Universität Zürich[UZH]

Bachelor Thesis

BSc Physics

Development and Testing of a Control Software for a Radio Telescope

February 2025 - May 2025

Author:
Felix Meyer
felix.meyer@uzh.ch

Supervisors:
Dr. Achim Vollhardt
Prof. Dr. Christof Aegerter

University of Zurich

Contents

Chapter 1

Introduction

The Department of Physics of the University of Zurich is presently installing a new radio telescope. This antenna should serve educational purposes, giving students the ability to gain first-hand experience in data collection and analysing real astronomical signals. It is meant to be used by high school students and university students and will be able to observe satellites, both in orbit around Earth, as well as spacecraft in deep space and celestial targets like pulsars.

Therefore, an easy-to-use control software is needed that can track targets and orient the antenna automatically with the push of a button.

Different solutions for individual requirements already exist. Gpredict [1] is a popular open-source tool for tracking Earth satellites and antenna control (cf. Section 5.11); while SPICE [2] is an algorithm capable of tracking deep space missions (cf. Section 2.2.4). However, presently, no software covers all requirements at once.

This thesis aims to develop and test a custom control software for this new radio telescope, which can track deep space and celestial targets as well as Earth satellites and communicate directly with the motor controller of the radio telescope.

Note that in this thesis, the words *satellite*, *spacecraft*, and *target* will be used interchangeably. There is no universal nomenclature, but in general, a satellite refers to an uncrewed spacecraft in orbit around the Earth, while spacecraft that fly interplanetary missions are just called spacecraft or probes. We will not strictly follow this nomenclature, and it does not matter either. The radio telescope will also be referred to as an antenna, because it is just a big antenna/satellite dish.

Chapter 2

Requirements and tools

2.1 Requirements

The goal of this thesis is to develop control software for a radio telescope. The radio telescope should be able to track satellites in orbit around Earth and in deep space. This is the list of requirements agreed upon at the beginning of the project:

Minimal requirements

- Angle calculations for TLE, OMM, SPICE kernels or astronomical targets (RA/DEC) or RA/DEC table based on location UZH and current time
- Prediction of tracks for the next 24/48/xx h
- Control of antenna motors based on angle calculations
- Graphical user interface, Linux preferred
- Visualisation of the next track from the perspective of the antenna system
- Calculation of Doppler shift based on nominal transmission frequency (database)

Optional

- Online data queries from CelesTrak (TLE, OMM) and NASA (SPICE Kernels)
- Visualisation of the satellite subpoint and footprint on a world map
- Script interface for planning and autonomous execution of tracks

2.2 Tools

2.2.1 Programming Language

Python is one of the most popular programming languages. [3] It is cross-platform compatible and widely distributed in academia. Furthermore, there already exist many packages

for things like the user interface (UI), the communication with the motor controllers, and the two major algorithms that we will use, SGP4 and SPICE Toolkit.

2.2.2 SGP4, GP, TLE and OMM

"The US government has provided GP or *general perturbations* orbital data to the rest of the world since the 1970s. These data are produced by fitting observations from the US Space Surveillance Network (SSN) to produce Brouwer mean elements using the SGP4 or *Simplified General Perturbations 4 orbit propagator*." [4]

The standard format to save this data in was the *Two-Line Element Set* (TLE). The format was designed to be as compact as possible, since at that time, storage and bandwidth were extremely limited compared to today, since it was the time of punch cards and dial-up modems.
The decisions taken then have created two problems for us in the 21st century. The first is a Y2K bug, because the years are saved as a two-digit number. The other is that the catalogue number is limited to 5 digits. Currently, the SSN is tracking about 26,000 objects, which is still within the limits of a 5-digit number, but on March 27th 2020, the Space Fence on Kwajalein Atoll got activated, and it is expected to track far more than 100,000 objects. Furthermore is the number of active satellites is increasing faster than ever, because of mega-constellations like SpaceX's Starlink and Amazon's OneWeb. [4]

To solve these problems, a new standard was created called the *Orbit Mean-Elements Message* (OMM), which is part of the Orbit Data Messages (ODM) Recommended Standard CCSDS 502.0-B-3 developed by the Consultative Committee for Space Data Systems (CCSDS). The OMM standard supports different file formats:

- XML: CCSDS OMM XML format including all mandatory elements.
- KVN: CCSDS OMM KVN format including all mandatory elements.
- JSON: OMM keywords for all GP elements in JSON format.
- JSON-PRETTY: OMM keywords for all GP elements in JSON pretty-print format.
- CSV: OMM keywords for all GP elements in CSV format.

Data is provided on the website CelesTrak.org. [4–6] For this program, we will use the OMM standard in the CSV format. However, we will also provide an option to use data in the TLE format.

2.2.3 Skyfield

The implementation of the SGP4 algorithm, which we will use, is a package called Skyfield. [7] In addition to the SGP4 algorithm, Skyfield provides many useful tools, such as converting from one coordinate system to another, calculating the subpoint of a satellite and many more. It is therefore a good choice for this project.

2.2.4 SPICE Toolkit

The SGP4 algorithm is focused on satellites in Earth's orbit. However, we are also interested in spacecraft that left the orbit of our planet and are exploring the depths of our solar system. For that, we will need to use the SPICE toolkit.
SPICE (Spacecraft, Planet, Instrument, C-matrix, Events) is developed at the Jet Propulsion Laboratory by NASA's Navigation and Ancillary Information Facility (NAIF) to plan and analyse missions. [2, 8]. It has been used on many missions since then. [9]

SpicyPy

The implementation of SPICE that we will use for this project is a package called SpicyPy. SpicyPy is a Python wrapper for the SPICE toolkit. [10]

Horizons System (API)

To use SpicyPy, one has to provide it with the necessary data. With SPICE, one has to provide this data in several files called kernels. (cf. Section 5.6.1) This can be hard to automate. However, there is a different way to use SPICE.

The Solar System Dynamics Group of the Jet Propulsion Laboratory in California, USA, provides a web application with an API[1] called Horizons System. This web application is limited in what can be achieved compared to a custom SPICE script. However, it is sufficient for our purposes since we are only interested in the position and velocity data of a spacecraft for a given time window. [11]

2.2.5 UI

Lastly, for the user interface, we will use PySide6. [12] which is a popular and easy-to-use framework to create modern looking graphical interfaces.

[1] Application Programming Interface

Chapter 3

Celestial calculations and Coordinate Systems

Before we can start with the software, we first need to introduce some basic terminology.[1]

3.1 Azimuth and Elevation

The probably most important coordinate system for this project is azimuth (AZ) and elevation (EL). Note that some literature uses the word *altitude* for elevation. There is no convention on which one to use, and both are correct; however, we will use elevation only to avoid any confusion with the altitude above ground in km.

The AZ-EL coordinates are always used in the reference frame of the observer[2]. As shown in Figure 3.1, azimuth describes in which direction to look, while elevation describes how high in the sky the target is. Accordingly, North is at 0° azimuth and East is at 90°. While 0° elevation means that the target is at the local horizon and 90° elevation is directly above the observer. This point is called *Zenith*.

3.2 Right Ascension and Declination

In an observer-based (rotating) system like azimuth-elevation, the apparent positions of celestial objects change constantly due to the Earth's rotation. This fact makes it very impractical to track or catalogue objects over longer periods. To solve this problem, we have to use a non-rotating system.

Let us first introduce the celestial sphere. This is an imaginary sphere around Earth,

[1] If not stated otherwise, my source for this chapter is the book *Astronomie* by Bennett and colleagues [13], and the illustrations are made by us.

[2] In our case, that is the antenna.

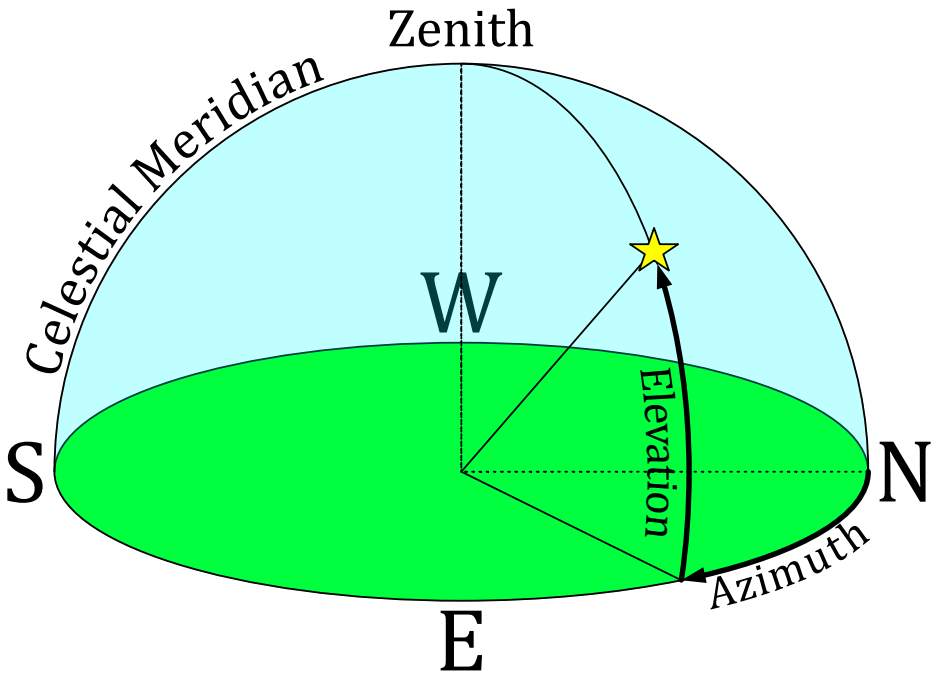


Figure 3.1: Azimuth and elevation are local polar coordinates.

on which we are projecting the latitude and longitude lines from Earth's surface (cf. Figure 3.2).

We refer to the "celestial longitude" *right ascension* (RA) and the "celestial latitude" *declination* (DEC). However, this reference frame is still rotating with Earth. We need a way to fix it in place, independently of the planet's rotation. To do that, we introduce the ecliptic.

The ecliptic is the apparent path that the Sun traces through the sky over a year, as seen from Earth. A different way of thinking about it describes it as sitting on the plane defined by Earth's orbit around the Sun (cf. Figure 3.3). In Figure 3.2, we can see that the ecliptic crosses the celestial equator at exactly two points. We can choose one of those points as the zero point for the right ascension. The chosen point is the one that points to the Sun on the day of the vernal equinox (cf. Figure 3.3). This point is called *First of Aries*[3].

We have now decoupled our coordinate system from the rotation of the Earth around itself. Technically, there is still a parallax effect while the Earth is orbiting the Sun, and the Earth's rotation axis is precessing very slowly. However, both effects are so small that they are usually neglected.

Declination is given in degrees, while right ascension is given in hours, where 24 hours is a full circle. To understand why it is convenient to use this unit, we need to introduce

[3]In the literature, it is also often referred to as vernal equinox, which can be confusing.

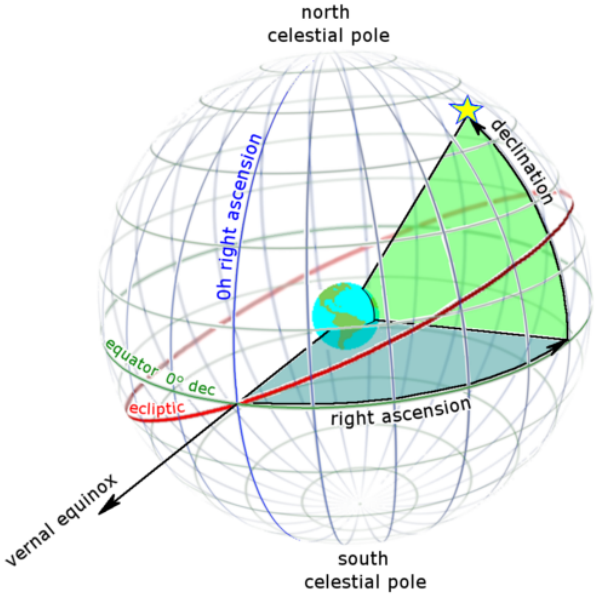


Figure 3.2: Right ascension and declination are like longitude and latitude on the celestial sphere, with the 0 meridian going through the first point of Aries, also known as the vernal equinox. Image source [14]

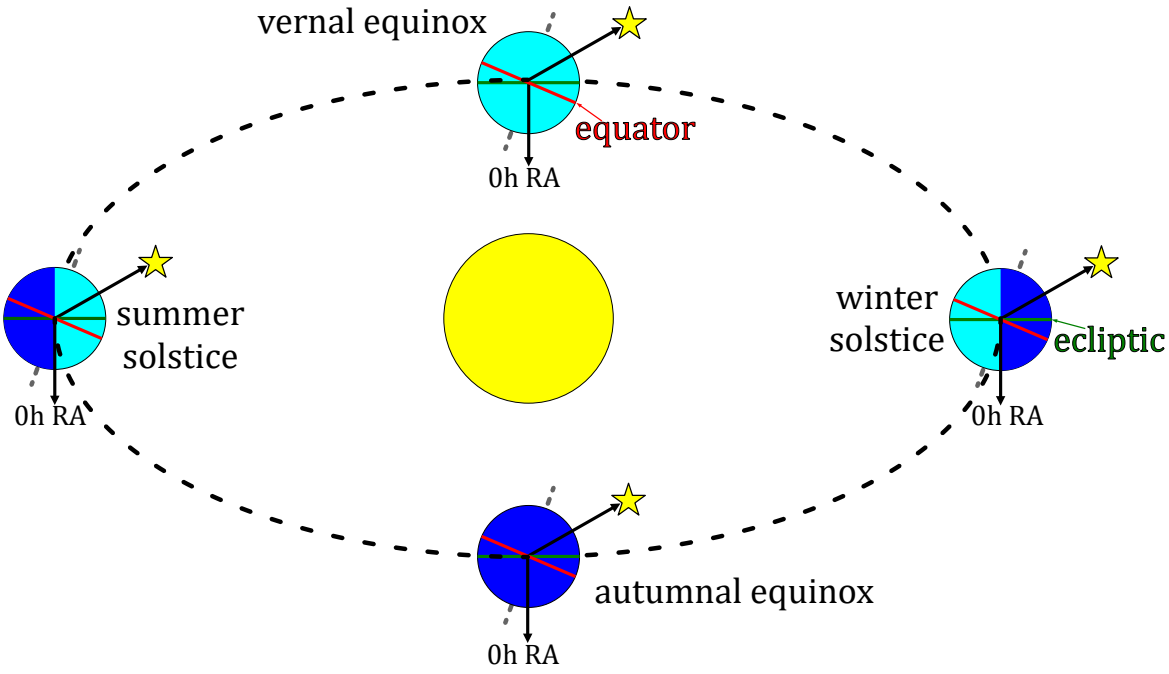


Figure 3.3: There are two points where the equator crosses the ecliptic. The one facing the Sun at the vernal equinox is called the first point of Aries. The image is not to scale.

the notion of sidereal time.

3.3 Local Sidereal Time

A day is 24 hours long. However, what exactly do we mean by that? In everyday life, this usually means the *mean solar day*. That means the time that the Sun takes to be in the same spot in the sky, averaged over a whole year. If one were to stop the time from when the Sun reaches its highest point in the sky today to when it reaches its highest point in the sky tomorrow, one would get on average 24 hours. However, as you can see in Figure 3.4, that is not the same as the Earth rotating once around its axis.

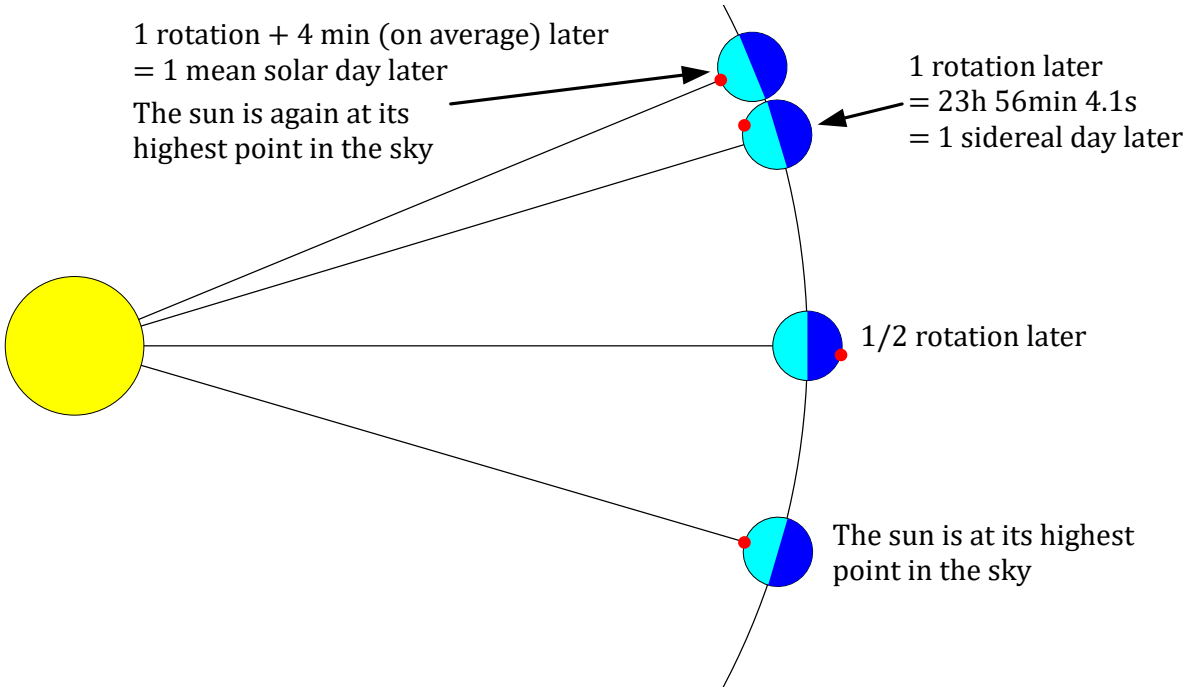


Figure 3.4: The sidereal day corresponds to one full rotation of the Earth around its axis, and it is about 4 minutes shorter than the solar day. The image is not to scale.

The Earth needs 23h 56min 4.1s to complete one rotation, and the last four minutes to make up for the distance that it rotates around the Sun during that day. This shorter time is called a sidereal day. A sidereal day is divided into sidereal hours, sidereal minutes, and sidereal seconds, the same way a solar day is. However, it is important to note that a sidereal hour is shorter than a "normal" solar hour. The same holds for minutes and seconds.

Since we now have a sidereal day that is made up of 24 sidereal hours, it makes sense to introduce a sidereal time. This is called the *local sidereal time* (LST)
The local sidereal day starts (LST = 00:00:00) when the first point of Aries[4] (RA =

[4]or any point with RA = 0

$0, \mathrm{DEC} = 0$) crosses the celestial meridian[5]. The celestial meridian is a great circle around the Earth that goes through the north and south poles as well as the zenith of the observer (cf. Figure 3.1). Therefore, the LST depends on the longitude of the observer. This also explains why it is called the *local* sidereal time, since it changes if we change location[6].

3.4 Hour Angle

Imagine you are stargazing in your backyard and you have brought along a camera to take a long-exposure image like the one in Figure 3.5.

Figure 3.5: A long-time exposure image of the night sky. This image was created by stacking several short-time exposure images. The image was recorded on the 9th of March 2025.

One would spontaneously think that it seems like all the stars would rotate around one single one. The centre of this phenomenon is the well-known North Star, Polaris/the North Celestial Pole. It has been used since antiquity to determine the direction of the geographic north. It is simply a fortunate coincidence that there is a star that happens

[5]Crossing the meridian is equivalent to being at the highest point in the sky (cf. Figure 3.6).

[6]Our (normal) solar time works in the same way. However, for convenience in everyday life, we have introduced time zones. So, the local solar time on our watches changes in a few discrete, big steps.

to align with the north celestial pole. Due to the slight precession of Earth's rotational axis, this alignment will not be true anymore in a few thousand years.

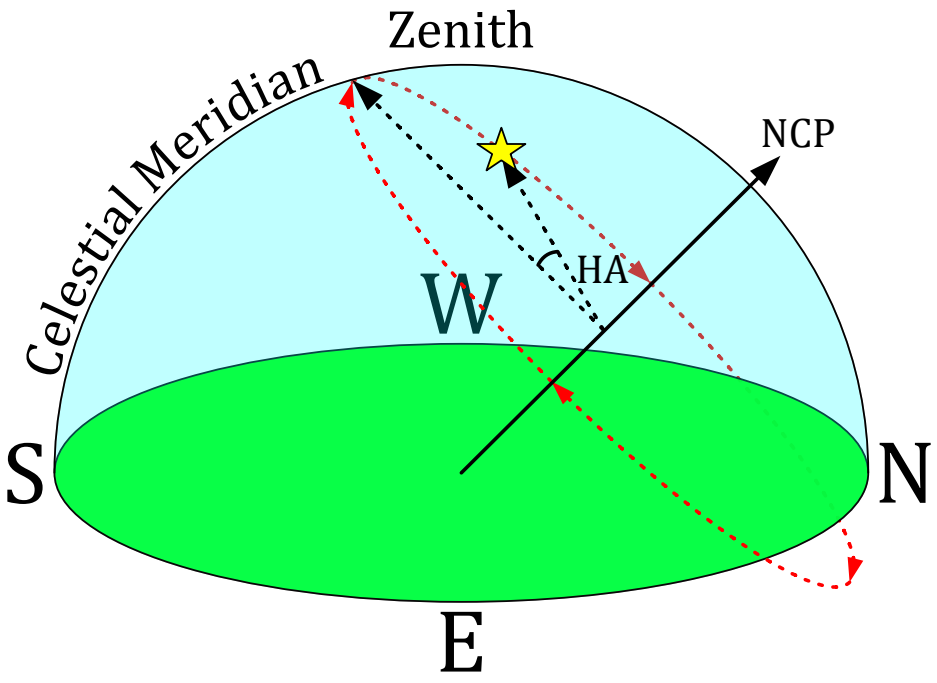


Figure 3.6: The hour angle is the angle between a star and its highest position in the sky, along an imaginary circle around Polaris. The hour angle also corresponds to the time since the star was at its highest position in the sky.

If we compare Figure 3.5 now with Figure 3.6, we can understand how the stars seem to travel on circles. We are also already familiar with the idea of measuring angles in (sidereal) hours. If we combine these ideas, we can describe the position of a star by the angle between its current position and when it was highest in the sky. This is the so-called *Hour Angle*. The advantage of giving this angle in hours is that it also directly tells us how much time has passed since it was highest in the sky.

3.5 Converting RA-DEC to AZ-EL

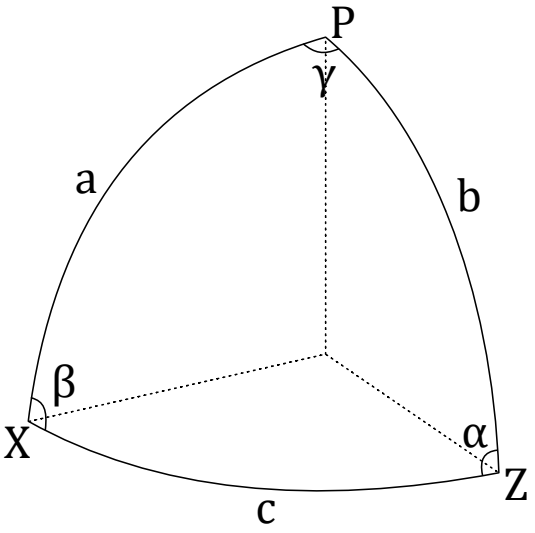
Now that we understand the basic terms well, we can derive the formula needed to convert the RA-DEC coordinates to AZ-EL. For that, we are going to use the spherical law of cosines:

$$\cos(c) = \cos(a)\cos(b) + \sin(a)\sin(b)\cos(\gamma) \tag{3.1}$$

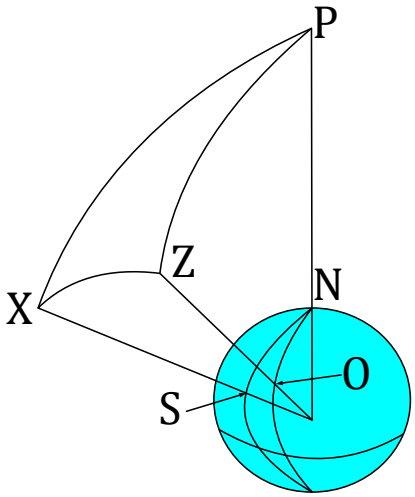
and the spherical law of sines:

$$\frac{\sin\alpha}{\sin a} = \frac{\sin\beta}{\sin b} = \frac{\sin\gamma}{\sin c}. \tag{3.2}$$

If we now look at the figures 3.7a and 3.7b, we can identify a as $90° - \delta$, with δ being the declination; b as $90° - \phi$, with ϕ being the latitude of the observer and c as $90° - el$, with el being the elevation angle; α as $-az$, with az being the azimuth angle and γ as the hour angle.



(a) The position of the target (X), the zenith of the observer (Z) and the north celestial pole (P) form a triangle on the celestial sphere.



(b) Projection of the position of the observer (O) and the subpoint of the target (S), onto the celestial sphere.

Figure 3.7

Combining equation (3.1) and (3.2) we find:

$$\sin(el) = \sin(\delta)\sin(\phi) + \cos(\delta)\cos(\phi)\cos(\gamma) \tag{3.3}$$

$$\tan(az) = \frac{\sin(\gamma)}{\cos(\gamma)\sin(\phi) - \tan(\delta)\cos(\phi)} \tag{3.4}$$

3.6 ICRF

We have an observer-centred coordinate system and an Earth-centred coordinate system. What we are still missing is a solar system-centred one. That is where the International Celestial Reference Frame (ICRF) comes in. For this purpose, we are essentially copying the RA-DEC system, also often known as the equatorial system, and moving its origin to the *solar system barycentre*[7] (SSB) at epoch[8] J2000.0. If one prefers to use Cartesian coordinates, then we can define the x-axis as pointing toward the first point of Aries,

[7] A barycentre is the centre of mass of a system of objects. Hence, the SSB is the point in space around which all our planets and the Sun rotate. The SSB is inside or very close to the Sun.

[8] Epoch just means a specific point in time. Epoch J2000.0 refers to January 1, 2000, at 12:00 TT (Terrestrial Time), or January 1, 2000, 11:58:55.816 UTC.

the z-axis points towards the north celestial pole and the y-axis is chosen such that it completes a right-handed coordinate system. [15, 16]

If one knows the position of a spacecraft in the ICRF relative to the SSB v_s (cf. Figure 3.8), then one can find its position relative to the observer v_o, with some simple vector geometry:

$$v_o = v_s - (v_1 + v_2 + v_3) \tag{3.5}$$

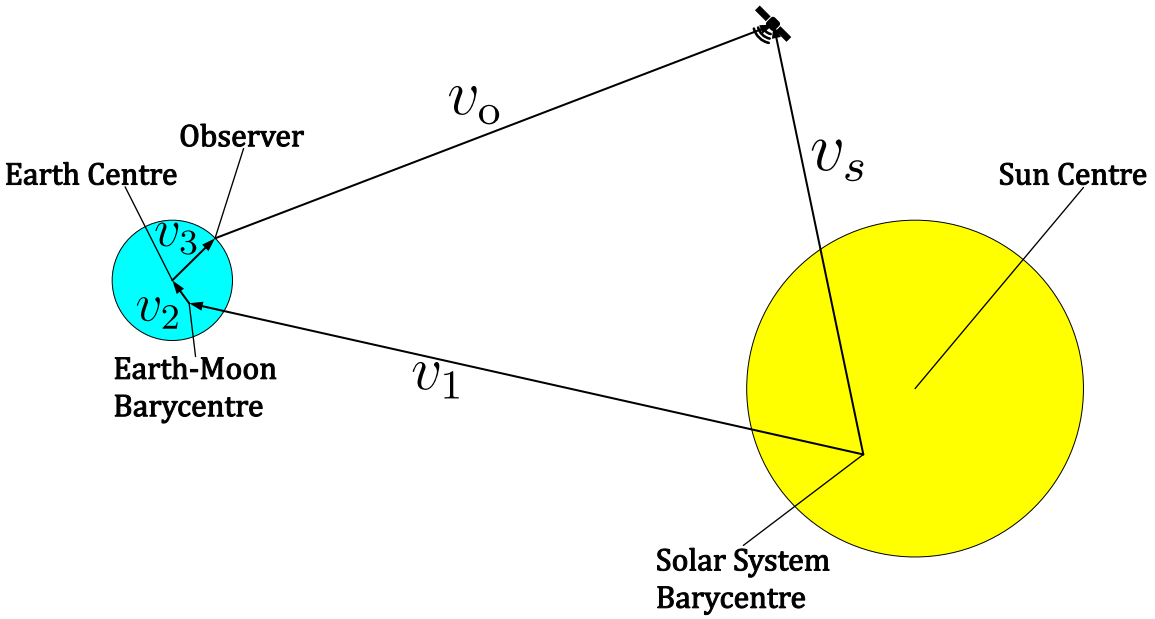


Figure 3.8: The position of celestial bodies is usually given relative to their barycentre. One can find their relative position by using vector geometry. The image is not to scale.

Chapter 4

User Interface

The following chapter gives an overview of the features of the satellite tracker and explains the user interface. However, it should also serve as a quick start guide for users who are interested in using the application to control the antenna. More rigorous explanations of how the individual subsystems work can be found in the referenced sections.

4.1 Overview

After launching the program, it will look like Figure 4.1. This is the interface used to track satellites and control the antenna. It is comprised of several sections called widgets. In the following sections, we will explain in detail what the different widgets are for and how to use them.

Note that the program might freeze directly after being launched. This is because it is downloading the latest data. If this is the case, a corresponding message will be displayed in the console on the bottom right of the interface.

4.2 Tracking Modes

At the top of the Tracking Modes widget, there is a drop-down menu that allows the user to switch between the five different tracking modes.

- List
- RA/DEC
- OMM/TLE
- SPICE
- AZ/EL

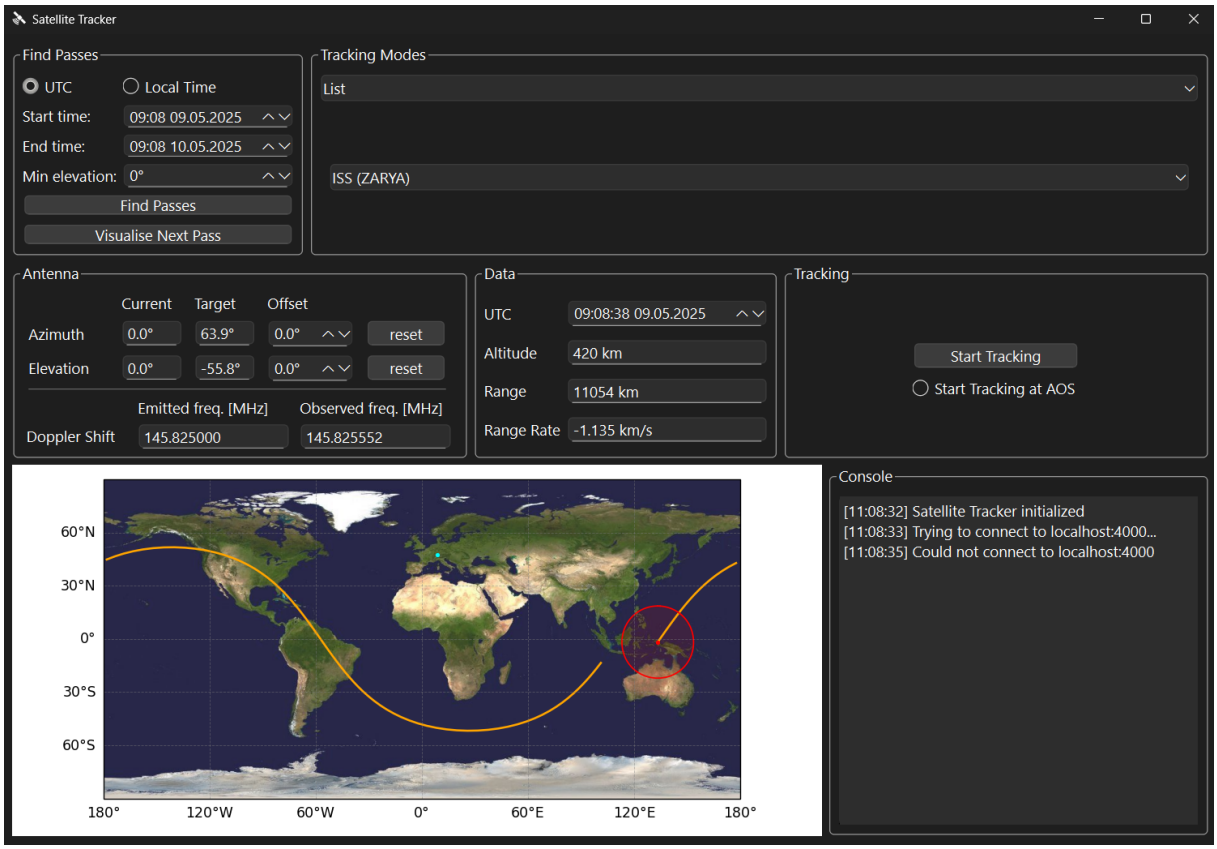


Figure 4.1: The interface

The different modes give the user the ability to choose between different data sources depending on their needs.

Not every parameter is supported in all modes. Table 4.1 gives an overview of which tracking mode supports which feature and which parameters are calculated.

4.2.1 List

Figure 4.2: In List mode, the user can select a target from a predefined list.

As the name implies, the tracking mode List consists of a predefined list of targets that are selectable from a drop-down menu (cf. Figure 4.2). This is the primary way to use the satellite tracker and supports all features. All needed data will automatically be downloaded and saved (cf. Sections 5.4 and 5.5).

	List	TLE/OMM	SPICE	RA/DEC	AZ/EL
Azimuth	✓	✓	✓	✓	✓
Elevation	✓	✓	✓	✓	✓
Range	✓	✓	✓	-	-
Range Rate	✓	✓	✓	-	-
Altitude	✓	✓	✓	-	-
Doppler Shift	✓	✓	✓	-	-
Subpoint	✓	✓	✓	✓	-
Footprint	✓	✓	✓	-	-
Ground Track	✓	✓	✓	-	-
Find Passes	✓	-	-	-	-
Visualise Next Pass	✓	-	-	-	-

Table 4.1: Not all tracking modes support all features.

There are two ways to add satellites to the list. One is manually (cf. Section 5.1.2) and the other one is via a button in tracking mode TLE/OMM (cf. Section 4.2.3).

4.2.2 RA/DEC

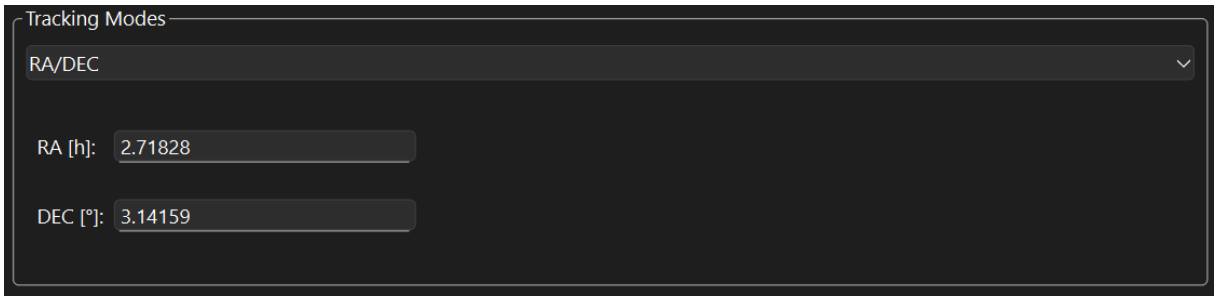


Figure 4.3: RA/DEC mode allows the user to use right ascension and declination coordinates.

In this mode, one can track a fixed point in the sky (e.g. a specific star) using right ascension and declination coordinates (cf. Section 3.2). The user needs to provide the coordinates in decimal hours for the right ascension and decimal degrees for declination. However, in the literature, the coordinates are often given like this: $02^h31^m48.7^s, +89°15'51''$ where the right ascension is given in hours, minutes, and seconds while the declination is given in degrees, arcminutes, and arcseconds. This can be converted into decimal form using:

$$\text{RA} = h + \frac{m}{60} + \frac{s}{3600}, \qquad \text{DEC} = d + \frac{m}{60} + \frac{s}{3600} \tag{4.1}$$

with h hours, d degrees, m (arc-)minutes and s (arc-)seconds. Hence, the example above would be RA = 2.530194 h and DEC = 89.26417°.

4.2.3 TLE/OMM

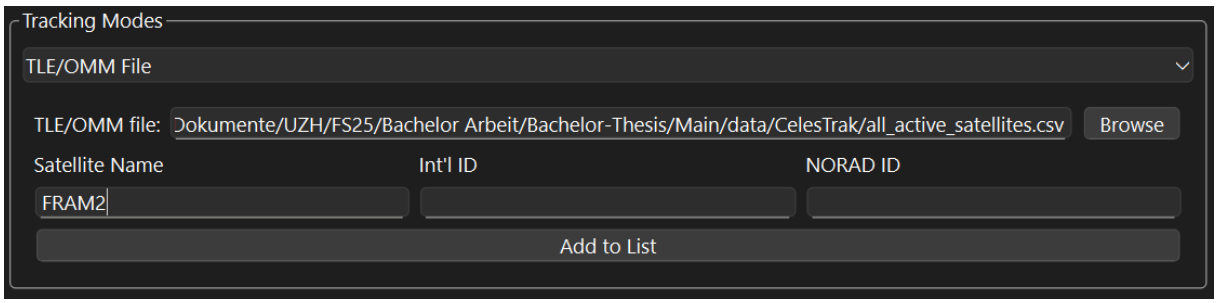


Figure 4.4: In TLE/OMM, the user can select a satellite by its Name, international designator or NORAD id.

This mode allows the user to select a specific OMM or TLE file as a source (cf. Section 5.7). This is useful if one wants to track a satellite that is not available in List mode. To select a satellite from the data, the user needs to provide the name, international designator (Int'l ID), or NORAD catalogue number (NORAD ID); one is sufficient. If the user enters several valid ones, the program will select from left to right. For example, assume that the user entered the satellite name TEMPSAT 1 and tracked it for a while. Now they want to switch to TDRS 3, however, they only know its Int'l ID: 1988-091B, so they enter that without removing TEMPSAT 1 from the *Satellite Name* field. The antenna will still try to track TEMPSAT 1.

Also, note that the name has to be exactly the same as in the data. That means if the user wants to track OSCAR 7, they have to enter OSCAR 7 (AO-7). The only exception is that the input is not case-sensitive. The program will handle that automatically.

Once the user has selected a satellite by entering its name or ID, they can track it or add it to the list, so that it becomes available in List mode. For that, the *Add to List* button at the bottom of the widget is used. One can also provide the frequency the satellite is sending on in the *Antenna* widget (cf. Section 4.3), and it will be saved to the list as well.

4.2.4 SPICE

This mode (cf. Figure 4.5) works similarly to the TLE/OMM mode but for interplanetary missions. Here, the user will have to provide a meta kernel and the name of the satellite. Note that, while providing a meta kernel (and all other needed kernels) is not hard (cf. Section 5.6.1), it is much easier to use the List mode. So, SPICE mode is mainly a backup for when the Horizons API does not work.

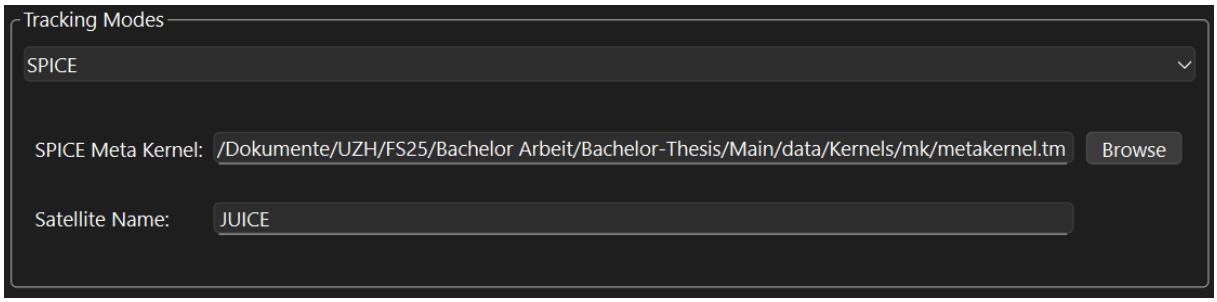


Figure 4.5: The SPICE mode allows the user to track interplanetary missions. However, it is more of a backup solution if List mode does not work.

4.2.5 AZ/EL

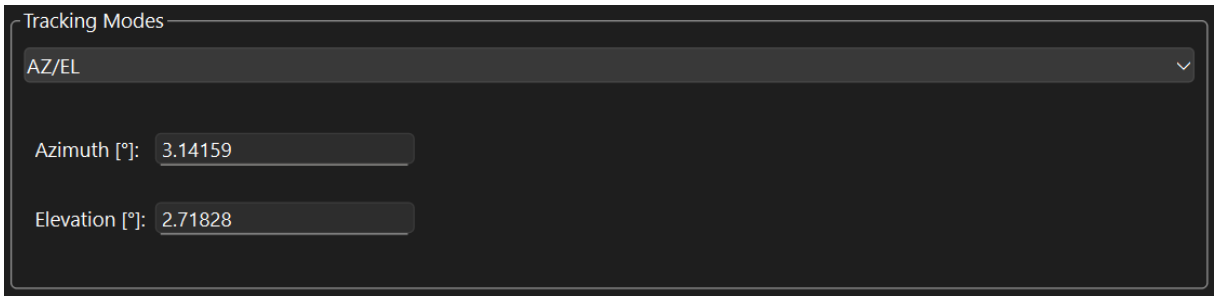


Figure 4.6: In AZ/EL mode, the orientation of the antenna can be controlled directly.

In this mode, the user can manually provide azimuth and elevation values (cf. Section 3.1) to point the antenna in a given direction. This mode is mainly intended for testing and maintenance purposes.

4.3 Antenna and Data

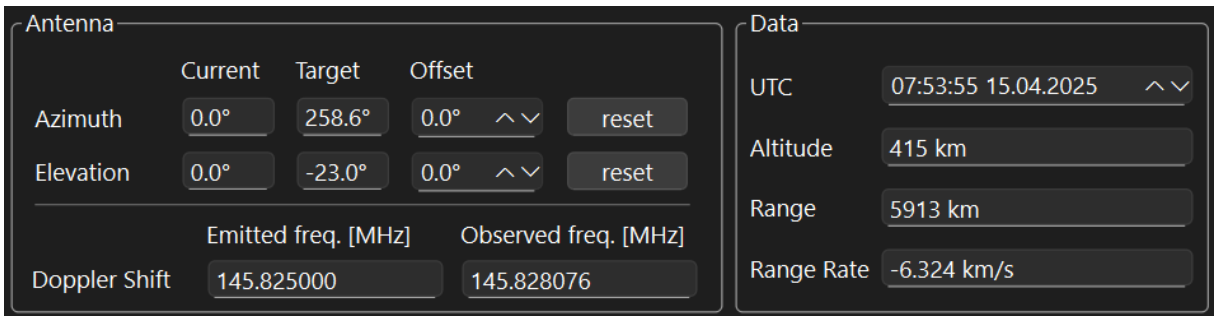


Figure 4.7: Antenna and Data widget display all available information about the antenna and satellite.

The two widgets *Antenna* and *Data* provide the user with all the available information about the antenna and the satellite. The *Current Azimuth / Elevation* shows how the antenna is currently oriented. The *Target Azimuth / Elevation* values show where the

satellite is. Note that the antenna will only move if the difference between *Current* and *Target* values exceeds a certain threshold (cf. Section 5.4.8).

The user can add an offset to the *Target* values. So, if the target azimuth is 203.1° and the offset is 3.4°, then the antenna will point at 206.5° instead of 203.1°. This is needed because the calculated target values have a certain uncertainty, and the actual position can sometimes differ by several kilometres from the calculated position, especially when the data are older. [17]

The offset can be changed using the arrow keys on the keyboard, or W, A, S, and D can be used. Should this not work, pressing Escape on the keyboard resets the focus. Now it should work. One can also simply select the text and enter the desired number. The *reset* buttons reset the offsets to 0.

Since the satellite is moving relative to the antenna, the signal will be Doppler-shifted. That means that we have to look for the signal at a frequency slightly different from the one on which it was emitted.

If in List mode, the program will get the values for the emitted frequency from the satellite list file (cf. Section 5.1.2). Therefore, if the user wants to change the frequency, they need to do it in the file.

If in TLE/OMM or SPICE mode, the user can change the frequency in the UI only. In RA/DEC and AZ/EL modes, the Doppler shift will not be calculated because it makes no sense for those modes.

Lastly, in the Data widget is the current time in UTC, the altitude of the satellite above the Earth's surface, the direct distance from the satellite to the antenna, and its relative velocity.

4.4 Tracking

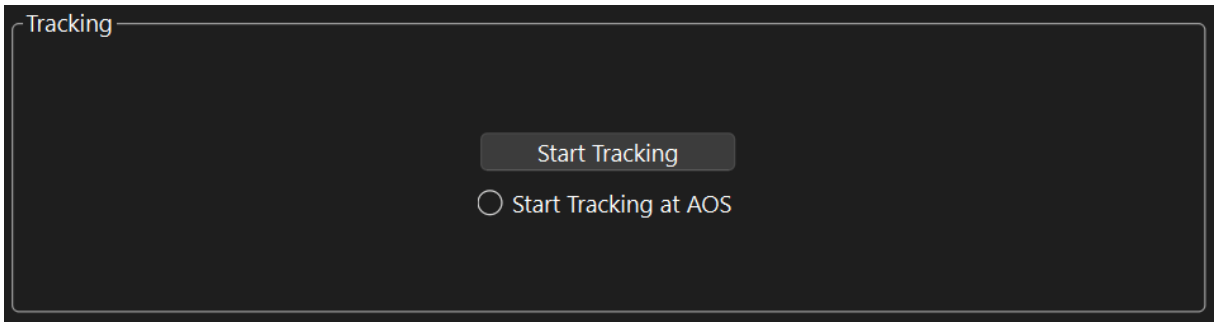


Figure 4.8: The antenna only moves when tracking is turned on.

We have to be a little bit careful with notation. Until now, we have mentioned track-

ing and tracking modes, and what we meant was *calculating where the satellite is*. This happens continuously in the background. However, there is also tracking as in *the antenna is pointing at the satellite and following its movements*. So, when we say *tracking is turned on*, we mean that we are sending commands to the antenna to move and follow the satellite. We will try to avoid conflicting notation as much as possible and hope that it is clear from context which kind of tracking we mean.

We only want the antenna to move when there is a chance that we will see something. So, the antenna will only move when the *Start Tracking* button is pressed. To turn tracking off, the same button can be pressed again. Alternatively, the space bar can be used to turn tracking on and off.

When the satellite is below the horizon, the tracking mode is turned off automatically. That also means that one cannot start tracking before the satellite has risen above the horizon. To help with that, one can click on *Start Tracking at AOS*. With this active, the tracking will automatically turn on as soon as the satellite rises above the horizon.

4.5 Map

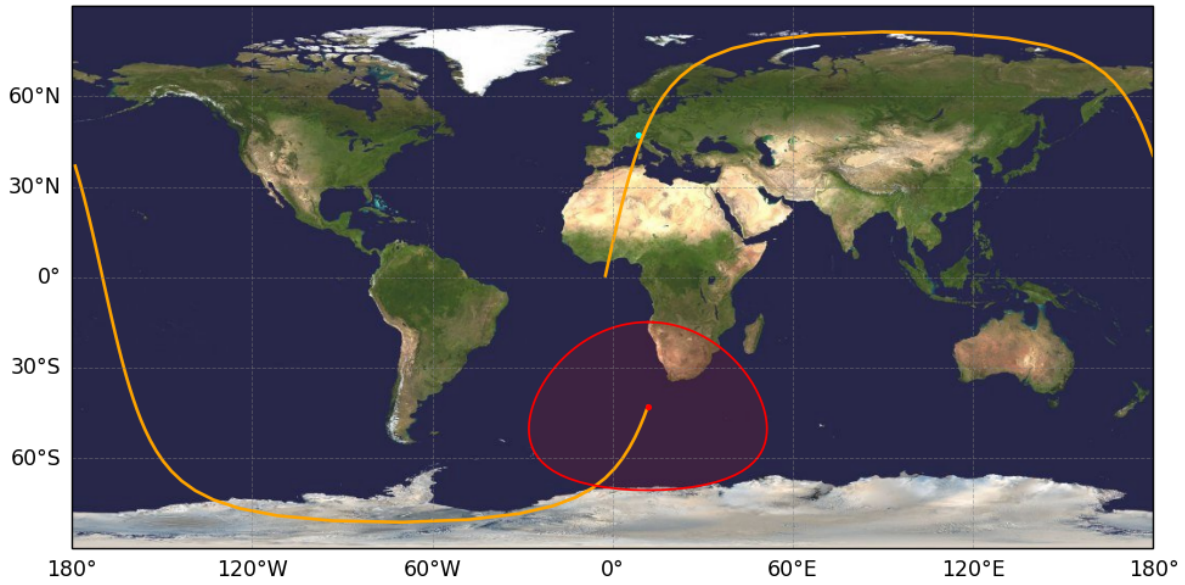


Figure 4.9: The Map shows the subpoint of the satellite (red dot), where the satellite is visible (red area), the ground track for the next 90 min (orange) and the position of the antenna (cyan). The world map image is provided by NASA. [18]

The red dot on the map (cf. Figure 4.9) marks the subpoint, the point on the Earth's

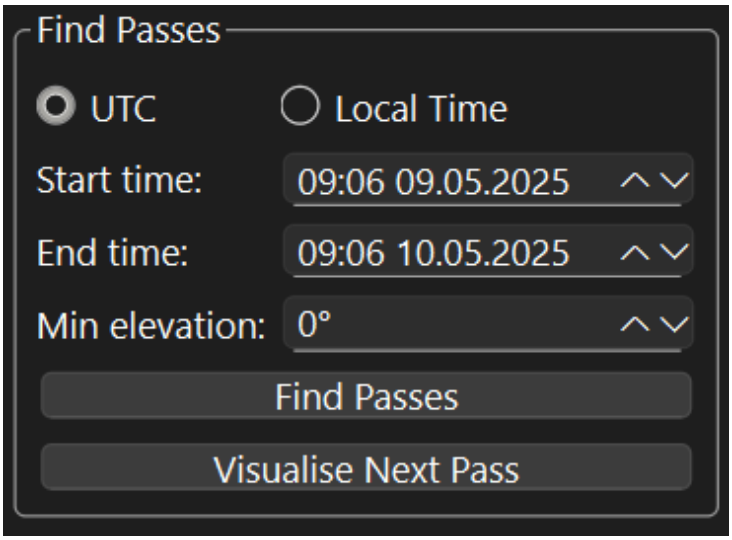


Figure 4.10: The Find Passes tool finds the time windows when the satellite will be visible to the antenna.

surface directly under the satellite. The shaded red area shows from where the satellite is visible (cf. Section 5.4.7). The orange line shows the ground track for the next 90 minutes. This time can be adjusted in the *config file* (cf. Section 5.4.6). And finally, the cyan dot is where the antenna is.

If there is no red dot on the map, it means that no valid satellite is currently selected.

4.6 Find Passes

The *Find Passes* widget is a tool to plan observations (cf. Figures 4.10 and 4.12). With this tool, one can find the time windows during which the satellite will be visible to the antenna. The results will be displayed on the console. Note that if one uses a minimum elevation angle of greater than zero, the times displayed in the console will be according to when the satellite reaches this angle. The user can switch between UTC and local time.

The *Visualise Next Pass* button will open a separate window, showing the next pass from the point of view of the antenna (cf. Figure 4.11).

4.7 Console

The last widget is the console. This is just a general output for all types of information (cf. Figure 4.12). For example, it informs the user when new data is being downloaded.

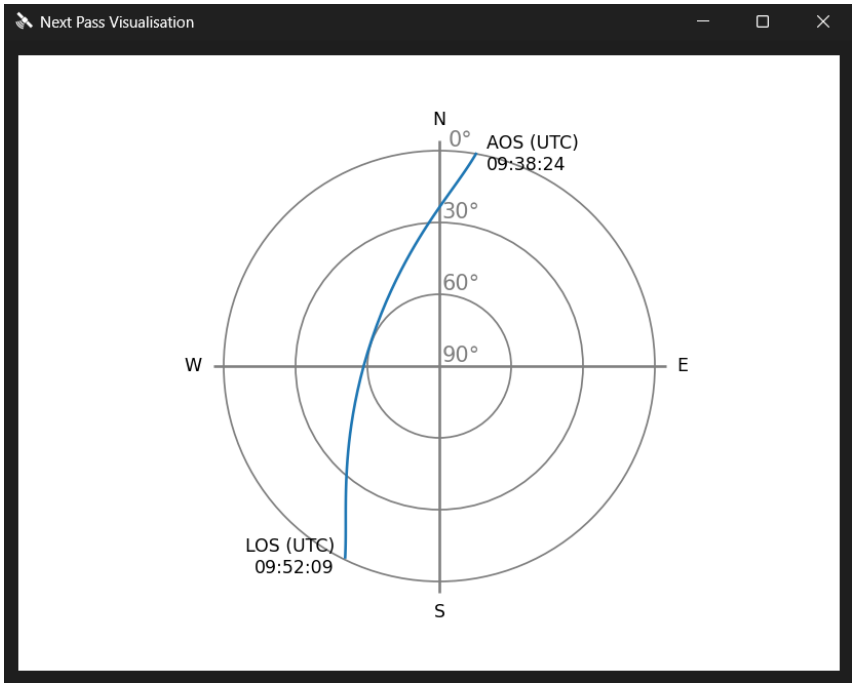


Figure 4.11: Visualisation of the next pass from the point of view of the antenna.

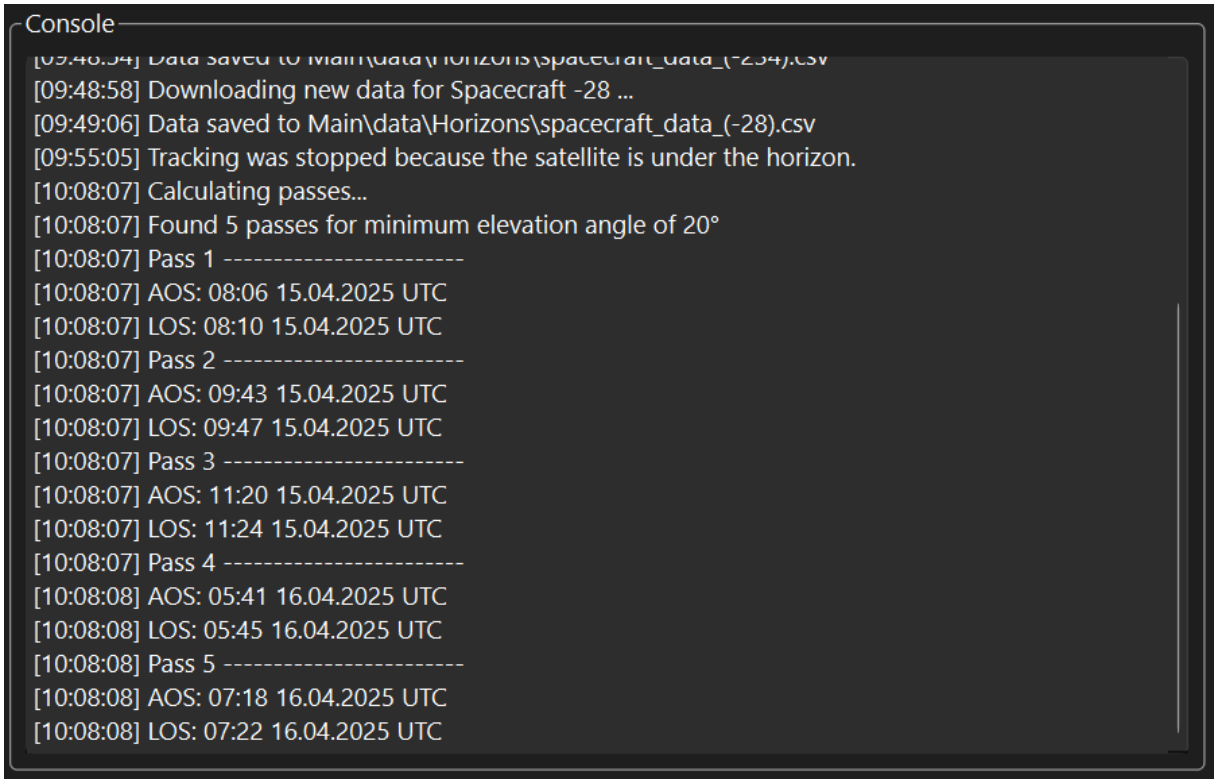


Figure 4.12: The Console provides the user with information about the status of the program and serves as output for the find passes feature.

4.8 Errors

To avoid the user getting spammed, no error messages will be displayed until tracking is turned on. This is because of a technical limitation. We can not differentiate between the case where the satellite is invalid because the user made a mistake when entering the information, and the case where the satellite is invalid simply because the user is not finished entering the information. Only once the user turns on tracking can we be sure that they are finished, and we can check for mistakes.

Not displaying error messages is problematic because we should not wait until the satellite is above us only to discover that something is not working. However, there is a solution to the problem. If one can see the satellite[1] on the map, everything is working smoothly; if not, something is going wrong. To get the error message displayed, one can quickly turn on tracking.

The most common error that the user will encounter is that they have not selected a valid satellite or that the satellite cannot be found in the data. In this case, an error message like this is shown:

```
[12:01:33] Error calculating satellite data: cannot unpack non-iterable
NoneType object
[12:01:34] Could not find FRAM2 in file C:/Users/felix/OneDrive/Dokumente/
UZH/FS25/Bachelor Arbeit/Bachelor-Thesis/Main/data/CelesTrak/all_active_sa
tellites.csv
```

If in TLE/OMM or SPICE mode, one should check the spelling and that the right file is selected. If that does not help or they are in List mode, we recommend trying it with the NORAD or Int'l ID instead. Often, several names and abbreviations are used for the same satellite.
It is also possible that the satellite is no longer available in the latest data. Reasons for that could be that their orbit decayed so much that they burned up in Earth's atmosphere, or (e.g. for crewed missions) they have landed back on Earth. It could also be an interplanetary mission that has left Earth's orbit. In that case, one should find its Horizons (JPL) ID and add it to the list (cf. Section 5.1.2).

[1] meaning the red dot

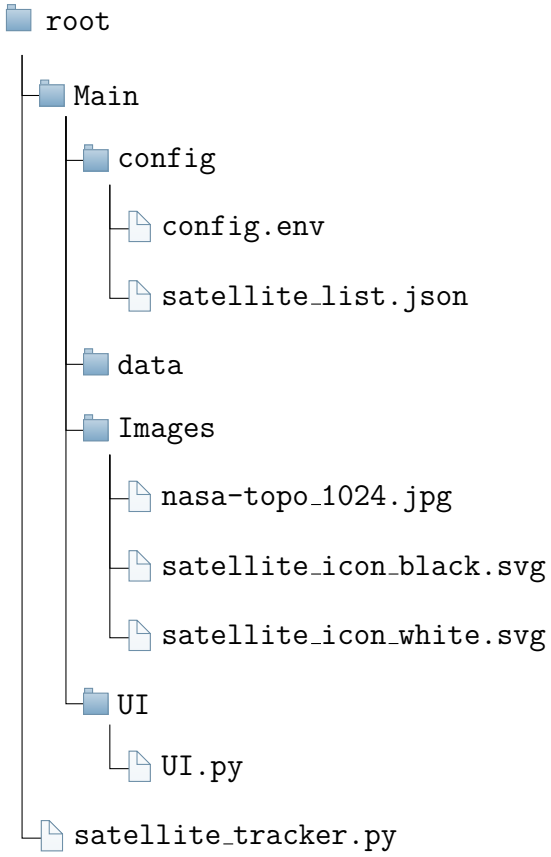
Chapter 5

Software Architecture

The public and stable version of the code is available on GitHub: https://github.com/Akut-Luna/Satellite-Tracker-v1/tree/main

5.1 Project File Structure

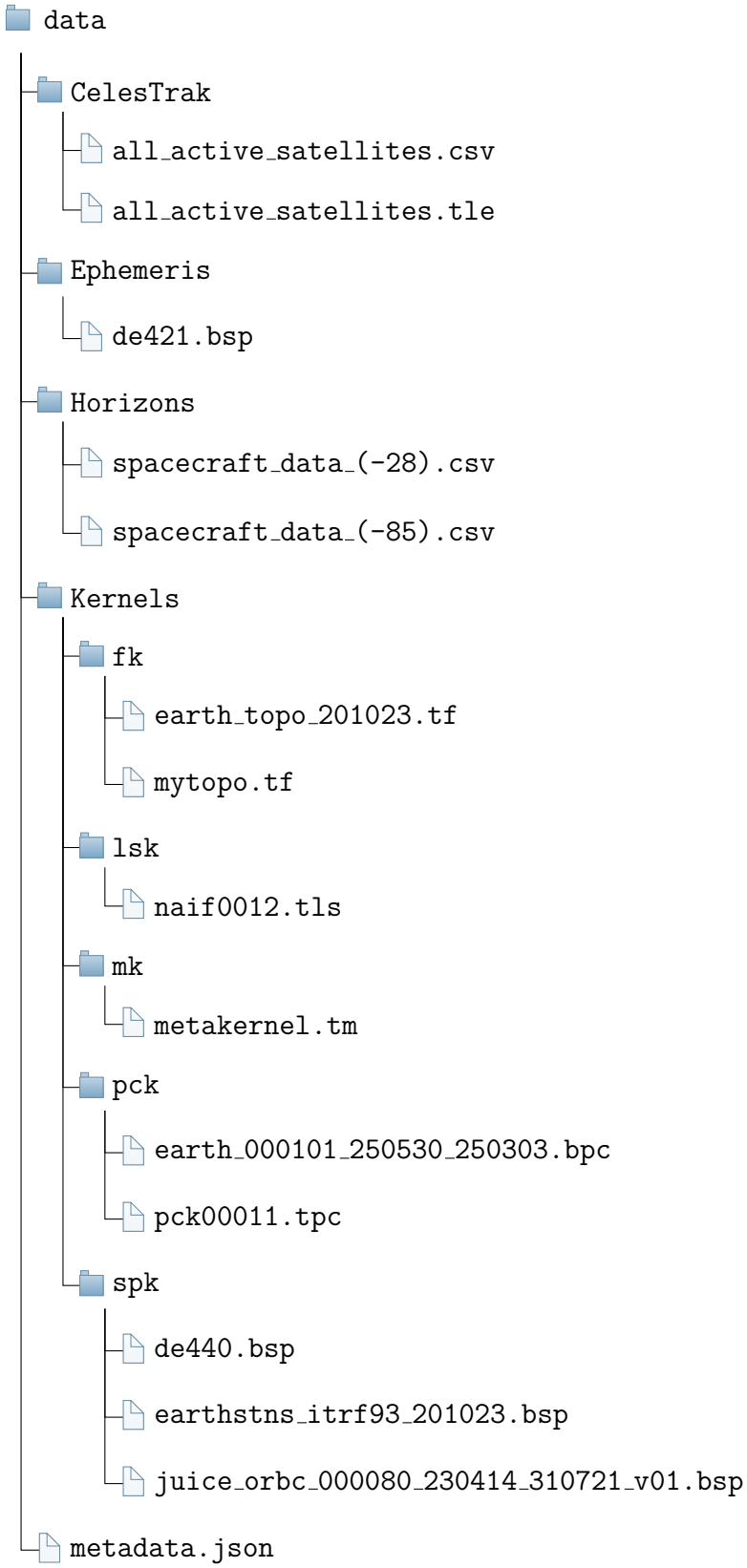
This is the file structure for the present project:



Note that there are only two Python files. One can start the program by running `satellite_tracker.py`, which in turn will run `UI.py`. `UI.py` contains everything else. It is certainly not an ideal solution to run the whole application via the UI. The project

just grew historically like this.

This is an example of how the file structure for the data folder can look:



5.1.1 Data

CelesTrak

In `CelesTrak` all GP data is saved. Note that `all_active_satellites.csv` needs to exist, otherwise the program will not run.

Ephemeris

`de421.bsp` contains ephemeris data, i.e. data on where which planet is at what time. Note that this data is only valid until 2050, and the file needs to exist for the program to start correctly.

Horizons

The files in `Horizons` contain position and velocity data, from the Horizons API, for specific spacecraft. Those files are automatically downloaded by the program (cf. Section 5.4.1).

Kernels

Kernels are data files that are only needed for the tracking mode SPICE. They will be explained in more detail in Section 5.6.1.

Metadata

Finally, `metadata.json` contains data on when CelesTrak and Horizons data were downloaded and until when the data is valid. The file has to exist for the program to start correctly, and it has to be in this format:

```
{
    "readme": "This file contains data on when the data was downloaded",
    "CelesTrak": {
        "last download": "2025-04-30T10:18:12.641305+00:00",
        "valid until": "2025-05-01T10:18:12.641305+00:00",
        "valid from": "2025-04-29T10:18:12.641305+00:00"
    },
    "Horizons": {
        "-85": {
            "last download": "2025-04-30T11:49:36.292753+00:00",
            "valid from": "2025-04-29T11:49:36.292753+00:00",
            "valid until": "2025-05-01T11:49:36.292753+00:00"
        },
```

```
        "-28": {
            "last download": "2025-04-30T11:49:45.363908+00:00",
            "valid from": "2025-04-29T11:49:45.363908+00:00",
            "valid until": "2025-05-01T11:49:45.363908+00:00"
        }
    }
}
```

5.1.2 Config

The `config.env` file contains configuration settings and parameters. The most important ones are the position parameters for the observer: latitude, longitude, altitude and time zone, as well as communication parameters for the motor controller: IP address and port.

The other settings will be explained in the corresponding sections, where they become relevant.

The `satellite_list.json` file contains a predefined list of targets. They can be Earth satellites, deep space spacecraft, or celestial bodies such as planets, moons or the Sun. Celestial targets such as stars cannot be included in this list. However, such targets can be tracked using the RA-DEC tracking mode. The file has to follow this format:

```
[
    {
        "name": "ISS (ZARYA)",
        "catalogs": {
            "NORAD": 25544,
            "Int'l": "1998-067A"
        },
        "frequency": 145.825
    },
    {
        "name": "Lunar Reconnaissance Orbiter",
        "catalogs": {
            "NORAD": 35315,
            "Int'l": "2009-031A",
            "Horizons": -85
        },
        "frequency": 0.0
    },
    {
```

```
        "name": "JUICE",
        "catalogs": {
            "NORAD": 56176,
            "Int'l": "2023-053A",
            "Horizons": -28
        },
        "frequency": 0.0
    },
    {
        "name": "Moon",
        "catalogs": {
            "Horizons": 301
        },
        "frequency": 0.0
    }
]
```

The values for `name`, `catalogs`, and `frequency` must not be empty. However, `catalogs` only need to contain at least one of the three options: `NORAD`, `Int'l` and `Horizons`. If `Horizons` is present, the other two are ignored. This is used to decide which data should be used for this specific target. Either the data is coming from the Horizon API or CelesTrak. For any future reference, this will be called the "Horizon case" or the "CelesTrak case".

5.2 Initial startup and loading of data

5.2.1 loading of data

When the program is launched, it starts by loading local data into memory for quick access. It first reads the parameters from `config.env`, followed by the contents of the files `satellite_list.json` and `metadata.json`, which are loaded into the variables `self.satellite_list` and `self.satellite_metadata`, respectively. Should the data change, either locally, i.e. the files, or in memory, we will have to reload it or save it to the corresponding file, respectively.

We are also loading the content of `all_active_satellites.csv` as a pandas DataFrame. The rationale behind this step now is to create a master dataset that contains all the information about every satellite we want to track[1]. To achieve that, we iterate through

[1]That means all satellites in the list, not every satellite available.

the satellite list. In the Horizon case, we add the content of the corresponding CSV file, which contains the data about this specific satellite, to the element that represents this satellite. Such an element would now look like this:

```
{
    'name': 'JUICE',
    'catalogs': {
        'NORAD': 56176,
        "Int'l": '2023-053A',
        'Horizons': -28
    },
    'frequency': 0.0,
    'df': <DataFrame [4321 rows x 10 columns]>
}
```

In the CelesTrak case, we will search in the data from `all_active_satellites.csv` for the satellite. Once found, we can use it to create a `EarthSatellite`[2] object. We again add to the corresponding element of the satellite list. Such an element would now look like this:

```
{
    'name': 'ISS (ZARYA)',
    'catalogs': {
        'NORAD': 25544,
        "Int'l": '1998-067A'
    },
    'frequency': 145.825,
    'EarthSatellite': <EarthSatellite ISS (ZARYA)>
}
```

5.2.2 checking for connection with motor controller

After the data has been loaded, the program will try to establish a connection with the motor controller. The motor controller functions like a server with an IP address and a port. So, we can ping it and wait for a response. To not freeze the rest of the program while waiting for replies from the motor controller, we will perform all motor controller-related tasks on a separate thread.

If there is no reply within five seconds, the program will assume that it is not connected to the motor controller. In that case, it will it will continue to function normally, except that it will not attempt to send or receive data from the motor controller.

[2]That is a Skyfield class.

If the connection was successful, a permanent TCP[3] socket will be established. Should the connection fail again while the program is running, it will attempt to reconnect. If it succeeds, then everything can continue as before. Should the reconnect timeout it will not attempt a second reconnect, but assume that the connection has been lost and continue as described above.

5.3 Main loop

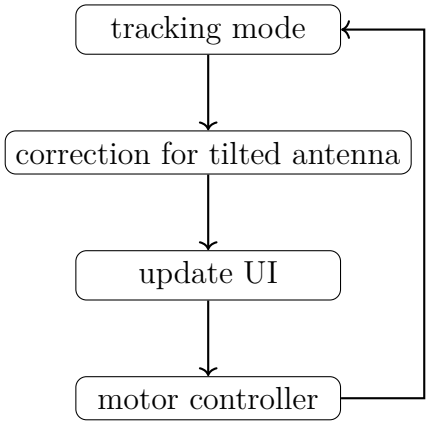


Figure 5.1: The main loop is set to cycle every 500 ms.

After the initial setup, we enter the main loop, which cycles every through every 500 milliseconds. In the following sections we will see how such a cycle looks for each tracking mode. For the tracking mode List, we will have to look at the Horizon case and the CelesTrak case separately.

5.4 Horizons stack

In Figure 5.2, you can see a schematic of what we are going to call the *Horizons stack*. It gives a general overview of how we calculate all necessary parameters in the Horizons case.

5.4.1 Update data

After the user has selected a satellite from the list, and before we do any calculation, we first check if the local data is still valid. For that, we just need to check the metadata (cf. sec 5.1.1). Should the spacecraft we are looking for not be in the metadata, then we need to update.

We do so by sending a request to the Horizons API and receiving data about the spacecraft's position and velocity for the next 24 hours in 1-minute steps. This data needs to be

[3]Transmission Control Protocol

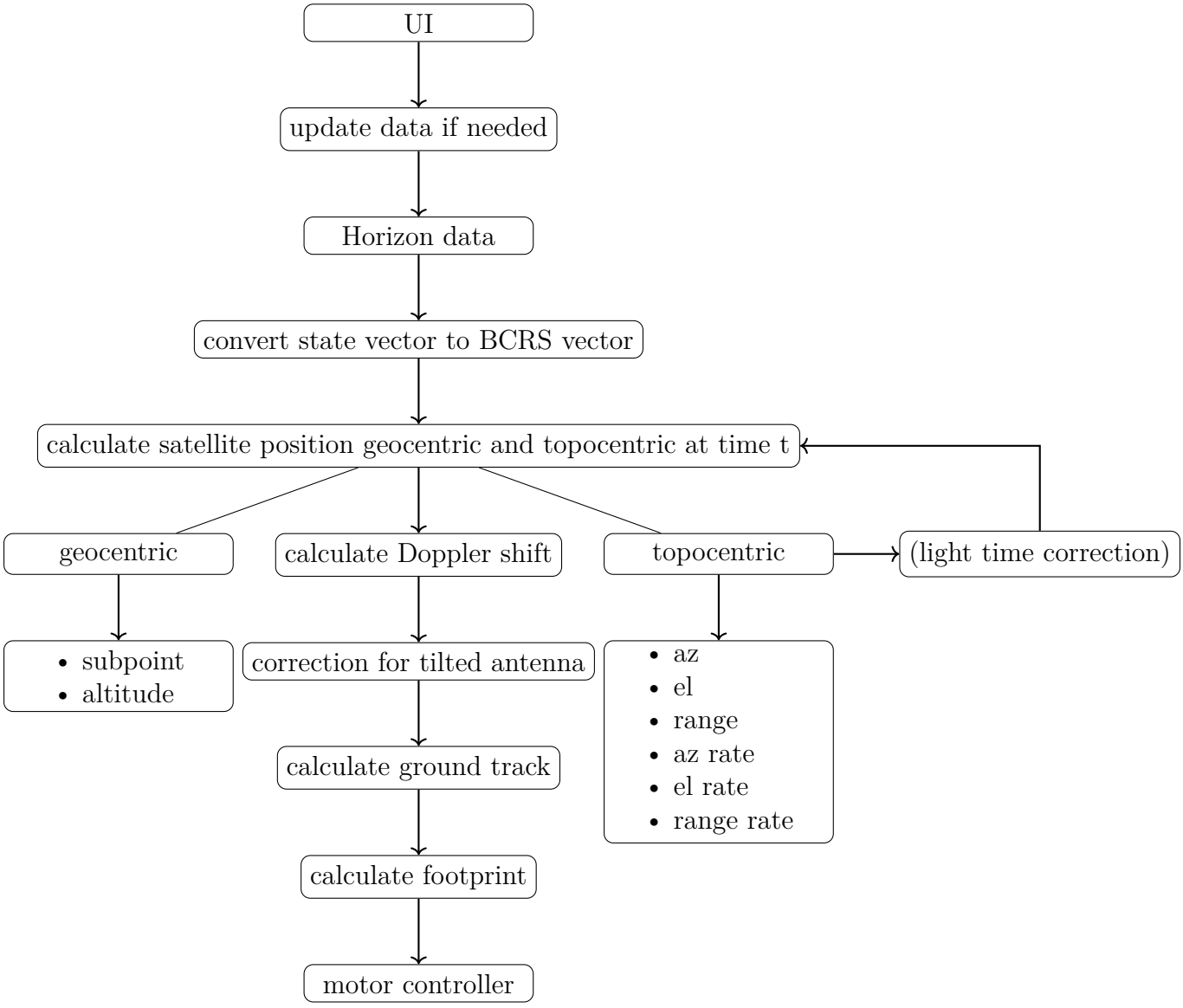


Figure 5.2: After selecting a satellite in the interface, the local data gets updated if needed and all necessary calculations and corrections are performed, before displaying the data on the UI and sending it to the motor controller.

parsed and then saved as a CSV file. During the parsing, we also convert from Barycentric Dynamical Time (BDT) to UTC. We do this by first converting to Universal Time (UT) and then applying a small correction UTC = UT − $\frac{\text{DUT1}}{86400}$. The Difference to Universal Time 1 (DUT1) can not be simply calculated but has to be looked up. The correction term is not implemented since the value for DUT1 is, by definition, always smaller than 0.9 s, and it can be safely neglected.

We want to avoid updating the data while tracking is turned on, because it freezes the program for a few seconds. Therefore, we update when we still have 30 minutes of data left. Should we reach this moment while tracking is turned on, we can continue for half an hour with the local data. Should we nevertheless run out of data, we force an update, even if tracking is still turned on. Lastly, we update and save the metadata.

5.4.2 Satellite position

If we combine the position and velocity vector into one, it is called the *state vector*. The Horizon API could have given us the azimuth and elevation directly. That begs the question of why we do our own calculations. The reason is that this way we are not too dependent on Horizons. Should the API stop working in the future, it will be relatively simple to implement a different source of data. This source only needs to provide the state vector for the target, which is more general than the azimuth and elevation angles, which are specific to the position of the observer.

First, we convert the state vector, which is given in the ICRF, into a Barycentric reference system (BCRS) vector. Geometrically, they are the same thing. The second is just the Skyfield version of the first. This is somewhat confusing because Skyfield also has a `ICRF` class. But the following calculations don't work if this class is used. What we have to use is the `Barycentric` class. The data that we need for that is position, velocity and time. Since the Horizon data is in 1-minute steps, we make a linear interpolation between the two data points closest in time.

We want to know the *topocentric*[4] and the *geocentric*[5] position of our spacecraft. Both can now be found using some simple vector geometry. The BCRS vector corresponds to v_s in Figure 3.8, and the topocentric vector corresponds to v_o.

To get the altitude of the spacecraft above Earth's surface and its subpoint from the geocentric vector, we can use the built-in Skyfield functions `altitude` and `subpoint`.

From the topocentric vector, we get, via the use of the Skyfield function `frame_latlon_and_rates`, the azimuth, elevation and range. The range, in contrast to

[4] relative to the observer
[5] relative to the centre of the Earth

the altitude, is the length of a straight line from the antenna to the spacecraft. We also get the rate of change for those three parameters.

5.4.3 Light time correction

The data that we receive from the Horizons API is already light time corrected. That means it is already corrected for the fact that if the distance to the spacecraft is, e.g. two light minutes, to the observer the spacecraft will not appear to be where it currently is, but where it was two minutes ago.

We have also written a light-time correction. This is again for the case that we have to use a different data source in the future, which does not provide light-time corrected data. In order not to overcorrect, our light time correction is deactivated. By setting `DISPLAY_LIGHT_TIME_CORRECTION_OPTION` in the config file to `True`, one can display an additional button on the UI, which allows turning this feature on and off.

5.4.4 Doppler Shift

Since the spacecraft is moving relatively quickly, we will have to account for the fact that the signal is Doppler-shifted.

$$f_o = \frac{f_e}{1 - \frac{v}{c}} \tag{5.1}$$

with f_o being the observed, f_e the emitted frequency and c the speed of light. It is reasonable to use the range rate for the relative velocity v. However, when the spacecraft flies towards the observer, the range decreases, and the range rate is negative. At the same time, we expect the frequency to increase because the spacecraft is coming closer. But if we require $\frac{f_e}{1-\frac{v}{c}} > 1$ and solve for v, we find that $v > 0$. Therefore, we have to choose the negative of the range rate as the relative velocity.

5.4.5 Correction for tilted antenna

All our calculations up to this point have assumed that the antenna is perfectly oriented towards North and it is perfectly perpendicular to the ground. However, in reality, this is not the case. The antenna will be ever so slightly tilted.

To correct for that, we convert our calculated azimuth and elevation angles into Cartesian coordinates[6], rotate by the Euler angles[7] roll ϕ, pitch θ, and yaw ψ, and convert back to azimuth and elevation.

[6] Since we are only interested in rotations, we can ignore the range and just use a unit vector.
[7] in the ZYX convention

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \; R_y = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \; R_z = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$\vec{v'} = R_x R_y R_z \vec{v}$$

For now, the angles are all 0. Once the antenna is operational, measurements of targets, whose positions are well known, need to be taken to determine by how much the antenna is tilted in each axis.

5.4.6 Ground track

The ground track of the satellite is the path on Earth's surface drawn by its subpoints in the near future. Specifically, we redo the calculations of the geocentric position of the satellite, but for the next 90 minutes in 1-minute steps. The number of steps can be adjusted in the config file by adjusting the variable `FLIGHT_PATH_STEPS`. This calculation happens in every loop of the programme. On less powerful computers, this may cause performance issues. Therefore, the variable `MIN_BEFORE_RECALCULATING_FLIGHT_PATH` is included in the config file. By default, it is set to 0. In this case, the flight path gets recalculated in every loop. If the variable gets set to an integer value greater than zero, the flight path will be saved to memory and only recalculated after the set number of minutes has passed.

5.4.7 Satellite Footprint

We can now display all the parameters that we calculated so far on the UI. However, there is one thing that we haven't calculated yet, and that is the footprint of the satellite. The footprint is the region on Earth's surface in which the satellite is visible above the horizon.

We project a cone from the satellite onto Earth's surface, and solve for the radius of the circle at the base of the cone. As we see in Figure 5.3, this results in a right-angled triangle, with a being the Earth's radius and c the sum of the Earth's radius and the altitude of the satellite above the surface. Taking the curvature of the Earth's surface into account, we find the radius with simple geometry

$$r = a \cdot \arccos\left(\frac{a}{c}\right). \tag{5.2}$$

When drawing the footprint on the map, we have to take into account that the map is also a projection of a 3D sphere onto a 2D plane. This results in the circle looking more

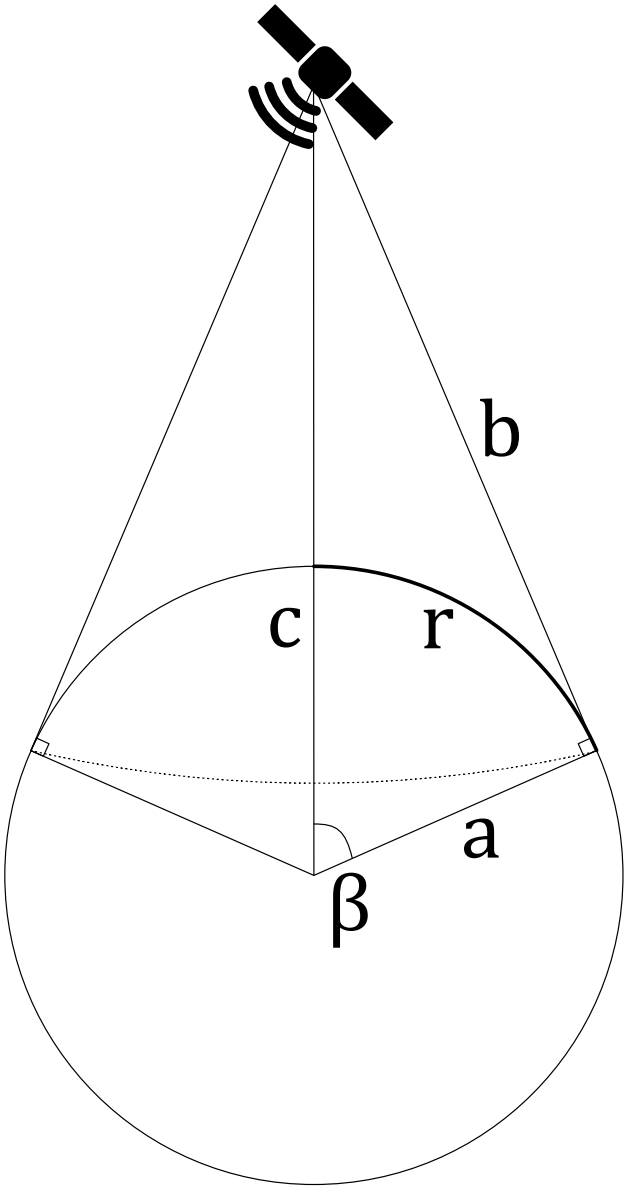


Figure 5.3: In order to correctly mark the area on the map, from which the satellite can be seen, we need to take the curvature of the Earth into account.

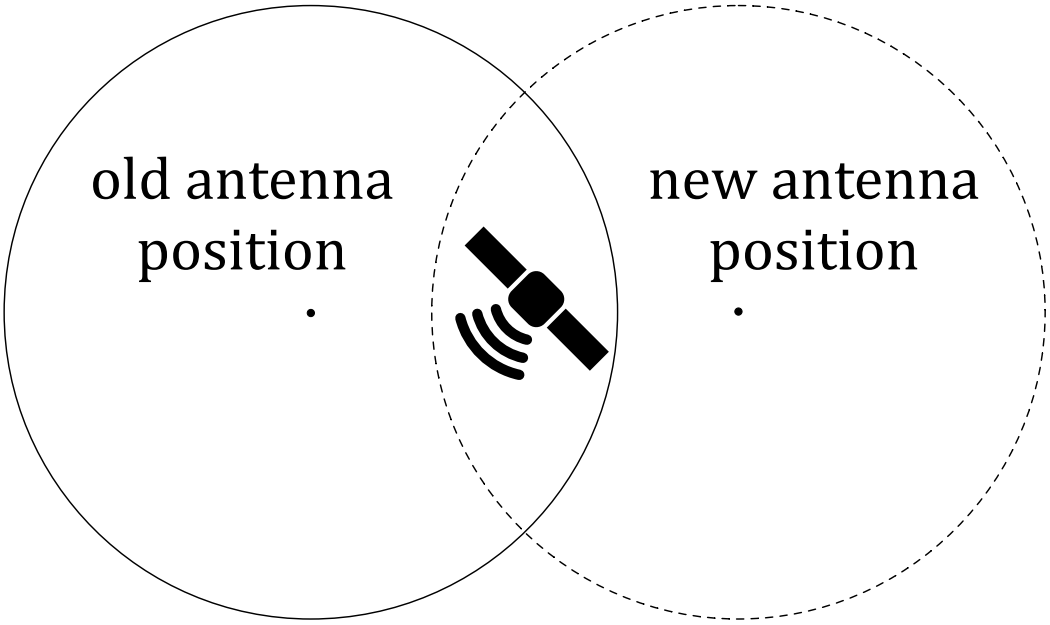


Figure 5.4: To prevent lagging behind the satellite, while tracking, we predict its position using its angular rate of change.

squished, the closer to the poles we get (cf. Figure 4.9.)

5.4.8 Motor Controller

Before we send the final azimuth and elevation values to the motor controller of the antenna, there are two more steps we need to take. The first is that we add the manual offset explained in Section 4.3. The second is that we check if we even need to send new values to the motor controller.

Moving the antenna causes vibration in the system, which is not ideal. As long as the satellite is in the narrow field of view of the antenna, there is no point in moving it. So, we need to determine if the satellite is about to leave the field of view. For that, we compare the angle between the current position of the antenna and the position of the target, aka the satellite. This is achieved by converting the azimuth and elevation values of both antenna and target into vectors and computing the angle between those two. If this angle exceeds a certain value, which is set in the config file via the variable `MIN_ANGLE_CHANGE_BEFORE_UPDATE`, we will send new target values to the motor controller.

Once the satellite is about the leave the field of view, we could move the antenna to where the satellite is now. However, while we are moving the antenna, the satellite is also moving. This way, we are always lagging behind the satellite. A better way of doing it would be to lead the target (cf. Figure 5.4).

Once we have determined that we should update the antenna's position, we check how

much time Δt has passed since the last update. We are still working on relatively small timescales of a few seconds. So, we can approximate the movement as linear and the angular range of change as constant[8]

$$\theta_{\text{antenna, new}} = \theta_{\text{target, now}} + \dot{\theta}_{\text{target}} \cdot \Delta t. \tag{5.3}$$

We do that for both azimuth and elevation, using the angular rates of change calculated before in Section 5.4.2.

Now it is finally time to talk to the motor controller itself. The communication with the controller happens via a TCP socket. We are sending a 13-byte binary package to the controller `[START, H1, H2, H3, H4, PH, V1, V2, V3, V4, PV, K, END]` [19]. The values are explained in Table 5.1.

Field	**Value**	**Description**
`START`	0x57 ('W')	Start byte
`H1`–`H4`	ASCII 0–9	Azimuth angle encoded as four ASCII digits
`PH`	(ignored)	Azimuth resolution in steps per degree
`V1`–`V4`	ASCII 0–9	Elevation angle encoded as four ASCII digits
`PV`	(ignored)	Elevation resolution in steps per degree
`K`	0x0F / 0x1F / 0x2F	Command byte: stop / status / set
`END`	0x20 (space)	End byte

Table 5.1: We communicate with the motor controller via a 13-byte-long message. The controller replies with a message that is 12 bytes long, since it does not include a command byte.

There are three possible commands that we can send to the controller. The first is `stop`, and it tells the antenna to ignore all previous commands and stop where it is.

The second is `status`. We send this command when we want to know where the antenna currently is. We will then receive a 12-byte package back, which has essentially the same format as the one we are sending, except that it is missing the command byte.

The last one is `set`. This command is used to tell the motor controller where it should point the antenna. We need to encode the azimuth and elevation values as four-digit numbers with leading zeros:

$$\mathrm{H} = \mathrm{PH} \cdot (360 + az) \tag{5.4}$$

$$\mathrm{V} = \mathrm{PV} \cdot (360 + el) \tag{5.5}$$

[8]For targets that are farther away, i.e. in deep space, Δt may be longer. However, the angular rates of those targets will also change less throughout Δt.

`PH` = `PV` = 10 are the resolution in steps per degree, and az, el are the azimuth and elevation values in decimal degrees. In the same way, we can decode the package, which we get back from the motor controller, after having sent the `status` command:

$$az = \mathrm{H1} \cdot 100 + \mathrm{H2} \cdot 10 + \mathrm{H3} + \mathrm{H4}/10 - 360 \tag{5.6}$$

$$el = \mathrm{V1} \cdot 100 + \mathrm{V2} \cdot 10 + \mathrm{V3} + \mathrm{V4}/10 - 360 \tag{5.7}$$

5.5 CelesTrak stack

We went through the whole stack from getting data from the Horizons API to sending the final azimuth and elevation values to the motor controller. Now we are doing it again for CelesTrak data. However, if we compare Figure 5.2 to Figure 5.5, we realise that they only differ in a few steps. So, we only need to focus on the sections that are different for the CelesTrak case.

5.5.1 Update data

As we have done before in the Horizon case, we want to check first if our local data is still valid. So, we check the metadata to see if we need to update. The data that we get from CelesTrak is GP/OMM data (cf. Section 2.2.2). Different from Horizon data, it does not contain position and velocity data at different times. It contains information about the orbit of the satellite itself, including data on how it will probably change over time due to atmospheric drag and other factors [20]. The SGP4 algorithm can use this data to predict where the satellite will be at a certain time. Therefore, we can't "run out of data" as in the Horizon case. However, the error in the prediction will become bigger over time.

CelesTrak itself checks for new data every two hours [4]. Therefore, we consider CelesTrak data as out of date when it is older than two hours. As in the Horizon case, we will not update while tracking is turned on.

The data which we receive from CelesTrak is the GP data for every active satellite in a CSV file. We can save it locally and load it into memory as described in Section 5.2.1. Similarly, we update and save the metadata.

5.5.2 Satellite position

We have our satellite as a `EarthSatellite` object (cf. Section 5.2.1). Now, getting the geocentric and topocentric position of the satellite is trivial. Skyfield is mainly made for satellites in an orbit around the Earth, therefore, the `EarthSatellite` is already the geocentric position. The topocentric position we obtained by subtracting the position of the antenna, relative to Earth's centre, from the geocentric position.

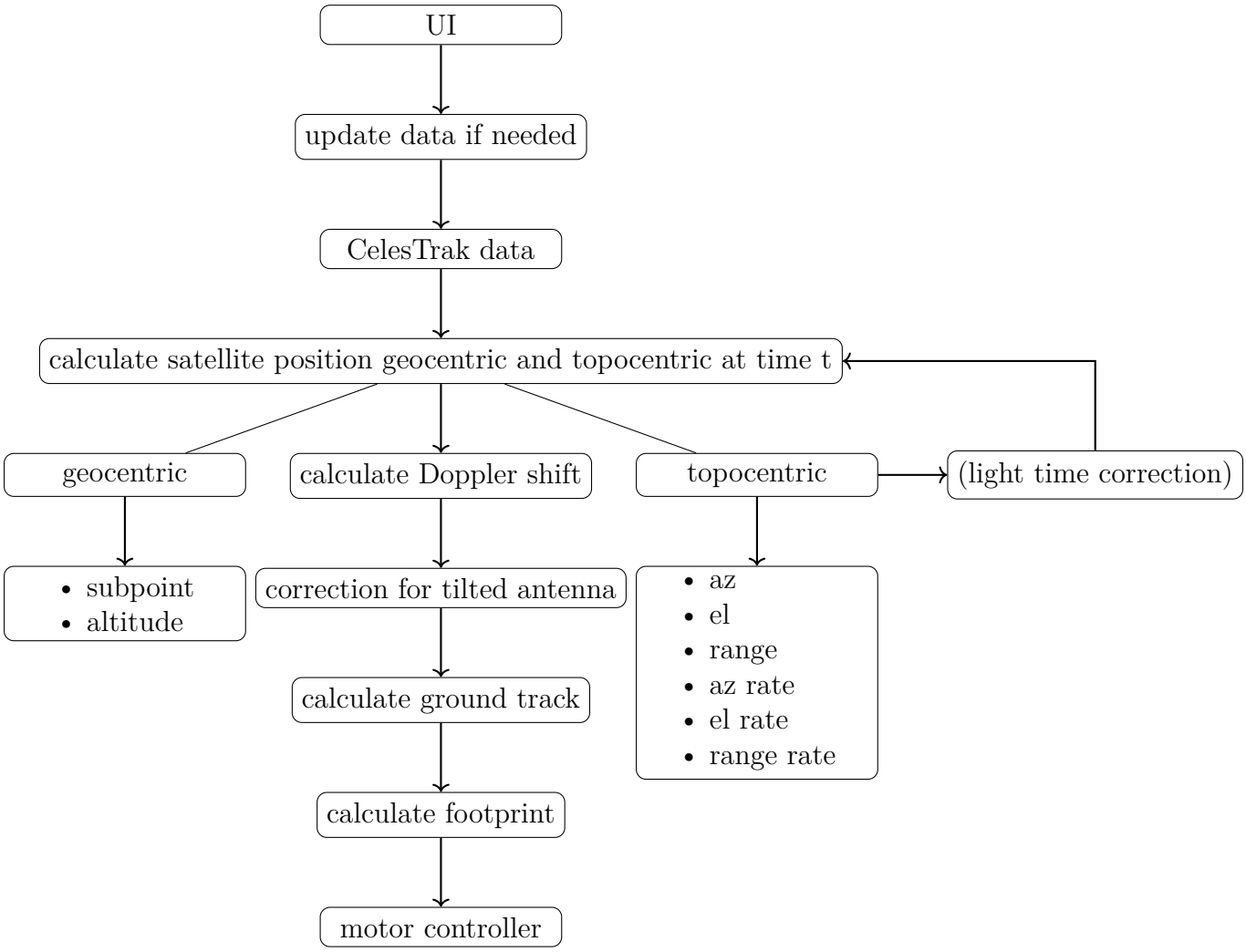


Figure 5.5: After selecting a satellite in the interface, the local data gets updated if needed and all necessary calculations and corrections are performed, before displaying the data on the UI and sending it to the motor controller.

5.5.3 Light time correction

The Skyfield function that we use to get azimuth, elevation, range and the corresponding rate of changes is `frame_latlon_and_rates`. This function does not take light time correction into account. Skyfield has a function that could account for light time correction. However, we have written the code in such a way that we are first evaluating the topocentric position at a given time t and then calculating azimuth and elevation. Skyfield can do it the other way around as well, using the `observe` function. In that case, light time correction would be taken into account. Why did we not do that? The reason is that this only works if you start with a topocentric position that was calculated from orbital information, not position information. Skyfield can do vector math for general position vectors. That means it just remembers that it is a vector pointing from the antenna to the satellite, but only once you specify at what time you want to observe the satellite, it will actually give you the "arrow" in 3D space. The `observ` function makes use of that by evaluating the position at an earlier point in time, the same way we did in Section 5.4.3.

However, this is not possible with a topocentric position that was calculated from just position data, as we did in the Horizons case, as this contains only the information for a single point in time. We needed this portion of the code to work with both the topocentric position in the Horizon case and the CelesTrak case. So, there is no light time correction in the CelesTrak case as well. Our own light time correction could be enabled, as described in Section 5.4.3. However, it is not needed. In the CelesTrak case, we are only dealing with satellites in Earth's orbit. Therefore, the distances are so short that the errors introduced by not correcting for light time are insignificant.

From here on out, everything else is the same as described in Sections 5.4.4 to 5.4.8.

5.6 SPICE stack

5.6.1 Kernels

The tracking mode SPICE differs from the tracking mode List[9], in that it only uses local data and does not download any new data from the internet. This local data is organised on so-called kernels. A kernel is really nothing more than a file that contains some sort of data. There are several types of kernels. We will only mention the ones that are relevant for our case:

- FK – reference frame specifications
- LSK - converts between UTC and SPICE Ephemeris Time (ET)

[9] aka the Horizon or CelesTrak case

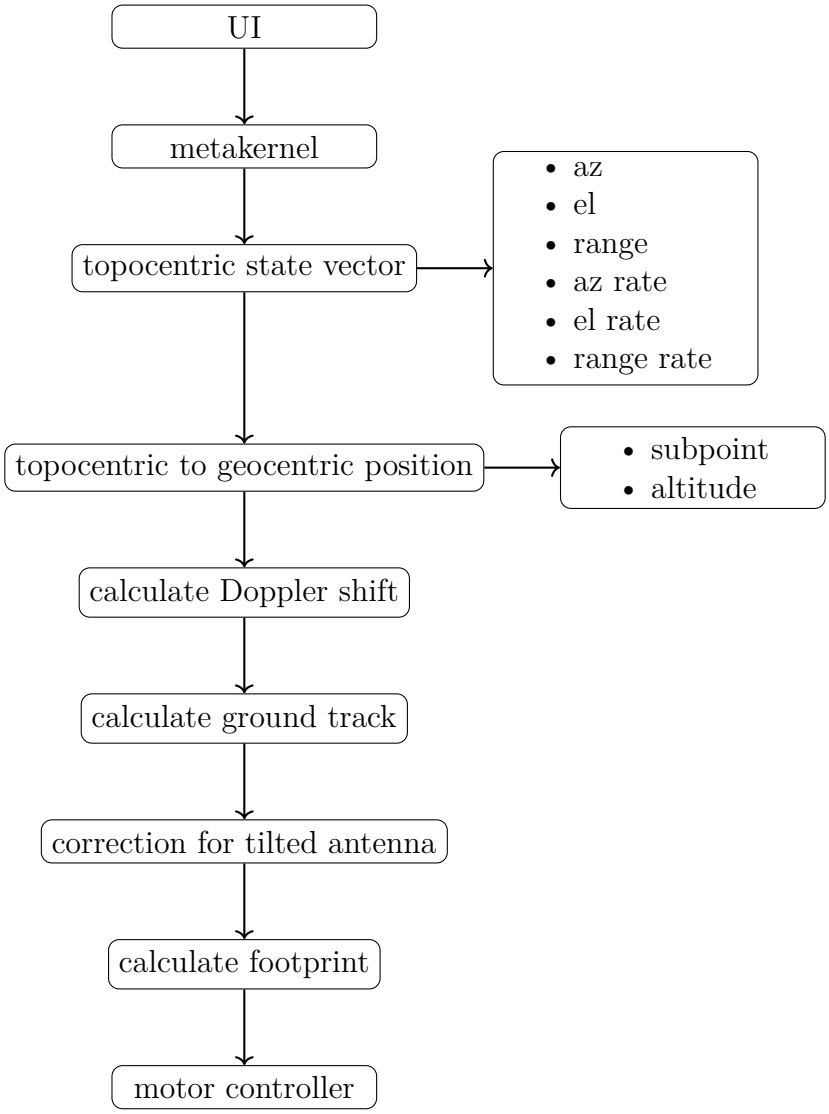


Figure 5.6: After selecting a metakernel and a satellite in the interface, all necessary calculations and corrections are performed, before displaying the data on the UI and sending it to the motor controller.

- PCK - planetary constants (orientation, size and shape)
- SPK - spacecraft and planet ephemeris (trajectory data)
- MK - metakernel

To track a spacecraft, one needs several different kernels, providing different information. Note that it is the job of the user to provide the kernels necessary to track a certain spacecraft. They are published by NASA and can be found on this website: naif.jpl.nasa.gov/pub/naif/. All those kernels need to be loaded. The easiest way to do that is by using a metakernel. Such a metakernel lists all other kernels that should be loaded. The paths must be given relative to the root of the project:

```
\begindata

PATH_VALUES  = ( 'Main/data/Kernels' )

PATH_SYMBOLS = ( 'KERNELS' )

KERNELS_TO_LOAD = (
    '$KERNELS/fk/earth_topo_201023.tf'
    '$KERNELS/fk/mytopo.tf'
    '$KERNELS/lsk/naif0012.tls'
    '$KERNELS/pck/earth_000101_250530_250303.bpc'
    '$KERNELS/pck/pck00011.tpc'
    '$KERNELS/spk/de440.bsp'
    '$KERNELS/spk/earthstns_itrf93_201023.bsp'
    '$KERNELS/spk/juice_orbc_000080_230414_310721_v01.bsp'
)

\begintext
```

The other special kernel is `mytopo.tf`. It is surprisingly difficult to specify a general observer location for SPICE. The only workaround that we could find was defining a new reference frame:

```
\begindata

  FRAME_MYTOPO              = 1234567
  FRAME_1234567_NAME        = 'MYTOPO'
  FRAME_1234567_CLASS       = 4
```

```
FRAME_1234567_CLASS_ID   = 1234567
FRAME_1234567_CENTER     = 399

TKFRAME_1234567_SPEC     = 'ANGLES'
TKFRAME_1234567_RELATIVE = 'IAU_EARTH'
TKFRAME_1234567_ANGLES   = ( -8.550440, -42.60251, 180 )
TKFRAME_1234567_AXES     = (       3,        2,   3 )
TKFRAME_1234567_UNITS    = 'DEGREES'
```

```
\begintext
```

Please note that the geographical coordinates of the observer need to be specified in an unusual format. The longitude in `mytopos` is given as the negative of the longitude, and the latitude is given as the negative of 90° minus the latitude. [21] Should the position of the antenna ever be changed, this needs to be adjusted manually. Adjusting the coordinates in the config file is not sufficient, because the `mytopos` file does not get created by the program automatically.

5.6.2 Calculating parameters

Once the metakernel gets specified by the user via the interface, it gets loaded. Now we can use SpicyPy functions[10] to calculate the topocentric state vector. This position and velocity data is light time corrected. From there, we get the range rate by projecting the velocity onto the line of sight.

Azimuth, elevation and range we find with another SpicyPy function[11]. The angular rates for azimuth and elevation we work out by calculating the topocentric position at different points in time and dividing the difference in angle by the difference in time.

To get the geocentric position, we can use SpicyPy again[12] to convert from the `mytopo` reference frame to the ITRF93[13] reference frame. From there, we get the subpoint and altitude.

Similar to the CelesTrak stack, everything that follows now is the same as described in Sections 5.4.4 to 5.4.8.

[10] georec and spkcpo
[11] recazl
[12] using the function pxform
[13] International Terrestrial Reference Frame

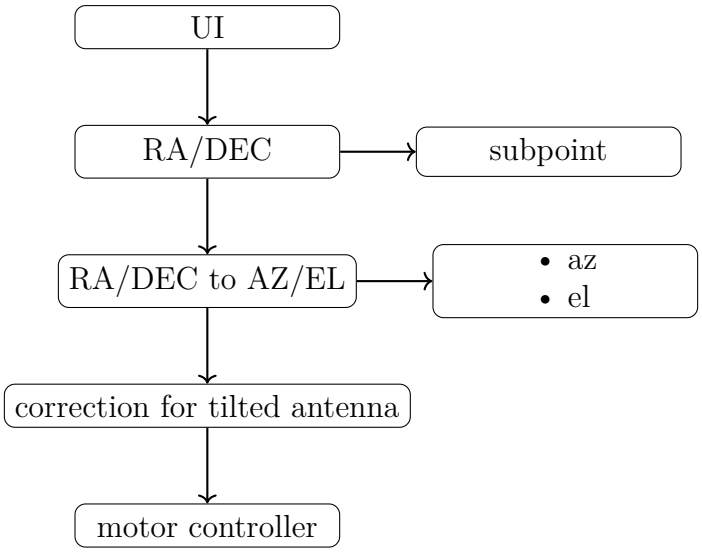


Figure 5.7: In RA/DEC mode, we only need to convert right ascension and declination into azimuth and elevation.

5.7 TLE/OMM stack

The stack for the TLE/OMM tracking mode is basically the same as for CelesTrak (cf. Figure 5.5). The only difference is that the user selects a local OMM[14] or TLE file in the UI, instead of it getting downloaded from the internet.

5.8 RA/DEC stack

The RA/DEC tracking mode is the first tracking mode in which we are not calculating all parameters. Since this tracking mode is for celestial targets like stars, black holes and other intergalactic radio sources, it doesn't make much sense to calculate the range or altitude. Furthermore, these targets don't emit signals on one well-defined frequency. So, calculating the Doppler shift does not make sense either. That only leaves us with the subpoint, for which we can use the Skyfield function `position_of_radec`, as well as azimuth and elevation, for which we can use the equations (3.4) and (3.3). The rest is as described in Sections 5.4.5 to 5.4.8.

5.9 AZ/EL stack

The AZ/EL stack is the simplest one, since this tracking mode is just for the user to orient the antenna in a certain way, for example, for maintenance or calibration. The rest is as described in Sections 5.4.5 to 5.4.8.

[14] file format CSV

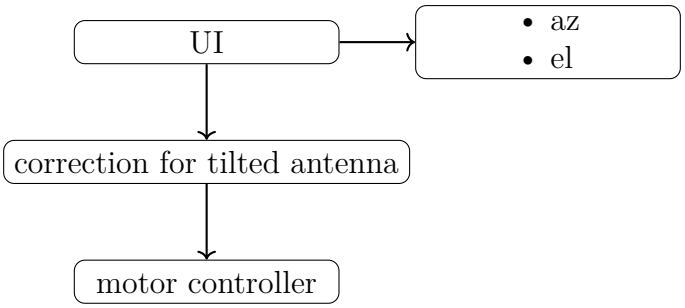


Figure 5.8: In AZ/EL mode, the azimuth and elevation values are not calculated, but directly taken from the UI.

5.10 Find passes

The *Find passes* feature is there to help plan observations. The aim is to find the point in time when the satellite is rising over the minimum elevation threshold (acquisition of signal, AOS) or setting below the minimum elevation threshold (loss of signal, LOS), respectively. This feature is only available in the tracking mode List. We therefore only have to check the Horizons case and the CelesTrak case.

The CelesTrak case is thankfully very easy, since there is a Skyfield function called `find_events` for that. In the Horizons case, we have to search for it ourselves. The reason why we can't use the Skyfield function here is the same as with the light time correction. We can not create a Skyfield object that can look forward and backwards in time, from the data we get from the Horizons API (cf. Section 5.5.3).

The naive approach would be to calculate the elevation angle for every point in time between the start and end time. Since this time window is by default 24 hours long, doing it in 1-minute steps would be way too computationally intensive. Therefore for we start with 10-minute steps. In the Horizons case, we are dealing with passes that are usually several hours long. Therefore, a 10-minute step size should be precise enough.[15] If we cross the threshold, we need to go back by 10 minutes and then go forward again in smaller time steps with a minimum size of 1 minute.

5.10.1 Visualise next pass

The *Visualise Next Pass* button on the UI opens a new window, which shows the next pass of the satellite as a trajectory on a crosshair or bullseye (cf. Figure 4.11). This is a common way of visualising satellite passes.

We do that by calculating the azimuth and elevation for several points between AOS and LOS. Since passes in the CelesTrak case are usually just a few minutes long, doing it in

[15]If it turns out in the future that there are performance issues on less powerful machines, this step size could also be increased. But there is no such setting in the config file yet.

1-minute steps would not be precise enough. Therefore, we are doing it in 1-seconds steps. However, this is computationally expensive, especially when we do it for the Horizons case, where one pass can last several hours. To solve this, we dynamically increase our step size until the total number of steps is below 500. More than 500 steps give us no visual improvement to the plot, so it is a good maximum value.

5.10.2 Known bugs

We encountered a bug in the find passes feature once, however, we were never able to replicate it. Therefore, it was never fixed. The bug occurred while the satellite was already above the minimum threshold. In such a case, the algorithm treats the current time as an AOS event. The LOS event should have occurred only a few minutes later, but the search algorithm did not detect it correctly. This resulted in the next event, which was an AOS event, being displayed wrongly as an LOS event. The same happened to all subsequent events.

Running the same algorithm with the same satellite, about half a minute later, resulted in everything being detected correctly. Therefore, this bug seems to be a rare edge case. Some minor changes have been made to the algorithm since then, and the bug was never observed again. This could mean that the bug is fixed, however, we consider it more likely that the conditions for the bug just have never been encountered again.

Fixing and testing could require a significant amount of time due to the inability to reliably reproduce the issue. Given its extreme rarity and the fact that this feature is not mission-critical for the operation of the antenna, we have decided to leave the code as it is.

5.11 Validation

The values for the current position of the antenna that we get from the motor controller have one significant digit after the decimal point. Therefore, we consider all azimuth and elevation values with an error less than or equal to 0.1° as acceptable.

We tested our calculations against several sources that are known to be reliable. The first is the Horizons API. As mentioned before in Section 5.4.2, the Horizons API can also give us the azimuth and elevation angles directly, as well as all other parameters that we calculate. For the Horizons case and the tracking mode SPICE, the azimuth and elevation values agree with the values from the Horizons API within 0.1°.

For the range, we get a difference of about 4000 km when tracing JUICE. This sounds like a lot. However, if we normalise it by dividing it by the range, we get a relative error of 0.002%, which we consider acceptable. If we track Jupiter, whose position is much better known, the difference is only 90 km. So, this difference might be explained by a

slight difference in the data used. For the range rate, we get a difference of about 0.02 km/s, which would be a relative error of $9 \cdot 10^{-9}$, also acceptable.

In the CelesTrak case, we are not comparing against the Horizons API. Instead, we are comparing it against the open-source software Gpredict. [1] It is a popular tool for tracking satellites in Earth orbit and also uses TLE data and the SGP4 algorithm. Our azimuth and elevation values agree with Gpredict within 0.1°.

We also use Gpredict to validate our Doppler shift calculations. Our values differ from Gpredict by 1 Hz. Considering that the antenna is expected to operate in the 1 GHz to 10 GHz region, this is acceptable.

To validate our implementation of the conversion from right ascension and declination to azimuth and elevation, we checked it against several online calculators. [22–24] They all agree with in 0.1° with our results.

Chapter 6

Conclusion

The aim of this thesis is the development and testing of a control software for a radio telescope. The software that we developed is capable of tracking the position of Earth satellites, deep space spacecraft and celestial targets and translating it into azimuth and elevation commands for the motor controller of the antenna. It can automatically collect the latest data from the internet, using CelesTrak and the Horizons System as sources, or it can use locally stored data. It supports GP data in TLE and OMM format, as well as SPICE kernels.

It calculates and displays important information about the satellite, like its Doppler shift. It can also calculate and display the next passes of the satellite over the antenna. Finally, it has a map that displays the subpoint of the satellite, together with its footprint and ground track.

All calculations have been tested against reliable sources and found to be in good agreement.

Declaration on the use of AI tools

We made extensive use of AI tools for this project, for some parts more than for others. We therefore want to break it down into the three categories: research, software development, and writing, and discuss them separately.

Research

When we considered this to be the topic of our bachelor thesis, we used ChatGPT [25] to gain a quick overview of the available software packages. This was later supplemented by more traditional research.

We also used ChatGPT to help us understand some initial concepts, such as the different coordinate systems. We must emphasise that all the information provided by generative AI, which was used directly or indirectly in this thesis, was manually checked and confirmed to be correct. None of the listed references has been taken directly from generative AI.

Software development

The actual software development was where we made the most use of AI. Tools such as ChatGPT, Claude AI [26], and GitHub Copilot [27] have proven to be invaluable. It helped us a lot working with software packages that we had never used before. Important to note is that we approached AI-generated code the same way we approach any code we get from the internet: Trust but verify. We made sure that we understood every line of the code before we implemented it. We also tested almost all the code snippets separately before implementing them into the main codebase. The only exceptions were snippets that were so simple and short that we could verify their correctness just by looking at them. We used the same procedure for any code that we found online.

Writing

For writing, we used Grammarly AI writing assistance. [28] This tool mainly corrects spelling and grammar mistakes and suggests corrections to punctuation. Furthermore, it suggests better formulations for improved clarity. What it does not do is generate full text (not even sentences) based on prompts or existing text.

Generative AI as ChatGPT, was used on rare occasions when we were not satisfied with the formulation of a sentence and asked the AI for variations on that sentence. It was also used to find a good formulation in English from a description of what we wanted to say. Therefore, some of the sentences in this thesis could technically be understood as generated by AI. However, literally everything was checked manually. Hence, generative

AI was used more in a way similar to how one would ask a colleague for help with a difficult sentence or formulation.

We also used the DeepL AI translator [29] on rare occasions. None of the illustrations included have been generated by AI.

Bibliography

[1]A. Csete, *Gpredict*, version 2.3.37, (2017) `https://oz9aec.dk/gpredict/` (visited on 05/22/2025).

[2]C. Acton, N. Bachman, B. Semenov, and E. Wright, "A look towards the future in the handling of space science mission geometry", Planetary and Space Science, Enabling Open and Interoperable Access to Planetary Science and Heliophysics Databases and Tools **150**, 9–12 (2018).

[3]P. Jansen, *TIOBE Index*, (May 2025) `https://www.tiobe.com/tiobe-index/` (visited on 04/18/2025).

[4]T. S. Kelso, *CelesTrak: A New Way to Obtain GP Data*, (Sept. 14, 2024) `https://celestrak.org/NORAD/documentation/gp-data-formats.php` (visited on 04/18/2025).

[5]D. Vallado and P. Crawford, "SGP4 Orbit Determination", in AIAA/AAS Astrodynamics Specialist Conference and Exhibit (Aug. 2008).

[6]D. A. Vallado, P. Crawford, R. Hujsak, and T. Kelso, "Revisiting Spacetrack Report #3: Rev 3", AIAA, Tech. Rep. (2007).

[7]B. Rhodes, "Skyfield: High precision research-grade positions for planets and Earth satellites generator", Astrophysics Source Code Library, ascl:1907.024 (2019).

[8]C. H. Acton, "Ancillary data services of NASA's Navigation and Ancillary Information Facility", Planetary and Space Science, Planetary data system **44**, 65–70 (1996).

[9]NASA, *List of missions using SPICE*, (July 8, 2024) `https://naif.jpl.nasa.gov/naif/SPICE_Users.pdf` (visited on 05/07/2025).

[10]A. M. Annex, B. Pearson, B. Seignovert, B. T. Carcich, H. Eichhorn, J. A. Mapel, J. L. F. v. Forstner, J. McAuliffe, J. D. d. Rio, K. L. Berry, K.-M. Aye, M. Stefko, M. d. Val-Borro, S. Kulumani, and S.-y. Murakami, "SpiceyPy: a Pythonic Wrapper for the SPICE Toolkit", Journal of Open Source Software **5**, 2050 (2020).

[11]NASA, *Horizons System*, (Sept. 6, 2022) `https://ssd.jpl.nasa.gov/horizons/app.html#/` (visited on 05/07/2025).

[12]falt, QtForPython, and Qt Project, *PySide6, Python bindings for the Qt cross-platform application and UI framework*, version 6.9.0, (2025) https://pypi.org/project/PySide6/ (visited on 05/22/2025).

[13]J. O. Bennett, M. Donahue, N. Schneider, M. Voit, and H. Lesch, *Astronomie, Die kosmische Perspektive*, 5th ed., Pearson Studium (Pearson Studium, München [u.a.], 2011), 1162 pp.

[14]Tfr000, *RA and DEC on celestial sphere*, (June 15, 2012) https://commons.wikimedia.org/wiki/File:Ra_and_dec_on_celestial_sphere.png (visited on 05/07/2025).

[15]NASA, *Chapter 2: Reference Systems - NASA Science*, en-US, (2025) https://science.nasa.gov/learn/basics-of-space-flight/chapter2-2/ (visited on 02/18/2025).

[16]B. Rhodes, *Skyfield Documentation - Positions*, (2025) https://rhodesmill.org/skyfield/positions (visited on 05/07/2025).

[17]T. S. Kelso, *CelesTrak: Current Supplemental GP Element Sets*, (2025) https://celestrak.org/NORAD/elements/supplemental/ (visited on 05/20/2025).

[18]NASA/Goddard Space Flight Center, *Whole world - land and oceans*, (Feb. 11, 2002) https://commons.wikimedia.org/wiki/File:Whole_world_-_land_and_oceans_12000.jpg (visited on 05/22/2025).

[19]N. H. Ryeng, *Description of SPID Rot1Prog and Rot2Prog protocol.* (Jan. 30, 2011) https://ryeng.name/blog/3 (visited on 03/31/2025).

[20]T. S. Kelso, *CelesTrak: FAQs: Two-Line Element Set Format*, (1998) https://celestrak.org/columns/v04n03/ (visited on 02/18/2025).

[21]G. Miller, *SPICE Alt Az example*, (Mar. 2020) https://astronomy.stackexchange.com/q/33095/67773 (visited on 05/06/2025).

[22]G. Miller, *Convert ra/dec to alt/az*, (2021) https://astrogreg.com/convert_ra_dec_to_alt_az.html (visited on 05/07/2025).

[23]NOAA, *Solar position calculator*, (May 22, 2025) https://gml.noaa.gov/grad/solcalc/azel.html (visited on 05/07/2025).

[24]S. R. Schmitt, *Celestial to horizon coordinates calculator*, (2004) http://www.convertalot.com/celestial_horizon_co-ordinates_calculator.html (visited on 05/07/2025).

AI Tools

[25]OpenAI, *ChatGPT-4 and ChatGPT-3.5 (versions February to May 2025)*, [Large language model], 2025.

[26]Anthropic, *Claude 3.7 Sonnet (versions February to May 2025)*, [Large language model], 2025.

[27]OpenAI, *GitHub Copilot (versions February to May 2025)*, [Large language model], 2025.

[28]Grammarly, inc., *Grammarly (versions April to May 2025)*, [AI writing assistance], 2025.

[29]DeepL SE, *DeepL (versions February to May 2025)*, [AI translator], 2025.