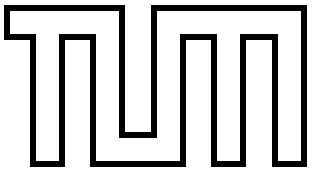


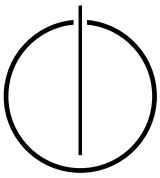
FAKULTÄT FÜR INFORMATIK

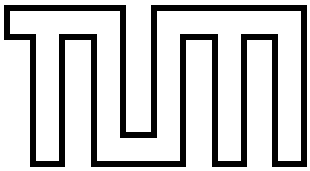
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Modeling Upcoming Megaconstellations in Space Debris Environment Simulations

Albert Noswitz





FAKULTÄT FÜR INFORMATIK

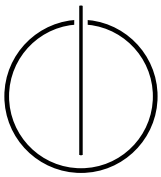
DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Modeling Upcoming Megaconstellations in Space Debris Environment Simulations

Modellierung Geplanter Megakonstellationen im Kontext von Simulationen von Weltraumschrott

Author:	Albert Noswitz
Supervisor:	Univ.-Prof. Dr. Hans-Joachim Bungartz
Advisor:	Fabio Alexander Gratl, M.Sc. and Dr.-Ing. Pablo Gómez
Date:	15.2.2022



I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, 15.2.2022 Albert Noswitz

Acknowledgements

I want to thank my advisors Fabio Gratl and Pablo Gómez for giving me the opportunity to contribute to, and to write a thesis about this simulation project. Both of you helped me out a bunch and I learned a lot in the process.
Another thank you to my friend Patrik for his support on my way to mastering informatics and for proofreading this thesis.

Abstract

In the near future, upcoming megaconstellations will drastically increase the number of operating satellites in low Earth orbit. They will operate in an environment with a large amount of space debris and increase the already existing chance of collisions. The simulation software LADDS aims to predict the evolution of space debris in order to assess risks associated with operations such as the insertion of constellations. The purpose of this work is to add constellations into the simulation LADDS. This is done by firstly implementing a component which generates constellation data from orbital shell parameters. Then, a component is implemented to insert the generated constellation input into the simulation over time. This enables the inclusion of constellation insertion scenarios in the simulation. An 11.5-year simulation run with the implemented functionality and a realistic constellation insertion scenario of about 50,000 satellites brought results in which constellations were involved in over 60% of all conjunctions. A second simulation run demonstrated that the altitude in which a constellation operates significantly influences the collision risk.
With the ability to include the upcoming megaconstellations and with the future integration of a breakup model, LADDS will be capable of predicting the evolution of the space debris environment more realistically.

Zusammenfassung

Bereits in absehbarer Zeit werden geplante Megakonstellationen die Anzahl an eingesetzten Satelliten innerhalb der niedrigen Erdumlaufbahn drastisch erhöhen. In einer Umgebung mit Unmengen von Weltraumschrott werden sie das Risiko von Kollisionen weiter erhöhen. Die entstehende Simulationssoftware LADDS nimmt sich vor, die Entwicklung des Bestandes von Weltraumschrott vorherzusagen, damit die Einschätzung der Folgen von Unternehmungen, wie der Einsatz von Satellitenkonstellationen, gestützt werden kann. Der Zweck dieser Ausarbeitung besteht darin, Konstellationen in die Simulation LADDS zu integrieren. Dazu wird zuerst eine Komponente implementiert, die von der Simulation verarbeitbare Daten anhand geeigneter Eingangsparameter generiert. Anschließend wird eine Komponente implementiert, welche Konstellationen anhand der generierten Daten zeitüberdauernd in die Simulation einsetzt. Dies ermöglicht die Simulation von Szenarien möglicher Zeitpläne zum schrittweisen Einsetzen der Konstellationen. Ein Programmdurchlauf mit der neuen Komponente, der elfeinhalb Jahre simuliert und Konstellationen im Umfang von 50.000 Satelliten nach einem realistischen Zeitplan hinzufügt, ergab, dass 60% der nahen Begegnungen von Objekten in der niedrigen Erdumlaufbahn Konstellationssatelliten involvieren. Ein weiterer Durchlauf zeigte, dass die Höhe, in der Konstellationen eingesetzt werden das Kollisionsrisiko erheblich beeinflussen.
Mit der nun implementierten Funktionalität die geplanten Megakonstellationen mitzusimulieren und der kommenden Integration einer Komponente, die kollidierende Objekte realitätsgetreu in Fragmente zerteilt, wird LADDS dazu fähig sein, die Entwicklung des Bestandes von Weltraumschrott realistischer einzuschätzen.

Contents

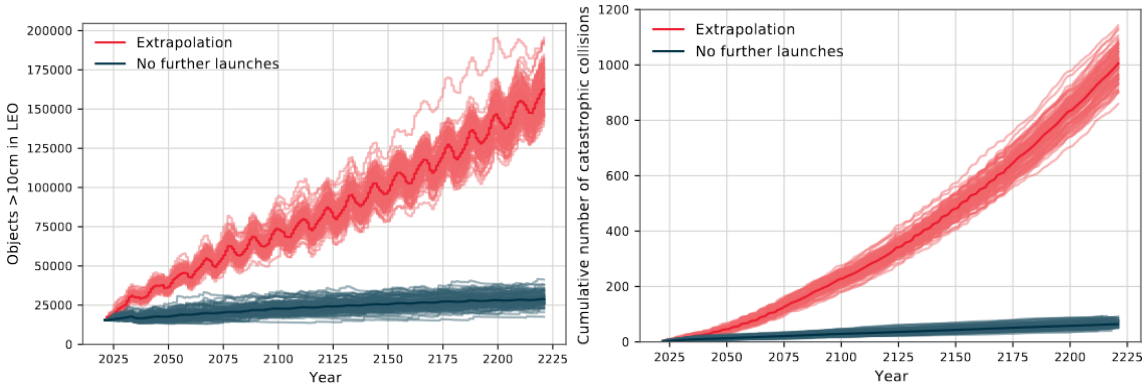
Part I.

Introduction and Background

1. Introduction

On November the 11th, 2019, the first satellites that are part of a megaconstellation were launched by the company SpaceX[1]. SpaceX as well as OneWeb, Amazon and several other companies are set to deploy their megaconstellations in order to provide telecommunication services such as internet connection. The term "megaconstellations" is used due to the amount of satellites within such a constellation. With 12,000 satellites, SpaceX's Starlink constellation alone significantly outnumbers all already present satellites, which counted about 2000 in 2018[2]. This drastic increase of satellites in the low Earth orbit (LEO) worsens concerns over the space debris population, which have existed even prior to megaconstellations. As possible collisions involving space debris can create substantially more space debris, which again increases the chance of further collisions, a scenario of exponential growth in the density of fragments becomes imaginable. In this scenario known as the Kessler syndrome [KJLM10], the feasibility of space flight and the operation of satellites would be impaired significantly. This problem is being addressed in various ways. Space debris mitigation measures such as the removal of space debris, or guidelines to minimize the production of debris during and after space missions aim to lower the overall risk of collisions and prevent the Kessler syndrome[3].
Another important task related to this problem is to assess the risks associated with our use of the near Earth environment by modeling its future space debris population. Approaches to simulate the space debris environment have been realized by several simulations such as LEGEND [LHKO04] developed by NASA and DELTA [Vir16] developed by ESA. In that manner, ESA's annual space environment report of 2021[4] utilized DELTA to predict the amount of collisions and of resulting objects in LEO shown in Figure 1.1. Over the long term, the results project an exponential growth in catastrophic collisions and a drastic increase of objects in space if the same measures and launch rate of recent years are applied in the future.
LEGEND, DELTA and other simulation tools utilize Monte Carlo methods for stochastic conjunction tracking in order to avoid small time steps, which comes at the cost of reduced accuracy. In order to tackle this problem, the simulation "Large-scale Deterministic Debris Simulation" (LADDS), which relies on deterministic conjunction tracking, is being developed[5]. LADDS is a project in the context of an ARIADNA study of ESA's Space Debris Office, ESA's Advanced Concepts Team, and the SCCS chair of the Technical University Munich. In the context of LADDS, this work is concerned with modeling the future megaconstellations and with their insertion into the simulation in order to study their effects on the space debris environment.
The first part of this work introduces necessary theoretical background and information about the planned megaconstellations, before describing the interface of LADDS and the additions of this work. As for the second part of the text, an implementation for the generation of constellations as well as the implementation that integrates constellations into the simulation are detailed. In order to assess the threat associated with the upcoming

megaconstellations, a long-term simulation run with a realistic timetable of constellation launches as well as a second run for enabling the comparison of constellations are conducted. Their results are subsequently presented and analyzed.



(a) Graph depicting amount of objects in LEO.

(b) This graph depicts the amount of catastrophic collisions over time.

Figure 1.1.: Graphs depict the results of 100 Monte Carlo simulations for both scenarios. The "Extrapolation" scenario assumes mitigation measures and spacecraft launching patterns of the previous three years. The "No further launches" scenario assumes no launch to take place in the future. Marked in darker color is the average of all the scenario's simulations.

2. Theoretical Background

2.1. Orbital Elements

Orbital elements are parameters that can be used as a mathematical model for describing the position of objects orbiting a central body at a given time. These objects can either be natural satellites like moons or planets, human made artificial satellites, or space debris. What is essential for modeling this instance of a so called two-body problem is a difference in mass that is large enough to assume negligible movement by the heavier body as a result of the lighter body's exercised force. A satellite orbits a central body, for example a planet, in the form of an elliptical trajectory. The ellipsis exists in a three-dimensional Cartesian coordinate system that describes the central bodies environment by placing the origin at the center of mass and by defining conventional direction vectors. These are the only necessary points of reference, as the central body's rotation is not considered when modeling the ellipsis. When the Earth takes the place of the central object, the system is called "geocentric equatorial coordinate system". [Wal18]
Before the semantics of each orbital element are explained according to [Wal18], a few terms and definitions are introduced.

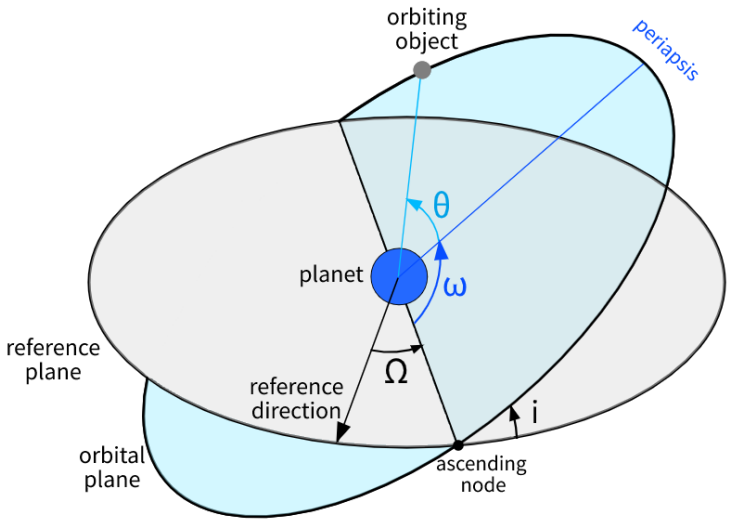


Figure 2.1.: Sketch illustrating some of the orbital elements: inclination i, longitude of the ascending node Ω, argument of periapsis ω, true anomaly θ

For the sake of conventionality and simplicity, a reference plane, that is ideally spanned

by two of the system's axes, is defined, as depicted in Figure 2.1. Pointing from the origin to an arbitrary point of the reference plane is the reference direction. Its value is also chosen to follow conventionality and usually points to the vernal point [Wal18]. Within the orbital plane exists the ellipsis that is the satellite's trajectory. Like any ellipsis, it is defined by two focal points F_1, F_2 as well as a constant sum of distances to the shape's focal points that every point on the ellipsis has. It has to be noted that the central body, in case of an elliptical orbit, is not at the center of the ellipsis, but on one of the focal points [Wal18]. Because of this, two more points are relevant when analyzing orbits: The periapsis, which is the point with the least distance to the central body and its counterpart, the apoapsis, which is the point with the biggest distance to the central body.

Modeling the Ellipsis' Shape

The first two parameters, the semi-major axis a and eccentricity e, are sufficient to determine the exact shape of the ellipsis. The semi-major axis a is defined as the length of the line connecting periapsis and apoapsis divided by two (see Equation 2.1), while the eccentricity parameter quantifies the distance between the two focal points according to Equation 2.2. It can be observed that the lower the eccentricity is, the more circular the ellipsis becomes. In the special case $e = 0$, the ellipsis resembles a circle and the two focal points are at the same point in space.

$$a = \frac{1}{2}(r_{apo} + r_{per}) \tag{2.1}$$

$$e = \frac{r_{apo} - r_{per}}{r_{apo} + r_{per}} \tag{2.2}$$

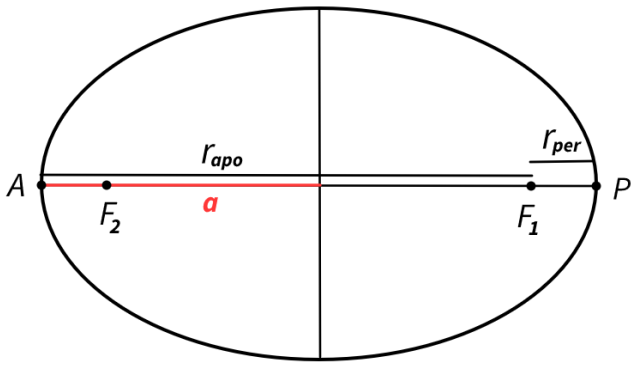


Figure 2.2.: An ellipsis and its landmarks relative to a focal point F_1: the apoapsis marked by A, the periapsis marked by P, and their respective distances to F_1: r_{apo} and r_{per}. The semimajor axis a equals half of the line connecting the apsides.

Most constellation satellites in LEO orbit the Earth in an approximately circular trajectory[6] and the parameters e, a are therefore set to 0 and to the radius respectively. This will hold for all constellation satellites modeled in this work.

Modeling the Ellipsis' Orientation

In order to define how the ellipsis is oriented w.r.t. the origin, the three parameters inclination i, longitude of the ascending node Ω, and the argument of periapsis ω must be specified. The inclination i determines the angle between the reference plane and the plane of the trajectory. The inclination of a satellite orbiting the Earth is automatically the maximum value of latitude the satellite reaches during orbit, assuming the Earth's equatorial plane serves as the reference plane. For instance, a satellite with $i = 55°$ alternates between 55° latitude on the northern hemisphere and the southern hemisphere.
By assigning an angular value to the longitude of the ascending node Ω, the ellipsis is rotated around the reference plane's normal vector on the origin by the set value. This determines the location of the ascending node, which is the point where the ellipsis intersects the reference plane and where the satellite ascends w.r.t. the reference frame.
The final element for creating the trajectory, the argument of periapsis ω, represents the angle between the periapsis and the ascending node on the orbital plane and thereby enables moving the apses of the ellipsis. This parameter does not change the geometry of circular trajectories that are characterized by $e = 0$ since every point on the trajectory could potentially be the periapsis. What does change for circular orbits is the starting position for the true anomaly θ.

Positioning the satellite on the ellipsis

Since the process of orbiting is a repeating process and the satellite will revisit the same position periodically, the true anomaly θ can be seen as a temporal parameter as much as it can be viewed as a spatial parameter. In that sense, the true anomaly describes the phase of the satellite at a given point in time, and enables placing the satellite onto any point of the trajectory. It does so by determining the angle between the satellite and the periapsis relative to the origin on the orbital plane. The mean anomaly M, an alternative means of determining the location of the satellite, is based on passed time rather than a pure angle, but both are equivalent for circular orbits. For the rest of this work, the shape of satellite trajectories is assumed circular.

2.2. Satellite Constellation Design

Proper satellite constellation design is concerned with optimizing various factors. In the case of the upcoming megaconstellations that provide telecommunication services, two important goals are to comprehensively cover surface area of the Earth and to minimize the risk of collisions with catastrophic impact. A common approach that is used to create constellations for communication purposes is to apply the Walker Pattern [GXG+20]. A Walker constellation is a set of so-called orbital shells which each consist of orbital planes. An orbital plane is again a set of satellites that move in the same trajectory. For the satellites within a plane, which are only different by their mean anomaly value M, as well as the planes within a shell, which only differ in their longitude of the ascending node Ω, it applies that their respective orbital elements are evenly distributed within the range of 0° to 360°. This is visualized in the Figures 2.5, 2.6 and expressed mathematically in the Equations 2.3 and 2.4 with p,s being the indices and P, S being the total number of planes and satellites

respectively [GXG+20].

$$\Omega_p = \frac{2\pi}{P}(p-1) \quad , \quad p \in \{1, ..., P\} \tag{2.3}$$

$$M_s = \frac{2\pi}{S}(s-1) \quad , \quad s \in \{1, ..., S\} \tag{2.4}$$

Another property of Walker constellations is that the satellites share a common value for the semi-major axis a, eccentricity e, the inclination i, and the argument of periapsis ω [GXG+20]. Walker constellations are separated into two categories: A Walker Delta constellation refers to constellations made up of satellites with an inclination value that is noticeably smaller than 90°, whereas Walker Star constellations consist of satellites with an inclination close to 90° [LCY21]. Examples of both constellation designs are shown in Figure 2.3. Coverage of the Earth's surface by satellites is maximized by utilizing the Walker Star pattern which makes all satellites reach the poles, whereas Walker Delta constellations concentrate their satellites on a smaller area. The area covered is also dependent on both the amount of satellites and the operating altitude of each satellite. The covered area decreases with a lower altitude as demonstrated in Figure 2.4. Many satellites are therefore needed for constellations in low altitudes [GXG+20] which can also be observed in Chapter 3, where orbital shells with low altitudes tend to have more satellites, and thus, a higher contribution to the LEO population.

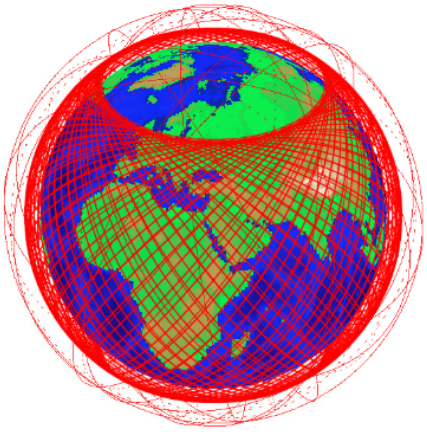


Figure 2.3.: Shells of the Starlink constellation containing both Walker Delta shells that do not reach the poles and Walker Star shells that cover all of the Earth's surface
Source: [APSODG21]

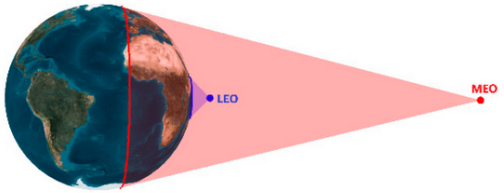


Figure 2.4.: The figure shows the Earth's surface area covered by satellites in different altitudes.
Source: [GXG+20]

Another parameter of Walker constellations, the phasing parameter F, is concerned with the collision risk, which is the most relevant factor for satellite constellation design. It determines the relative phasing between neighboring planes which describes an offset that is added to the mean anomaly and accumulates with every new plane. The introduction

of relative phasing extends the mean anomaly Equation of a satellite s in a plane p from Equation 2.4 to Equation 2.5.

$$M_{sp} = \frac{2\pi}{S}(s-1) + \frac{2\pi}{T}F(p-1)\ ,\ s \in \{1,...,S\},\ p \in \{1,...,P\},\ F \in \{0,...,P-1\} \quad (2.5)$$

$T = P \cdot S$ stands for the total number of satellites in the shell and the property of F to be an integer between 0 and $P-1$ ensures the same relative phasing to hold between satellites in the planes P and 1. As it can be seen in Equation 2.5 as well as in Figure 2.6, the relative phasing between satellites in neighboring planes depends on the plane index p, while the spacing between satellites in the same plane depends on the satellite index s.

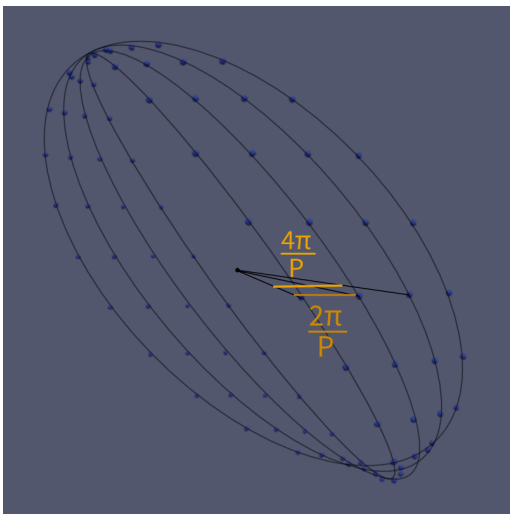


Figure 2.5.: A cutout of four planes of an orbital shell. The planes orientation only differs in their longitude of the ascending node Ω and planes always have the same angular distance to their neighboring planes. The F parameter of this shell is set to 0.

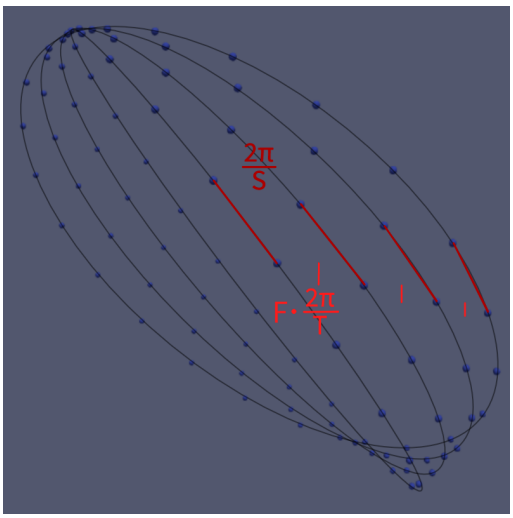


Figure 2.6.: Marks the two step sizes that determine the mean anomaly M of a satellite: the relative phasing step in lighter red, and the step between satellites in the same plane in darker red. Marked distances should be understood as angular distances between the points w.r.t. the center of mass.

According to [LCY21], the phasing parameter F determines the minimum distance between satellites of a constellation. Determining an optimal parameter F, which can be done by simulating the orbital shell for each possible value of F, can drastically increase the minimum distance between satellites. It is therefore an essential parameter to be considered for Walker constellations, which are denoted by the expression **i: T/P/F**.

3. Upcoming Megaconstellations

In this chapter, satellite megaconstellations are presented by describing their orbital shell parameters and by discussing their general launching conditions. While the launches of all presented constellations are planned, the exact execution of their plans is differently certain for each constellation. OneWeb and SpaceX's Starlink have already launched parts of their constellation, while constellations of other companies, which include Telesat and Amazon, are expected to launch in the near future. For most constellations, the Federal Communications Commission (FCC) plays a key role, as the constellations need to be FCC approved in order to launch. The FCC is an American commission that regulates interstate and international communications and distributes the frequency bands necessary for the constellation's service. All previously mentioned companies have parts of their proposed constellation approved by the FCC [PdPCC21]. For Astra's constellations, that are also included in this space debris environment simulation, an FCC request has been filed[7], but has not been authorized as of January 2022. The constellation GuoWang planned by the Chinese government[8] is not dependent on FCC approval. Since some information of constellation parameters are only partially available, one of the sources used for constellation information [BHGE21] creates representative shell parameters based on the incomplete data. Nevertheless, the following presented data is used for creating a possible and representative future scenario that is simulated and then analyzed in Chapter 6.

SpaceX intends to deploy two constellations in LEO. About 1900 satellites of the first constellation with a total of nearly 12,000 satellites, are in operation as of January 2022[9]. 4408 of them inhabit altitudes between 540 to 570 kilometers and about 7500 are planned to inhabit low altitudes between 336 and 346 kilometers, as shown in Table 3.1. All satellites of the constellation have been approved by the FCC[10] and the constellation is the biggest approved constellation yet in terms of the number of satellites. A large number of 30,000 further satellites is planned to operate as part of the Starlink Generation 2, which will also be considered in Chapter 6, although it is not a part of the 11.5-year simulation scenario [BHGE21]. Its parameters are listed in Table 3.3.

OneWeb, the other company to start their constellation launch early, also plans to operate two constellations. Both the first as well as the second constellation will operate in a 1200 kilometer altitude. Most of the first constellation's 716 satellites orbit the Earth in polar planes with an inclination of 87.9°. All 716 satellites are approved by the FCC. OneWeb's second set of satellites comprise 6372 more satellites [PdPCC21]. The orbital shell parameters are listed in Table 3.1.

Telesat's constellation, which is announced to launch in early 2023[11], comprises two orbital shells, one polar shell of 351 satellites in 1015 kilometer altitude and another shell of 1320 satellites in a high altitude of 1325 [PdPCC21]. The Kuiper constellation by the company Amazon is fully approved by the FCC and is expected to start with its constellation deployment in late 2022[12]. It comprises 3236 satellites that will be placed in altitudes between 590 and 630 kilometers [BHGE21]. The parameters of both Telesat and Kuiper are

shown in Table 3.2.
Astra plans to launch up to three constellations depending on their customers demand [13]. The first constellation of 40 satellites is a single equatorial plane with an inclination of 0° at an altitude of 700km and the second constellation of about 2300 satellites will be deployed at the same altitude. Their third constellation is over 11,000 satellites large and would be located within 380-400 kilometer altitudes[14]. All constellations with their parameters are shown in Table 3.2.
Lastly, the GuoWang constellation includes about 13,000 satellites in three discernible altitude regions: 3600 satellites in a 508km altitude, about 2500 in altitudes of 590km and 600km, and four shells worth about 6900 satellites in 1145km altitudes [BHGE21]. The exact shell parameters are shown in Table 3.3.

	Altitude (km)	i (°)	Planes	Satellites per plane	N
Starlink 1	540	53.2	72	22	1584
	550	53.0	72	22	1584
	560	97.6	6	58	348
	560	97.6	4	43	172
	570	70.0	36	20	720
	346	53.0	42*	60*	2547
	341	48.0	42*	60*	2478
	336	42.0	42*	60*	2493
OneWeb 1	1200	87.9	12	49	588
	1200	55.0	8	16	128
OneWeb 2	1200	87.9	36	49	1764
	1200	55.0	32	72	2304
	1200	40.0	32	72	2304

Table 3.1.: Orbital shell parameters of Starlink Generation 1, OneWeb's first and second constellation. Numbers marked with (*) are estimates according to the source [BHGE21] that roughly amount to the known total count of satellites.

	Altitude (km)	i (°)	Planes	Satellites per plane	N
Telesat	1015	98.98	27	13	351
	1325	50.88	40	33	1320
Kuiper	590	33.0	28	28	784
	610	42.0	36	36	1296
	630	51.9	34	34	1156
Astra 1	700	0	1	40	40
Astra 2	690	98.0	14	36	504
	700	55.0	56	32	1792
Astra 3	380	97.0	20	112	2240
	390	30.0	51	96	4896
	400	55.0	61	68	4148

Table 3.2.: Orbital shell parameters of Telesat, Kuiper and Astra's constellations

	Altitude (km)	i (°)	Planes	Satellites per plane	N
GuoWang	590	85.0	8	60	480
	600	50.0	40	50	2000
	508	55.0	60	60	3600
	1145	30.0	27	64	1728
	1145	40.0	27	64	1728
	1145	50.0	27	64	1728
	1145	60.0	27	64	1728
Starlink 2	328	30.0	7178	1	7178
	334	40.0	7178	1	7178
	345	53.0	7178	1	7178
	360	96.9	40	50	2000
	373	75.0	1998	1	1998
	499	53.0	4000	1	4000
	604	148.0	12	12	144
	614	115.7	18	18	324

Table 3.3.: Orbital shell parameters of GuoWang and Starlink Generation 2

4. Implementation Context

The objective of this chapter is to describe LADDS[15], the overarching simulation software for this work, and how it interfaces with the addition of constellations that will be implemented. At first, the general methodology of the simulation is presented, before the structure of the simulation's code is introduced by describing the simulation cycle and the involved components. Following that is an overview over how the constellation component is related to the simulation, before the component's actual implementation is presented in Chapter 5.

4.1. Simulation Approach

LADDS is a collaboration between the Chair of Scientific Computing in Computer Science of the Technical University Munich, ESA's Space Debris Office as well as ESA's Advanced Concepts Team. It aims to create the first deterministic space debris environment simulation, in contrast to approaches that use Monte Carlo methods for finding collisions. The simulation currently simulates objects tracked by Celestrak[16] that inhabit the lower Earth orbit between 200 and 2000 kilometers above the Earth's surface, while the inclusion of future constellation satellites is the matter of this thesis.

4.2. Simulation Structure

In order to create a simulation for making a prediction of the future space debris environment, certain procedures have to be performed continuously. One task is to propagate the objects based on the effective laws of physics. Secondly, the interactions between the objects, which are collisions or close encounters in more general terms, have to be considered and finally be communicated by creating output. The component of this simulation that propagates the objects, applies leapfrog integration based on various accumulated forces [Bö21]. Detecting the collisions is based on AutoPas [GSBN21], a library for N-Body simulations that provides containers for the simulated objects. These objects, in this case satellites and debris, are modeled by particles, which are points in space without shape. AutoPas containers utilize efficient data structures, parallelization, and dynamic auto-tuning[17] for the pairwise iteration of particles serving as a base for particle interactions. In LADDS, the concrete simulation cycle is realized as depicted in Figure 4.1. The first part of the simulation cycle `integrate()` is performed by the integrator component that updates the position and velocity values of every particle. `updateConstellation()` is a function that will be inserted to follow the `integrate()` function and is explained later in this work. As the positions of particles change due to the integration step and particles are potentially added

in `updateConstellations()`, AutoPas must update internal structures. This requires the invocation of the `updateContainer()` function.

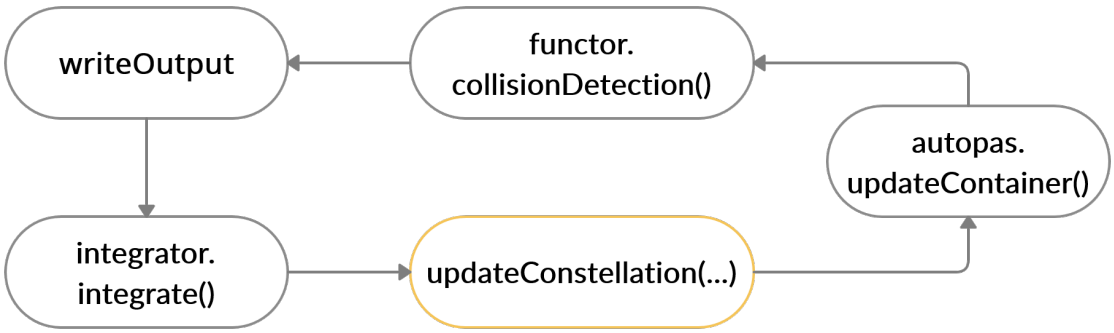


Figure 4.1.: This figure shows the simplified cycle of code executed. Marked in orange is the function to be added in this thesis.

While the collision detection is based on AutoPas, the component to call the necessary function `collisionDetection()` is a dedicated functor implementing the detection of collisions. This functor, the `CollisionFunctor`, utilizes an AutoPas function to iterate over pairs of particles efficiently, while performing the operation on these handed pairs. The distance of particle encounters is not computed by calculating the distance between the two particles at one time step, but by linearly interpolating the trajectories of each particle by sampling their positions at the current and the previous time step, before calculating their minimum distance. This allows for a bigger time step that increases the simulation speed, while adding a negligible error value to the measured distance. At the very end of the iteration, the detected collisions are printed into a HDF5 output file and serve as the main information for the results analysis in Chapter 6. Further output includes snapshots of the simulation state, which is also relevant for analyzing the simulation result, as well as visualization output files (e.g. .vtu). These output operations can be, and often should be, performed not at every single iteration, but in intervals, as outputting the simulation state in each iteration consumes a lot of memory.

4.3. Integrating Constellations

Between the integration step and the container update, the functionality to insert satellites of constellations into the simulation can be implemented. This is done by implementing a function `updateConstellations()` that adds satellites from objects of a class `Constellation` to the simulation's particle container. Among other attributes, the `Constellation` objects contain attributes that allow scheduled insertion, all of which is discussed further in Section 5.2.

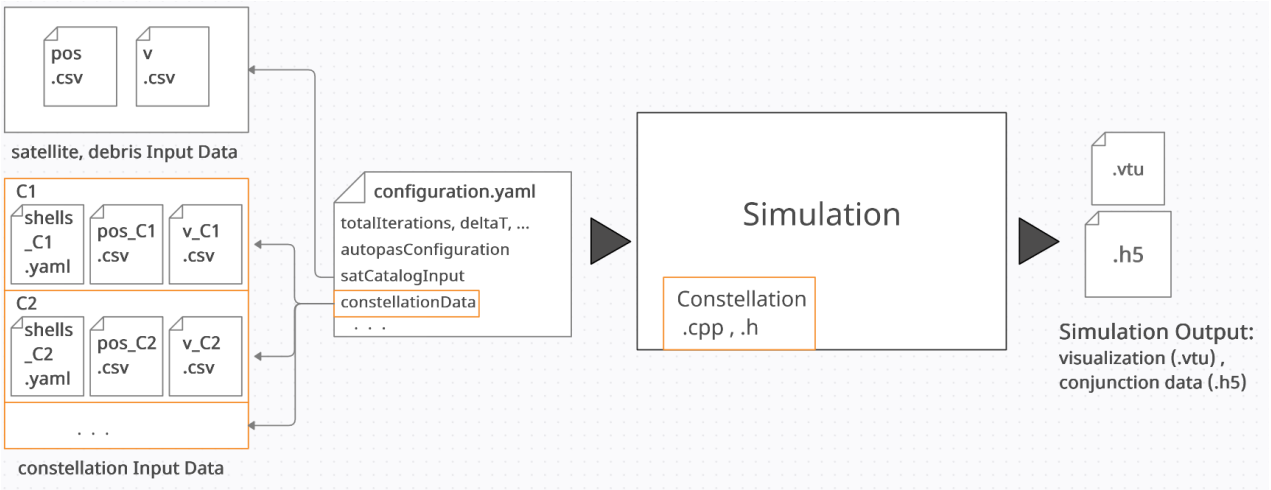


Figure 4.2.: Overview over how this work is related to the space debris environment simulation. Orange rectangles mark additions to the simulation. The generation of constellation input data is discussed in section Section 5.1, the addition of the constellation input in Section 5.2, and produced simulation data is presented and analyzed in Chapter 6.

A configuration file serves to configure the simulation by setting the parameters which include the number of iterations, the simulated time step between each iteration, etc. (see Figure 4.2). In addition to the already included particles originating from data of tracked satellites[18] and debris presently in orbit, constellation data is fed into the simulation to initialize the `Constellation` objects. The generation of the input data is also a part of this work and is done separately by a Python program described in Section 5.1. It creates directories with the constellation data that is distributed in three files: Two .csv files, each containing one of the two state vectors, the position and the velocity, for every satellite and a .yaml file with the constellation parameters. Both the constellations and the other objects in space are fed into the simulation by specifying their file paths in the configuration file.

Part II.

Implementation and Results

5. Implementation

5.1. Generation of Constellations

The following implementation enables a user to create orbital shells by specifying the altitude above the Earth's surface, the inclination, the number of planes within the shell, and the number of satellites in each plane. These orbital shell parameters are commonly used in literature to characterize shells [PdPCC21] [BHGE21], as it is also done in Chapter 3. Further parameters, that are rather secondary but nevertheless necessary in some circumstances, are addressed later in Subsection 5.1.2. This implementation of the constellation generation utilizes the libraries numpy (np) and pykep (pk), a "scientific library to provide basic tools for astrodynamics research."[19]

5.1.1. Creation of Shells from Parameters

Since constellations are a set of shells, the program needs to read in and process one 4-tuple (`altitude` , `inclination` , `nPlanes` , `nSatellites`) at a time. The first objective is to convert this tuple into a list of orbital elements for all satellites in the shell, which is achieved by the code in Listing 5.1. At first, variables representing the orbital elements are initialized, which includes the semi-major axis `a`, eccentricity `e`, inclination `i`, longitude of the ascending node `W`, argument of periapsis `w`, and mean anomaly `M`.

```
1   a = altitude * 1000 + 6371000
2   e = 0
3   i = inclination * pk.DEG2RAD
4   W = 0
5   w = 0
6   M = 0
7
8   pStep = 2 * math.pi / nPlanes
9   sStep = 2 * math.pi / nSatellites
10
11  for x in range(nPlanes):
12      for y in range(nSatellites):
13          elements_list.append([a,e,i,W,w,M])
14          M = M + sStep
15      W = W + pStep
16      M = 0
17  shell_list.append((altitude,inclination,nPlanes,nSatellites))
18  nShells += 1
```

Listing 5.1: Python 3 code for generating a shell given parameters altitude, inclination, nPlanes, nSatellites and initialized empty lists elements_list, shell_list

Apart from the variables `W` and `M`, the orbital elements remain unchanged for all satellites of the shell. As the satellites are placed w.r.t. the center of Earth, the Earth's radius of 6371

kilometers is added. After the step sizes for the variable orbital elements `W`, `M` are accordingly assigned, the orbital elements of every satellite have to be stored in a list. Increasing `M` by its step size `sStep` for each new satellite in a plane and increasing `W` by its step size `pStep` for each new plane places the satellites to form a Walker constellation. In order to store the shell information, which is necessary data for the insertion of constellations discussed in Section 5.2, the parameters are stored in `shell_list`. The previous steps must be executed for every shell, before the accumulated list of orbital element sets is converted into the state vectors by the library function `pk.par2ic()`:

```
satellites = []
for elements in elements_list:
  pos,v = pk.par2ic(elements,pk.MU_EARTH)
  pos = np.asarray(pos) / 1000.0
  v = np.asarray(v) / 1000.0
  satellites.append((pos,v))
```

The constant `pk.MU_EARTH` describes the Earth's gravity parameter and must be passed to ensure that the satellites orbit the Earth. Conversely, the function `pk.ic2par()` would create orbital elements from state vectors.

5.1.2. Setting Further Parameters

Four further parameters are added to account for arising problems and to increase the accuracy of some constellation models. In order to be able to generate any circular Walker constellation, relative phasing is added as the variable `offsetM`, which determines the phase difference between neighboring planes. The phase difference between the last and the first plane is only the same as the other differences if `offsetM` is assigned a value `F*2*math.pi/(nPlanes*nSats)` according to the second summand of Equation 2.5. In the implementation, multiples of `offsetM` based on the plane index have to be added to the total mean anomaly `M` of each orbital element set.
Another parameter `startW` is needed when multiple shells overlap[20] due to similar altitude and inclination, which is the case for shells in Starlink's first constellation listed in Table 3.1 and other presented constellations. The problem is solved, if an offset is added to the longitude of ascending node `W`. Assigning the value of the following equation to this offset causes that no overlapping occurs for N shells that would overlap otherwise:

$$startW = \frac{360°}{G} \cdot \frac{i}{N} \quad , \quad i \in \{0, ..., N-1\} \tag{5.1}$$

G denotes the smallest common multiple of the shells' `nPlanes` values and every shell must have a unique index i. The idea behind this equation can be explained by using the example of two Starlink shells listed in Table 3.1, which have identical altitude and inclination values. When `W` values are distributed to each plane using the same offset 0, both shells have planes at `W = 0` and `W = 180`, which overlap entirely. In order to prevent the overlapping, both shells are first assumed to have the same number of planes G, which is 12 in this example. The resulting step size 30°, which is the evaluated left fraction of Equation 5.1, is the angle between two planes in each 12-plane shell. By multiplying with the right fraction, the offsets 0° and 15° are created in order to maximize the distances between planes from the different shells. Because both original shells of 4 and 6 planes are each a subset of the extended shell

with 12 planes, the application of different offsets according to Equation 5.1 also prevents overlapping between planes of the original shells.
The third parameter `W_area` enables shells to span a different angle than the usual range of 360°. This is implemented by replacing the numerator of line 8 in Listing 5.1 with the radians value of `W_area`.
Finally, the parameter `argPeriapsis` corresponds to the argument of periapsis variable `w`. Modifying `argPeriapsis` practically determines the phase of the entire shell, which can be done in order to prevent overlapping satellites at the simultaneous insertion of constellations.

5.1.3. State Vector and Metadata Output

Creating the constellation input for the simulation requires the data gathered in the lists `satellites` and `shell_list` to be written into files. Firstly, a directory is created to contain the data of the files. The .csv file for the positions and the .csv file for the velocities are created by extracting the respective vector list stored in `satellites`. Writing the .yaml file that contains the constellation's meta information, such as used parameters and the time of launch and its insertion duration, requires the import of the `yaml` package. .yaml files are structured similarly to python dictionaries and can be set up by creating a dictionary. In order to produce the desired output format depicted in Listing 5.2, the dictionary entries are implemented as shown in Listing 5.3.

```
constellation:
  name: OneWebPhase1
  startTime: 2020/02/06
  duration: 1057d
  nShells: 2
shell1:
  altitude: 1190
  inclination: 87.9
  nPlanes: 12
  nSats: 49
shell2:
  altitude: 1200
  inclination: 55.0
  nPlanes: 8
  nSats: 16
```

Listing 5.2: Example of the desired file structure

```
constellation_cfg = {
  "constellation": {
    "name": constellation_name,
    "startTime": startTime,
    "duration": duration,
    "nShells": nShells
  }
}
for i in range(nShells):
  constellation_cfg["shell" + str(i+1)] = {
    "altitude": shell_list[i][0],
    "inclination": shell_list[i][1],
    "nPlanes": shell_list[i][2],
    "nSats": shell_list[i][3]
}
with open(shells_path, "w") as fh:
  yaml.dump(constellation_cfg, fh,sort_keys=
      False)
  fh.close()
```

Listing 5.3: Python 3 code for the creation of output formatted according to Listing 5.2

For the reason of tracing back the parameters needed to generate the state vectors, the values of the additional parameters can be included in the .yaml file analogously to the orbital shell parameters. They are however not needed for the implementation of the insertion into the simulation.

5.2. Integration of Constellations into the Simulation

In this approach, the integration of constellations into the simulation consists of three parts: the implementation of a class `Constellation`, a method `initConstellations()` in the `Simulation` class invoking the constructor for each `Constellation` instance during the initialization, and the `updateConstellations()` method that updates each `Constellation` instance in the simulation loop.

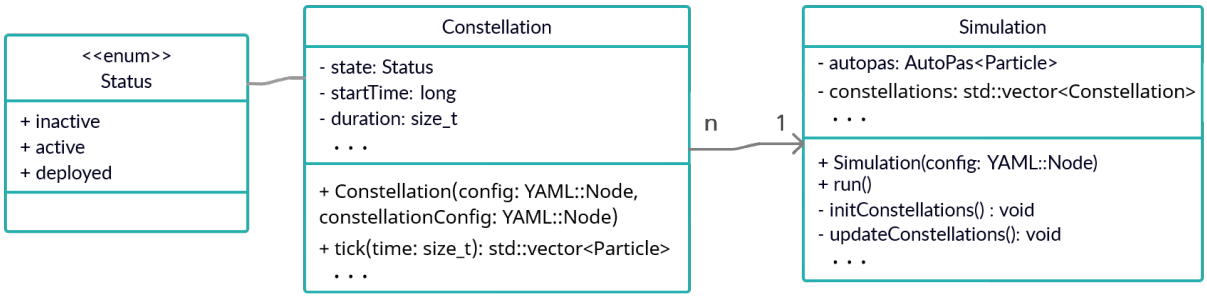


Figure 5.1.: A simplified UML diagram that depicts the `Constellation` class and its relation to the `Simulation` class.

As it can be seen in Figure 5.1, the interface of `Constellation` objects consists of a single method `tick()`, apart from the constructor. This method is called in the simulation loop in order to retrieve particles from the constellations according to a schedule. The schedule partially depends on a time of launch `startTime` and a duration `duration`, which are parameters from the constellation's .yaml file. Each constellation object is on the one hand constructed with information about simulation parameters contained in the `config` object and by the `constellationConfig` object with constellation data on the other hand. These objects are `YAML::Node` objects that read the .yaml files that have been depicted in Figure 4.2.

5.2.1. Initialization of Constellations

Initializing the constellations in the `Simulation` class is done by the member function `initConstellations()`. It creates a vector of `Constellation` objects by reading a configuration parameter containing the names of the constellations chosen to be included. After finding the .yaml file for each constellation based on the name, the corresponding `YAML::Node` is passed to the constructor call.
By using the passed configuration information, the constructor sets the attributes `deltaT`, the time simulated in one iteration, and `interval`, the number of iterations between two invocations of `updateConstellations()`. These members are necessary for tracking the simulation time and for converting the launch schedule information from calendar time into simulation time measured in iterations. This conversion sets the two parameters `startTime` and `duration` and is explained in Subsection 5.2.2.
The actual element of the constellation, the satellites that are modeled as `Particle` objects, are loaded from the .csv files containing the state vectors and then stored in a `std::deque`.

Determining the .csv files' paths is based on the YAML-attribute `constellationName`, which is used as a key to differentiate between the files associated with different constellations. Metadata for each shell of the constellation is included in its configuration file and is needed for the particle insertion algorithm. They are read and stored in a vector of 4-tuples `shells`. Unique ID values have to be distributed to all the simulated particles in order to identify the particles involved in a collision. As it is convenient for analyzing the collision data, constellations are each allocated their own range of integers. A way to additionally achieve visual discernibility is by allocating each constellation a range as big as a power of ten that safely exceeds any constellation size. Additionally, the non-constellation input, which consists of the tracked satellites and debris objects in orbit, is also allocated a range that safely exceeds their count of objects. For that matter, a static variable can be used in the Constellation class, that is initialized out of line with the value 10,000,000 and added 1,000,000 at the end of every `Constellation` constructor execution.

5.2.2. Launch Schedule Computation

Before the particles of `Constellation` objects are inserted to the simulation, the launch schedule of each object needs to be calculated. First, the constellation's YAML-parameter, that describes the time of launch as a date string, must be converted into the simulation's unit of time: the iteration. This involves a series of conversions between five descriptions of time depicted in Table 5.1.

date string	struct tm	standard reference	simulation reference	iteration
2022/07/22	{0,0,0,22,6,2022}	1658448000s	17452800s	1745280

Table 5.1.: Example of a conversion from calendar time to iteration time

In order to get a time in units of a second, the date string is firstly parsed and assigned to a C++ `struct tm`[21], before `std::mktime()` is called with the created `struct tm` as the parameter in order to return a time in seconds of the type `time_t`. As the time reference of the product is not related to the reference time of the simulation at iteration zero, another conversion must be performed. A subtraction with a `time_t` value describing the reference time of the simulation returns the time difference between the constellation launch and the simulation start. In the last step of the conversion series, the iteration of launch is obtained through division by `deltaT`. The `duration` YAML-parameter value is interpreted as a number of days and the conversion only requires a conversion to seconds and a division by `deltaT`.
If the `startTime` string of the configuration file is not formatted as a date, but as a plain integer and if the duration string is not concluded by the 'd' character, they can just be interpreted as iteration values from the start.

After the two temporal attributes `startTime` and `duration` have been set, the launch schedule, which is used by the `tick()` method, can be calculated and stored in a two-dimensional vector `schedule`. At each slot with the indices i, j the launch time of plane j in shell i is assigned in the unit of iterations. Inserting the constellation particles plane-wise and linearly at the same time requires a certain strategy considering the different amount of

planes per shell and particles per plane. To achieve linearity on a higher level, the shells are allocated time spans in proportion to their total size, as shown in Figure 5.2. On a lower level, that is for each shell, the planes are added in identical time steps, as the size of each plane is the same.

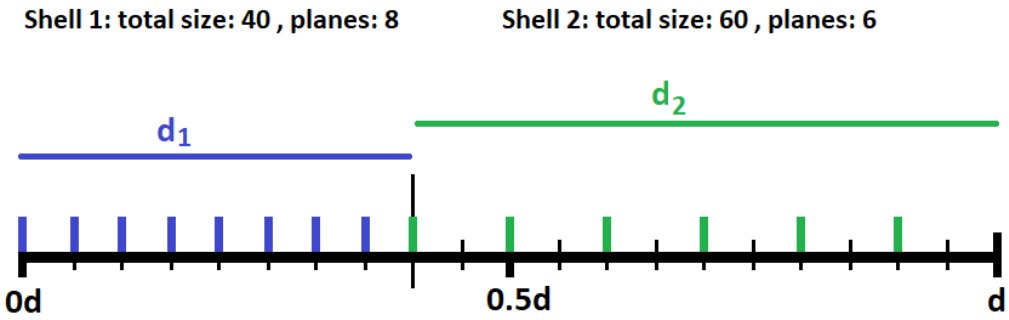


Figure 5.2.: Example of an insertion scenario and the associated schedule. A plane's time of insertion is indicated by a colored bar. d is a symbol for the duration parameter. d_1 and d_2 are symbols for the partitions allocated to each shell.

5.2.3. Particle Insertion Algorithm

During the main simulation loop, the `Simulation` member method `updateConstellations()` calls the method `tick(simulationTime)` on every constellation object in fixed intervals. The method employs different behavior depending on its state stored in the member variable `status`. Possible states are inactive, meaning the simulation time has not reached the constellation's `startTime` yet, active, meaning the simulation time is larger or equals `startTime`, and deployed, which signals that the constellation is fully inserted. In the "inactive" state, the `status` variable is set to "active" when `simulationTime` passes `startTime`. Additionally, the object starts keeping track of the time internally using a counter variable `timeActive` that begins its count at `startTime`.
Once the constellation is active, the algorithm demonstrated in Figure 5.3 is employed that determines the particles to be inserted into the simulation. The variables `currentShell` and `planesDeployed` keep track of the insertion progress by pointing to the timestamp of the next plane in the `schedule` variable. Once `timeActive` passes this timestamp, particles of an amount according to the plane size are included in the set of particles returned and the indices are updated to point to the next timestamp. The particles are inserted into the AutoPas particle container in the same order as they have been generated. After all particles have been inserted, the state of the constellation is set to "deployed". In all following calls of the `tick()` method, it performs a no-operation while returning an empty vector.
Since it is possible that new particles are inserted close to other particles, conjunctions could instantly be triggered. This can be prevented by inserting such particles with a delay. The particle container `AutoPas` provides a region iterator that iterates over all particles in a defined region. It can be used to detect whether other particles are within a critical range of the particle that should be inserted. If that is the case, the particle is placed in a vector and the insertion is attempted in the next iteration.
As the generated particles are inserted statically over a duration that is unrelated to the

```
1  particles ← {}
2  while timeActive > schedule[currentShell][planesDeployed] do
3      for i ← 0 to planeSizes[currentShell] do
4          particles.push_back(pop_front(satellites))
5      planesDeployed++
6      if planesDeployed == nPlanes[currentShell] then
7          currentShellIndex++
8          planesDeployed ← 0
9          if currentShell == nShells then
10             status ← deployed
11             break
12 timeActive ← timeActive + interval
13 return particles
```

Figure 5.3.: Code executed when tick() is called and the status is set to active.

phase of neighboring planes, the constellation only reliably retains its relative phasing property when inserted within one iteration. In order to be able to model constellations with a minimized chance of intra-constellation collisions[22], the implementation would need to retain the relative phasing. This is one reason why intra-constellation collisions are not considered in the next chapter.

6. Simulation Results

Now that constellations can be generated and included in the simulation, it is possible to start simulation runs that support assessing the collision risk of upcoming megaconstellations. By utilizing AutoPas and by running the simulation on a computing cluster[23] node with 28 cores, it is possible to simulate years with time steps of 10 seconds in a matter of a few days. In this chapter, the results of simulation runs with constellations and input data consisting of tracked objects that are already in space[24] are presented. At first, the methodology for running the simulation in order to obtain the results is described. Throughout this chapter, the term "tracked objects" only refers to the non-constellation input data.

6.1. Methodology

The simulation output consists of periodic snapshots of the state vectors associated with object IDs and a log of all close encounters registered by the pairwise collision detection. Close encounters, which are also referred to as conjunctions, are defined by a conjunction threshold determining the maximum distance for registered conjunctions. In the conducted simulations, conjunctions of up to 250 meters were tracked. Furthermore, the only conjunctions taken into account are those where at least one of the particles is a tracked non-constellation object. Non-constellation objects are inserted at simulation start and no further tracked objects are added afterwards. All objects were propagated by the same component, meaning they were subjected to the same Keplerian motion and perturbations. These perturbations include the J2-, the C22-, and the S22-component, the solar radiation pressure, and atmospheric drag. Only objects in the LEO region from 200-2000km altitudes above the Earth's surface are included in the simulation. Objects with a 10,000km distance from the Earth's center are deleted. Factors that are not yet accounted for are satellite station-keeping[25], evasive maneuvers performed by active satellites, and the generation of space debris following a collision. Determining whether conjunctions result in collisions are not of primary importance in the analysis for these reasons and, because breakups that generate further debris are not yet implemented.
In the first of two simulation runs, 11.5 years were simulated with estimated times of constellation insertions according to Figure 6.1. Most estimates regarding insertion duration are based on the FCC's general rule that companies can only launch their constellation within nine years after the approval has been given[26]. Approximate, or accurate starting times for launch were available for Starlink Generation 1, OneWeb's first constellation, Kuiper and Telesat, while times of launch for Astra, GuoWang and OneWeb 2 are not based on concrete supporting information. For OneWeb's first constellation, the choice of the insertion speed is based on the current speed of insertion. As described in the previous chapter, satellites are assumed to launch linearly over time. Shells are inserted in the same order as listed in Chapter 3.

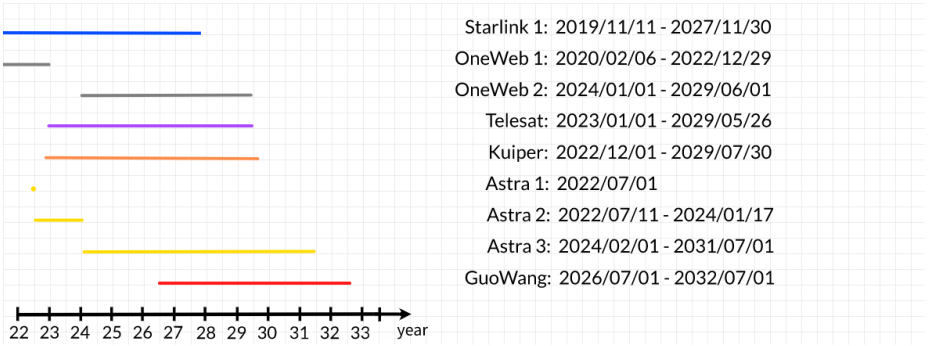


Figure 6.1.: This figure shows the insertion times of each constellation included in the first simulation from 2022/01/01 until 2033/07/01. The amount of satellites at each point in time is shown in Figure 6.2

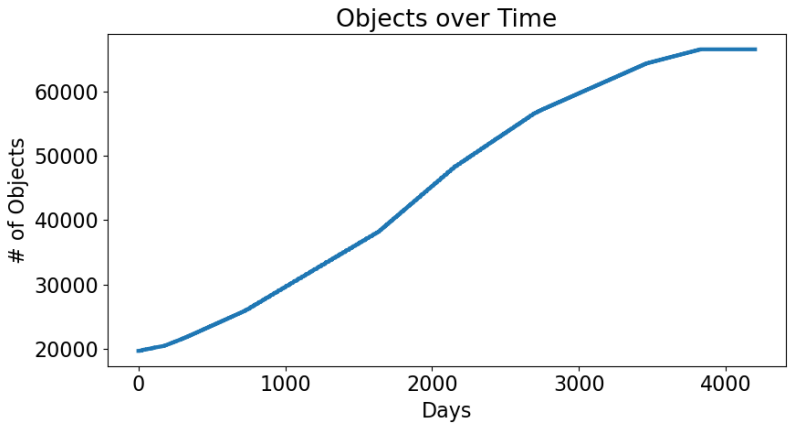


Figure 6.2.: Graph describing the amount of objects in the simulation over the simulated time. The number of non-constellation objects equals 15,975 and at the simulation start the object size equals 19,693 due to already inserted Starlink and OneWeb satellites.

In the second run, which is a one year run, all constellations including Starlink Generation 2 are fully inserted at Iteration 0. Here, the full constellations can be compared directly and a shorter time span that is more tangible can be looked at.
Before the results are analyzed, information about the tracked input data and the constellations is presented in order to be able to better assess the general collision risk of constellation satellites according to their operating altitude. Figure 6.3 and Figure 6.4 show the altitude distributions of the non-constellation objects and the constellation satellites respectively.

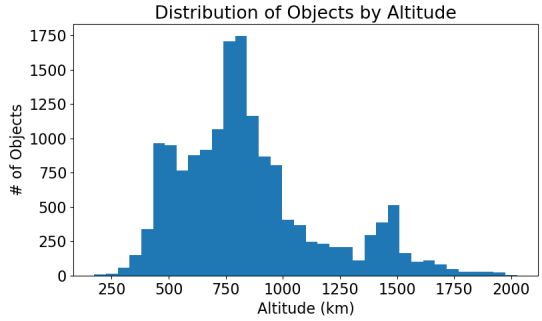


Figure 6.3.: Histogram of non-constellation objects based on their altitude.

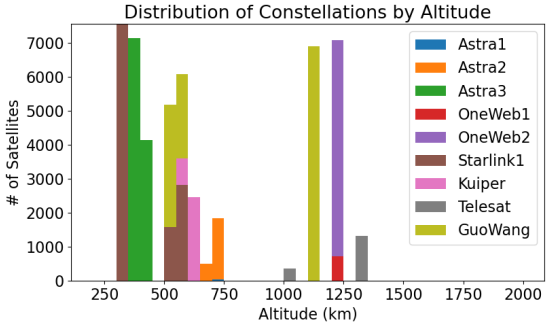


Figure 6.4.: Histogram of constellation satellites based on their altitude. The contribution of each constellation is marked by a labeled color.

6.2. Holistic Conjunction Analysis

In the simulated scenario of eleven and a half years, the cumulative amount of conjunctions depending on the conjunction threshold are shown in Figure 6.5. Samples of values are to be found in Table A.2 in the appendix. Over the simulated time, 317,668 encounters within 250m and 50,812 encounters within 100m have been tracked. When the first thousand days are compared to the last thousand days, the amount of 250m-conjunctions as well as the 100m-conjunctions double[27]. This shows that conjunctions of this class occur more than daily, even in the early stage of the constellation insertion.

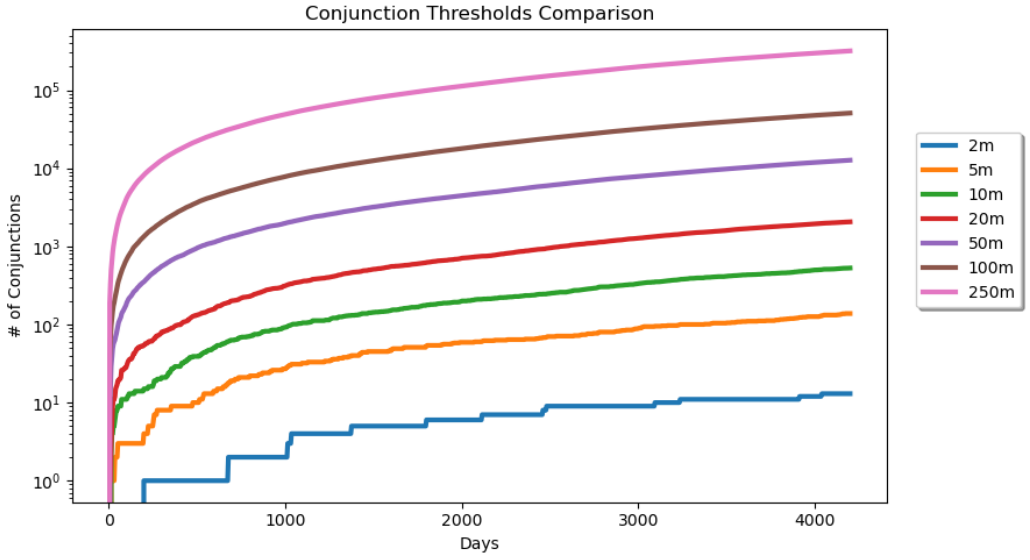
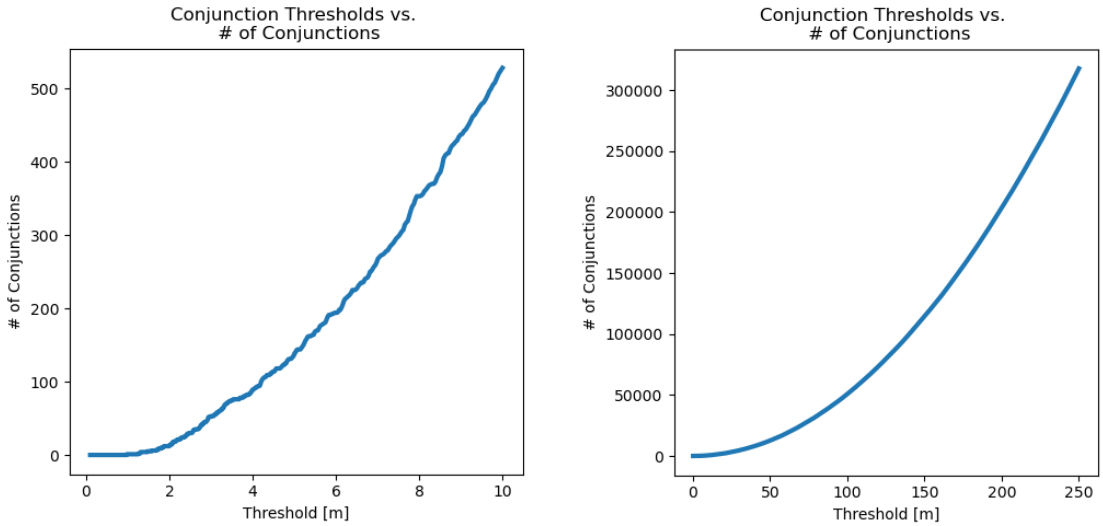


Figure 6.5.: Graphs showing the amount of conjunctions over time depending on the set threshold. The axis counting the counjunctions is logarithmic in order to depict various orders in magnitude.

The graph also shows the drastic decrease in the chance of a conjunction occurring when the conjunction threshold is decreased. For instance, 10m-conjunctions, which count 528 at the end of the simulation, are nearly 100 times less likely than a 100m-conjunction. 2m-conjunctions, which are comparatively rare, have a very high chance of collision if no avoidance maneuvers are performed. The cuboid satellites of Starlink and OneWeb, for example, have a maximum side length of 3.2m and 1.0m respectively as well as attached solar arrays [APSODG21]. 13 of these and 138 5m-conjunctions occurred throughout the simulation. Figure 6.6a shows the close encounters in more detail by mapping all conjunction threshold values between 0 and 10m to a count of total conjunctions. The shown relationship has the characteristic of a polynomial function, which is also indicated by Figure 6.6b. This characteristic is expected, as the number of satellites within a sphere, which grows in a cubic manner for an increasing conjunction threshold, is counted to determine the conjunctions.



(a) Short range

(b) Long range

Figure 6.6.: Continuous graphs describing the relation of the amount of conjunctions and the threshold for the low range [0m,10m] and the wide range [0m,250m]. Graph samples are listed in Table A.4

Since the figures previously shown additionally consider collisions between two tracked objects, it is relevant to know the ratio of these conjunctions. Over all thresholds roughly 38.5% out of all conjunctions did not involve constellations (see Table A.3). This means that conjunctions involving constellations amount to 62.5% of all conjunctions and they occurred about 1.62 times more often.

6.3. Conjunction Analysis by Constellations

In this section, results are presented w.r.t. individual constellations. At first, results from the 11.5-year scenario are analyzed, before each constellations' relation to the occurred conjunctions are analyzed in a scenario that allows comparison of fully deployed constellations.

The total amount of close 20m-conjunctions involving one constellation satellite and one tracked object are shown in Figure 6.7. Additionally, the contribution of each constellation is highlighted by color. Out of the total 1291 conjunctions, Starlink's constellation, which was also deployed long before the other large contributors, has the highest share of conjunctions equaling 424. The GuoWang constellation, which has the second highest conjunction count, conributed 273 conjunctions, even though it was inserted late in the scenario. Despite the fact that the Astra 3 constellation is the single largest constellation with a comparable size to Starlink and GuoWang, its number of conjunctions equals only 148.

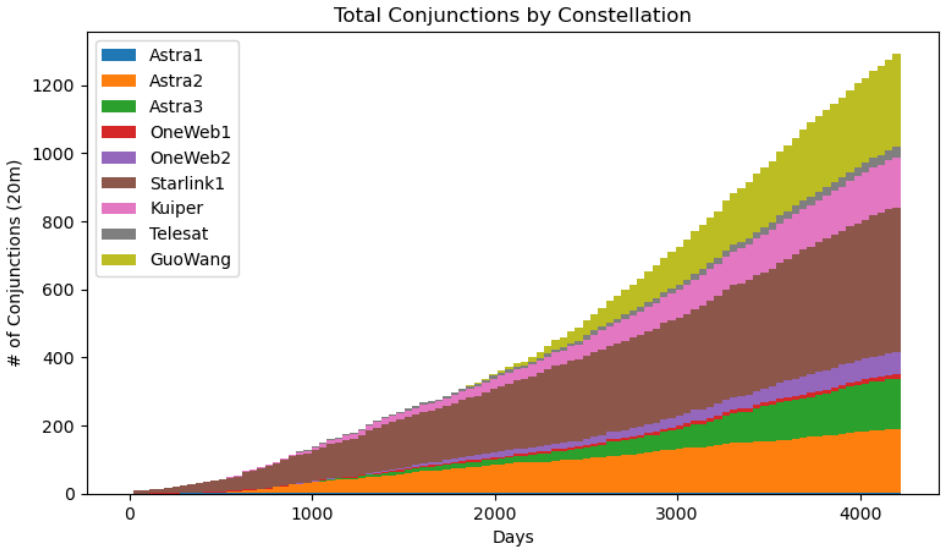


Figure 6.7.: This figure shows the accumulated amount of 20m-conjunctions between constellation satellites and non-constellation objects. The contribution of each constellation is indicated by a labeled color. Value samples are listed in Table A.5

Besides the similar number of satellites, the constellations GuoWang and Starlink are comparable in their altitude distribution, as they respectively have 6080 and 4408 satellites deployed in the altitude region between 500-600km, which has a high density in tracked objects. Furthermore, the majority of each constellation's satellites are in altitude regions with low object densities. This indicates that the different amount of conjunctions is caused by the fact that Starlink has been inserted earlier. The low amount of conjunctions in the Astra 3 constellation is explained by the fact that the whole constellation is deployed within altitudes of 380-400km with a low density of objects.
OneWeb 1, OneWeb 2 and the Telesat constellation are similarly deployed in altitudes with comparatively low object densities and score a low amount of conjunctions compared to its size, with 12, 67 and 30 conjunctions respectively. Astra's small first constellation consisting of 40 satellites in equatorial orbits caused one conjunction in total.
Lastly, the Kuiper constellation and Astra 2, which inhabit altitude regions with comparatively high object densities, contributed 147 and 189 conjunctions respectively.

1-Year Simulation Results for Constellation Comparison

In the second simulation of one year, the constellations are fully inserted at Iteration 0, which enables comparison that is independent from projected times of deployment. The constellations are compared based on the their accumulated count of 100m-conjunctions at the end of the simulation. Picking the conjunction threshold of 100m aims to make the comparison more statistically reliable, as more conjunctions are tracked.
Out of all conjunctions involving constellations, the share of each constellation is depicted in Figure 6.8. The GuoWang constellation is involved in 25.7% of the conjunctions and has the highest share of all the constellations, even though the Starlink 2 constellation has more than twice the number of satellites. Starlink's second constellation has only the second highest share, which remarkably shows how much influence the choice of the satellites' operating altitudes has on collision risk, or the amount of evasive maneuvers that would have to be performed to prevent a collision. Aside from 4468 Starlink 2 satellites in higher density altitudes, compared to the already mentioned 6080 GuoWang satellites in similar altitudes, the spare 25,532 satellites operate in low density altitudes (see Table 3.3).

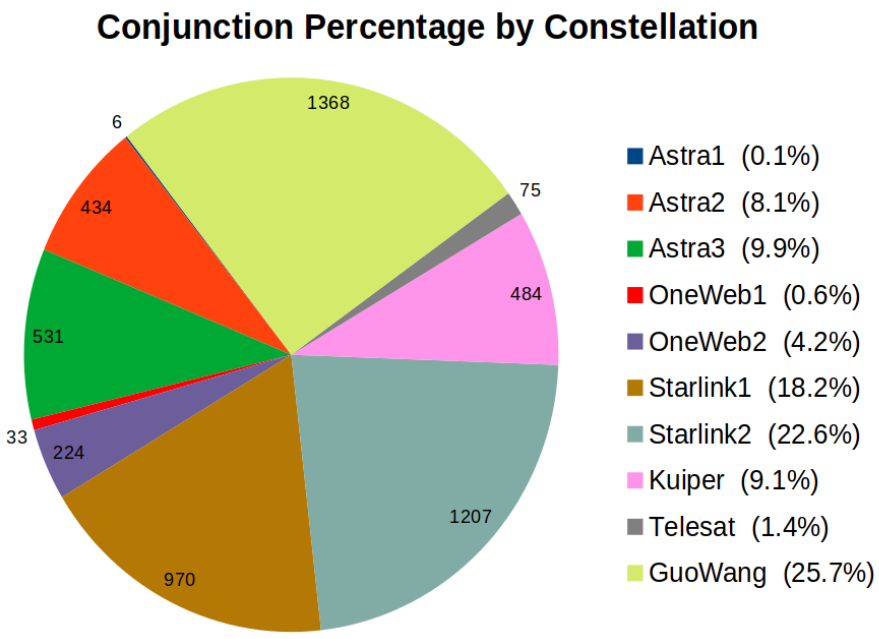


Figure 6.8.: This pie chart shows the contribution of conjunctions for every constellation.

This is also shown by the 7088 OneWeb satellites that are involved in about 4.8% of conjunctions, while the 2336 Astra satellites deployed in 690-700km altitudes claim a higher percentage of 8.2%. The Telesat, the Astra 3, and the OneWeb constellations behave highly similar. All other constellations contain shells in higher density altitudes and have an accordingly higher amount of conjunctions.
While making up 80,577 simulation objects, the constellation satellites are involved in 75.9% out of all recorded 100m-conjunctions, as opposed to the 15,975 tracked objects making up 24.1%. This means that on average 0.0662 conjunctions occurred per constellation

satellite, while 0.106 of conjunctions occur for every tracked object. In other words, a tracked non-constellation object has higher chances of close encounters with other tracked objects than a constellation satellite.

7. Conclusion

As a result of this work, constellations can be generated as well as inserted into the space debris environment simulation LADDS. Resulting data from simulation runs that include upcoming megaconstellations show, how the insertion of constellations in lower Earth orbit affects the frequency of conjunctions and how the choice of the orbital shell's altitude significantly influences the overall risk of collision. The quality of the modeled constellations can be increased by conserving relative phasing within orbital shells when they are added over time. Furthermore, the addition of individual constellation satellite properties such as mass and size can be implemented and set according to the constellation.
In order to be able to better estimate the future space debris population in lower Earth orbit, breakup events triggered by collisions will be integrated into the simulation.

Part III.

Appendix

A. Graph Samples

Time in days	0	1000	2000	3000	3199	4000	4199
	19693	29580	45307	59781	61750	66550	66550

Table A.1.: Samples of graph in Figure 6.2. The Values are object counts.

Threshold/Days	1000	2000	3000	4000	4199
2m	2	6	9	12	13
5m	27	59	89	126	138
10m	92	200	334	498	528
20m	314	707	1275	1950	2059
50m	1983	4471	7853	11876	12673
100m	7943	17916	31692	47577	50812
250m	49179	112880	199081	297641	317668

Table A.2.: Samples of graphs in Figure 6.5. Values describe counts of conjunction

Threshold	2m	5m	10m	20m	50m	100m	250m
	0.4615	0.3985	0.3977	0.3729	0.3840	0.3840	0.3856

Table A.3.: Ratio of conjunctions not involving constellations in all conjunctions. Values describe counts of conjunctions

Threshold	2m	4m	6m	8m	10m
	13	89	194	353	528

Table A.4.: Samples of graph in Figure 6.6a. Values describe counts of conjunctions

Constellation/Days	1001	2003	3004	4005	4199
Astra1	1	1	1	1	1
Astra2	33	83	129	181	189
Astra3	0	17	58	139	148
OneWeb1	2	5	9	12	12
OneWeb2	1	15	29	61	67
Starlink1	89	190	289	406	424
Kuiper	9	31	81	138	147
Telesat	2	9	15	28	30
GuoWang	0	6	111	246	273

Table A.5.: Samples of graph in Figure 6.7. Values describe counts of conjunctions.

B. Footnotes

Notes

[1] Accessed 2022/02/08: https://en.wikipedia.org/wiki/List_of_Starlink_launches

[2] Accessed 2022/02/08: https://www.statista.com/statistics/897719/number-of-active-satellites-by-year/

[3] Accessed 2022/02/08: https://www.esa.int/Safety_Security/Space_Debris/Mitigating_space_debris_generation

[4] Accessed 2022/02/08: https://www.sdo.esoc.esa.int/environment_report/Space_Environment_Report_latest.pdf

[5] Accessed 2022/02/08: https://www.esa.int/gsp/ACT/projects/debris_hpc/

[6] https://celestrak.com/

[7] Accessed 2022/02/06: http://www.parabolicarc.com/2021/11/05/astra-space-applies-to-launch-more-than-13000-satellites-proposed-broadband-constellations-exceed-79000-spacecraft/

[8] Accessed 2022/02/06: https://spacenews.com/china-is-developing-plans-for-a-13000-satellite-communications-megaconstellation/

[9] https://celestrak.com/

[10] Accessed 2022/02/13: https://spacenews.com/spacex-submits-paperwork-for-30000-more-starlink-satellites/

[11] Accessed 2022/02/06: https://www.reuters.com/technology/canadas-telesat-takes-musk-bezos-space-race-provide-fast-broadband-2021-04-11/

[12] Accessed 2022/02/06: http://www.parabolicarc.com/2021/11/05/astra-space-applies-to-launch-more-than-13000-satellites-proposed-broadband-constellations-exceed-79000-spacecraft/

[13] Accessed 2022-02-06: https://spacenews.com/astra-says-focus-is-on-launch-as-it-files-application-for-satellite-constellation/

[14] Accessed 2022/02/06: http://www.parabolicarc.com/2021/11/05/astra-space-applies-to-launch-more-than-13000-satellites-proposed-broadband-constellations-exceed-79000-spacecraft/

[15] Accessed 2022/02/08: https://www.esa.int/gsp/ACT/projects/debris_hpc/

[16] https://celestrak.com/

[17] Here, auto-tuning refers to the automatic configuration of optimal algorithms and other strategies based on the simulation state [GSBN21]

[18] Already launched constellation satellites are excluded from the set to avoid redundancy

[19] Accessed 2022/02/12: https://esa.github.io/pykep/

[20] Shells contain planes with identical orbital elements

[21] struct tm is defined to code the month as a number from 0 to 11

[22] Collisions between satellites of the same constellation

[23] The simulation was run on CoolMUC-2: https://doku.lrz.de/display/PUBLIC/CoolMUC-2, Accessed 2022/02/12

[24] Constellation satellites that have been launched at this time are excluded from the input set as well as docked satellites

[25] The term refers to satellites that counteract perturbations to remain in a stable orbit

[26] Accessed 2022/02/07: https://spacenews.com/oneweb-asks-fcc-to-authorize-1200-more-satellites/

[27] The factors describing the increase equal 2.03 and 2.05 respectively

List of Tables

List of Figures

Bibliography

[APSODG21] C Alvaro Arroyo-Parejo, Noelia Sánchez-Ortiz, and Raúl Domınguez-González. Effect of mega-constellations on collision risk in space. In *8th European Conference on Space Debris*, volume 8, 2021.

[BHGE21] CG Bassa, OR Hainaut, and D Galadí-Enríquez. Analytical simulations of the effect of satellite constellations on optical and near-infrared observations. *arXiv preprint arXiv:2108.12335*, 2021.

[Bö21] Oliver Bösing. Efficient trajectory modelling for space debris evolution. Bachelor's thesis, Technical University of Munich, 9 2021.

[GSBN21] Fabio Alexander Gratl, Steffen Seckler, Hans-Joachim Bungartz, and Philipp Neumann. N ways to simulate short-range particle systems: Automated algorithm selection with the node-level library autopas. *Computer Physics Communications*, 273:108262, 2021.

[GXG+20] Meiqian Guan, Tianhe Xu, Fan Gao, Wenfeng Nie, and Honglei Yang. Optimal walker constellation design of leo-based global navigation and augmentation system. *Remote Sensing*, 12(11):1845, 2020.

[KJLM10] Donald J Kessler, Nicholas L Johnson, JC Liou, and Mark Matney. The kessler syndrome: implications to future space operations. *Advances in the Astronautical Sciences*, 137(8):2010, 2010.

[LCY21] Jintao Liang, Aizaz U Chaudhry, and Halim Yanikomeroglu. Phasing parameter analysis for satellite collision avoidance in starlink and kuiper constellations. In *2021 IEEE 4th 5G World Forum (5GWF)*, pages 493–498. IEEE, 2021.

[LHKO04] J.-C Liou, D.T Hall, P.H Krisko, and J.N Opiela. Legend – a three-dimensional leo-to-geo debris evolutionary model. *Advances in Space Research*, 34(5):981–986, 2004. Space Debris.

[PdPCC21] Nils Pachler, Inigo del Portillo, Edward F. Crawley, and Bruce G. Cameron. An updated comparison of four low earth orbit satellite constellation systems to provide global broadband. In *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, pages 1–7, 2021.

[Vir16] B Bastida Virgili. Delta debris environment long-term analysis. In *Proceedings of the 6th International Conference on Astrodynamics Tools and Techniques (ICATT)*, 2016.

[Wal18] Ulrich Walter. *Astronautics: The Physics of Space Flight, Third Edition.* Springer, 2018.