Simulation of Low Earth Orbit Objects

Semester Master Project

Dunant Joachim

Supervisor: Rosilho De Souza Giacomo
Professors: Abdulle Assyr, Knieb Jean-Paul

Contents

1 Introduction

Since the beginning of the space race in the 50s, humankind has put an enormous amount of objects into orbit. We estimate that more than 3000 satellites are currently around the earth [12], while around 400 new ones are sent each year [13]. Furthermore, there are a few hundred of thousands of debris of different kinds below 10 centimeters, plus around 7000 cataloged non-active satellites and other larger debris are in orbit around the earth [14], creating a massive space traffic. Collision risk is one of the biggest issues when it comes to space traffic, as a collision between two objects would create even more debris, raising the chance to have future collisions and thus inducing a vicious circle which would pollute space even more. Hence it is today really important to keep track of objects in orbit around our planet and to predict when collisions may happen, to be able to prevent them.

This project is done in collaboration with the Space Situational Awareness (SSA) EPFL Team, founded in july 2020 by students from the EPFL. Its objective is to set up a catalog of important information on each detectable object in Earth orbit, in order to predict, and thus avoid dramatic collisions [8].

Thus, the project aims at designing an efficient method to optimize long-term computations of collision probabilities for every object in low earth orbit (LEO), this means all satellites, active or not, and debris that are less than 2000 km above sea level. We only check for LEO because more than 75% of the satellites and almost all debris are in this type of orbit [12]. To be able to compute this collision risk, it is necessary to build a database to store space objects' information, such as position through time for instance. Thus, the construction of such a database will be the main focus of this project.

At the present day there are already many public and regularly updated databases of orbiting objects, but hardly any collision avoidance ones, which show data such as the collision risk between two given satellites or their miss distance. The most known resource for collision risk is LeoLabs [9], available publicly for companies and individuals that would need this kind of information. On the other hand NASA and ESA use their own private databases and methods. However, it is still important to build a new database from scratch, so that it can be managed internally and updated according to the needs of the SSA Team, EPFL, or any external partner.

This project is divided in two parts. The first part will focus on building a database which efficiently supports insertion of newly detected objects and the load of information about each space objects in LEO, by doing trajectory simulations of these objects through time. Multiple database structures will be explored, by looking at different ways to compute and optimise a Nearest Neighbour (NN) search. Then the goal of the second part is to explore (but not implement) multiple methods for the computation of collision probabilities of each satellite and debris, to see their pros and cons in the case of space orbits. In this manner, we can achieve the objective of having a regularly updated database containing all the objects detected in LEO and that can also provide an effective basis for calculating the risk of collision between every space object.

2 Space objects database construction

A detailed description of the methods used to construct an efficient and complete database for space objects in LEO is offered in this section. It is separated in three parts, firstly how to identify objects in LEO, then how we can simulate trajectories around the earth and finally, by putting the first two parts together: what is the complete structure of the database.

We need to build a database that can support complex computations on a large number of objects, store and reference multiple types of information about spatial objects for regular and long-term use. In this part we will explore and understand how to build a database that meets those conditions and what type of information it must store. The database will have to track, simulate objects and compute collision risks for each catalogued object, therefore it must contain at least these elements :

- TLE informations, one of the data format to store space objects details such as its orbit parameters, see section 2.1.
- An array of simulated positions through time, meaning that there would be one position vector per time step, computed by making a simulation of said object. This vector will have cartesian $\{x, y, z\}$ coordinates where the origin $\{0, 0, 0\}$ is the center of the earth.
- An array of velocity vectors through time, computed the same way as the position vectors, where the values $\{x, y, z\}$ from those vectors are the velocities of the object in the given direction.
- For each object the nearest neighbours at each time step, because each space object needs to know at any point in time their adjacent satellites or debris, to be able to determine against which ones collision risk has to be evaluated.

2.1 LEO objects identification with TLE format

To be able to build such a database, we firstly need to know what types of objects will be present in it. As we only work with space objects orbiting in LEO, we can build a list of these objects by parsing a list of all detected space objects and choose only those in LEO. We can find such a list for instance on the public Space-Track database [11], where they describe objects in different formats such as json, csv or html, but most importantly the *Two-Line Element* (TLE) format, that will be used throughout the project.

A TLE data format for space objects has 3 lines (hence it is also commonly called *Three-Line Element* or 3LE) : Its name in the first line (called line 0), and two other lines containing different details as its launch period and parameters of the ellipse defining its orbit. Each number or value in the lines 1 and 2 represent a parameter such as the ballistic coefficient, inclination of the orbit or the argument of perigee. It also contains indications about the object itself, for instance its launch year or

the launch number of the year. See Figure 1 for an example. This TLE format is updated regularly for each detected space object, to prevent estimations and loss of precision from every detection method. Furthermore, every time a new space object is detected then its related information is put in a new TLE.

```
0 VANGUARD 1
1     5U 58002B   20070.27364589 +.00000228 +00000-0 +26265-3 0  9999
2     5 034.2578 173.8647 1847930 266.0134 072.8371 10.84821291194486
```

Figure 1: *Example of TLE informations*

Thus, a lot of information is present in a TLE, but we only need a few for this project. As we work only with LEO objects, the first thing to do is to discard TLEs describing objects outside of LEO. Objects in LEO have orbits below 2000 kilometers above sea level and an orbit that isn't too eccentric. The term "eccentricity" is explained in more details after. To summarize, a TLE of a LEO object must fulfill the following requirements :

- Motion mean being greater than 11.25. The motion mean is defined as the number of revolutions around the earth per day. We need this requirement because LEO is composed of objects orbiting at 2000 kilometers or below, so objects at this altitude need a minimum speed to stay on their orbit and not fall back on the earth. If the motion mean of an object would be below 11.25 revolutions per day, they would then have to be higher in altitude, so in *Medium Earth Orbit* (MEO), ranging from 2000 kilometers to geostationary orbits (35'786km). This number is found in a TLE as the last number of the second line, up to the 8th decimal, so 10.84821291 in Figure 1.

- Eccentricity being below 0.25. It is the amount by which an orbit deviates from a perfect circle, where 0 is a perfect circle, 1 is a parabolic escape orbit and more than that would be an hyperbolic one. If the eccentricity would be greater than the requirement of 0.25 it would be classified as an HEO, as in *Highly Elliptical Orbit* (not to be confused with *High Earth Orbit* which is the orbit of objects higher than geostationary altitude). This is located in a TLE as the fourth number of the second line, with a decimal point assumed, so 0.1847930 in Figure 1.

2.2 Orbit simulation

Starting from the information found in a TLE, we can obtain a rough estimation of the position and velocity of an object at any point in time. This is done using the information defining its elliptic orbit and the multiple external forces acting on the object. It is a rough estimate, because the trajectories of these objects are not perfectly elliptic due to a high number of perturbations, such as the attraction of the moon and the sun, the imperfect shape of the earth, its radiation or its drag.

Therefore, a lot of different models were made to take into account these perturbations and obtain more accurate trajectories predictions. The most common models are the five *simplified perturbations models*. Those five models are called SGP, SGP4, SGP8, SDP4, and SDP8, where SGP stands for *Simplified General Perturbations* and are mostly used for space objects being in LEO. On the other hand, SDP, or *Simplified Deep Space Perturbations*, are models used, as its name indicates, on objects that are higher in altitude, more precisely objects higher than 5900 kilometers. The number after the name of the model is the indication of its version, where the higher the number, the more recent it is. As an indication, for instance SGP8 handles a more precise orbital decay of the space objects.

We will use SGP4's model [1] for this project, as we work only with LEO objects and because it is highly sufficient in precision, while not being too expensive to compute. Also, there is a complete and detailed dedicated Python library [10] which gives us the estimated position of an object at a given time, given their TLE. This library is used throughout the whole project.

2.3 Database structure

Multiple approaches were examined on what type of database structure to use to support fast moving objects in a 3D-space. Knowing that we need to compute collision avoidance, we need to often perform Nearest-Neighbor (NN) searches. Therefore, the first idea was to use R-Trees [2], built to handle very efficiently NN searches in 2D or 3D, by building encapsulated rectangles over points in space. An example of an R-Tree in a 2D environment is given in Figure 2. It makes it also very efficient to search, delete and insert in the database.

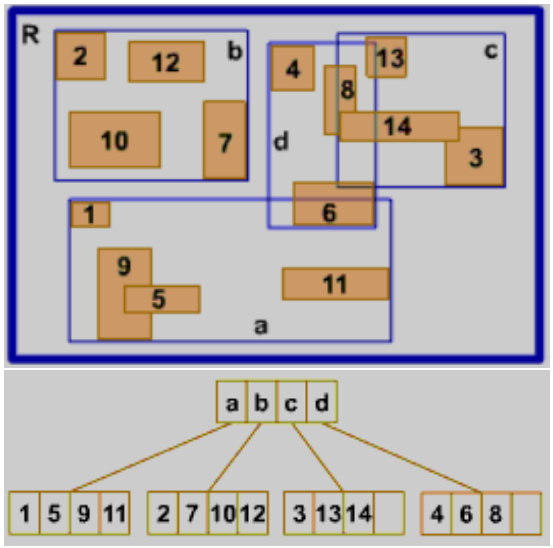

Figure 2: *Example of an R-Tree in 2D-space*

However, satellites and debris are moving at high speeds, up to 8.0km/s. As standard R-Trees are made for non moving objects, either we would need to rebuild

a new R-Tree at each time step, which is very costly ($O(n \cdot log(n))$ with n being around 20'000, just for the initialization of the database, then again $O(n \cdot log(n))$ to search the NN for every object), or use a different approach. There are ways to build an R-Tree differently so that it takes into account movement, for instance parametric R-Trees [3] or indexing R-Trees [4], but unfortunately in our case the objects move too fast to be handled efficiently by those models. Indeed, after only one second of satellite movement in space we would need to rebuild the R-Tree, because around 20'000 objects moved for 8 kilometers in any direction. So, keeping track of collisions using R-Trees is overly expensive. Hence a naive approach would in fact work better in our case, where at each time step every object goes through every other ($O(n^2)$) to find which is the closest.

Furthermore, we can reduce the number of NN searches we perform per object by taking into account that we only have to compute collision avoidance for neighbours that approach the object in question, because otherwise the risk of collision would be equal to 0 in the next few time steps. An example of such a search is given in Figure 3. Then, we can also greatly reduce the overall number of searches we perform on the whole simulation by looking at the k nearest neighbor for each object, where k is a small integer. Indeed it is sufficient to take a small number of approaching nearest neighbours, because every object will compute them, so corner cases where the next collision, for a given object, is not in its list of nearest approaching neighbour, are quickly no longer happening (at around $k = 3$). For this project we chose $k = 1$, because even if corner cases exist, it is computationally less expensive and lighter in memory space, while corner cases would not happen often in our cases of LEO objects. Finally, it is only needed to run a new NN search at the time when the object and the neighbour intersect, i.e. when the norm of the distance between them gets larger at each next time step.

With all of that together, we can finally build an efficient and complete database that will contain around 20'000 objects and their corresponding information.

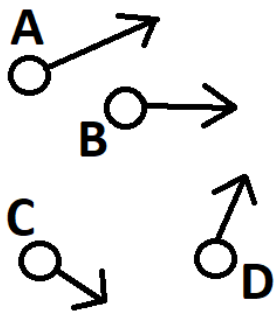

Figure 3: *Example of a nearest approaching neighbour search with four different objects. In this case we will find : A→D because neither B or C are approaching A; B→D because A is not approaching B and D is the nearest neighbour after A; C will have no nearest approaching neighbour and finally D→B because it is the nearest neighbour and is approaching D.*

3 Collision avoidance

This section will be focused on how to compute collision risk (please note that this an exploration of such methods, not their actual implementation), where we have to process this risk for every object in LEO, for a high number of time steps. Firstly, to compute collision probability between different space objects efficiently, we have to consider the unknowns and variables that we have, as well as the high number of objects and iterations we are going through.

As stated before, the position of space objects at a given time is an estimation and thus is not exact. Even the position of an object when it is detected by the most precise devices is estimated to within 1 to 3 kilometers. The computation of its new position using SGP4 is also estimated with an error of 1-3km per day. Therefore when we calculate the probability of collision of two objects, we have an approximation for the start and end positions, so we need to take into account those estimations in the probability calculation, instead of a straight-forward approach.

Propositions of such computations are made in some papers (Bérend [5], Morselli et al. [6], Losacco et al. [7]). Each method brings a trade-off between computation time and precision of the result. Every one of them is using a given number of samples of trajectories with small variations, to give a statistical information about start and end points of both objects. So, one of the main criteria to know if a given method is expensive or not is in part given by its number of samples required.

In our case, as we have to compute collision risk between every space object, we need to have efficient and quick computations, otherwise it could take longer than real time, which would not be useful. However, we also need to have a precise result otherwise we would not be able to know when it is necessary to prevent the collision. Hence, the best way to compute collision risk would be to have a compromise between both and by computing collision risk only when crossing distance is below a certain value, to greatly reduce the number of times it is needed to have this risk computed.

Thus, we would start with a cheap calculation of collision probability over every satellite, where *Line Sampling* [6] seems to be a good choice, as it is cheaper to compute than for instance *Monte Carlo* methods, because there are a lot more samples to make overall. From there, if the results of the *Line Sampling* gives a sufficient probability of collision at a given time step between two space objects, it could be interesting to use a more precise method and time of intersection to confirm it, for instance by using *Subset Simulation* with the Taylor expansion of the Distance of Close Approach of both objects [6]. By doing this process, we will have a precise and complete collision probability between each object in LEO for the whole duration of the simulation.

4 Results

Index	Name	Line 1	Line 2	Times	Positions	Velocities	Nearest	NNTimes	Real Nearest	Real NNTimes
0	EXPLORER 1	1 00004U 58001A 70090.03500497 .07718844 +0...	2 00004 033.1468 334.6171 0024739 311.5310 048...	(2021-06-09T14:13:46.862133, 5)	[[813.8536559058701, -5470.604811530397, -3559....	[[7.722697373394465, 0.7797481251834 15, 0.6033...	[7, 63, 77, 45, 53, 18, 70, 41, 74, 65, 65, 65...	[164, 268, 278, 367, 377, 588, 606, 658, 682, ...	[VANGUARD 3, SCOUT X-1 DEB, DISCOVERER 23 DEB,...	[2021-06-09T14:27:26.862133, 2021-06-09T14:36:...
1	SPUTNIK 3	1 00008U 58004B 60095.16166626 .02607090 +0...	2 00008 065.0599 163.5585 0088318 135.6760 224...	(2021-06-09T14:13:46.862133, 5)	[[3641.963220456387, 1868.58543045031 4, -5122....	[[1.556165099178945 3, 6.724832884802905, 3.641...	[38, 68, 94, 83, 38, 58, 98, 98, 98, 98, 98, 9...	[129, 416, 442, 450, 673, 696, 697, 698, 699, ...	[COURIER 1B, SPUTNIK 10, THOR ABLESTAR DEB, SO...	[2021-06-09T14:24:31.862133, 2021-06-09T14:48:...
2	EXPLORER 4	1 00009U 58005A 59275.28696823 .00892850 +0...	2 00009 050.2549 082.4216 0254534 057.7226 303...	(2021-06-09T14:13:46.862133, 5)	[[-5615.819601094353, 4056.766797078636, -643....	[[-3.278349290410224, -3.553356313478101, 5.70...	[35, 99, 32, 35, 99, 50, 16, 16, 16, 16, 16, 1...	[26, 139, 347, 595, 684, 709, 710, 711, 712, 7...	[SPUTNIK 5, DISCOVERER 29, ECHO 1 DEB (MYLAR O...	[2021-06-09T14:15:56.862133, 2021-06-09T14:25:...
3	VANGUARD 2	1 00011U 59001A 21159.69257427 .00000034 0...	2 00011 32.8639 157.7993 1467999 102.7262 274...	(2021-06-09T14:13:46.862133, 5)	[[3243.003910693040 3, 7060.029839835879, -5015...	[[-5.412155524231037 5, 2.799086699978565, -0.1...	[33, 8, 58, 72, 61, 25, 70, 28, 69, 38, 94, 91...	[88, 155, 192, 238, 274, 393, 486, 504, 526, 6...	[ECHO 1 DEB (METAL OBJ), EXPLORER 7, ATLAS AGE...	[2021-06-09T14:21:06.862133, 2021-06-09T14:26:...
4	VANGUARD R/B	1 12U 59001B 21160.17374792 -.00000013 0...	2 12 32.8957 182.7665 1666032 151.5210 218...	(2021-06-09T14:13:46.862133, 5)	[[-6057.942305833695, 6130.101129409849, -4050...	[[-4.917505948681089 5, -2.895150761723169 5, 1....	[84, 83, 78, 49, 21, 96, 88, 29, 68, 1, 8, 58,...	[15, 70, 93, 163, 215, 432, 480, 499, 584, 601...	[THOR ABLESTAR R/B, SOLRAD 3/INJUN 1, EXPLORER...	[2021-06-09T14:15:01.862133, 2021-06-09T14:19:...

Figure 4: *Example of a complete database*

Figure 4 shows an example of a final database, using the first 100 space objects (from a total of 20'087) of the whole TLE file given by the query from space-track.org using the conditions to get LEO objects, as defined in section 2.1. We can see every column used in the complete implementation of the database :

- **Index** : The index number of each satellite inside the database.
- **Name** : The name of the satellite given in its TLE.
- **Line 1** : The second line in its TLE.
- **Line 2** : The third and last line of its TLE.
- **Times** : A tuple giving the exact date of the first time-step, and the number of seconds between each time-step.
- **Positions** : The $M \times 3$ array of position vectors through time of each object, where T is the number of time-steps in the whole simulation.
- **Velocities** : The $M \times 3$ array of velocity vectors through time, giving the speed and direction of the object at each time-step.
- **Nearest** : The array of indexes of each nearest approaching neighbour through the whole simulation.
- **NNTimes** : The array of the intersection time for each NN given by the previous column. For instance, the satellite "EXPLORER 1" will cross the satellite indexed 7 ("VANGUARD 3") at the 164th time-step.
- **Real Nearest** : The array of the names of each nearest approaching neighbour, so that we can have easier insight on both crossing satellites.
- **Real NNTimes** : The array of exact dates of intersection times, to have more insight on when a collision might happen.

This database thus contains a lot of information per space object, but is still pretty efficient to compute collision risks, by minimizing the cost of the catalogue initialization and updates and also reducing the number of information stored to the minimum. It is then possible to insert new detected objects, delete or search inside the database efficiently, while having a regular update possible, to continuously keep track of collision risks through time. Indeed, it would not be a good idea to do a simulation for more than the next few days, as space object orbits might change slightly every day and is updated regularly in public catalogues. Hence, an initialization of around 2 or 3 days with an update every 5 to 10 hours seem sufficient.

Such a database gives the tools to improve usual methods and computations of a lot of domains, starting with collision risks which was the main objective of the project, but also interactive 3D visualizations, classification and comparison with newly detected objects, and so on.

5 Conclusion

In this project we have seen how to build an efficient database to optimize the computation of collision risk in the case of space objects orbiting in *Low Earth Orbit*. We have seen how we can compute and simulate objects in LEO to store inside our database and that there are multiple possible approaches to build and maintain such a database. Surprisingly the best one seems to be the "simplest" way where each object looks at the whole database at any needed time step, to search its nearest approaching neighbours.

Then, by using the information of intersection time and relative positions, we are able to determine between which objects and at what time it is needed to compute a collision probability. We have seen multiple possible ways to compute it, and concluded that a mix between multiple methods and granularity seems to be the most efficient and useful one for our project, because we have a lot of objects and collision risks to calculate over the whole simulation.

Our model seems to be sufficiently efficient time and memory wise to store all detected objects in LEO, while also providing an easy access to regular updates and allowing to get collision risks for every object at any point in time. It could however be further improved by doing a regression on trajectories, so that they are more accurate and would prevent outlier coordinates that would mess up the calculations. Further work can also be applied on the implementation of the database, to parallelize it and accelerate the overall computation time of the initialization. Indeed, the trajectory computations are completely independent from each other and once they are all computed, every NN search can be independent and thus parallelized with little effort.

References

[1] David A. Vallado & Paul Crawford. *SGP4 Orbit Determination*, 2008.
Conference: *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*

[2] Antonin Guttman. *R-Trees – A Dynamic Index Structure for Spatial Searching*, 1984.
Conference: *SIGMOD'84, Proceedings of Annual Meeting, Boston, Massachusetts*

[3] Mengchu Cai & Peter Revesz. *Parametric R-Tree: AnIndex Structure for Moving Objects*, 2000.

[4] Simonas Saltenis, Christian S. Jensen, Scott T. Leutenegger, Mario A. Lopez. *Indexing the Positions of Continuously Moving Objects*, 2000.
Conference: *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data*

[5] Nicolas Bérend, *Estimation of the probability of collision between two catalogued orbiting objects*, 1999.
Advances in Space Research, Volume 23, Issue 1, Pages 243-247

[6] Alessandro Morselli, Roberto Armellin, Pierluigi Di Lizia & Franco Bernelli Zazzera, *A high order method for orbital conjunctions analysis: Monte Carlo collision probability computation*, 2014.
Advances in Space Research, Volume 55, Issue 1, Pages 311-333

[7] Matteo Losacco, Matteo Romano, Pierluigi Di Lizia & Camilla Colombo, *Advanced Monte Carlo sampling techniques for orbital conjunctions analysis and Near Earth Objects impact probability computation*, 2019.
Conference: 1st NEO and Debris Detection Conference

[8] SSA-Team : *EPFL Space Situational Awareness Team*
www.epfl.ch/epfl-space-situational-awareness-team visited on may 9th, 2021

[9] LeoLabs : *EPFL Space Situational Awareness Team*
https://www.leolabs.space visited on april 4th, 2021

[10] Brandon Rhodes : *SGP4 Python library*
https://pypi.org/project/sgp4/ visited on march 4th, 2021

[11] Space-Track : *Public catalog for space objects*
https://www.space-track.org/ visited on april 4th, 2021

[12] UCS-USA : *UCS Satellite Database*
www.ucsusa.org visited on april 19th, 2021.

[13] Celine Deluzarche : *Combien de satellites tournent autour de la Terre?*
www.futura-sciences.com visited on april 11th, 2021.

[14] ESA : *Space debris by the numbers*
www.esa.int visited on april 5th, 2021