12-2021

Processing and Visualizing Satellite Data

Caleb Collier
Taylor University, caleb_collier@taylor.edu

Processing and Visualizing Satellite Data

Caleb Collier
Advisor: Dr. Stefan Brandle

December 10, 2021

Abstract

Satellites are a useful way of gathering data at high altitudes. To be able to properly view the data, however, there are many important steps that one must take to ensure the data received is readable and usable. The data must be transmitted from the satellite to the ground, then must be decommuted and can then be used in various ways. This paper is an exploration of various ways of processing and visualizing data received from satellites, as well as various ways of using the data.

1 Introduction

There are many aspects of satellites that are important to the discovery of new things. Among the software related practices in the satellite field, the processing and visualization of data received from a satellite is an important part of discovery. The first satellite launched was Sputnik 1 in 1957 [14]. It orbited for around 3 months before hitting the ground. Today, satellites can stay in orbit for much longer and are much more robust. While communication satellites today can both send and receive data, the main focus of this paper is how satellites send data to ground stations on Earth, how to interpret the received data, and how to visualize the data.

2 Personal Background

Over the summer of 2021, I worked for a company called NearSpace Launch (NSL). NSL creates satellites that send clients specific data. Prior to working at NSL, I had zero experience in working with satellite data. As I worked over the summer, I learned more about processing satellite data. Some of the ideas I will explain will be in the context of NSL, but these ideas are not only used by NSL.

3 Globalstar

NearSpace Launch uses a service from Globalstar. Globalstar is a popular communications company that uses a constellation of low Earth orbit satellites for data communication [12]. A satellite constellation consists of multiple satellites (24 in Globalstar's case)[2] orbiting around the Earth that can provide coverage all around the Earth [13].

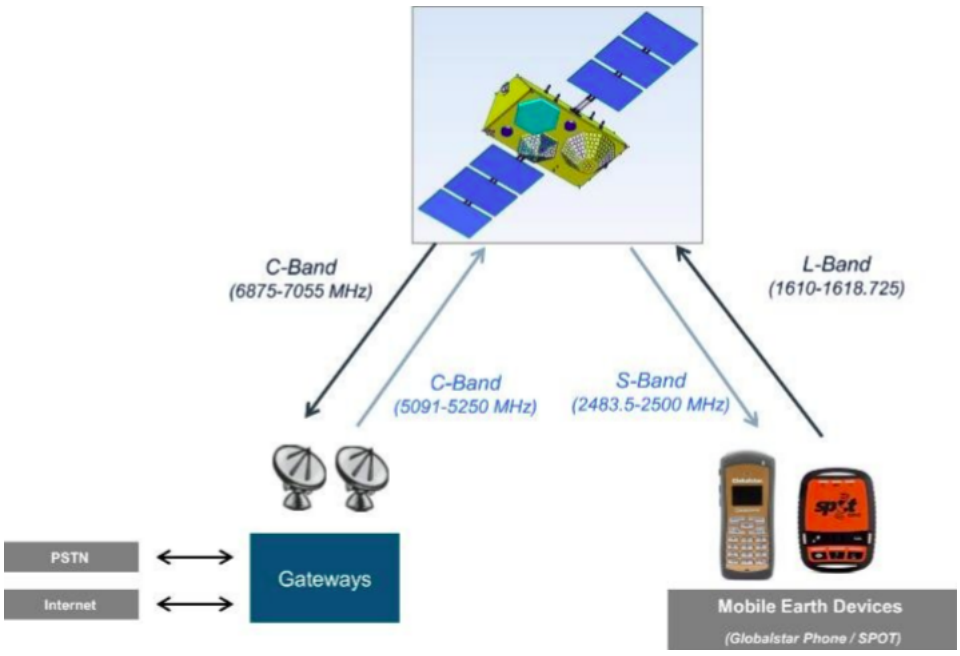
3.1 Simplex Radios

One way that Globalstar sends data back and forth is through simplex communication. Simplex communication is a communication channel that only sends data in one direction [10].

3.2 Information Path

The satellite that is sent into orbit will have various sensors on it to collect data. This data, once collected by the sensors, is sent to the Globalstar satellites via simplex radio. This information is then sent to Globalstar's ground station, and is then sent to the client via the internet. A more detailed diagram is shown in Figure 1.

Figure 1: Globalstar's data communication path



4 Data Format and Storage of Data

When NSL receives data from Globalstar, it is received through the internet as an XML document. With this string, NSL receives the timestamp that it was transmitted from Globalstar to them, and the time when NSL receives the data. In this XML string, there is some information such as the simplex radio ID, and the actual data that came from the satellite. This XML document is stored in a database and then needs to be parsed. The string of actual data that is received is in a hexadecimal format. The length of the string is determined by NSL or NSL's client.

5 Decommutation

The data received in its raw form is not very useful, so it needs to be parsed and then NSL can receive the values in a more useful form. Each string of data begins with 8 bits, or 2 hexadecimal characters. These 8 bits are known as the function code, and will determine how the string is to be parsed. Each string is divided into a specific number of bits specified by the client. Each division will represent one data value, like altitude, velocity, etc. Once the hex string is stored in the database, a various number of conversion functions are used to convert from hexadecimal to the specified format.

Figure 2: A partial parsing of a hexadecimal string received from Globalstar

FnCode	Seq	Lat	Lon	
F1	221	27.8508	76.9275	
Last Pkt	2021-10-01 05:59:57		F1DD1099B0DB2DDA33C5FFED91D5039A7C39	

5.1 Stitching Multiple Packets Together

Sometimes there will be more data wanted from a satellite than a single packet can hold. To account for this, I made a program that took multiple packets and combined them into one. Which packets are stitched together is up to the one launching the satellite. In NSL's case, there were two packets, between which contained lat, lon, and alt, as well as VX, VY, and VZ among other things. If the program stitches together a packet with a function code F1 and a packet with a function code F2, it must check two things before stitching them together. First, it must check if the sequence codes match. Each sequence code starts at 0 and climbs to 255. After the sequence code hits 255, it rolls back to 0 and climbs to 255 again. This is not enough to ensure that the proper data will be stitched together, so it will also check the time the packet was sent. If a packet with the function code F1 and a sequence number of 2 was sent at 10:00:00 UTC on one day, but a packet with the function code F2 and a sequence number of

2 was sent at 10:00:00 UTC on the next day, those two packets would not be stitched together as they do not correspond.

5.1.1 Stitching Packets Together Based on Time

For most cases, the algorithm described above connects two packets together very well. However, because of the way that the time is stored, there are some edge cases that will cause this algorithm to break. The time is stored as a GNSS time standard. GNSS time is stored in weeks and seconds. When the week changes from one week to another, the GNSS seconds roll back to 0. This presents a problem if one packet is sent right before the change of a week and the other packet is sent right after the change of the week. Before stitching together packets, the algorithm must also check if one of the packets was sent at the change of the week.

6 Satellite Location

One important piece of data that should be known for a given satellite is the location in its orbit. Latitude, longitude, and altitude can give some idea of where a satellite is at a given time, but it is not as precise as may be desired. For more precise locations, there are two formats, either an ephemeris or a TLE.

6.1 Ephemeris

Ephemerides have been used since the first millennium BC, and can be used for naturally occurring or artificial satellites. An ephemeris can be used to calculate the path and position of a given object at any time. Not only can ephemerides be used to view positions in the past and present, they can be used to predict a future orbit. When calculating an orbit with an ephemeris, however, one needs to be aware of how far ahead in the future the prediction is. It is best to only predict an orbit about 2 or 3 days in advance from the present. Any time out of this window may be considerably less accurate. The layout of an ephemeris is a table of various properties of whatever object is being tracked. There is not one standard of ephemerides, so what properties are tracked may be different. The most common properties tracked by an ephemeris are the X, Y, and Z location of the object, as well as the velocities in the X, Y, and Z direction. Ephemerides are large but are very precise, and much more precise than a TLE (which is discussed in the subsection below).

6.2 TLE

A two-line element set (TLE) is another way of representing the position of a satellite in an orbit. A TLE is much more concise than an ephemeris, but not as precise. TLEs can also be used to predict an orbital path of a satellite in the future, or can be used to determine what the orbital path was in the past. A TLE consists of two lines of information in a standard format. TLEs are used

by NORAD to keep track of artificial satellites currently in orbit around the Earth. While TLEs are useful, they do need to be generated every couple of days to ensure that they are still up to date with the current orbit.

6.3 Ephemeris and TLE Generation Webpage

During my summer at NSL, I made a webpage that would generate either an ephemeris or a TLE for any given satellite. While a decent portion of my work went into creating the webpage itself, a large portion of my work went into connecting various data paths together. When the user clicks the button to generate an ephemeris/TLE, the first thing that happens is the fetching of data for a given satellite. A query is run on a database to get the X, Y, and Z values, as well as the velocities VX, VY, and VZ. Once they are all fetched, these values get passed to another server that contains software known as FreeFlyer. Two of my colleagues created a program that would generate a TLE or ephemeris from those given parameters, as well as a time window. Once the ephemeris or TLE was created, the data would be passed back to the previous server and then was displayed on the webpage.

Figure 3: A TLE generated from the NSL webpage

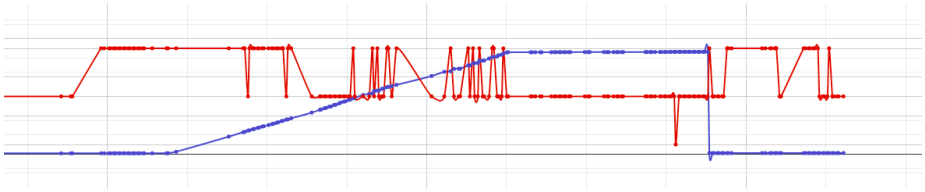
```
success True
run_type 0
solution_epoch middle
uuid 16944
manually_set_apriori 0
use_cartesian_fit 1
Start of fit span:          Jul 06 2021 02:37:47.990000000
End of fit span:            Jul 06 2021 20:53:48.990000000
Epoch:                      Jul 06 2021 11:45:48.490000000
Number of Observations:     3
Eccentricity:               0.0016620057707973826
Argument of perigee (deg):  211.46115593630913
RAAN (deg):                 316.2017159333257
Inclination (deg):          97.50965427001886
Mean Anomaly (deg):         244.5438086891483
Mean Motion (orbits/day):   15.13903836562295
Mean Motion Dot:             .00000000
Mean Motion DDot:            00000-0
Bstar:                      -0.0017809140114781739
Revolution number:          *** currently not being calculated ***

1 00001U 00000AAA 21187.49014456  .00000000  00000-0 -17809-2 0  9991
2 00001 097.5097 316.2017 0016620 211.4612 244.5438 15.13903837    68
```

7 Data Visualization

Receiving data from satellites is extremely useful, but with the sheer amount of data that one may get from them requires some sort of data visualization. There are many ways to visualize data recieved from a satellite. One needs to decide how exactly they are planning to do so (for example, whether the visualization will be in 2D or 3D), and one also needs to decide what data to visualize. A large amount of data may come back, but only some of it is related to each other. One example of this may be taking the altitude of a balloon and relating it to the external temperature as shown in Figure 4.

Figure 4: A 2D visualization of the altitude (blue) vs external temperature (red) over time.



7.1 Choosing Software

There are many types of software and libraries that can visualize data. For 2D visualization, one can use proprietary software like Tableau, or on greater scales one may use something like WebFOCUS. There are some libraries local to Python that are powerful in data visualization. One such library is pandas. Pandas is used for both data manipulation and data visualization. I had decided that for my data visualization, I wanted to visualize data in 3D instead of 2D. In my research, I had come across three JavaScript libraries to choose from that offer 3D visualization. Those three were Three.js, CesiumJS, and D3.js.

7.1.1 Deciding What Software to Use

Originally, I had not heard of CesiumJS before. My plan was to use either Three.js or D3.js to visualize data. D3 seemed more promising to me than Three.js, but I knew that it was going to be hard to visualize this type of data in 3D becuase while D3 is powerful, it was not created for this sort of data directly. I went to talk to Dr. Brandle, my research advisor, and he had mentioned CesiumJS. After looking at what Cesium had to offer, I decided that I was going to use that to visualize my data.

8 CesiumJS

Cesium was created by the company AGI. It is an API that is well known in the space data visualization community. Cesium has been open source from 2012, and will continue to be indefinitely [1]. One can create with Cesium and host their creation on the cloud using Cesium Ion, but I decided that I was going to host it locally, so I created a simple web server using Apache. After I had created my server, I received an API key for Cesium and got to work on figuring out how to visualize data.

8.1 First Attempt at Data Visualization

First, I had to find some data to work with. After talking with my research advisor, I decided to use satellite data from a recent launch from NearSpace Launch.

The satellite's name was TAGSAT-1. Using NSL's simplex console, I downloaded the altitude, latitude, and longitude of the satellite that was launched as a CSV file. Once I had the data, I needed to learn how Cesium works, and what I could use to visualize the data I had just received. I came across a tutorial that was used to visualize the flight path of an airplane. I decided that for a first test, I would use this tutorial to get used to Cesium and see if this type of visualization would work. I took the data from TAGSAT-1 and turned it into a JSON string for Cesium to use. After following the tutorial, I had created a very basic visualization of the data as shown in Figure 5.

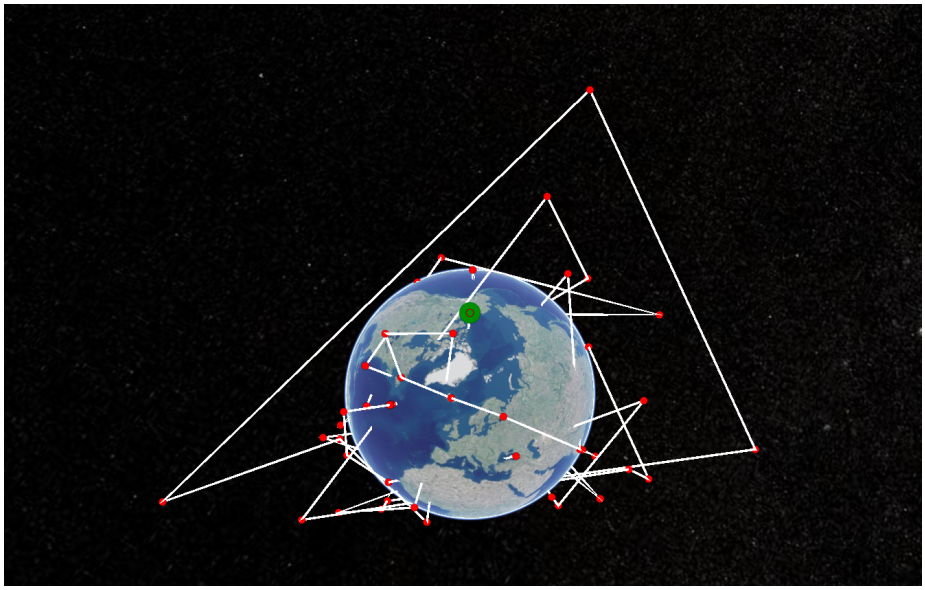


Figure 5: Visualization of TAGSAT-1 data.

Obviously, this visualization is not accurate of a proper orbit for many reasons. However, this visualization was useful for me and my development of data visualization. For one, this visualization showed me that, assuming there are no underlying problems in the way I visualized everything, the data I received from this satellite is faulty. After discussing the results with my advisor, he confirmed that indeed, only one single point of data was confirmed to be correct. Without this visualization, I would not have known that unless I explicitly asked him. Second, this visualization showed me that using the type of visualization used for flight path tracking would not be a good way to visualize data for multiple reasons. Creating the data as a JSON file was too much of a process, especially if I had even more data points than I had previously, which is always a possibility. Also, when direction is changed according to latitude and altitude, it abruptly does so, unlike an orbit which is a spherical (possibly elliptical) and smooth path around the Earth. Because of the blunders of this type of visualization, I knew that I needed to find another way to go about my visualization.

8.2 Switching Data Sets

By the time I had finished my first attempt at data visualization, the semester had ended and I began to work at NSL. During my time at NSL, a second TAGSAT satellite was launched. The data we recieved was much more reliable, and from it a TLE and Ephemeris format were able to be generated. Because I had more familiarity with the way this satellite was set up, and because the data was more reliable than TAGSAT-1, I decided to switch the data set from TAGSAT-1s data to TAGSAT-2s.

8.3 Further Attempts at Visualization

After switching data sets, I went back to Cesium's website to see if there was anything that would show a smoother and more consistent orbit around the Earth. The website has a showcase of different ways that Cesium can be used, along with the code. After perusing some code examples, I came across an example titled CZML. As shown below in Figure 6, it consists of multiple orbital paths, some of which are Low Earth orbits like the one that I have, and others that are much further from the Earth.

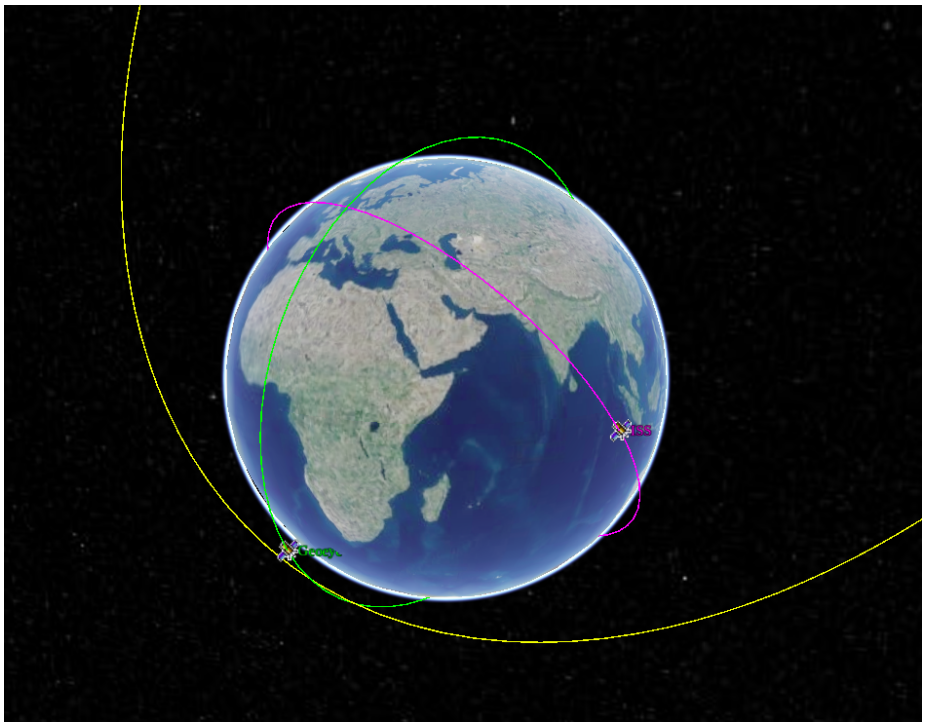


Figure 6: Smoothed orbits of multiple satellites.

This was a highly desirable way to visualize an orbit, so I began doing

research on how I would be able to accomplish this. I discovered that Cesium had made a string known as a `.czml`. The `.czml` file is a valid JSON string, but is much easier to comprehend and create when compared to the previous JSON string I had created in my previous attempt. The main idea of this string is that it describes certain attributes and positions of a given object (in this case a satellite) across time. I spent a lot of time trying to understand the exact format of this string. Cesium has put some documentation on their GitHub about the format of CZMLs, but it is lackluster and could use a bit more explanation. Searching for more explanation, I discovered a python package that will take the TLE of a satellite and generate a CZML file for it. After installing and testing it out, I received the results I was looking for as shown in Figure 7. I attempted to use the TLE calculated by the webpage that I had created at NSL over the summer, but that did not seem to work very well. Instead, I decided to go to a website known as Celestrak to get a TLE. I found the TLE for our satellite and it worked much better.

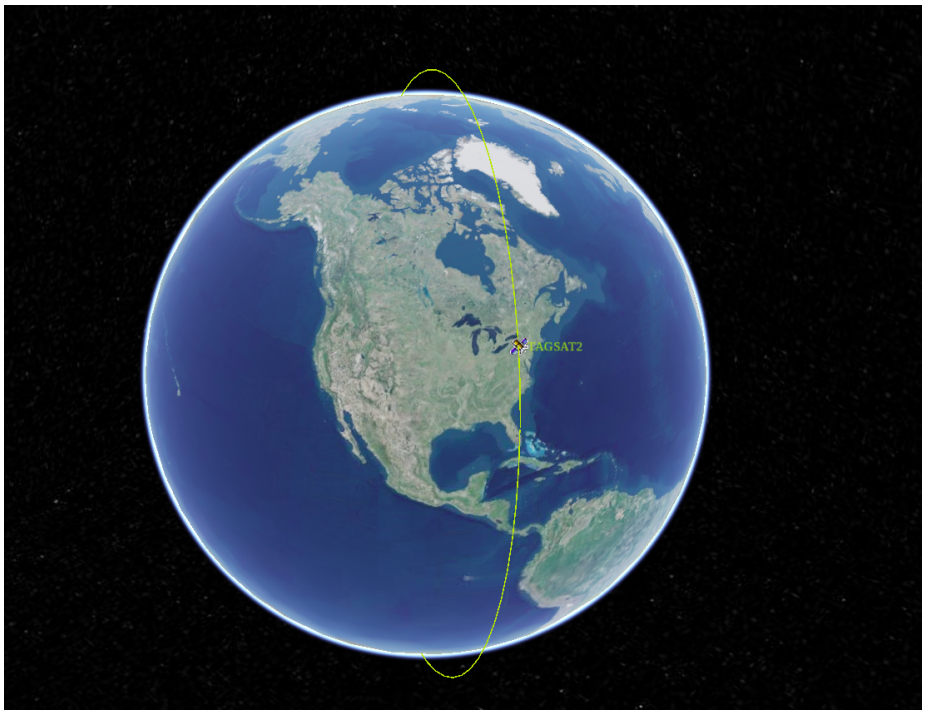


Figure 7: Orbit of TAGSAT-2 created with the python package satellite-czml

9 Conclusion and Future Work

Over the summer and in the process of writing this paper, I have learned a lot about the processing and visualization of satellite data. I have certainly learned that the processing of data is an important part of data visualization. Processing data into a workable format makes data visualization much easier than if you were to have raw data, and is an essential part of data visualization.

Discovering the python package that generates a `.czml` file based on a given TLE has opened the possibility for more research in this type of visualization. There are some papers that discuss delivering satellite data in real time. While TLEs need to be updated every couple of days, I believe that it would be useful to update an orbital path in real time given a TLE. The idea would be to use Celestrak to feed the TLE of a given satellite into a python script whenever the TLE is updated. Once it generates the `.czml` file, the script would replace the old file with the new file in the server. This would differ from the other papers I came across in my research as it would be updating the satellite *orbit* in real time as opposed to other kinds of data.

Bibliography

[1] AGI. *CesiumJS*. 2021. URL: https://cesium.com/platform/cesiumjs/ (visited on 10/11/2021).

[2] Globalstar. *Globalstar Constellations*. 2021. URL: https://www.globalstar.com/en-us/about/our-technology (visited on 09/26/2021).

[3] Dr. T. S. Kelso. *Celestrak*. 2021. URL: https://www.celestrak.com/NORAD/elements/stations.txt (visited on 10/27/2021).

[4] NearSpace Launch. "Globalstar Communication Link for CubeSats: TSAT, GEARRS1, and GEARRS2". In: 2014.

[5] NearSpace Launch. "TSAT Globalstar ELaNa-5 Extremely Low-Earth Orbit (ELEO) Satellite". In: 2014.

[6] NSL. *NearSpace Launch*. 2021. URL: https://data2.nsldata.com/console/ (visited on 10/27/2021).

[7] Brandon Rhodes. *sgp4*. 2019. URL: https://pypi.org/project/sgp4/ (visited on 10/27/2021).

[8] RhodesMill. *Skyfield*. 2019. URL: https://rhodesmill.org/skyfield/ (visited on 10/27/2021).

[9] a.i. solutions. *FreeFlyer*. 2021. URL: https://ai-solutions.com/freeflyer-astrodynamic-software/ (visited on 11/02/2021).

[10] "The Authoritative Dictionary of IEEE Standards Terms, Seventh Edition". In: *IEEE Std 100-2000* (2000), p. 1053. DOI: 10.1109/IEEESTD.2000.322230.

[11] Wikipedia contributors. *Ephemeris — Wikipedia, The Free Encyclopedia*. [Online; accessed 2-November-2021]. 2021. URL: https://en.wikipedia.org/wiki/Ephemeris.

[12] Wikipedia contributors. *Globalstar — Wikipedia, The Free Encyclopedia*. [Online; accessed 26-September-2021]. 2021. URL: https://en.wikipedia.org/wiki/Globalstar.

[13] Wikipedia contributors. *Satellite Constellation — Wikipedia, The Free Encyclopedia*. [Online; accessed 26-September-2021]. 2021. URL: https://en.wikipedia.org/wiki/Satellite_constellation.

[14] Wikipedia contributors. *Sputnik 1 — Wikipedia, The Free Encyclopedia*. [Online; accessed 26-September-2021]. 2021. URL: https://en.wikipedia.org/wiki/Sputnik_1.

[15] Wikipedia contributors. *Two-line element set — Wikipedia, The Free Encyclopedia*. [Online; accessed 27-October-2021]. 2021. URL: https://en.wikipedia.org/wiki/Two-line_element_set.

[16] Lan Zhao et al. "Delivering Real-Time Satellite Data to a Broader Audience". In: *Proceedings of the 5th Grid Computing Environments Workshop*. GCE '09. Portland, Oregon: Association for Computing Machinery, 2009. ISBN: 9781605588872. URL: https://doi.org/10.1145/1658260.1658264.