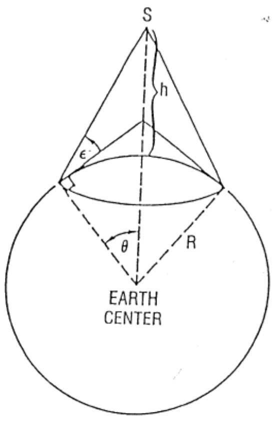


Figure 4.6: Coverage of a satellite given its elevation and altitude

τ	**Altitude [km]**	2θ	**S**
12:1	1680.86	13.38	3
25:2	1464.49	12.01	2
13:1	1262.09	10.66	3
27:2	1072.26	9.31	2
14:1	893.80	7.98	4
29:2	725.65	6.65	2
15:1	566.90	5.33	5
31:2	416.73	4.02	3
16:1	274.42	2.71	9

Table 4.3: Parameter S for the studied configurations

Moreover, a minimum revisit time is needed. For now, each satellite revisits its ground track once it completes the cycles, so more planes will have to be added too. In conclusion, the configurations that have a 24-hour cycle will need 12 planes, and the configurations that have a 48-hour cycle will need 24 planes. Table 4.4 lists the final parameters for all configurations.

$\tau = n_s : n_E$	**Altitude [km]**	**S**	**P**	**T**
12:1	1680.86	3	12	36
25:2	1464.49	2	24	48
13:1	1262.09	3	12	36
27:2	1072.26	2	24	48
14:1	893.80	4	12	48
29:2	725.65	2	24	48
15:1	566.90	5	12	60
31:2	416.73	3	24	72
16:1	274.42	9	12	108

Table 4.4: Initial Walker parameters for the studied configurations

The first six configurations have the least number of satellites, which at the end has the most relevance at the total cost. These are the ones that are going to be optimized.

4.4 Optimization algorithms

After some semi-analytical guesses, we will try to reach a most efficient architecture, using some optimization algorithms with some of the schemes already explained.

Among all the possible ones, the ones more appropriate are the following: [28] [29]

Gradient descent: its implementation is really simple, and it is easy to understand, the main problem is that it is very likely to end up in a local minimum, which could be too close to the initial guess (and depending on the case, the initial guess could be itself a local minimum). This can be solved picking a better initial point, or running it a lot of times from different points, reducing its efficiency. Gradient-based optimizers use this idea. but in a more complex way.

Genetic Algorithm: it is a powerful algorithm based on a Darwinian evolution, creating an initial population, and improve it by randomly mutating them and doing crossovers among the individuals who perform better. It is conceptually easy to understand, but more complex at the time to implement it. [7] [30]

1. Create an initial population.
2. Evaluate its fitness.
3. Select the individuals with the best fitness score.
4. If convergence criteria is not met, crossover and mutate the individuals, and go to step 2 again.
5. If convergence criteria is met, stop.

Figure 4.7: Flowchart of a Genetic Algorithm

The main elements of a Genetic Algorithm are the following:

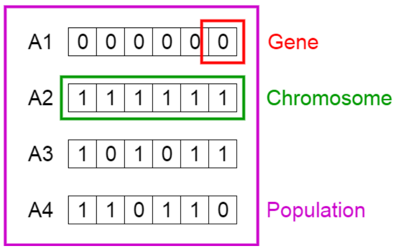


Figure 4.8: Basic elements of a genetic algorithm

Gene: each one of the parameters that may be changed: the orbit paremeters, `h, P, S`
Chromosome: the whole group of genes that configure a single solution: any valid constellation.
Population: a number of chromosomes set by the user, it can vary.

Implementation

The random nature of a GA makes very suitable for problems were little knowledge of the optimal solution is available. It may take approaches that the user didn't consider, providing new frameworks to solve the problem, and with constant mutation it will help to avoid getting stuck at local minima.

It is also for its random nature that for every time it is run, a different solution may be reached. There are some programs that automatically re-run the optimizer a couple of times and then select the best configuration above all. It is useful though to keep track of all those solutions, in order to better understand its behaviour.

There is a wide variety of optimizers based on GA or on gradient. Some important parameters are the population size, how many members are going to be mutated, how much relevance have previous generations, etc. However, none of them can give a 100% guarantee that the found value is the best solution (the only guarantee would be trying all the possible parameters, a.k.a. brute-force study).

The first runs were performed by the `DEAP` package of Python, providing very simple configuration, and mutating by a plain Gaussian distribution. Later on, there was access to a licensed software known as *ModelCenter*, which is capable to interface with a wide range of programs (e.g. MATLAB, Python, Excel), and includes a library of optimizers. It allows automatic data transfer among applications, and as a consequence it is able to optimize a Python function "from outside" and record its values in an Excel spreadsheet, without actually opening any of both programs.

One of the optimizer packages included is `Dakota`. [31] Its gradient-based algorithm learns itself from the model it is optimising, and keeps readjusting its parameters. From time to time, when it reaches an apparent local minimum, it goes back several stages to try to find better solutions. It was eventually the algorithm used for the last runs.

Chapter 5

Simulations

5.1 Initialisation

The simulations were at first executed at a normal PC using Python. At the final stages, *ModelCenter* was used to optimize the constellations. Figure 5.1 shows its behaviour. As mentioned before, any time it found a local minimum, it went back to try new paths. It happened successively several times until it reached an optimal solution. All the optimizations had a similar behaviour.

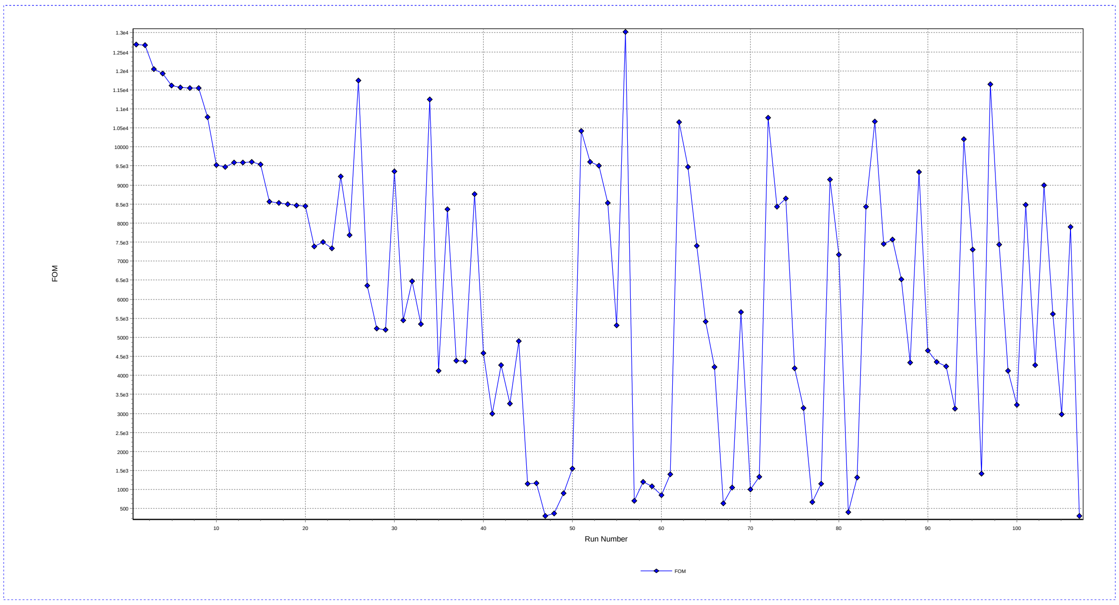


Figure 5.1: Example run of the optimizer, minimizing the Figure of Merit

5.2 Results

Those are the coverage rates of the optimized constellations for each τ value. The color scale represent the revisit time (legend at the right):

$\tau = 12 : 1$
Coverage: 90.13%
Mean revisit time: 1.996 h

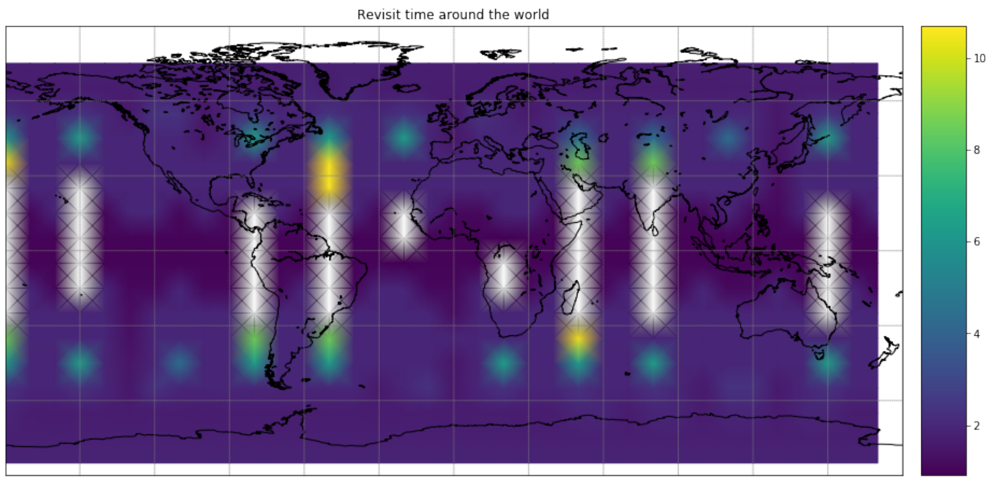


Figure 5.2: Optimal solution for $\tau = 12 : 1$

$\tau = 13 : 1$
Coverage: 100%
Mean revisit time: 2.006 h

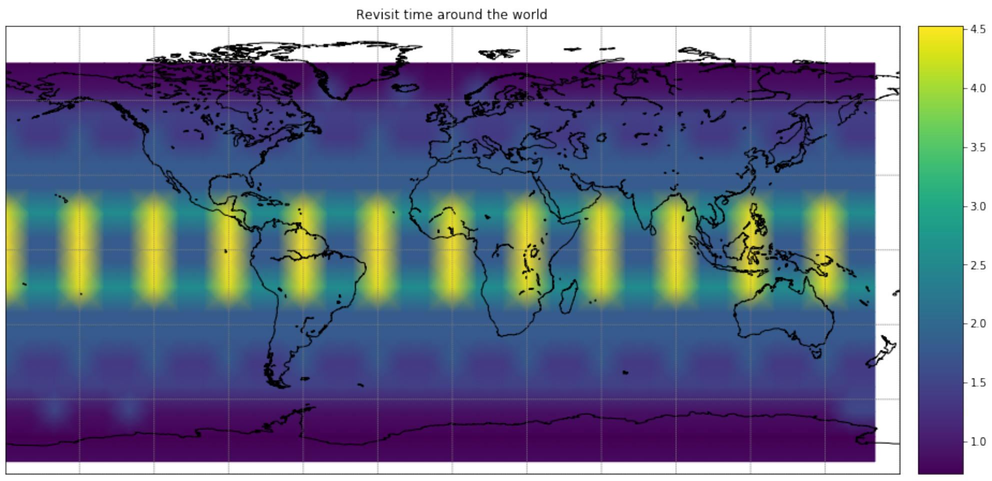


Figure 5.3: Optimal solution for $\tau = 13 : 1$

$\tau = 14:1$
Coverage: 100%
Mean revisit time: 1.797 h

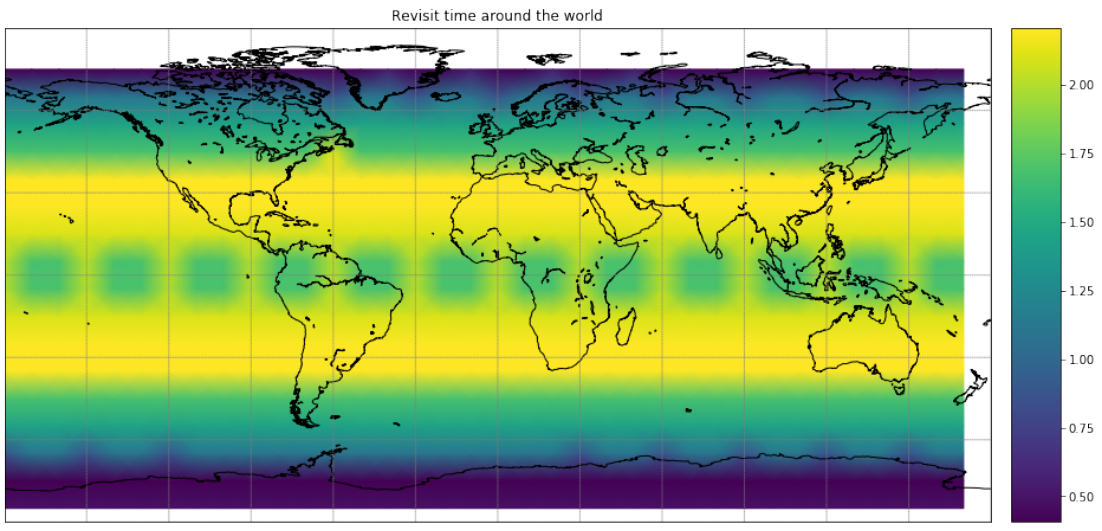


Figure 5.4: Optimal solution for $\tau = 14:1$

$\tau = 25:2$
Coverage: 100%
Mean revisit time: 1.746 h

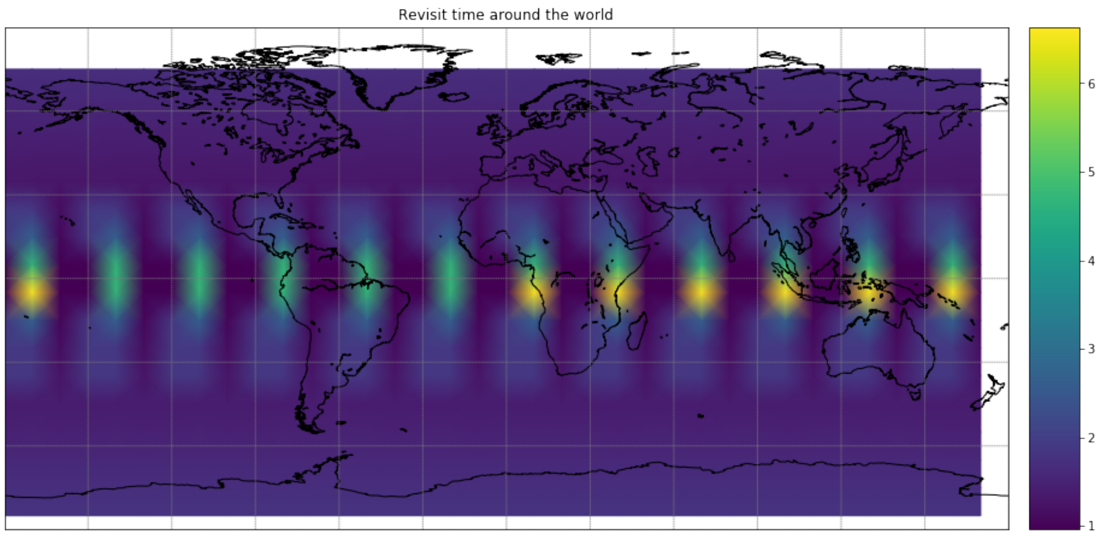


Figure 5.5: Optimal solution for $\tau = 25:2$

$\tau = 27 : 2$
Coverage: 100%
Mean revisit time: 1.868 h

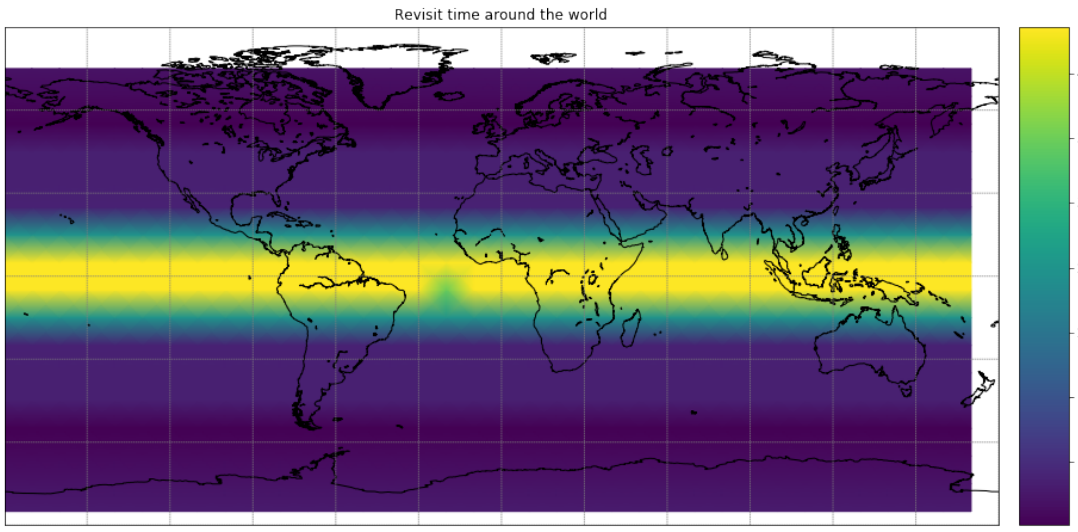


Figure 5.6: Optimal solution for $\tau = 27 : 2$

$\tau = 29 : 2$
Coverage: 100%
Mean revisit time: 1.868 h

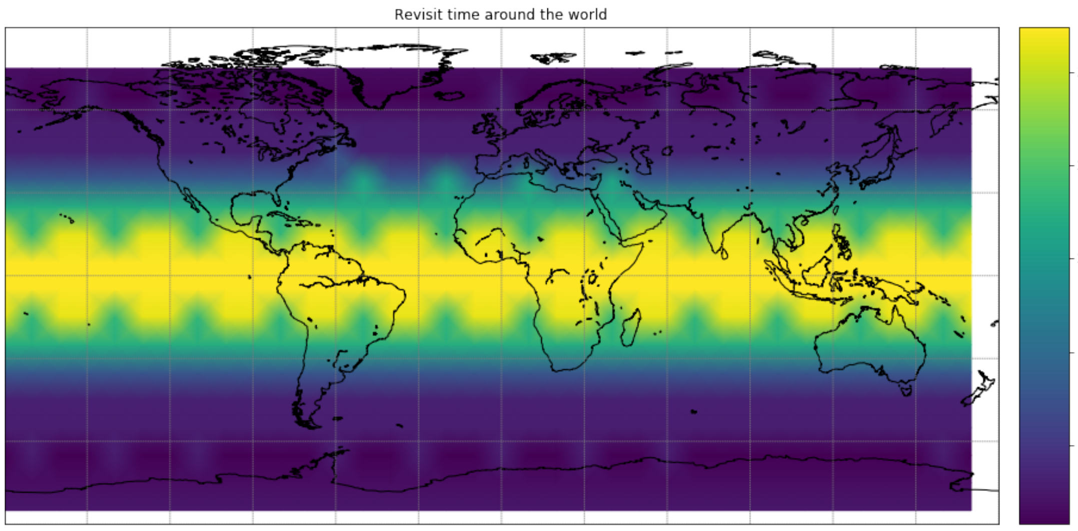


Figure 5.7: Optimal solution for $\tau = 29 : 2$

Chapter 6

Conclusions

6.1 Summary of findings

An algorithm was developed from scratch, using purely open source code. That helped compute the performance of any constellation, whose inputs could be either the Walker-Delta parameters ($i : T/P/F$), or the keplerian elements of each one of the satellites.

A semi-analytical initial guess was developed, and it came to the conclusion that, indeed, the best way to perform an Earth Observation mission is by using a Sun-synchronous orbit in Low Earth Orbit.

Afterwards, from the constellations that were considered more suitable, they were optimized to reach the minimum cost, where the requirements were still met. In most cases, the final configuration was different than the original one found previously in an analytical way.

		Initial guess		**Final value**		
$\tau = n_s : n_E$	Altitude [km]	P	S	P	S	Cost (Δv)
12:1	1680.86	12	3	11	1	264.722 km/s
13:1	1262.09	12	3	12	2	586.704 km/s
14:1	893.80	12	4	12	3	893.060 km/s
25:2	1464.49	24	2	24	1	582.199 km/s
27:2	1072.26	24	2	24	2	1182.183 km/s
29:2	725.65	24	2	24	2	1199.109 km/s

Table 6.1: General results

A general conclusion from table 6.1 is that increasing the number of revolutions around the Earth, although it may create a denser ground track, it is countered by the fact that the satellite has less altitude, which makes the footprint to be much smaller. This pattern could already be noticed at table 4.4, where the discarded alternatives were the ones with smaller altitude.

One the other hand, it is interesting to see that the number of planes hardly changed. Assuming this tendency was to be kept, increasing the day-cycles would only make the matters worse. A simple explanation for this is that for as little as 3-day cycles, even though the ground track was dense enough to cover the whole Earth, the low-latency of its period would require some thirty-ish satellites to keep the constant monitoring.

Taking the cost (delta-v) as the ultimate decision criterion, would leave the first configuration as the most appropriate one. However, it must be taken into consideration that it orbits at an altitude of nearly 1680 km, way higher than most conventional EO satellites (usually at 700-800 km). That high altitude has two major side effects. On one hand, the ground resolution is lower and more refraction elements take part, on the other hand, the Van Allen radiation belts may greatly affect the satellites trajectory and its communications. The total cost to counter those effects (better camera, trajectory corrections) might very well surpass the other values found.

A study considering such effects is left as further work, and instead the best constellation configuration for each altitude is given.

6.2 Recommendations for future work

The optimization was made with only the Walker parameter of the constellation. With a more powerful hardware, more complex constraints could be introduced, and use as optimizing variables all the orbital elements for every single satellite.

Although a LEO constellation that accomplishes the initial goals was developed, a study about the data communication from the satellites to the Earth has not been carried out. A good temporal resolution is pointless if that information is not able the reach the Earth quickly. Some possibilities are proposed that should be studied later on. For instance, as the satellites orbit around the Earth several times a day in Repeating Ground Tracks, there could be some strategic points to install receptors and collect the data. This approach would still depend on the period of the satellites, and would require building receptors all around the globe. It would be interesting to study the communication possibilities intra- and inter-planes, which would require very few receptor points in the Earth. This would, nevertheless, add a lot of complexity to the problem, as the satellites would constantly send information among themselves.

Another feasible option is to depend on a totally different constellation (probably in MEO or GEO) to facilitate the communications. As communication satellites are not a mystery nowadays, it wouldn't be difficult to implement in a theoretical point of view, but would instead increase a lot the data traffic, given that we are dealing with high-resolution images from several satellites in real time.

Moreover, should this project ever be implemented in real life, a lot of caution should be put in the ever-increasing problem of the Kessler syndrome (commonly known as space garbage). The LEO area is one of the most infested of debris and ancient disposed missions, making a collision highly possible. Even the constellation itself contributes to the problem. With a revisit time as short as two hours (and even smaller at the crossing points), any launch to be performed later on should check that the rocket will not interfere with the constellation orbit, in order to avoid a catastrophic outcome. There should also be developed a way to dispose of a satellite should it expire its lifetime.

Appendix A

Python script

```python
## Import all necessary packages
import numpy as np
import matplotlib as plt
import orekit
vm = orekit.initVM()
print('Java version:',vm.java_version)

from orekit.pyhelpers import setup_orekit_curdir
setup_orekit_curdir()

#package fr.cs.examples.propagation;

from java.io import File;
from java.util import Locale;

from org.hipparchus.geometry.euclidean.threed import Vector3D;
from org.hipparchus.util import FastMath;
from org.orekit.attitudes import CelestialBodyPointed
from org.orekit.bodies import BodyShape;
from org.orekit.bodies import GeodeticPoint;
from org.orekit.bodies import OneAxisEllipsoid;
from org.orekit.bodies import CelestialBodyFactory;
from org.orekit.data import DataProvidersManager;
from org.orekit.data import DirectoryCrawler;
from org.orekit.errors import OrekitException;
from org.orekit.frames import Frame;
from org.orekit.frames import FramesFactory;
from org.orekit.frames import TopocentricFrame;
from org.orekit.orbits import KeplerianOrbit, PositionAngle;
from org.orekit.propagation import Propagator;
from org.orekit.propagation import SpacecraftState;
from org.orekit.propagation.analytical import KeplerianPropagator;
from org.orekit.propagation.events import ElevationDetector,
    EventDetector, NodeDetector, EventsLogger;
from org.orekit.propagation.events.handlers import *;
from org.orekit.propagation import events;
from org.orekit.time import AbsoluteDate;
from org.orekit.time import TimeScalesFactory;
from org.orekit.utils import Constants;
from org.orekit.utils import IERSConventions;
from org.orekit.utils import PVCoordinates;
from org.orekit.utils import PVCoordinatesProvider;
from org.orekit.utils import ExtendedPVCoordinatesProvider;

from math import pi,radians
import matplotlib.pyplot as plt
import mpl_toolkits
from mpl_toolkits.basemap import Basemap
import time
import json
from pprint import pprint

###########################################################

## Data input

# Parameters: (h, i_deg, P, S)
h = 1688000 # Altitude above Earth surface
i_deg = 60 # Inclination (in degrees)
P = 4 # Number of orbital planes
S = 2 # Number of satellites in each plane

## Earth grid, grid-points that are going to be analysed
XX = []
for i in range(-180,180,5):
    XX.append(i)

YY = []
for i in range(-85,85,5):
    YY.append(i)

# How many hours are we going to propagate?
prop_time = 24.0 # Twenty-four hours

# Which is the minimum elevation required for the "observatory" to
    detect the satellite? (Angular distance from zenith)
elev_degrees = 60

######################################################

## Beginning of subroutine
```

```


def main(h,i_deg,P,S):

    a = Constants.WGS84_EARTH_EQUATORIAL_RADIUS+h
    e = 0.001
    i = radians(i_deg)
    pa = 0
    raan = np.arange(0,360,360/P)
    lv = np.arange(0,360,360/S)

    sat = []
    for ii in range(P):
        for jj in range(S):
            sat.append([a, e, i, pa, radians(raan[ii]), radians(lv[jj
                ])])
    n_sat=len(sat)

    ae = Constants.WGS84_EARTH_EQUATORIAL_RADIUS
    mu = Constants.WGS84_EARTH_MU
    utc = TimeScalesFactory.getUTC()
    r_GEO = (mu*(86400/(2*pi))**2)**(1/3)

    initialDate = AbsoluteDate(2018, 12, 21, 12, 0, 00.000, utc)
    inertialFrame = FramesFactory.getEME2000()
    earthFrame = FramesFactory.getITRF(IERSConventions.IERS_2010,
        True);
    earth = OneAxisEllipsoid(Constants.WGS84_EARTH_EQUATORIAL_RADIUS,
                                                   Constants.
                                                       WGS84_EARTH_FLATTENING
                                                       ,
                                                   earthFrame);

    orbitList=[]
    for i in range(n_sat):
        orbitList.append([])
    for i in range(n_sat):
        orbitList[i]=KeplerianOrbit(float(sat[i][0]), float(sat[i
            ][1]), float(sat[i][2]), float(sat[i][3]),
                                       float(sat[i][4]), float(sat[i
                                           ][5]), PositionAngle.TRUE,
                                           inertialFrame, initialDate, mu
                                           )
    # orbitList

    ZZ = np.zeros((len(XX),len(YY)))
    logger = []
    node = []
    cout = []
    masterList = [] # Big variable that stores all satellite visits (from all satellites) to all grid-points

    for ii in range(len(XX)):
        logger.append([])
        for jj in range(len(YY)):
            logger[ii].append([])

    for ii in range(len(XX)):
        node.append([])
        for jj in range(len(YY)):
            node[ii].append([])

    for ii in range(len(XX)):
        cout.append([])
        for jj in range(len(YY)):
            cout[ii].append([])

    for ii in range(len(XX)):
        masterList.append([])
        for jj in range(len(YY)):
            masterList[ii].append([])

    ## Subroutine propagator. Once for every satellite (n_sat times)
    for kk in range(n_sat):
        kk_PLUS_ONE=kk+1
        kepler = KeplerianPropagator(orbitList[kk]);
        for ii in range(len(XX)):
            for jj in range(len(YY)):

                #// Station
                longitude = radians(XX[ii]);
                latitude  = radians(YY[jj]);
                altitude  = 0.;
                station1 = GeodeticPoint(latitude, longitude, altitude);
                sta1Frame = TopocentricFrame(earth, station1, "station1");

                #// Event definition
                maxcheck  = 60.0;
                threshold =  0.001;
                elevation = radians(elev_degrees) ;
                sta1Visi = ElevationDetector(maxcheck, threshold, sta1Frame).withConstantElevation(elevation).withHandler(ContinueOnEvent().of_(ElevationDetector));

                logger[ii][jj] = EventsLogger();
                node[ii][jj] = logger[ii][jj].monitorDetector(sta1Visi);

                #// Add event to be detected
                kepler.addEventDetector(node[ii][jj]);
                #kepler.setEphemerisMode();

        # Processing: Propagation of keplerian orbit

        print("Propagating orbit", kk_PLUS_ONE, "of", n_sat, "...")
        tic = time.time()
        finalState = kepler.propagate(initialDate.shiftedBy(prop_time*3600.)); #Number of days: one (1)
        toc = time.time() - tic
        print("Time elapsed:",toc,"seconds")

        # Post-processing: Storage of "visibility" event in masterList[ii][jj], in terms of hours from the initialDate
        #                  Also, computing of COUT[ii][jj], maximum revisit time at that point given that orbit
        #                  Also, once for every satellite (n_sat times)

        for ii in range(len(XX)):
            for jj in range(len(YY)):
                eventList=logger[ii][jj].getLoggedEvents() #Local list of events for every grid-point
                N=eventList.size() #Number of events=revisits
                dates = []
                revisitTime = []

                if N==0:
                    #print("error")
                    cout[ii][jj]=48 #or inf
                else:
                    for i in range(N):
                        #print(list1.get(i).getState().getDate())
                        dates.append(eventList.get(i).getState().getDate())
                        masterList[ii][jj].append(eventList.get(i).getState().getDate().durationFrom(initialDate)/3600)



    for ii in range(len(XX)):
        for jj in range(len(YY)):
            dates=np.sort(masterList[ii][jj])    #masterList[ii][jj]=sort(masterList[ii][jj])
            N=np.size(dates)
            revisitTime = []

            if N<3:
                #print("error")
                cout[ii][jj]=48 #or inf
            else:
                for i in range(1,N):
                    revisittime = dates[i]-dates[i-1]
                    revisitTime.append(revisittime)
                revTime[ii][jj]=max(revisitTime)
    return revTime


revTime=main(h,i_deg,P,S)

######################################################

## Plot


plt.figure()
fig=plt.figure(figsize=(16, 12) )

m = Basemap(projection='cyl',
            resolution='l',
            area_thresh=None)

m.drawcoastlines()
m.drawmeridians(np.arange(-180, 180, 30), color='gray')
m.drawparallels(np.arange(-90, 90, 30), color='gray');

m.pcolormesh(XX,YY,np.transpose(cout))
m.colorbar(location='right')
plt.title("Revisit time around the world")

####################################################

## Computation of coverage and mean revisit time

cvg = np.copy(cout)
cout_corrected = np.copy(cout)
total = np.copy(cout)

for ii in range(len(XX)):
    for jj in range(len(YY)):
        if cout[ii][jj]==48:
            cvg[ii][jj]=0
        else:
            cvg[ii][jj]=1*cos(radians(YY[jj]))

for ii in range(len(XX)):
    for jj in range(len(YY)):
        if cout[ii][jj]==48:
            cout_corrected[ii][jj]=0
        else:
            cout_corrected[ii][jj]=cout[ii][jj]*cos(radians(YY[jj]))

for ii in range(len(XX)):
    for jj in range(len(YY)):
        total[ii][jj]=cos(radians(YY[jj]))

coverage = sum(cvg)/sum(total) # Total coverage (out of 1)
mean_revTime = sum(cout_corrected)/sum(cvg)
```

Bibliography

[1] ONION H2020. http://www.onion-h2020.eu/ [Electronic][Accessed:2018-09-15].

[2] Ahmad Alkhatib. A review on forest fire detection techniques. *International Journal of Distributed Sensor Networks*, 2014, 03 2013.

[3] David de la Torre. Space Engineering notes, UPC. http://atenea.upc.edu/ [Electronic][Accessed:2017-10-27].

[4] Howard D. Curtis. *Orbital Mechanics for Engineering Students.* Elsevier, 2005.

[5] Kepler orbit. http://en.wikipedia.org/wiki/Kepler_orbit [Electronic][Accessed:2018-11-05].

[6] Yanchao He, Xu Ming, Xianghua Jia, and Roberto Armellin. High-precision repeat-groundtrack orbit design and maintenance for earth observation missions. *Celestial Mechanics and Dynamical Astronomy*, 02 2017.

[7] Sihang Liu, Li Pengxu, Gaofeng Cui, and Weidong Wang. Design of satellite constellation with inter-satellite links for global communication using genetic algorithm. pages 367–373, 12 2017.

[8] R.R. Bate, D.D. Mueller, and J.E. White. *Fundamentals of Astrodynamics.* Dover Books on Aeronautical Engineering Series. Dover Publications, 1971.

[9] F. G. Lemoine, S. C. Kenyon, J. K. Factor, R.G. Trimmer, N. K. Pavlis, D. S. Chinn, C. M. Cox, S. M. Klosko, S. B. Luthcke, M. H. Torrence, Y. M. Wang, R. G. Williamson, E. C. Pavlis, R. H. Rapp, and T. R. Olson. *The Development of the Joint NASA GSFC and the National Imagery and Mapping Agency (NIMA) Geopotential Model EGM96.* NASA/TP-1998-206861, 07 1996.

[10] Ronald J Boain. ABC-s of sun-synchronous orbit mission design. 2004.

[11] Sun-synchronous orbit. http://en.wikipedia.org/wiki/Sun-synchronous_orbit [Electronic][Accessed:2018-12-20].

[12] D Mortari, Matthew Paul Wilkins, and Christian Bruccoleri. The flower constellations. *Journal of the Astronautical Sciences*, 52, 01 2004.

[13] W S. Adams and L Rider. Circular polar constellations providing continuous single or multiple coverage above a specified latitude. *Journal of the Astronautical Sciences*, 35:155–192, 05 1987.

[14] Lloyd Wood. *Satellite Constellation Networks*, pages 13–34. 04 2003.

[15] Yuri Ulybyshev. Near-polar satellite constellations for continuous global coverage. *Journal of spacecraft and rockets*, 36(1):92–99, 1999.

[16] John Gerard Walker. Circular orbit patterns providing continuous whole earth coverage. Technical report, Royal Aircraft Establishment Farnborough (United Kingdom), 1970.

[17] Galileo: a constellation of navigation satellites. http://www.esa.int/Our_Activities/ Navigation/Galileo/Galileo_a_constellation_of_navigation_satellites [Electronic][Accessed:2019-01-09].

[18] Giovanni Palmerini. Design of global coverage constellations based on elliptical orbits. 07 1996.

[19] Stanley Kidder and Thomas H. Vonder Haar. On the use of satellites in Molniya orbits for meteorological observation of middle and high latitudes. *Journal of Atmospheric and Oceanic Technology*, 7:517–522, 05 1990.

[20] Guangming Dai, Xiaoyu Chen, Maocai Wang, Elena Fernandez, Tuan Nam Nguyen, and Gerhard Reinelt. Analysis of satellite constellations for the continuous coverage of ground regions. *Journal of Spacecraft and Rockets*, 54:1–10, 08 2017.

[21] Nesrin Sarigul-Klijn, Chris Noel, and Martinus Sarigul-Klijn. Air launching eart-to-orbit vehicles: Delta v gains from launch conditions and vehicle aerodynamics. 01 2004.

[22] Anaconda. https://www.anaconda.com/ [Electronic].

[23] Orekit. https://www.orekit.org/ [Electronic].

[24] Gérard Petit and Brian Luzum. IERS Conventions (2010). *Tech. Rep. DTIC Document*, 36:180, 01 2010.

[25] Alessandro Golkar and Edward F. Crawley. A framework for space systems architecture under stakeholder objectives ambiguity. *Systems Engineering*, 17, 07 2014.

[26] Salvo Marcuccio and Rafel Heitkoetter. Environmental monitoring of the Amazon basin with a low cost small satellite constellation in equatorial leo. pages 1–7, 03 2018.

[27] Xin Luo, Maocai Wang, Guangming Dai, and Xiaoyu Chen. A novel technique to compute the revisit time of satellites and its application in remote sensing satellite optimization design. *International Journal of Aerospace Engineering*, 2017, 2017.

[28] Xiaolong Zhu and Yang Gao. Comparison of intelligent algorithms to design satellite constellations for enhanced coverage capability. pages 223–226, 12 2017.

[29] Theresa W. Beech, S. Cornara, Miguel Mora, A. GMVS., and C Newton. A study of three satellite constellation design algorithms. 2013.

[30] M Asvial, Rahim Tafazolli, and B.G. Evans. Satellite constellation design and radio resource management using genetic algorithm. *Communications, IEE Proceedings-*, 151:204 – 209, 07 2004.

[31] Dakota. https://dakota.sandia.gov/ [Electronic][Accessed 2019-01-31].