

Umlaufbahnen von Satelliten: messen, simulieren und programmieren



**Joshua Dreier – Kantonsschule Heerbrugg – November 2021
Mentor: Benedikt Götz – Klasse 4P**

Neulandstrasse 33, St. Margrethen 9430

1 Kurzzusammenfassung

Diese Arbeit mit dem Thema «Umlaufbahnen von Satelliten» behandelt die Frage, inwiefern es möglich ist mithilfe einer Messung den nächsten Überflug des gemessenen Satelliten vorherzusagen. Für den Zweck eine solche Umlaufbahnen zu simulieren, wurde eine Applikation namens «Tarvos» programmiert, welche fünf Tage vorausrechnet. Bei den gemessenen Satelliten wurde sich auf die Internationale Raumstation (ISS) fokussiert, wobei andere Satelliten auch Erwähnung finden. Die Arbeit beschreibt wie man einen gemessenen Satelliten in eine Position in verschiedenen Koordinatensystemen umwandeln kann und wie man vorgehen muss um die Drehung eines solchen Satelliten um die Erde zu simulieren. Diese Arbeit wurde von Joshua Dreier aus der Klasse 4P der Kantonsschule Heerbrugg verfasst und von Benedikt Götz mentoriert.

2 Inhalt

1	Kurzzusammenfassung.....	2
2	Inhalt.....	2
3	Einleitung	4
3.1	Grundidee	4
3.2	Einführung in die Orbitalmechanik	4
3.2.1	Inklination	5
3.2.2	Höhe	6
3.3	Astronomische Koordinatensysteme	8
3.3.1	Das horizontale Koordinatensystem.....	8
3.3.2	Das äquatoriale Koordinatensystem.....	9
4	Material und Messmethoden	10
4.1	Sammeln der Daten	10
4.1.1	Auswahl der Überflüge	10
4.1.2	Equipment	11
4.1.3	Standort	12
4.1.4	Zeit	13
4.1.5	Prozess der Messung	13
4.2	Daten Auswertung.....	14
4.2.1	Plate-Solving	14
4.2.2	Berechnen relevanter Werte mittels Excel-Tabelle	17

Inhalt

4.2.3 Beispiele	21
5 Entwicklung und Funktionsweise des Programms.....	26
5.1 Entwicklung.....	26
5.2 Bedienung.....	28
5.2.1 Programmoberfläche	28
5.2.2 Vorgehensweise.....	29
5.2.3 Datei auswerten	31
5.3 Funktionsweise.....	33
5.3.1 Grundstruktur	33
5.3.2 Koordinatensysteme	34
5.3.3 Erde	34
5.3.4 Ort.....	36
5.3.5 Satellit.....	37
5.3.6 Sichtbarkeits-Rechner	40
6 Ergebnisse	41
6.1 Ergebnisse des Verfahrens	41
6.2 Vergleich mit Literaturwerten.....	44
7 Diskussion	45
7.1 Ungenauigkeiten in der Simulation	45
7.2 äussere Ungenauigkeiten.....	46
7.3 Verbesserungsmöglichkeiten.....	47
8 Literaturverzeichnis.....	47
9 Abbildungsverzeichnis:	49
10 Tabellenverzeichnis.....	50
11 Dank	51
12 Selbstständigkeitserklärung.....	51
13 Anhang	52
13.1 Lexikon.....	52
13.2 Ungezielte Beobachtungen.....	54

13.3	Erläuterungen	55
13.3.1	Erdfigur	55
13.3.2	Literaturvergleiche weiterer Überflüge	56
13.4	Ausdruck des Programm Codes	57

3 Einleitung

3.1 Grundidee

Als ich mich freizeitlich bedingt und vom Freifach Astronomie inspiriert mit Satelliten befasst habe, war eine wiederkehrende Frage «Wie viel davon könnte man selbst mit einer Messung bestimmen?». Deswegen nahm ich mir für die Maturarbeit vor, mithilfe von einer Messung und mit möglichst wenigen äusseren Angaben, den nächsten Überflug eines Satelliten vorherzusagen.

3.2 Einführung in die Orbitalmechanik

Die Orbitalmechanik oder auch Raumfahrtmechanik ist ein Teilgebiet der Physik, welches sich mit der Bewegung von Satelliten befasst. Grundlegend kann man sagen, dass die Orbitalmechanik auf das Newtonsche Gravitationsgesetz¹ (siehe rechts) und den Gesetzen der Kreisbewegungen aufbaut. In Worten ausgedrückt besagt dieses Gesetz, dass die Anziehungskraft zwischen zwei Körpern sich proportional zum Produkt ihrer Masse und indirekt proportional zum Quadrat der Distanz der Schwerpunkte zwischen den Körpern verhält. Sie nimmt also linear zu, je grösser die Massen der Körper sind und quadratisch ab, je weiter sie voneinander entfernt sind.

Newton'sches Gravitationsgesetz:

$$F_G = G * \frac{m_1 * m_2}{r^2}$$

F_G = Anziehungskraft [N]

G = Gravitationskonstante, beträgt

$$6.67430(15) * 10^{-11} \frac{m^3}{kg*s^2} [17]$$

m_1, m_2 = Masse der betrachteten Körper [kg]

r = Distanz zwischen den
Schwerpunkten [m]

¹ Siehe Quelle [17]

Im Kontext der Erdsatelliten kann man annehmen, dass die Erde um mehrere Größenordnungen massiver ist als ein gegebener Satellit. Damit kann man die Anziehungskraft auf einen Satelliten unabhängig von dessen Masse beschreiben:

$$m_1 = m_{Sat}, m_2 = M_{\oplus} \rightarrow m_2 \gg m_1 \rightarrow F_G = \frac{\mu}{r^2}$$

$$\mu = G * M_{\oplus} \text{ (Erdmasse).}$$

Der Themenbereich «Kreisbewegungen» im Teilgebiet der Dynamik beschreibt eine Kreisbewegung als Resultat einer Konstanten, zum gleichen Punkt zeigende Kraft (Zentripetalkraft, in Abbildung 1 \vec{F}_Z) und eine senkrecht dazu stehende Geschwindigkeit (in Abbildung 1 \vec{v}). Ein Satellit erfährt eine solche Kraft in der Form der Gravitationskraft. Dies führt dazu, dass der Mittelpunkt der Umlaufbahn (auch Orbit genannt) eines Satelliten immer im Mittelpunkt der Erde bzw. bei elliptischen Umlaufbahnen in einem der Brennpunkte liegt. Weiterhin weiss man dadurch, dass die Umlaufbahn eines Satelliten in einer Ebene liegen muss.

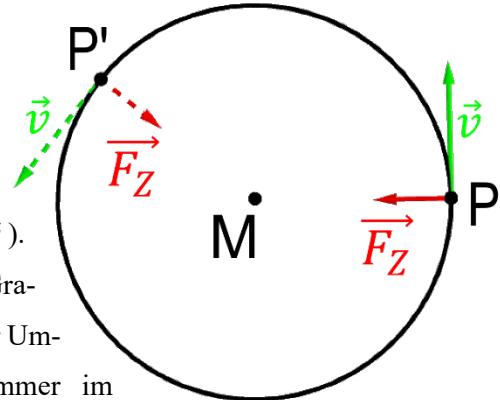


Abbildung 1: Kreisbewegung

Im Allgemeinen benötigt es sechs sogenannte Bahnelemente, um die Bahn eines beliebigen Satelliten zu beschreiben. Im Rahmen dieser Matura Arbeit werden jedoch elliptische Bahnen vernachlässigt und es wird nur auf Kreisbahnen fokussiert. Dadurch, dass die Simulation mit dem selbstgeschriebenen Programm «Tarovs» durchgeführt wird, müssen von ausserhalb der Simulation nur zwei Parameter angegeben werden: Die Höhe h , Die Inklination i .

3.2.1 Inklination

Die Inklination i oder auch Bahnneigung gibt den Winkel zwischen der Bahnebene und der Äquatorialebene an. Die Äquatorialebene ist die Ebene, in welcher der Erdäquator liegt und die Bahn-ebene die oben erwähnte Ebene in der die Umlaufbahn eines Satelliten liegt. Dies diktiert zudem, von welchen Längengraden der Satellit sichtbar ist. So ist z.B. ein erd-naher Satellit mit Inklination 0° nie vom Breitengrad 47° Nord aus sichtbar, jedoch ist es bei einem weiter entfernten Satelliten möglich. Die ISS besitzt eine Inklination von 51.664 und ist somit für alle Breitengrade bis 51° Nord und Süd und ein paar wenige Breitengrad darüber/darunter regelmässig sichtbar.

3.2.2 Höhe

Wie in Abbildung 2 teilweise zu sehen ist, gibt die Höhe h die Distanz zwischen der Meereshöhe und der Umlaufbahn des Satelliten an. Da die Erde vereinfachend⁵ als Kugel angesehen werden kann, bleibt diese Höhe bei Kreisbahnen konstant. Grob kann man die Höhe von Satelliten in 4 Kategorien einteilen⁶:

Name	Höhe
LEO ²	200-2'000 km (Genutzt: 200-1'300km)
MEO ³	2'000-35'000 (Genutzt 3'000-35'000)
GEO ⁴	35'786 km
SSO	mehr als 35'786 km
HEO	stark elliptisch

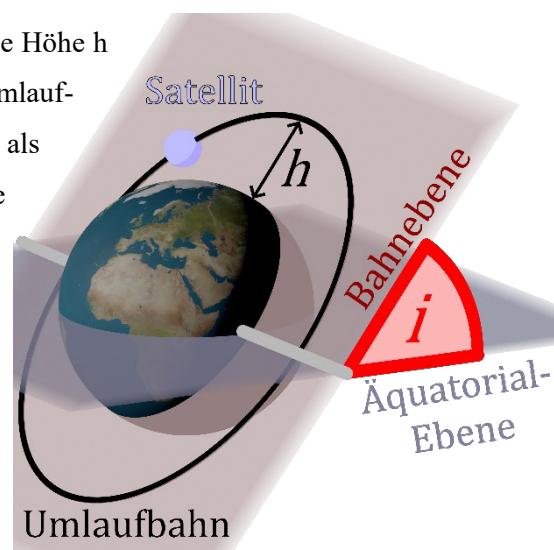


Abbildung 2: relevante Bahnelemente

LEO: steht für **Low Earth Orbit** (erdnahe Umlaufbahn). Satelliten in diesem Bereich haben aufgrund dieser Erdnähe eine Periode von unter 130 Minuten. Sie sind von der Erde aus nur kurz, aber sehr hell sichtbar. Umlaufbahnen im LEO sind am kostengünstigsten zu erreichen und aufgrund dessen befinden sich die meisten aktiven Satelliten im LEO. Beispiele für gegenwärtige (November 2021) LEO-Satelliten sind:

- Die Internationale Raumstation (ISS)⁷. Konstant bemannt seit November 2000. (für mehr Informationen siehe Kapitel 4.1.1 «Auswahl der Überflüge» auf Seite 10.)
- Die Chinesische Raumstation⁸ Zhōngguó Kōngjiānzhàn: Aktiv seit April 2019.
- SpaceX' «Starlink» Satelliteninternetdienst (geplant bis 2027: 11'927 Satelliten) und ähnliche Angebote.
- Weltraumteleskope (Hubble, Spitzer, TESS, Newton, Fermi, Cheops, Gaia, etc.)
- Erkundungs- und Wettersatelliten

² Siehe Quelle [14]

³ Siehe Quelle [18]

⁴ Siehe Quelle [1]

⁵ Siehe Anhang: Kapitel 13.3.1 «Erdfigur» auf Seite 55

⁶ Siehe Quelle [18]

⁷ Siehe Quelle [19]

⁸ Siehe Quelle [20]

Einleitung

- Kommunikationssatellitensysteme

LEO-Satelliten bilden den eindeutigen Schwerpunkt dieser Arbeit, da sie mit Standard Kamera-Equipment als einzige gut messbar sind.

MEO: steht für **Medium Earth Orbit**. Die Lücke an besetzten Umlaufbahnen zwischen LEO und MEO von 1'200 Kilometern bis 3'000 Kilometer lässt sich durch den Van-Allen Strahlungs-Gürtel erklären, welcher vom Erdmagnetfeld ausgelöst wird und in eben diesem Bereich eine hohe Strahlenbelastung auslöst. Die Umlaufszeit reicht von etwa 4 Stunden bis 12 Stunden und der Hauptnutzen für solche Umlaufbahnen stellen Kommunikationssatellitensysteme und Navigationssatellitensysteme (GPS, Galileo, GLONASS) dar.

GSO: steht für **Geostationary Orbit** (Geostationäre Umlaufbahn). Diese Bezeichnung beschreibt alle kreisförmigen Umlaufbahnen mit einer Inklination von 0° und einer Höhe von 35'786 Kilometern. Es wird als geostationär bezeichnet, weil die Periode somit die Dauer eines siderischen Tages entspricht. (für mehr Informationen siehe Kapitel 5.3.3 «Erde» auf Seite 34) Das bedeutet somit, dass GSO-Satelliten von der Erde aus gesehen am gleichen Ort im horizontalen Koordinatensystem stehen zu bleiben scheinen. Solche «schweben» stets über demselben Ort, welcher auf dem Äquator liegt. Die häufigsten Verwendungen sind Fernseh-Satelliten, Kommunikationssatelliten und meteorologische Satelliten. Diese Satelliten können jedoch für keine Breitengrade über 82° Nord oder unter 82° Süd Signale empfangen oder senden.

Im Falle eines **IGSO** (Inclined Geostationary Orbit = geneigte geostationäre Umlaufbahn) erfüllt der Satellit nicht die Bedingung, dass die Bahninklination 0° beträgt. Somit scheint die Bodenspur eine von der Inklination abhängige grosse 8-förmige Schlaufe zu bilden. Siehe Abbildung 3.



Abbildung 4: Mondaufnahme des 18.09.2021 zwischen Satellitenbeobachtungen

SSO: steht für **Supersynchronous Orbit** (übersynchrone Umlaufbahn). Dies bezeichnet alle Satelliten, welche weiter entfernt sind als die Höhe der GSO. Er beinhaltet somit alle Satelliten, welche länger als einen Tag für einen Umlauf benötigen, als Beispiel kann man den Mond nennen.

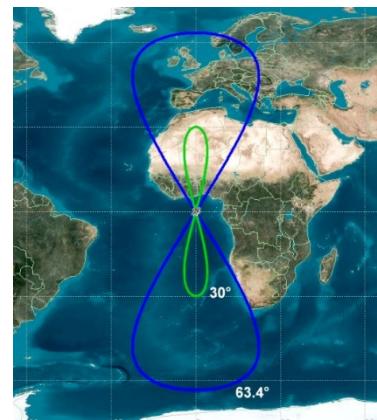


Abbildung 3: Bodenspur zweier IGSO-Satelliten [21]

HEO: steht für **Highly Elliptical Orbit** (hochelliptische Umlaufbahn). Eine solche Umlaufbahn zeichnet sich dadurch aus, dass der erdnächste und der erdfernste Punkt sehr stark unterschiedliche Höhen haben. Dies wird auch Exzentrizität genannt haben (z.B. 200km Erdnähe, 15'000km Erdferne). Beispiele beinhalten Strahlung- und Forschungssatelliten, welche den Van-Allen Strahlungs-Gürtel passieren sollten, aber auch sogenannte «Molnija»-Orbits, welche in Dreiergruppen für einen Pol die Funkverbindung mit den höheren Breitengraden (über 82° Nord, unter 82° Süd) garantieren.

3.3 Astronomische Koordinatensysteme

In der Astronomie kommen verschiedene Koordinatensysteme zum Einsatz. In den meisten Fällen sind es Kugelkoordinaten, welche durch zwei Winkel beschrieben werden. Die, welche für diese Arbeit von Bedeutung sind das äquatoriale und das horizontale Koordinatensystem.

3.3.1 Das horizontale Koordinatensystem

Das horizontale Koordinatensystem, zählt zu der Gruppe der topozentrischen Koordinatensystemen⁹. Es ist womöglich das am einfachsten zu verstehende, da es die Position eines Punktes am Nachthimmel des Beobachters beschreibt. Es ist in zwei Winkel aufgeteilt. Der vertikale davon ist der **Elevation h**, welcher den Winkel zum Horizont bezeichnet und somit zwischen 0° und 90° reichen kann. (Solange die Position über dem Horizont ist. Negative Werte bedeuten eine Position, welche sich unter der horizontalen Ebene befindet) Die horizontale Komponente ist der **Azimut a**, welcher vom Norden über den Osten herumgeht und somit zwischen 0° und 360° liegen kann. (Vom Norden über den Osten bedeutet: wenn man von Oben auf den Boden schaut im Uhrzeigersinn. Wenn man vom Boden in den Himmel schaut im Gegenuhrzeigersinn).

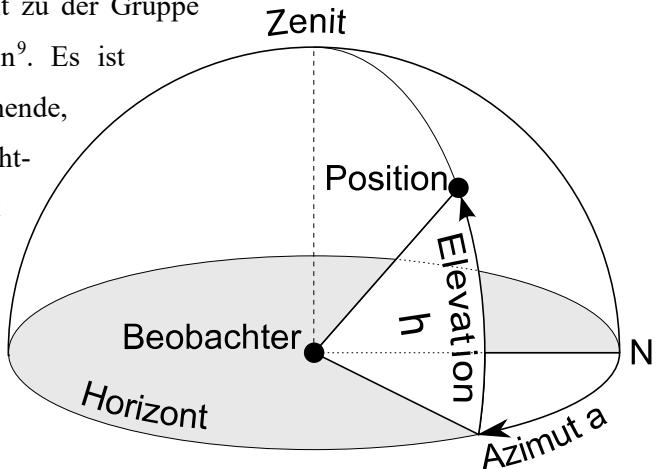


Abbildung 5: Darstellung horizontales Koordinatensystem mit Elevation h und Azimut a. ursprüngliches Bild: [15], Bearbeitet: selbst

⁹ Siehe Quelle [10]

3.3.2 Das äquatoriale Koordinatensystem

Das äquatoriale Koordinatensystem besteht ebenfalls aus zwei Winkelgrößen. In diesem Falle ist es die **Deklination δ** , welche die vertikale Koordinate ist und die **Rektaszension α** , Beziehungsweise der **Stundenwinkel τ** , welcher die horizontale Komponente darstellt. Das liegt daran, dass, das äquatoriale Koordinatensystem weiter in zwei Unterarten eingeteilt wird.

So kann sich das Äquatoriale Koordinatensystem wie das horizontale Koordinatensystem vom Zentrum der Erde aus vorgestellt werden, aber anstatt der Nordrichtung als Nullpunkt wählt man den Frühlingspunkt (also den Schnittpunkt zwischen Ekliptik und Äquatorebene). Die Deklination ist somit analog zur Elevation, die Rektaszension (**rotierendes äquatoriales Koordinatensystem**, da es sich von einem Beobachter auf der Erdoberfläche aus scheint zu drehen) analog zum Azimut (aber wie erwähnt mit anderem Nullpunkt). Dazu gibt es den Stundenwinkel, welcher sich mit der Erde dreht und den Winkel von dem in den Himmel projizierten Null-Meridian zu einer Position angibt. Dies ist das **ruhende äquatoriale Koordinatensystem**, da es sich von einem Beobachter auf der Erdoberfläche aus scheint zu drehen

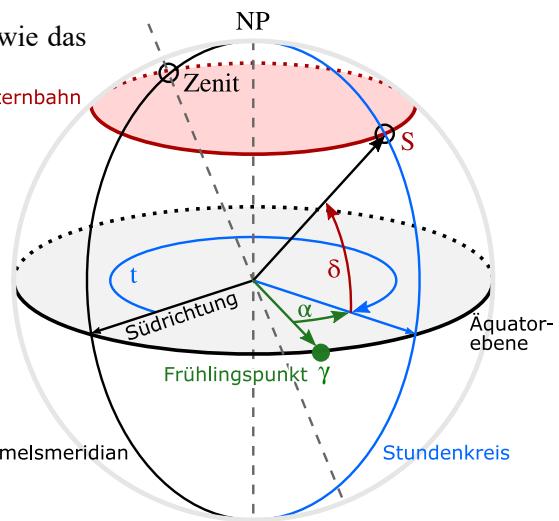


Abbildung 6: das Äquatoriale Koordinatensystem

4 Material und Messmethoden

4.1 Sammeln der Daten

4.1.1 Auswahl der Überflüge

Um die Umlaufbahn eines ausgewählten Satelliten bestimmen zu können, lohnt es sich für die Voraussage der Beobachtungstermine Onlineplattformen zu verwenden. Erfahrungsgemäß entsprechen deren zeitlichen und richtungsmässigen Voraussagen mit grosser Präzision der Zeit und Richtung der tatsächlichen Überflüge. Den Dienst, welchen ich für diesen Zweck gewählt habe, ist «heavens-above.com».

Um möglichst einfach nützliche Daten zu erhalten, bietet es sich an, möglichst helle Satelliten zu messen. Dabei stellt sich bei der Suche nach Satelliten ein klarer Favorit heraus: die Internationale Raumstation (ISS). Die scheinbare Helligkeit der ISS lässt sich ungefähr mit der von Venus vergleichen¹⁰.

Dies liegt nicht nur an der schieren Grösse der Raumstation (Grundriss: 109m × 73m)¹¹, sondern auch an der relativen

Nähe (Stand November 2021: zwischen 418 km und 424 km)¹² zur Erde. Dadurch lässt sich die ISS auch bei suboptimalen Sichtungsverhältnissen auf länger belichteten Fotos deutlich abbilden. Zusätzlich bedeutet der Winkel zwischen der Äquatorialebene und der Umlaufbahn der ISS (51.6°), dass sie regelmässig über allen Orten mit einem Breitengrad kleiner als 51.6° sichtbar ist.



Abbildung 7: ISS am 4. Okt. 2018 aus Sicht der Soyuz-Kapsel der «Expedition 56» [5]

Andere Satelliten sind auch möglich zu beobachten, benötigen aber wesentlich bessere Wetter- und Lichtverschmutzungs-Verhältnisse und sind zudem nicht immer von blossem Auge sichtbar.¹³ Es ist auch möglich, ohne konkretes Ziel beobachten zu gehen, da man zu den richtigen Stunden auch von Auge überfliegende Satelliten finden kann, welche heutzutage in jeder klaren Nacht zu sehen sind.

¹⁰ Siehe Quelle [8]

¹¹ Siehe Quelle [5]

¹² Siehe Quelle [3]

¹³ Für Beispiele siehe Anhang: Kapitel 13.2 «Ungezielte Beobachtungen» auf Seite 54

4.1.2 Equipment

Als Aufnahmegerät sollte man eine Kamera mit möglichst hochauflösendem und sensitivem Sensor nutzen.

Aufgrund der langen Belichtungszeit ist ein Stativ und ein Fernauslöser zu empfehlen. Weitwinkel-Objektive eignen sich gut, da man dadurch weniger oft nachführen muss und mehr identifizierbare Sterne auf dem Bild hat. Teleskope sind für LEO¹⁴-Satelliten nur mit spezieller Nachführungssoftware zu verwenden.



Abbildung 8: Kamera (Canon EOS 250D) mit Objektiv (SIGMA 20mm F1,4 DG HSM | Art)

In meinem Fall benutzte ich als Kamera eine **Canon EOS 250D** mit dem von der Schule bereitgestellten **SIGMA 20mm F1,4 DG HSM | Art** als Objektiv und dem **Hama Star 62** als Stativ. Da diese Kamera Bluetooth-Verbindungen mit Smartphones unterstützt, habe ich meines (Samsung Galaxy A52 5G) dafür nutzen können.



Abbildung 9: Stativ eingefahren, 60 cm (Hama Star 62)



Abbildung 10: Stativ ausgefahren, 160 cm (Hama Star 62)

¹⁴ Siehe Anhang: Lexikon; Seite 52

4.1.3 Standort

Für die Standortauswahl sind unterschiedliche Faktoren für unterschiedliche Satelliten zu beachten:

4.1.3.1 Helligkeit:

Hellere Satelliten, wie die ISS, können bereits aus vorstädtischen Gärten (in meinem Fall: 47°26'52.2"N 9°38'24.1"E, Neulandstrasse 33, 9430 St. Margrethen, mein Garten) beobachtet werden, da der Satellit auch durch die Lichtverschmutzung hindurch gemessen werden kann. Sobald man Satelliten bis hin zu einer Magnitude von etwa 4 beobachten will, sollte man mindestens aus den Dorfgrenzen (in meinem Fall: 47°26'49.4"N 9°37'42.2"E, Jägerstrasse, 9430 St. Margrethen, Feld ausserhalb von St. Margrethen) und/oder sich über die Höhe der Strassenlaternen (in meinem Fall: 47°24'56.6"N 9°37'42.1"E, Karl-Völker-Strasse 11, 9435 Heerbrugg, die Sternwarte der KSH) begeben, damit man sich vom grössten Teil der von Strassenlaternen eingebrachten Lichtverschmutzung absetzen kann.

Für alle Satelliten welche dunkler sind, sollte man sich wesentlich weiter von künstlichen Lichtquellen befinden. Dafür bieten sich lokale Berge und Hügel gut an (in meinem Fall: 47°24'40.2"N 9°32'32.7"E, Gonzern, Oberegg, Feld auf dem St. Anton).

4.1.3.2 Himmelsrichtung

Es ist darauf zu achten, dass man bei einem gegebenen Satelliten nicht einen Standort wählt, bei dem die Himmelsrichtung, in welcher die scheinbare Bahn des Satelliten im Höhepunkt steht, von Gebäuden oder von der Landschaft verdeckt ist. Dies ist lediglich bei Satellitenüberflügen mit einer Elevation von über 70° in den meisten Fällen irrelevant.

4.1.3.3 Elevation

Normalerweise sollte man von einem Mindest-Elevationswinkel von 10° ausgehen, damit der Satellit überhaupt durch die Erdatmosphäre und über eventuellen Hindernissen in der Landschaft sichtbar ist. Je höher die Elevation, umso lichtverschmutzter können die Verhältnisse sein, da in der Regel im Zenit die Lichtverschmutzung am wenigsten merkbar ist.

4.1.4 Zeit

Für gezielte Beobachtungen sollte man sich auf die Zeit verlassen, welche einem die Online-Programme wie «heavens-above.com» vorhersagen. Jedoch muss man stets darauf achten, die gegebene Zeit von UTC in lokale Zeit und ggf. in Sommerzeit umzurechnen. Es lohnt sich dazu jeweils mit Wetter-Radar-Apps wie «Meteo Swiss» die Bewölkungsvorhersagen anzuschauen, da man durch Wolken keine Satelliten sehen kann.

Für ungezielte Beobachtungen ist es gut zu wissen, dass die meisten beobachtbaren Satelliten sich im LEO befinden und damit nur bis zu etwa 1.5 Stunden nach der Morgen- bzw. vor Abenddämmerung sichtbar sind. Dies liegt daran, dass dies die Zeiten sind, in denen man auf der Erde kein direktes Licht von der Sonne bekommt, die Satelliten aber aufgrund ihrer höheren Position trotzdem noch das Sonnenlicht reflektieren. Satelliten mit höher gelegener Umlaufbahn, wie z.B. solche in GSO¹⁵ sind zu den gleichen Uhrzeiten zu beobachten, jedoch typischerweise länger als diese in LEO, da sie länger nicht in den Erdschatten eintreten.

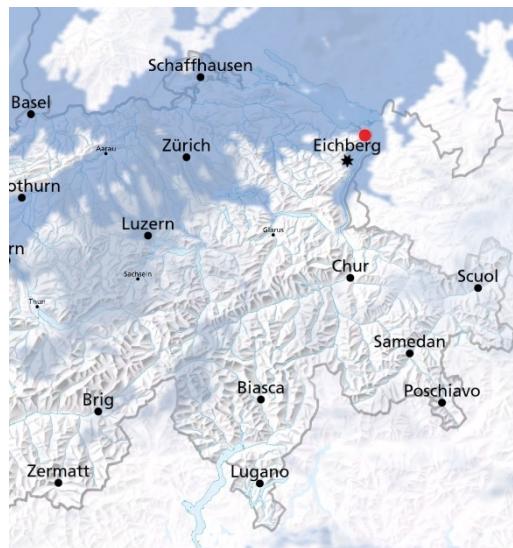


Abbildung 11: negatives Beispiel des Wetterradars der App "MeteoSwiss"

4.1.5 Prozess der Messung

Vor Ort ist es bereits vor dem Beobachten von Relevanz sich einen Plan zu machen, wann und wie man nachführen sollte. Dafür kann man am besten eine Aufnahme vom Himmel mit dem Kamera Modus «Programmautomatik» auslösen und sehen, wie lange es für eine Aufnahme braucht. Andere Belichtungsmodi, wie z.B. Dauerbelichtung können auch genutzt werden, sind aber eher hilfreich, wenn man ungezielt aufnimmt. Das Nachführen sollte dann jeweils nur zwischen den Aufnahmen geschehen und niemals währenddessen. Solange die Kamera ein Bild am Aufnehmen ist, sollte man die ganze Apparatur möglichst nicht berühren und keine Lichtquellen, wie z.B. Taschenlampen in der Nähe haben. Nach dem Einstellen des Stativs in die gewünschte Himmelsrichtung wartet man, bis sich der Satellit im Sichtfeld der Kamera befindet und benutzt den Fernauslöser, um abzudrücken. Danach führt man dem Pfad des Satelliten, welcher von Auge gut bestimmbar ist, nach und wiederholt den beschriebenen Vorgang, bis der Satellit in den Erdschatten oder unter den Horizont tritt.

¹⁵ Siehe Anhang: Lexikon; Seite 52

4.2 Daten Auswertung

4.2.1 Plate-Solving

Die aufgenommenen Bilder werden auf einen Computer, welcher Windows 10 installiert hat, geladen und können dann als Bildsequenz in «AstroImageJ» geöffnet werden.

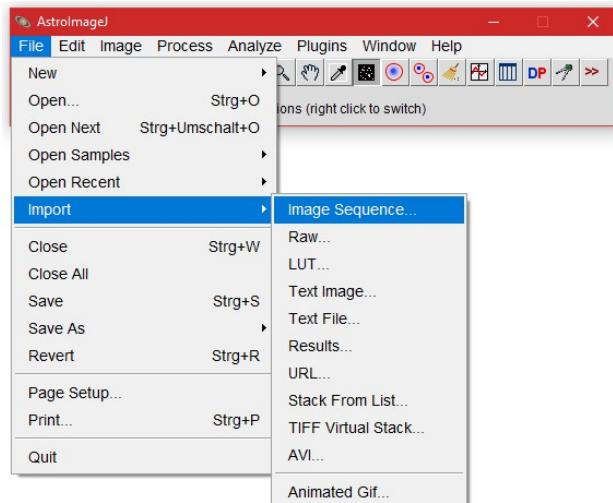


Abbildung 12: Logo von AstroImageJ [8]

Abbildung 13: Anleitung zum Importieren mehrerer Bilder in AstroImageJ

«AstroImageJ» ist eine Erweiterung der Open Source Software «ImageJ» und dient in diesem Falle dazu die Rektaszension und Deklination der Satellitenbahnen zu bestimmen. Dies leistet «AstroImageJ» durch **Plate-Solving**. D. h. es vergleicht die Sterne in einem Bild mit einer Datenbank aus bekannten Sternenformation und leitet dadurch die Position jedes Pixels im rotierenden äquatorialen Koordinatensystem ab. Um diese Funktion benutzen zu können muss man sich bei nova.astrometry.net zusätzlich einen Account einrichten und den dortigen API-Key in die Plate-Solving Optionen von «AstroImageJ» eingeben.

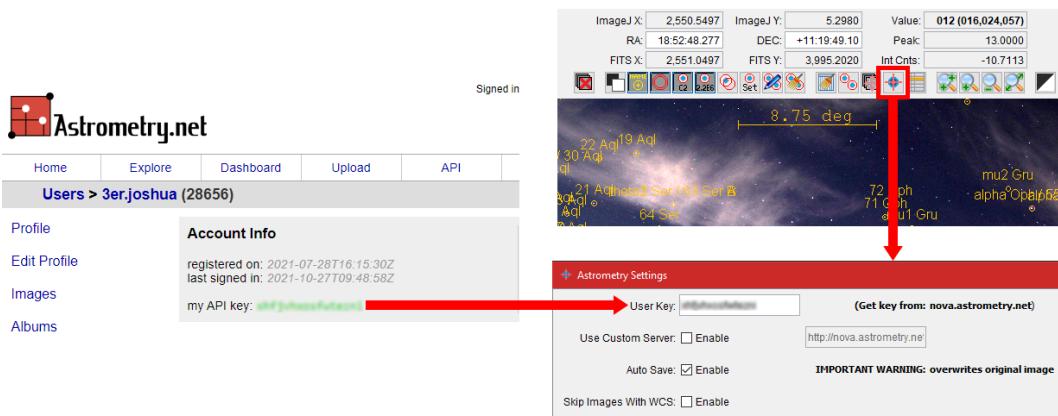


Abbildung 14: Eintragen des API-Keys

Damit das Plate-Solving auch erfolgreich wird, muss man die Einstellungen wie folgt gut beachten. Der wichtigste anpassbare Wert ist die sogenannte «**SIP-Order**», welcher bestimmt wie gross bzw.

Material und Messmethoden

klein der Bereich des Bildes im Himmel ist. Für Weitwinkelaufnahmen sollte dieser auf 5 stehen. Zusätzlich nützlich ist die «**Limit Max Peaks**» Option, welche es erlaubt, bei hellen Satellitenaufnahmen die maximale Helligkeit für einen Stern tiefer einstellen zu können als die Helligkeit der Satellitenspur. Ein weiterer Erkennungsmechanismus für Sterne wird durch den «**Median Filter**» bestimmt. Je grösser dieser ist, desto weniger werden Sterne, die teilweise durch die Atmosphäre oder verwackelten Aufnahmen verschwommen sind, als Stern wahrgenommen. Dies hilft vor allem bei Aufnahmen mit einem hohen ISO-Wert, da dort das Rauschen ansonsten unabsichtlich als Stern aufgefasst werden kann.

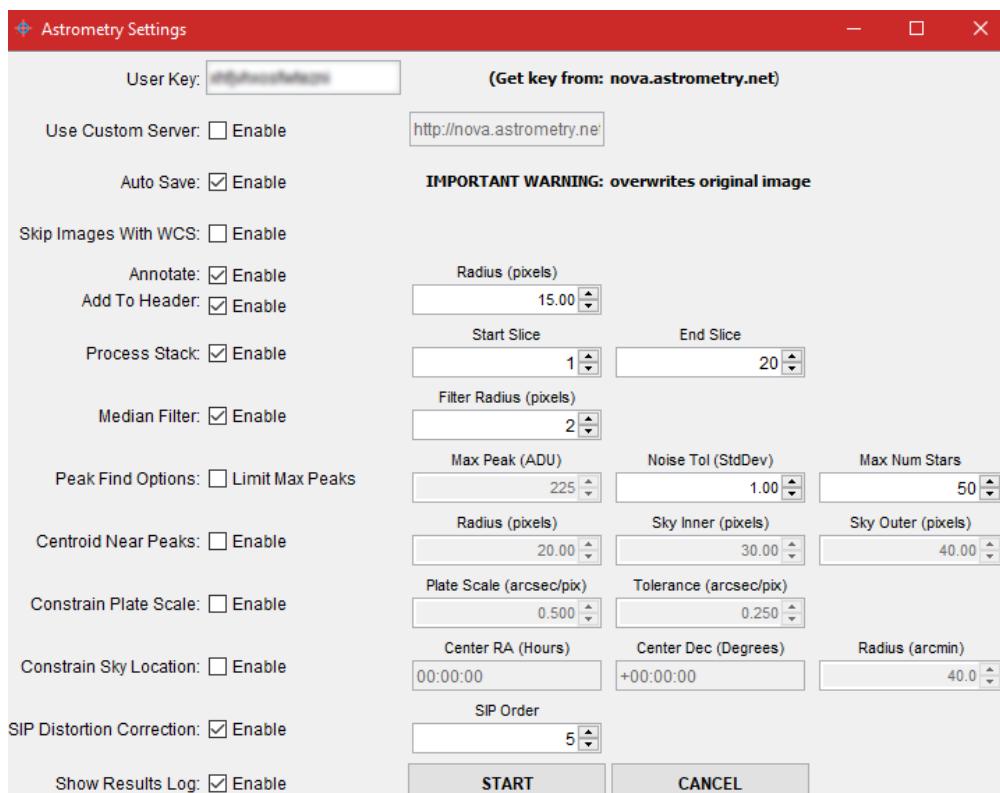


Abbildung 15: Plate-Solving Einstellungen in AstroImageJ

Bei erfolgreichem Plate-Solve kann man nun das «**astrometry Tool**» in «AstroImageJ» auswählen.

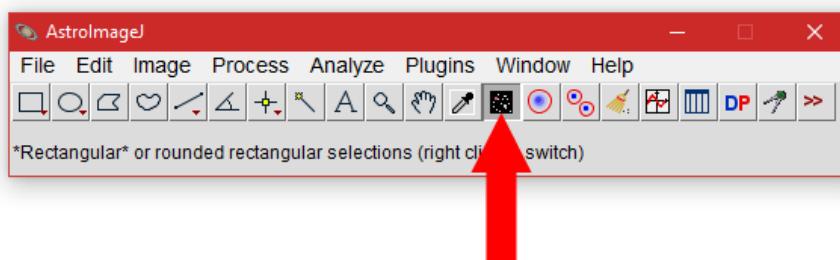


Abbildung 16: Auffindungsart des Astrometry Tools

Material und Messmethoden

Mit diesem kann man nun den Beginn der Satellitenspur mit einem Rechtsklick auswählen und ihn bis zum Ende der Satellitenspur ziehen.



Abbildung 17: Benutzung des Astrometry Tools

Direkt nach dem Loslassen der Strecke erscheint von AstroImageJ eine Tabelle, welche die gemessenen Punkte als RA/DEC-Koordinaten angibt.

Wiederholt man den obigen Prozess für alle Bilder einer Messungsreihe, ist man bereit dazu die Daten weiter mit Excel auszuwerten.

4.2.2 Berechnen relevanter Werte mittels Excel-Tabelle

4.2.2.1 Nutzung der Tabelle

Da sich Excel einwandfrei für Datenauswertung mit überschaubaren Datensätzen eignet und das Programm weit verbreitet ist, wurde sich bei dieser Arbeit dafür entschieden, um die durch Beobachtungen und AstroImageJ entstandenen Rohdaten für das selbstgeschriebene Programm vorzubereiten.

Der Nutzer kann die dazugelegte Exceldatei benutzen, indem er die Tabelle von AstroImageJ in das Arbeitsblatt «Daten» kopiert. Am besten ist dies möglich, indem man bei AstroImageJ mittels «Ctrl + A, Ctrl + C» die Daten in die Zwischenablage kopiert, bei der Exceletabelle die Zelle C2 auswählt und die Daten mittels «Ctrl + V» von der Zwischenablage in die Tabelle einträgt. Danach muss manuell die Belichtungszeiten sowie die Zeit der Aufnahme jedes einzelnen Bildes eingetragen werden. An diese Informationen kommt man, indem man die Details der Datei abruft (Im Windows File-Explorer: Rechtsklick → Eigenschaften → Details) oder im Windows File-Explorer eine eigene Spalte für diese Eigenschaften öffnet. Nun bleibt es lediglich noch die Angaben in den fettumrahmten Zellen auszufüllen.

	Beobachtungsort	KSH-Dach	Datum	31.05.2021
ISS Höhe[m]	Breitengrad	47.415738	JD00	2459365.5
413000	Längengrad	9.628334	MJD2000	7820.5
ISS Geschwindigkeit [m/s]	UTC Zeitzone		T (Julianischer Halbtag)	0.214113621
7600	plus	01:00		
ISS Umlaufszeit [s]	Sommer/Winterzeit		GSMT(0h) [s]	59691.80741
5580	Sommer		GSMT(0h) [d]	16:34:52

Abbildung 18: Ausschnitt des Daten-Blatts der Excel-Tabelle

Die Angaben für die Höhe, Geschwindigkeit und Umlaufszeit der ISS sind für die Vorbereitung der Daten auf das selbstgeschriebene Programm «Tarvos» unwichtig und dienen nur dem Vergleich der errechneten Winkelgeschwindigkeit mit der beobachteten und sind somit optional. Längen- und Breitengrad sowie Zeitzone können anhand von Diensten wie Google-Maps, allenfalls einer Karte oder eines Ortungsgerätes (z.B. GPS des Smartphones) bestimmt werden.

4.2.2.2 Funktionsweise der Tabelle

Das primäre Ziel der Tabelle ist es, die Koordinatenangaben, welche von AstroImageJ kommen mittels Sternzeit in eine quantitative Position am Himmel in Elevationswinkel (= ALT, h) und Azimut (=AZ, a) umzurechnen. Die Sternzeit ist abhängig vom Breitengrad des Beobachters und der Uhrzeit zum Zeitpunkt des Beobachtens. Aufgrund dessen teilt man die Rechnung in mehrere Schritte auf:

4.2.2.2.1 Julianisches Datum

Das julianische Datum zählt die vergangenen Halbtage seit dem 1. Januar 4713 v. Chr. und ist in der Astronomie eine übliche Zeitzählung¹⁶. Es hat jedoch mit dem julianischen Kalender wenig gemeinsam. Mittels folgendem Algorithmus¹⁷ kann man das gregorianische Datum (heute gängiger Kalender) in ein julianisches Datum umrechnen.:

```
wenn Monat > 2 dann  
    Y = Jahr; M = Monat  
sonst  
    Y = Jahr - 1; M = Monat + 12  
  
D = Tag  
  
B = 2 - [Y/100] + [Y/400]  
  
JD = [365,25(Y + 4716)] + [30,6001(M + 1)] + D + B - 1524,5
```

4.2.2.2.2 Sternzeit in Greenwich um 0h

Dadurch, dass der Polynom-Term, welcher die Sternzeit in Greenwich um 0h angibt, «T» in julianischen Jahrhunderten seit dem 1. Januar 2000 benutzt, muss man dies folgendemassen umrechnen (2451545 = JD am 1. Januar 2000, GSMT: Greenwich Mean Sidereal Time)

$$T = \frac{JD - 2451545}{365.25}$$

$$\begin{aligned} GMST(0h\ T)[s] \\ = 24110.54841 + 8640184.812866 * T + 0.093104 * T^2 - 0.0000062 * T^3 \end{aligned}$$

¹⁶ Siehe Quelle [11]

¹⁷ Siehe Quelle [11]

4.2.2.2.3 Sternzeit in Greenwich zu UTC

So weit haben wir für ein beliebiges Datum die Sternzeit auf dem Nullmeridian und um 00:00 Uhr ausgerechnet. Um die Umrechnung durchführen zu können, wird jedoch die «lokale» Sternzeit zur Uhrzeit der Beobachtung und am Meridian des Längengrades gesucht. Dies wird Folgendermassen erreicht. (ΔT : Korrektur Summand. Typischerweise 73^{18} sec. 1.002737909: Synodische Tageslänge). Für ein Beispiel siehe Spalte 3 in Tabelle 1 auf Seite 22.

$$GMST(UTC\ T)[s] = GMST(0h\ T)[s] + (UTC[d] + \Delta T) * 86'400 * 1.002737909$$

Um folgende Rechnungen einfacher zu machen, lohnt es sich GMST (UTC T) folgendermassen in einen Tagesbruchteil umzurechnen, da Excel Tagesbruchteile als Uhrzeiten darstellen kann. (Beispiel bei Tabelle 1 Spalte 4)

$$GMST(UTC\ T)[d] = GMST(UTC\ T)[s]/(86'400)$$

4.2.2.2.4 Lokale Sternzeit

Um nun die lokale Sternzeit LMST (UTC T) [d] auszurechnen, muss man nur noch den Längengrad als Kreisbruchteil addieren. (λ : geographischer Längengrad, LMST: Local Mean Sidereal Time). (Beispiel bei Tabelle 1 Spalte 5)

$$LMST(UTC\ T)[d] = GMST(UTC\ T)[d] + \lambda[deg]/360$$

4.2.2.2.5 Stundenwinkel

Der Finale Schritt vor der Ausrechnung der Himmelskoordinaten ist die Umrechnung in den Stundenwinkel τ , welcher die Winkeldifferenz der Rektaszension und des durch die LSMT (UTC T) dargestellte Himmels-Meridian darstellt. (α : Rektaszension, Beispiel bei Tabelle 1 Spalte 8)

$$\tau[rad] = (LMST(UTC\ T)[d] - \alpha[deg]/360) * 2\pi$$

¹⁸ Siehe Anhang Kapitel 13.3.2 «Erdrotationswinkel» auf Seite 55

4.2.2.2.6 Himmelskoordinaten

Schlussendlich können wir Elevationswinkel und Azimut durch folgende Formeln¹⁹ ausrechnen:
(δ: Deklination, h: Elevationswinkel, a: Azimut). Beispiel bei Tabelle 1 Spalte 9 und 10.

$$h = \sin^{-1}(\sin \delta * \sin \lambda + \cos \delta * \cos \lambda * \cos \tau)$$

$$w = \cos^{-1}\left(\frac{\sin \delta - \sin h * \sin \lambda}{\cos h * \cos \lambda}\right)$$

wenn $w > 0$ dann

$$a = w$$

sonst

$$a = 360 - w$$

4.2.2.3 Funktionsweise der Tabelle

Die Tabelle rechnet nach diesem Schema die Koordinaten für jeden gemessenen Punkt aus dem äquatorialen Koordinatensystem in das horizontale Koordinatensystem um. Anschliessend rechnet es den arithmetischen Durchschnitt zwischen dem Start- und Endpunkt der Satellitenspur aus und weist diesem Punkt die Zeit des Bildes (welches am Endpunkt entsteht) abzüglich der halben Belichtungszeit (da sich dort die Mitte der Spur befindet) zu.

Die Tabelle verarbeitet dies zu zwei Diagrammen:

- eine Darstellung der «Roh-Daten» von AstroImageJ im äquatorialen Koordinatensystem
 - o Siehe Kapitel 4.2.3.3 Seite 23: Abbildung 22, Abbildung 21, Abbildung 23.
- eine Darstellung der ausgewerteten Koordinaten im horizontalen Koordinatensystem
 - o Siehe Kapitel 4.2.3.4 Seite 24f.: Abbildung 24, Abbildung 25, Abbildung 26.

Aus dem Diagramm der ausgewerteten Koordinaten bekommt man eine quadratische Näherungsfunktion, für welche man den mathematischen Hochpunkt bestimmen kann.

¹⁹ Siehe Quelle [2]

4.2.3 Beispiele

4.2.3.1 Aufnahmen



Abbildung 19: ISS durch Orion (07.11.2021)

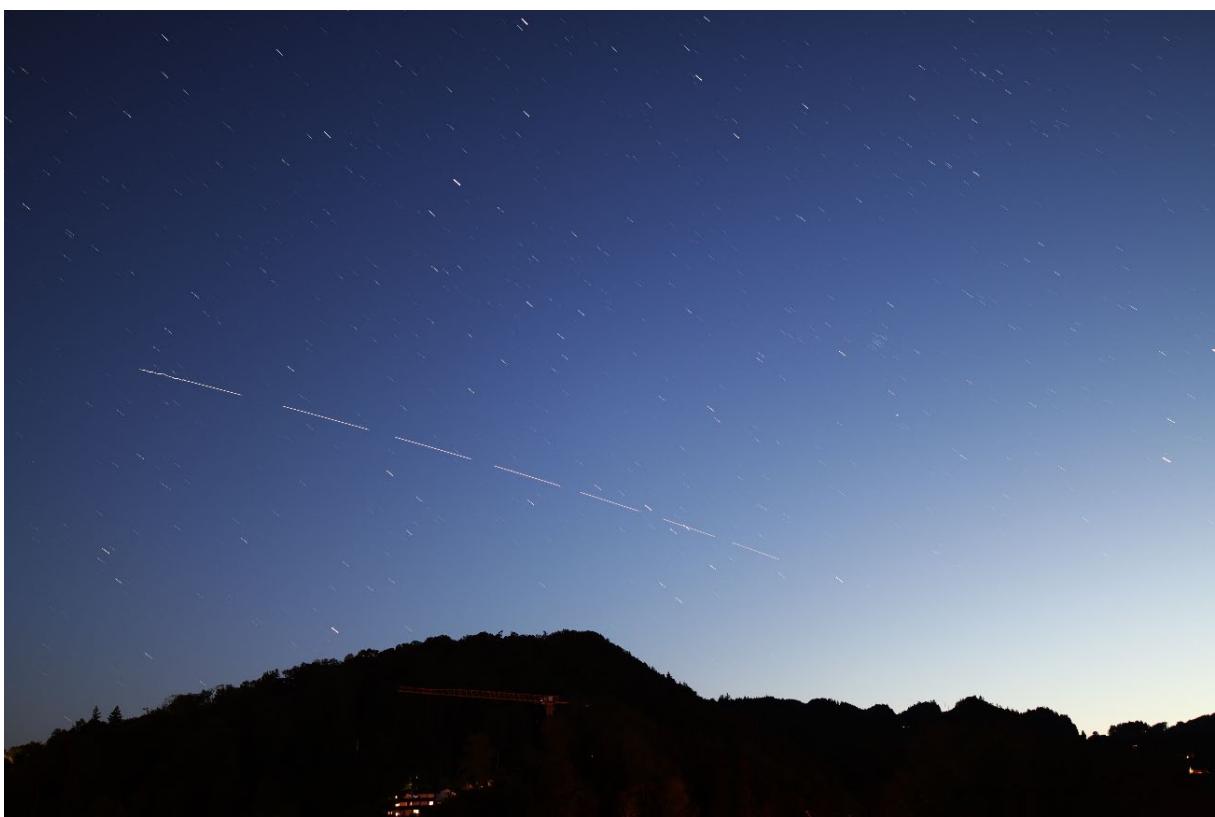


Abbildung 20: ISS über Sonnenuntergang (31.05.2021)

Material und Messmethoden

4.2.3.2 Tabellen

Tabelle 1: Umrechnung in Horizontales Koordinatensystem Anhand von Daten des 18.09.2021.

Julianisches Datum 18.09.2021: 2459475.5 → T = 0.217125257

Breitengrad $\lambda = 47.41116^\circ$

Zeit [d]	UTC	GSMT (UTC) [s]	GSMT (UTC) [d]	LSMT (UTC) [d]	\varnothing RA [deg]	\varnothing DEC [deg]	\varnothing HA [rad]	\varnothing AZ [deg]	\varnothing ALT [deg]
20:56:12	18:56:22	154155	18:49:15	12:39:01	261.50	-5.64	0.530	215.63	30.55
20:56:31	18:56:36	154169	18:49:29	12:39:13	268.66	-1.49	0.406	209.59	37.02
20:56:42	18:56:46	154179	18:49:39	12:39:27	274.03	1.47	0.313	204.19	41.47
20:56:53	18:56:58	154191	18:49:51	12:39:41	280.71	5.04	0.197	196.36	46.49
20:57:18	18:57:28	154221	18:50:21	12:39:55	296.33	12.49	-0.073	172.03	54.20
20:57:42	18:57:46	154239	18:50:39	12:40:09	315.46	19.92	-0.406	137.64	56.53
20:57:54	18:57:58	154251	18:50:51	12:40:25	326.50	22.89	-0.598	120.01	53.23
20:58:06	18:58:10	154263	18:51:03	12:41:11	335.54	24.64	-0.755	108.37	49.00
20:58:28	18:58:38	154291	18:51:31	12:41:45	349.95	25.94	-1.004	94.67	40.44
20:56:12	18:56:22	154155	18:49:15	12:42:15	261.50	-5.64	0.530	215.63	30.55
20:56:31	18:56:36	154169	18:49:29	12:42:47	268.66	-1.49	0.406	209.59	37.02
20:56:42	18:56:46	154179	18:49:39	12:43:11	274.03	1.47	0.313	204.19	41.47

Tabelle 2: Koordinaten der Hochpunkte (ISS Beobachtungen)

Datum	31.05.2021	18.09.2021	07.11.2021
Uhrzeit (UTC)	19:22:37	18:57:46	05:25:21
Näherungsfunktion	$y = -0.005x^2 + 2.2202x - 214.01$	$y = -0.0059x^2 + 1.7515x - 72.571$	$y = 0.0003x^2 - 0.5215x + 38.416$
h des Maximums [deg]	32.4544	57.4185	21.765
a des Maximums [deg]	222.02	148.432	218.095

4.2.3.3 Im äquatorialen Koordinatensystem

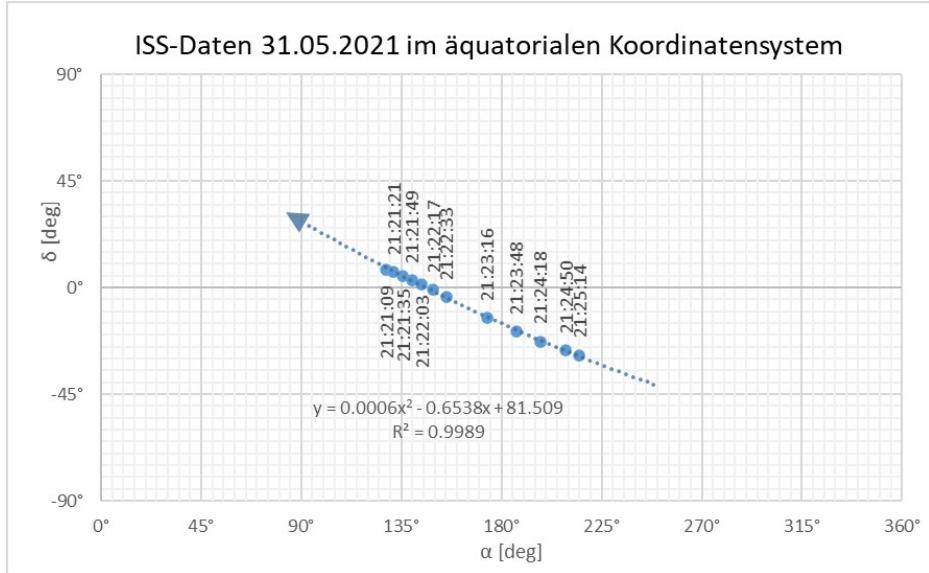


Abbildung 22: 31.05.2021 im äquatorialen Koordinatensystem

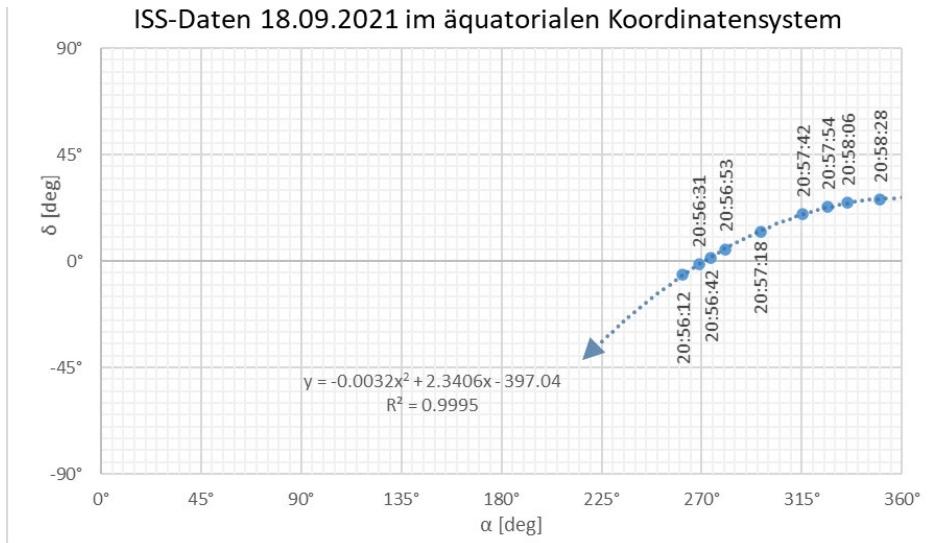


Abbildung 21: 18.09.2021 im äquatorialen Koordinatensystem

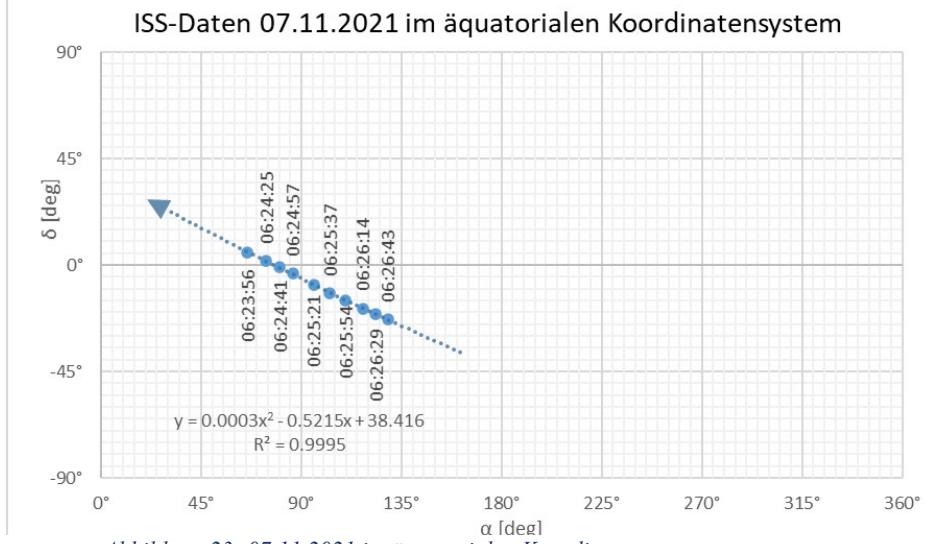


Abbildung 23: 07.11.2021 im äquatorialen Koordinatensystem

4.2.3.4 Im horizontalen Koordinatensystem

ISS-Daten 31.05.2021 im horizontalen Koordinatensystem

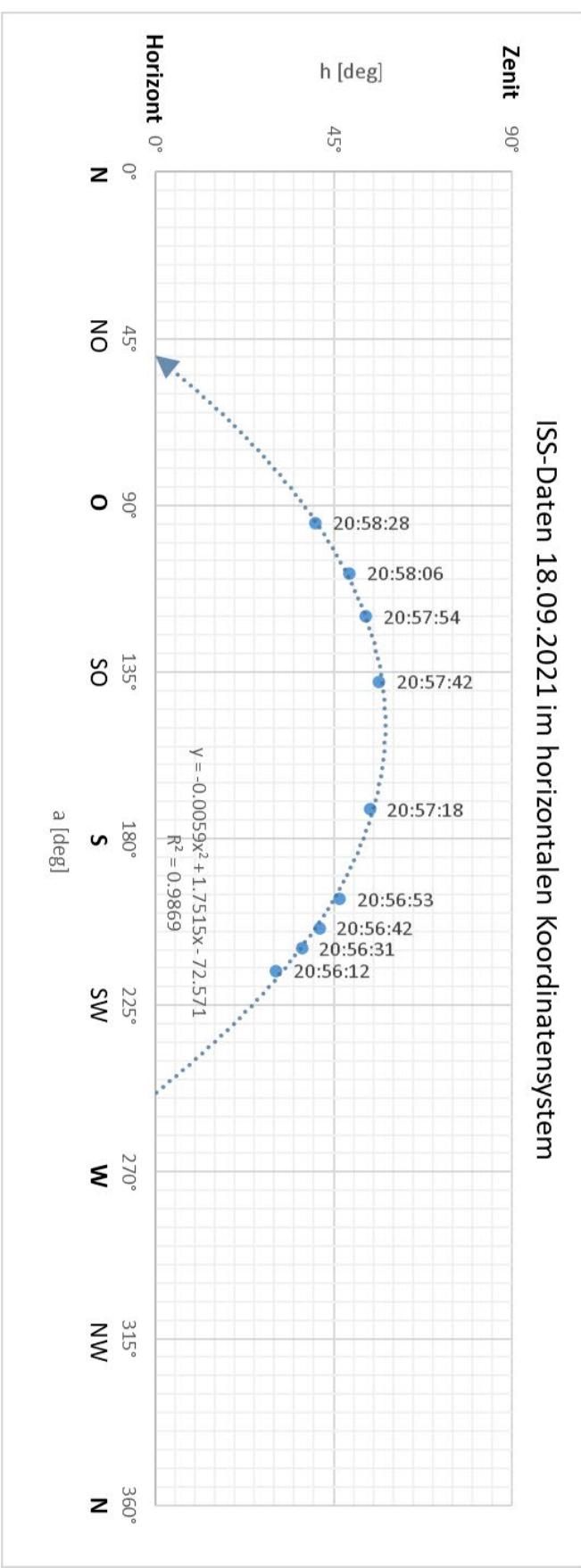


Abbildung 24: 18.09.2021 im horizontalen Koordinatensystem

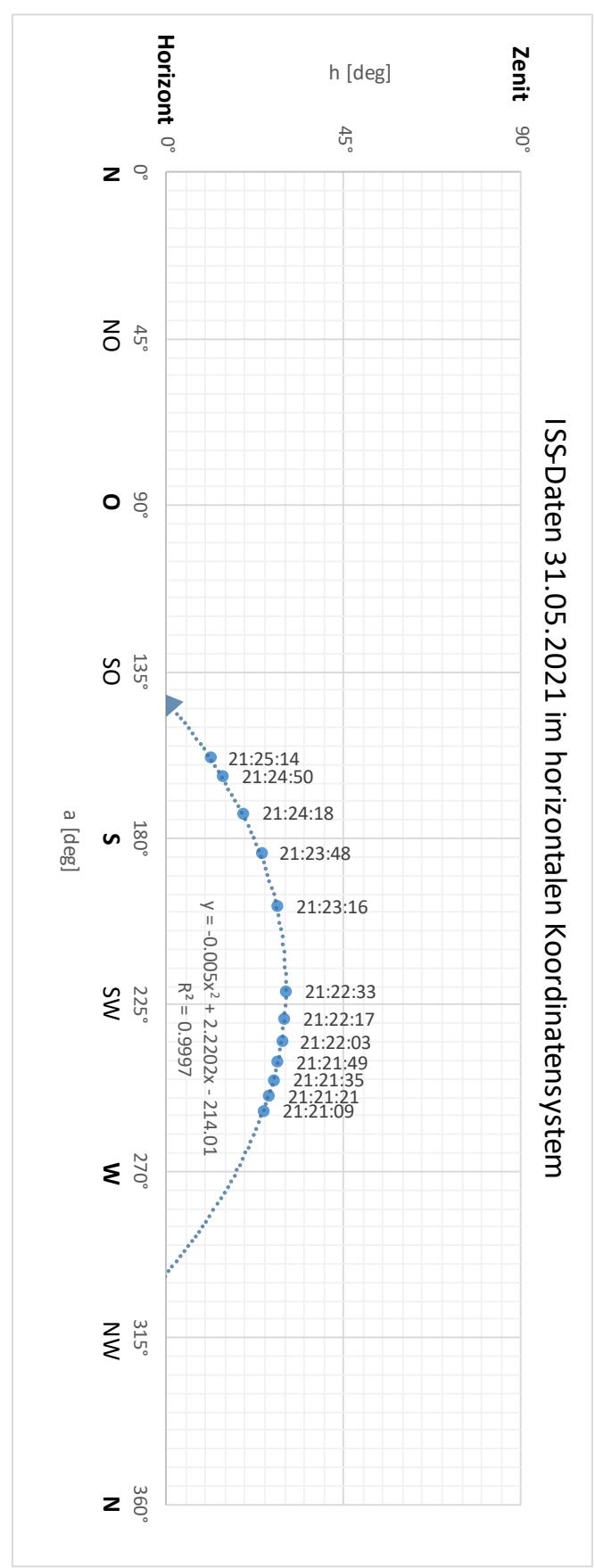


Abbildung 25: 31.05.2021 im horizontalen Koordinatensystem

ISS-Daten 07.11.2021 im horizontalen Koordinatensystem

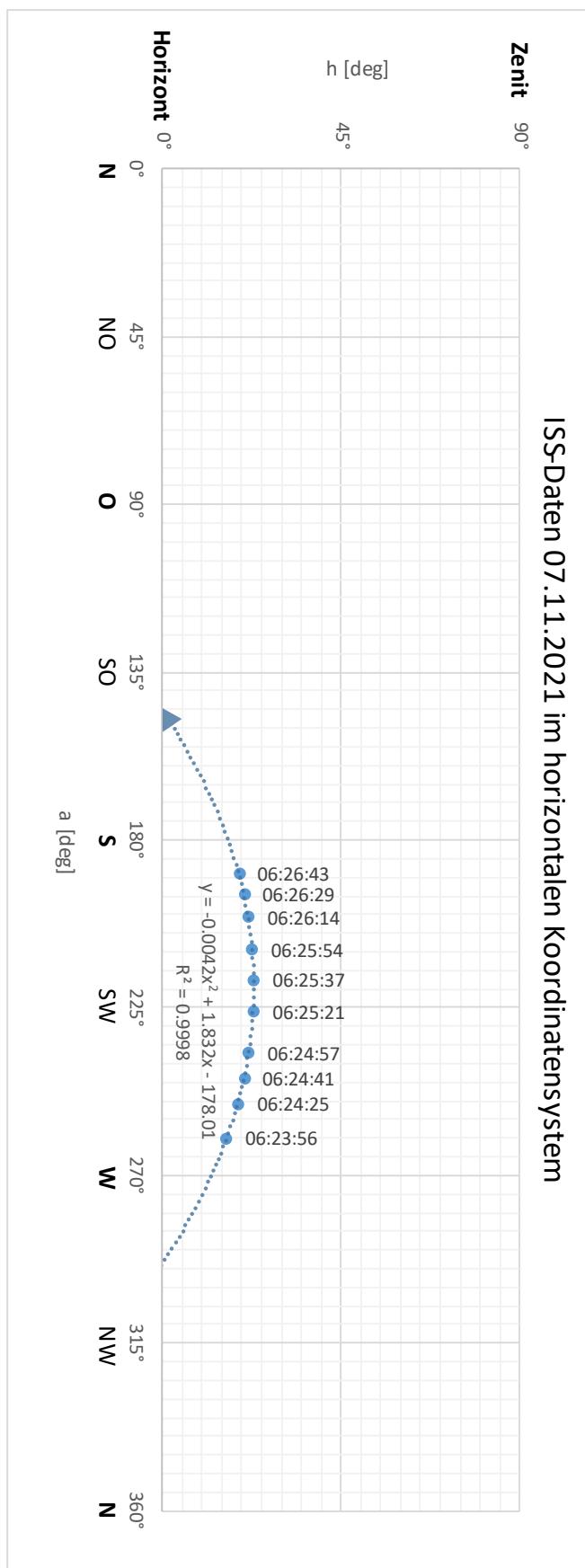


Abbildung 26: 07.11.2021 im horizontalen Koordinatensystem

5 Entwicklung und Funktionsweise des Programms

Da es aus den gemessenen Daten schwer möglich schien, rechnerisch die nächsten Überflüge zu bestimmen, wurde sich dafür entschieden, ein Programm zu entwickeln, welches diese Überflüge simulieren kann. Als Namen wurde sich für «Tarovos» entschieden. Dies ist der Name einer der kleineren und weit entfernten Monden Saturns. Es wurde mit dem Ziel programmiert, aufgrund einer Satellitenmessung (im horizontalen Koordinatensystem) die nächsten Überflüge für eine beliebige Zeit zwischen dem 1. März 2021 und 1. März 2022 vorherzusagen.

5.1 Entwicklung

Als Programmiersprache für das Programm wurde Java gewählt, da ich von der Schule aus Erfahrungen damit gesammelt habe und die Syntax dabei verständlich finde. Um mich aber wirklich mit der Sprache zurechtzufinden wurde nach der Themenwahl mehrere Monate an Zeit verwendet, um die die Programmiersprach zu lernen. Hierbei waren vor allem die Java Tutorials des YouTube-Kanales «Programming with Mosh» nützlich.

Als die Grundlagen erlernt waren, musste entschieden werden was für Libraries²⁰ verwendet werden sollten, da das Programmieren einer eigenen 3D-Engine viel zu viel Zeit benötigen würde. Diese Entscheidung fiel auf «JavaFX». Einer der Hauptgründe dafür ist, dass es FXML (was

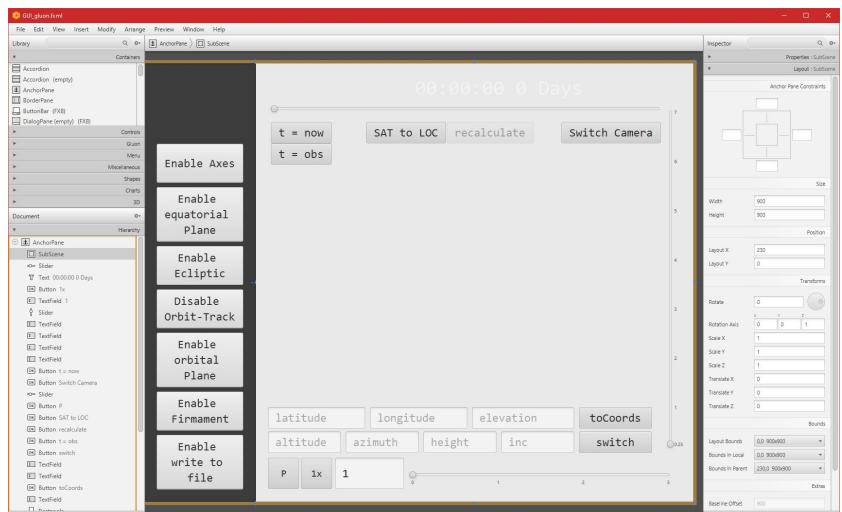


Abbildung 27: GUI des Programmes in Gluon Scene Builder geladen

che HTML ähnelt) unterstützt, was erlaubt in einer Nutzer-freundlichen Oberfläche das GUI²¹ grösstenteils separat vom eigentlichen Programmcode zu entwerfen. Für diesen Zweck wurde das Programm «Gluon Scene Builder» verwendet. Zudem war es äusserst wichtig, dass die Library das Anzeigen von 3D-Objekten und das Laden von 3D-Modellen ermöglicht, da ansonsten das Anzeigen der Erdoberfläche schwierig geworden wäre. Ausserdem enthält JavaFX einige

²⁰ Siehe Anhang: Lexikon; Seite 52

²¹ Siehe Anhang: Lexikon; Seite 52

mathematische Tools welche es erlaubt das Rechnen mit Vektoren performant auszuführen. Beim Erlernen von FXML und JavaFX war der YouTube Kanal «Bro Code» nützlich.

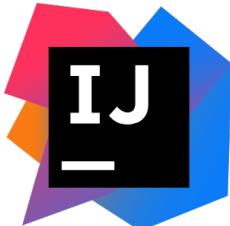


Abbildung 30: Logo von «IntelliJ IDEA» [25]

Als Entwicklungs-Umfeld (IDE) wurde, hauptsächlich aus persönlicher Vorliebe, «IntelliJ IDEA» verwendet. Für die in «Tarovos» verwendeten 3D-Objekte wurde das Open-source Programm «Blender» verwendet, mit welchem ich bereits vor der Arbeit Erfahrung hatte und welches auch zum erstellen einiger Graphiken nützlich war.



Abbildung 28 Logo von «Gluon Scene Builder» [24]



Abbildung 29 Logo von «Blender» [26]

Von der investierten Zeit her nahm die Entwicklung und sicherstellen der funktionsfähigkeit des Programmes bei weitem am meisten ein. Grosse Probleme dabei verursachten das bestimmen einer Rotationsachse (), sowie der Umgang mit der Zeit. Bei mehreren Problemen lag es daran, dass zu hohe Ziele gesteckt wurden, ohne dass genügend Vorarbeit geleistet wurde. So wurde zum Beispiel das implementieren von ellipischen Umlaufbahnen, welche es nie zu einer funktionierenden Version geschafft hat, früh probiert, bevor das Programm fassbare resultate produziert hatte. Schlussendlich konnten jedoch alle essentiellen Programm-Funktionen umgesetzt werden und wenn ich heute nochmals die gleichen funktionen versuchen würde zu implementieren, hätte ich bereits eine funktionierende Grundlage, auf der dies geschehen kann.

Das endgültige Programm, welches auch auf dem USB-Stick beigelegt ist enthält 16 Klassen und insgesamt rund 1'150 Zeilen an Code.

5.2 Bedienung

5.2.1 Programmoberfläche

Hier folgte ein beschrifteter Teil der Benutzer-Oberfläche mitsamt den Funktionen der einzelnen Knöpfe.

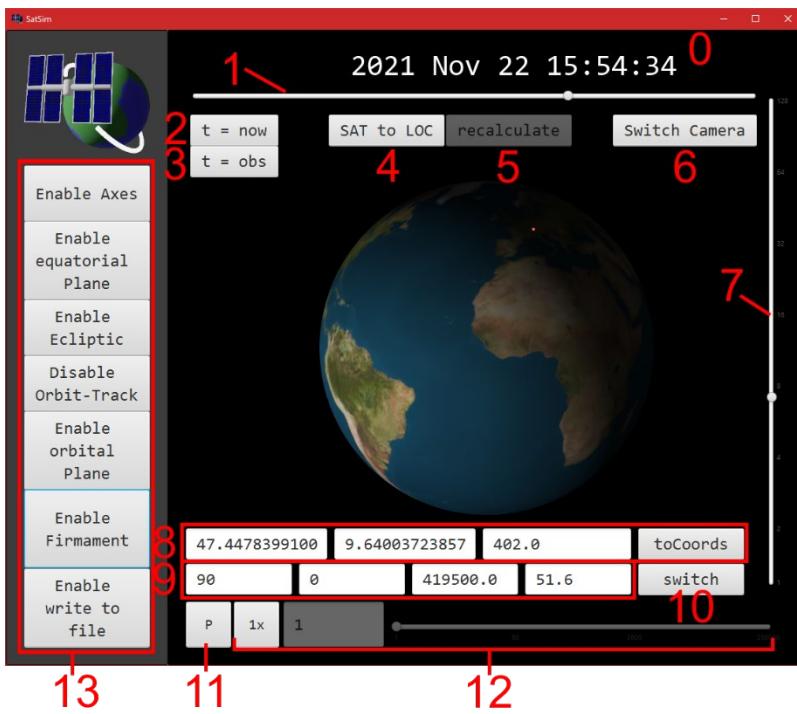


Tabelle 3: Legende zu Abbildung Abbildung 27

- | | |
|---|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 0 | Zeitangabe der ausgewählten Programm-Zeit in UTC. |
| 1 | Zeit-Schieberegler , dient zum Einstellen der Beobachtungszeit, reicht vom 21 März .2021 bis zum 21. März 2022 (Frühlingspunkt). |
| 2 | Zeit-zu-Jetzt-Knopf , die aktuelle Zeit wird zur ausgewählten Programm-Zeit |
| 3 | Zeit-zu-Beobachtungszeit , die ausgewählte Programm-Zeit wird auf die Zeit der eingegebenen Beobachtung gesetzt. Ist dies nicht möglich wird sie auf den 21. März 2021 gesetzt. |
| 4 | Satellit-zu-Ort-Knopf , setzt zusammen mit dem Elevationswinkel (altitude), dem Azimut (azimuth) und der Satellitenhöhe (height) den Satellit an den Ort im Himmel, an dem er gesichtet wurde. |
| 5 | Satellit-neu-berechnen-Knopf , berechnet mithilfe der Satellitenhöhe (height) und der Bahninklination (inc) aus 9 die möglichen Umlaufbahnen für die nächsten fünf Tage. |
| 6 | Kamera-Wechsel-Knopf , schaltet von der freien Kamera auf die Oberflächen-Kamera und zurück. (Oberflächenkamera funktioniert nur beschränkt). |



Abbildung 32: Logo von «Tarvos», erstellt mit Blender 2.93.6

Abbildung 31:
beschrifteter Screenshot von
«Tarvos»

7	Zoom-Schieberegler , bestimmt wie nahe die Kamera an der Erde ist, wird ebenfalls durch das Mausrad kontrolliert. (bei Oberflächen-Kamera ändert es die Blickrichtung)																
8	geographische Koordinaten-Bedienfeld , erlaubt dem Nutzer, die geographischen Koordinaten wie Breitengrad (latitude), Längengrad (longitude) und geographische Höhe (elevation) einzugeben und mit «toCoords» zu bestätigen.																
9	horizontale Koordinaten-Bedienfeld , erlaubt dem Nutzer, die ausgewerteten Daten im Form von horizontalen Koordinaten anzugeben. Dazu muss man weitere Bahnpараметer (Satellitenhöhe (height) und Inklination (inc)) angeben.																
10	Umlaufbahn-Wechsel-Knopf , wechselt zwischen den zwei möglichen Umlaufbahnen																
11	Pausier-Knopf , erlaubt dem Nutzer die Zeit anzuhalten und wieder laufen zu lassen.																
12	Zeit-Geschwindigkeit-Schieberegler , dient dazu, dem Nutzer die Möglichkeit zu bieten, die Geschwindigkeit des Zeitflusses einzustellen.																
13	Optionale Programm-Funktionen , können nach Belieben ein- und ausgeschaltet werden:																
	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="padding: 2px;">Enable/Disable...</th><th style="padding: 2px;">Ein-/Ausblenden der...</th></tr> </thead> <tbody> <tr> <td style="padding: 2px;">Axes</td><td style="padding: 2px;">— X, Y und Z-Achsen</td></tr> <tr> <td style="padding: 2px;">equatorial Plane</td><td style="padding: 2px;">— äquatorial Ebene.</td></tr> <tr> <td style="padding: 2px;">ecliptic</td><td style="padding: 2px;">— Ekliptik-Ebene</td></tr> <tr> <td style="padding: 2px;">orbit-Track</td><td style="padding: 2px;">— Umlaufspur</td></tr> <tr> <td style="padding: 2px;">orbital Plane</td><td style="padding: 2px;">— Umlaufbahn-Ebene</td></tr> <tr> <td style="padding: 2px;">firmament</td><td style="padding: 2px;">— Sternzeichen</td></tr> <tr> <td style="padding: 2px;">write to File</td><td style="padding: 2px;">— Dateispeicherung</td></tr> </tbody> </table>	Enable/Disable...	Ein-/Ausblenden der...	Axes	— X, Y und Z-Achsen	equatorial Plane	— äquatorial Ebene.	ecliptic	— Ekliptik-Ebene	orbit-Track	— Umlaufspur	orbital Plane	— Umlaufbahn-Ebene	firmament	— Sternzeichen	write to File	— Dateispeicherung
Enable/Disable...	Ein-/Ausblenden der...																
Axes	— X, Y und Z-Achsen																
equatorial Plane	— äquatorial Ebene.																
ecliptic	— Ekliptik-Ebene																
orbit-Track	— Umlaufspur																
orbital Plane	— Umlaufbahn-Ebene																
firmament	— Sternzeichen																
write to File	— Dateispeicherung																

5.2.2 Vorgehensweise

Damit das «Tarvos» die gewünschten Ergebnisse produziert ist eine korrekte Benutzung vorauszu-setzen. Eine solche wird folgend präsentiert:

1. Der Nutzer gibt die Koordinaten des Beobachtungsortes in die Felder von 8 ein und setzt somit die Position der orange-roten Kugel zum gewünschten geographischen Ort
2. Mittels den Bedienflächen 1, 2, und 12 soll der Nutzer die richtige Zeit einstellen und pausiert die Zeit.
3. Bei den Feldern aus 9 gibt man die Werte aus der Excel-Tabelle ein, Höhe und Inklination müssen ggf. literarischen Quellen entnommen werden. Anschliessend bestätigt man mit 4 seine Eingabe. Die blaue Kugel, welche den Satelliten darstellt, erscheint am eingegebenen Ort, relativ zur Orts-Position. Der Knopf von 5 sollte dadurch auswählbar gemacht worden sein.

Entwicklung und Funktionsweise des Programms

4. Man betätigt den eben erwähnten Knopf 5 und wählt anschliessen mit «switch» die passende Umlaufbahn aus. Nur die, bei welcher der Höhepunkt der eingegebenen Position entspricht, kann eine passende Umlaufbahn sein.
5. Falls man einen signifikant unterschiedliche Beobachtungsstandort hat (>50 km Distanz), als der Standort, für den man die Überflüge hervorragen will, so kann man in diesem Schritt in 8 Koordinaten ändern und mit «toCoords» bestätigen. Wenn man «Enable write to File» eingeschaltet hat, so muss man Knopf 5 erneut betätigen.
6. Nun lässt man die Zeit mit gewünschter Geschwindigkeit laufen und sieht, wann die Nächsten Überflüge stattfinden. Falls das Programm einen Überflug bemerkt, wird in der Konsole das Datum, die Uhrzeit, sowie der Elevation angegeben. Falls bei Schritt 4 «Enable write to File» eingeschaltet war, gibt eine längere Ladezeit, jedoch erstellt «Tarvos» dafür in einer Datei im gleichen Pfad wie die .exe-Datei des Programmes. in welche es sämtliche Vorausgesagten Überflüge für 5 Tage nach der Beobachtung hineinschreibt.

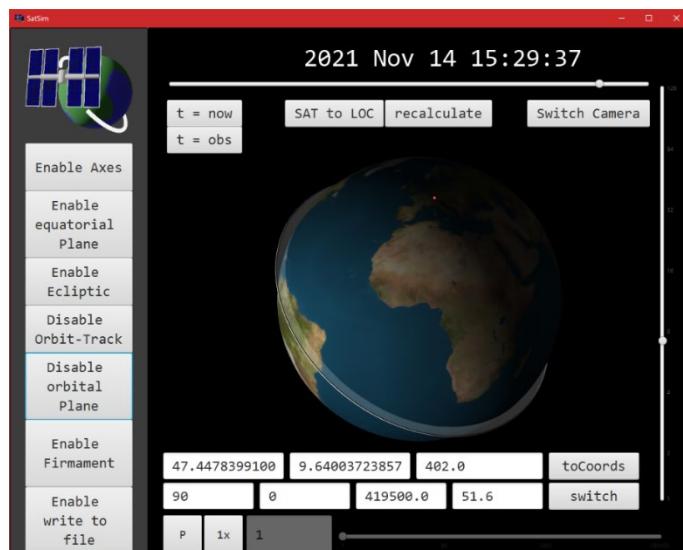


Abbildung 33: Screenshot von «Tarvos» mit ISS-Umlaufbahn (nach Schritt 4)

5.2.3 Datei auswerten

5.2.3.1 «simMode» bestimmen

In «Tarvos» steht er sogenannte «simMode» (Kurz für «simulation mode», da es aufgrund zweier möglicher Rotationsachsen auch zwei mögliche Simulationen der Umlaufbahn gibt) nach dem Ausrechnen der Satelliten-positionen auf 0. Er gibt an, welche der zwei möglichen Umlaufbahnen angezeigt wird. Mit Knopf 10 wird dieser zwischen 0 und 1 umgestellt.

Man schaut nun welche der Umlaufbahnen stimmen kann. dafür geht man mit Knopf 3 wieder zur Beobachtungszeit zurück. In jedem mir bekannten Fall gibt es eine eindeutig richtige und eine eindeutig falsche Umlaufbahn, denn die Momentane Satellitenposition muss vom Ort auf dem nächsten Punkt auf der Umlaufbahn liegen. Da die eingegebenen Satellitenposition ein Hochpunkt ist, muss der theoretische Hochpunkt der Umlaufbahn identisch sein. Hierfür ist das Standardmäßig eingeschaltete «Enable Orbit Track» aus 13 nützlich. Unten ein Beispiel mit einer falschen und einer richtigen Umlaufbahn.



Abbildung 34: negatives (links) und positives (rechts) Beispiel für die Auswahl der Umlaufbahn)

Man merkt sich, bei welchem «simMode» die richtige Umlaufbahn angezeigt wurde und kann dann weiterfahren.

5.2.3.2 Überflüge herauslesen

Die Output-Datei wird mit dem Namen «output.txt» im gleichen Ordner wie die Programmdatei abgespeichert. In dieser wird für beide «simModes» in chronologischer Reihenfolge aus den fünf Tagen nach dem Zeitpunkt der Beobachtung jeder Zeitpunkt, bei dem sich der Satellit im sichtbaren Himmelsbereich befindet, in fünf-Sekunden-Schritten mit Datum, Uhrzeit (UTC) und Elevation gespeichert. Falls ein Überflug zusätzlich zur richtigen Tageszeit geschieht, wird der Eintrag mit dem Präfix «Visible: » versetzt.

output2 07.11.2021.txt - Editor			
Datei	Bearbeiten	Format	Ansicht
SimMode: 0			
2021 Nov 07 05:25:21		ALT:22.0	
2021 Nov 07 05:25:26		ALT:23.0	
2021 Nov 07 05:25:31		ALT:24.0	
2021 Nov 07 05:25:36		ALT:25.0	
2021 Nov 07 05:25:41		ALT:26.0	
2021 Nov 07 05:25:46		ALT:27.0	
2021 Nov 07 05:25:51		ALT:28.0	
2021 Nov 07 05:25:56		ALT:29.0	
2021 Nov 07 05:26:01		ALT:31.0	
2021 Nov 07 05:26:06		ALT:32.0	
2021 Nov 07 05:26:11		ALT:33.0	
2021 Nov 07 05:26:16		ALT:35.0	
2021 Nov 07 05:26:21		ALT:36.0	
2021 Nov 07 05:26:26		ALT:38.0	
2021 Nov 07 05:26:31		ALT:39.0	
2021 Nov 07 05:26:36		ALT:41.0	
2021 Nov 07 05:26:41		ALT:42.0	
2021 Nov 07 05:26:46		ALT:44.0	
2021 Nov 07 05:26:51		ALT:45.0	
2021 Nov 07 05:26:56		ALT:47.0	
2021 Nov 07 05:27:01		ALT:49.0	
2021 Nov 07 05:27:06		ALT:50.0	

Abbildung 35: Ausschnitt einer Output-Datei

Nachdem man die Output-Datei geöffnet und am besten separat abgespeichert hat, kann man mit der Suchfunktion (Ctrl + F) des Windows-Texteditor nach dem Stichwort «simMode: » gefolgt von der Zahl des «simMode» mit der richtigen Umlaufbahn suchen. Dort angekommen kann man nun einen Überflug ausmachen, da dafür mehrere sichtbare Zeitpunkte aneinanderhängen. Sobald also der Zeitunterschied von einem Überflug zum nächsten grösser als 5 Sekunden ist, beginnt ein neuer Überflug. Falls «VISIBLE: » vor den Zeitpunkten steht, bedeutet es, dass «Tavros» den Überflug als sichtbar (d.h. zur richtigen Tageszeit) einstuft. Man kann einen Überflug nun charakterisieren, indem man den Beginn, das Ende und den Hochpunkt wie in Abbildung 31 und Tabelle 4 bestimmt.

Weiterhin kann man nun im Programm zu den Zeitpunkten des Überfluges navigieren und dabei wie in Abbildung 32 herausfinden in welcher Himmelsrichtung der Überflug stattfand. Für den Zweck dieser Arbeit wurden Himmelrichtungen bis zu einer Viertel der Hauptrichtungen angegeben (N, NNO, NO, ONO, O...) Diese können einfach in den Azimut umgerechnet werden, denn vom Norden aus entspricht jede Viertel-Hauptrichtung 22.5° .



Abbildung 37: Himmelsrichtung eines Überfluges bestimmen

→ Datum: 09.11.2021

	Zeit	Elevation	Himmelsrichtung	Azimut
Beginn	07:01:16	11°	WSW	225°
Hochpunkt	07:04:56	52°	NNW	337.5°
Ende	07:08:36	11°	ONO	67.5°

Beginn eines Überfluges (erster Eintrag)

Datei	Bearbeiten	Format	Ansicht	Hilfe
2021 Nov 09 05:31:36	ALT:12.0			
2021 Nov 09 05:31:41	ALT:11.0			
2021 Nov 09 05:31:46	ALT:11.0			
VISIBLE: 2021 Nov 09 07:01:16	ALT:11.0			
VISIBLE: 2021 Nov 09 07:01:21	ALT:11.0			
VISIBLE: 2021 Nov 09 07:01:26	ALT:12.0			
VISIBLE: 2021 Nov 09 07:01:31	ALT:12.0			
VISIBLE: 2021 Nov 09 07:01:36	ALT:13.0			
VISIBLE: 2021 Nov 09 07:01:41	ALT:14.0			
VISIBLE: 2021 Nov 09 07:01:51	ALT:15.0			
VISIBLE: 2021 Nov 09 07:01:56	ALT:15.0			
VISIBLE: 2021 Nov 09 07:02:01	ALT:16.0			
VISIBLE: 2021 Nov 09 07:02:06	ALT:17.0			
VISIBLE: 2021 Nov 09 07:02:11	ALT:18.0			
VISIBLE: 2021 Nov 09 07:02:21	ALT:19.0			
VISIBLE: 2021 Nov 09 07:02:26	ALT:20.0			
VISIBLE: 2021 Nov 09 07:02:31	ALT:20.0			
VISIBLE: 2021 Nov 09 07:02:36	ALT:20.0			
VISIBLE: 2021 Nov 09 07:02:41	ALT:22.0			
VISIBLE: 2021 Nov 09 07:02:46	ALT:23.0			
VISIBLE: 2021 Nov 09 07:02:51	ALT:24.0			
VISIBLE: 2021 Nov 09 07:02:56	ALT:25.0			
VISIBLE: 2021 Nov 09 07:03:01	ALT:26.0			
VISIBLE: 2021 Nov 09 07:03:06	ALT:27.0			
VISIBLE: 2021 Nov 09 07:03:11	ALT:28.0			
VISIBLE: 2021 Nov 09 07:03:16	ALT:29.0			
VISIBLE: 2021 Nov 09 07:03:21	ALT:30.0			
VISIBLE: 2021 Nov 09 07:03:26	ALT:31.0			
VISIBLE: 2021 Nov 09 07:03:31	ALT:33.0			
VISIBLE: 2021 Nov 09 07:03:36	ALT:34.0			
VISIBLE: 2021 Nov 09 07:03:41	ALT:35.0			
VISIBLE: 2021 Nov 09 07:03:46	ALT:37.0			
VISIBLE: 2021 Nov 09 07:03:51	ALT:38.0			
VISIBLE: 2021 Nov 09 07:03:56	ALT:39.0			
VISIBLE: 2021 Nov 09 07:04:01	ALT:41.0			
VISIBLE: 2021 Nov 09 07:04:06	ALT:42.0			
VISIBLE: 2021 Nov 09 07:04:11	ALT:44.0			
VISIBLE: 2021 Nov 09 07:04:16	ALT:45.0			
VISIBLE: 2021 Nov 09 07:04:21	ALT:46.0			
VISIBLE: 2021 Nov 09 07:04:26	ALT:48.0			
VISIBLE: 2021 Nov 09 07:04:31	ALT:49.0			
VISIBLE: 2021 Nov 09 07:04:36	ALT:50.0			
VISIBLE: 2021 Nov 09 07:04:41	ALT:51.0			
VISIBLE: 2021 Nov 09 07:04:46	ALT:51.0			
VISIBLE: 2021 Nov 09 07:04:51	ALT:52.0			
VISIBLE: 2021 Nov 09 07:04:56	ALT:52.0			
VISIBLE: 2021 Nov 09 07:05:01	ALT:52.0			
VISIBLE: 2021 Nov 09 07:05:11	ALT:51.0			
VISIBLE: 2021 Nov 09 07:05:16	ALT:50.0			
VISIBLE: 2021 Nov 09 07:05:21	ALT:49.0			
VISIBLE: 2021 Nov 09 07:05:26	ALT:48.0			
VISIBLE: 2021 Nov 09 07:05:31	ALT:46.0			
VISIBLE: 2021 Nov 09 07:05:36	ALT:45.0			
VISIBLE: 2021 Nov 09 07:05:41	ALT:44.0			
VISIBLE: 2021 Nov 09 07:05:46	ALT:42.0			
VISIBLE: 2021 Nov 09 07:05:51	ALT:41.0			
VISIBLE: 2021 Nov 09 07:05:56	ALT:39.0			
VISIBLE: 2021 Nov 09 07:06:01	ALT:38.0			
VISIBLE: 2021 Nov 09 07:06:06	ALT:37.0			
VISIBLE: 2021 Nov 09 07:06:11	ALT:35.0			
VISIBLE: 2021 Nov 09 07:06:16	ALT:34.0			
VISIBLE: 2021 Nov 09 07:06:21	ALT:33.0			
VISIBLE: 2021 Nov 09 07:06:26	ALT:31.0			
VISIBLE: 2021 Nov 09 07:06:31	ALT:30.0			
VISIBLE: 2021 Nov 09 07:06:36	ALT:29.0			
VISIBLE: 2021 Nov 09 07:06:41	ALT:28.0			
VISIBLE: 2021 Nov 09 07:06:46	ALT:27.0			
VISIBLE: 2021 Nov 09 07:06:51	ALT:26.0			
VISIBLE: 2021 Nov 09 07:06:56	ALT:25.0			
VISIBLE: 2021 Nov 09 07:07:01	ALT:24.0			
VISIBLE: 2021 Nov 09 07:07:06	ALT:23.0			
VISIBLE: 2021 Nov 09 07:07:11	ALT:22.0			
VISIBLE: 2021 Nov 09 07:07:16	ALT:21.0			
VISIBLE: 2021 Nov 09 07:07:21	ALT:20.0			
VISIBLE: 2021 Nov 09 07:07:26	ALT:20.0			
VISIBLE: 2021 Nov 09 07:07:31	ALT:19.0			
VISIBLE: 2021 Nov 09 07:07:36	ALT:18.0			
VISIBLE: 2021 Nov 09 07:07:41	ALT:17.0			
VISIBLE: 2021 Nov 09 07:07:46	ALT:17.0			
VISIBLE: 2021 Nov 09 07:07:51	ALT:16.0			
VISIBLE: 2021 Nov 09 07:07:56	ALT:15.0			
VISIBLE: 2021 Nov 09 07:08:01	ALT:15.0			
VISIBLE: 2021 Nov 09 07:08:06	ALT:14.0			
VISIBLE: 2021 Nov 09 07:08:11	ALT:14.0			
VISIBLE: 2021 Nov 09 07:08:16	ALT:13.0			
VISIBLE: 2021 Nov 09 07:08:21	ALT:12.0			
VISIBLE: 2021 Nov 09 07:08:26	ALT:12.0			
VISIBLE: 2021 Nov 09 07:08:31	ALT:11.0			
VISIBLE: 2021 Nov 09 07:08:36	ALT:11.0			
2021 Nov 09 08:38:26	ALT:11.0			
2021 Nov 09 08:38:31	ALT:11.0			
2021 Nov 09 08:38:36	ALT:12.0			
2021 Nov 09 08:38:41	ALT:12.0			
2021 Nov 09 08:38:46	ALT:13.0			

Hochpunkt (grösste Elevation)

Ende (letzter Eintrag)

Abbildung 36: Anleitung zur Auswertung einer Output-Datei

Tabelle 4: Ergebnisse von Abbildung 32

5.3 Funktionsweise

5.3.1 Grundstruktur

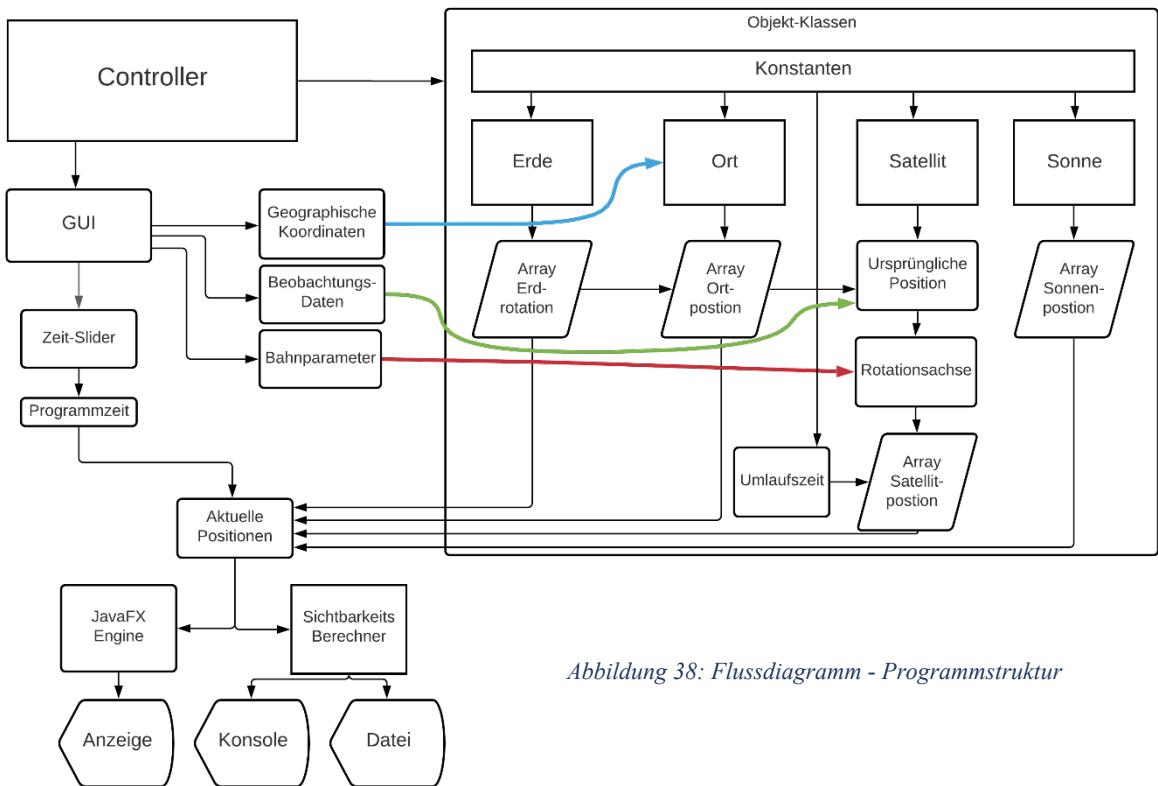


Abbildung 38: Flussdiagramm - Programmstruktur

Oben dargestellt ist eine vereinfachende Darstellung der Funktionsweise von «Tarvos». Es ist auf einer zentralen «**Controller-Klasse**» aufgebaut. Diese fungiert als eine Art «**Koordinations-Organ**», welches die anderen Teile des Programmes koordiniert und im Endeffekt das erzeugt, was auf dem Bildschirm angezeigt wird. Einerseits erzeugt die Controller-Klasse mittels eines FXML-Skripts das GUI (d.h. Graphical User Interface, Grafische Benutzeroberfläche), welches dem Nutzer erlaubt, die gemessenen Werte (Zeit, Azimut, Elevationswinkel), Standort-Informationen (Längengrad, Breitengrad, Meereshöhe) sowie Bahnparameter aus der Literatur (Inklination, Umlaufsbahnhöhe) einzugeben und die Simulation zu beobachten. Zudem erstellt es die Instanzen der «**Objekt-Klassen**», welche jeweils ein Aspekt der Simulation (z.B. die Erde, den Standort oder den Satelliten) numerisch berechnen. Diese Berechnungen werden jeweils in «**Arrays**» (in Abbildung 33: Rhomboiden) gespeichert, welches die Position oder Rotation in einer nach Zeit sortierten Liste aufbewahrt. Für die ausgewählte Zeit wird aus diesen Arrays die aktuelle Position herausgelesen, welche einerseits auf dem Bildschirm angezeigt sowie darauf untersucht wird, ob der eingegebene Satellit am eingegebenen Ort sichtbar ist, was in der Konsole in einer neuen Zeile notiert und in eine Datei geschrieben wird. In folgenden Unterkapiteln wird auf die genaue Funktionsweise dieser «**Programm-Organe**» genauer eingegangen.

5.3.2 Koordinatensysteme

Das von «Tarovos» benutzte Koordinatensystem basiert auf dem äquatorialen Koordinatensystem. Es baut jedoch nicht auf Kugelkoordinaten, sondern auf einem Achsenkoordinatensystem auf. In diesem bildet die XZ-Ebene die Äquatorialebene und die Y-Achse geht durch die geographischen Pole. (Siehe Abbildung 39)

Der Grund für dieses Koordinatensystem ist, dass die Umlaufbahn eines Satelliten dabei stationär bleibt und sich die Bewegungen der Erde und der Sonne einfacher in diesem Koordinatensystem darstellen lassen, als die Umlaufbahn eines Satelliten in einem anderen Koordinatensystem.

Als zeitlicher Nullpunkt wurde 12:00 Uhr des 1. März 2021 gewählt, da dies die «Tag-und-Nacht-Gleiche» darstellt und somit der Ursprung des äquatorialen Koordinatensystems ist, wo sich die Sonne genau auf der Z-Achse befindet.

5.3.3 Erde

Die Rotation der Erde kennt man als den bürgerlichen Tag mit 24 Stunden. Dies ist die Zeit, die die Erde benötigt, um sich so weit zu drehen, dass der gleiche Punkt wieder zur Sonne gerichtet ist. Aus diesem Grund kann man diesen Tag auch den Sonnentag nennen.

Vergleicht man jedoch die Rotation zu den Fixsternen (d.h. mit dem äquatorialen Koordinatensystem), fällt auf, dass die Sonne dafür eine kürzere Zeitspanne benötigt. Diese Zeitspanne nennt man den **«siderischen Tag»**. Er ist im Durchschnitt 3 Minuten und 55.9011 Sekunden kürzer als ein Sonnentag und beträgt somit rund 23 Stunden 56 Minuten 4,0989 Sekunden. Dies kann man ausrechnen, da es von Achsen-Präzession²² abgesehen in einem tropischen Jahr (Sonnenjahr, beträgt 365,242.190,52 Tage) ziemlich genau einen

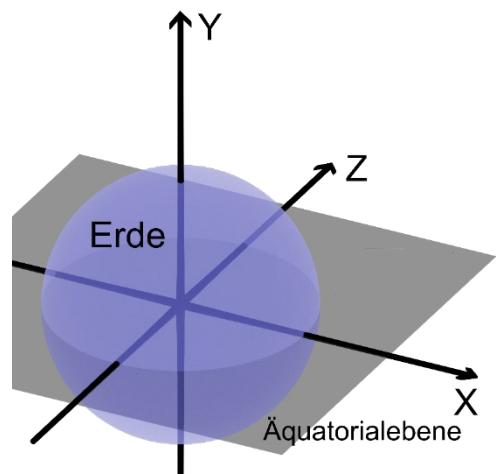
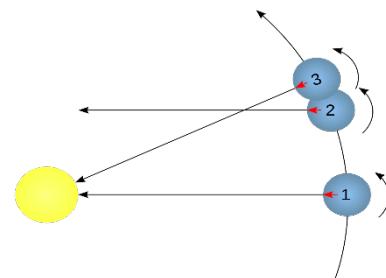


Abbildung 39: Koordinatensystem des Programmes



Ein Sonnentag (1 → 3) dauert jeweils länger als ein siderischer Tag (1 → 2), wenn Rotation und Revolution die gleiche Drehrichtung haben.

Abbildung 40: Sonnentag vs. siderischer Tag [10]

²² Siehe Anhang Kapitel 13.3.3 «Achsenpräzession» auf Seite 55

siderischen Tag mehr gibt, als es Sonnentage hat. Dadurch muss die Dauer eines Sonnentages um diesen Bruchterm kleiner sein:

$$t_{\text{sid. Tag}}[h] = 24 * 365.4219052 / (365.24219052 + 1) \approx 23.934471 h$$

Dieser Wert unterscheidet sich zum Literaturwert von $23,93447192 \text{ h}^{23}$ um etwa 0.1 Sekunden.

Dies kann nun umgewandelt werden in eine Drehgeschwindigkeit in $\frac{\text{grad}}{\text{Sekunde}} = \frac{360}{t_{\text{sid. Tag}}[s]}$. Das Programm rechnet damit für eine gewisse Anzahl an Tagen die zugehörige Rotation jeder fünften Sekunde aus und speichert sie in einem Array. In der «Konstanten-Klasse» werden die Anzahl an Tagen und die Unterteilungseinheit (welche standardmäßig 5 Sekunden beträgt) festgehalten. (Siehe Abbildung 38).

²³ Siehe Quelle [9]

5.3.4 Ort

Für die Zwecke dieser Arbeit bezieht sich das Wort «Ort» stets auf die geographischen Koordinaten des Beobachtungs-ortes im Koordinatensystem des Programmes. Nachdem der Nutzer die Koordinaten des Beobachtungsortes eingibt, muss das Programm diese in einen Punkt im kartesischen, dreidimensionalen Raum umwandeln. Dies geschieht mittels folgender Umwandlung²⁴. Siehe Abbildung 41.

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} r * \sin \theta * \cos \varphi \\ r * \sin \varphi * \cos \varphi \\ r * \cos \theta \end{pmatrix}$$

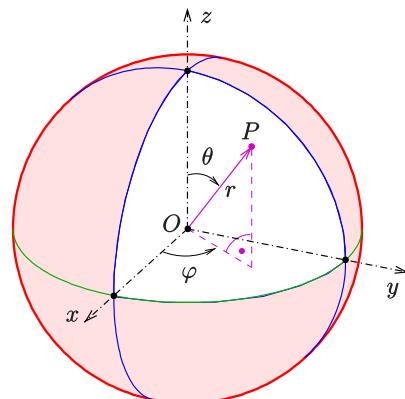


Abbildung 41: Kugelkoordinaten in XYZ-Achsenystem [13]

Das Programm benutzt statt des in Abbildung 36 gezeigten Koordinatensystems eines, bei dem die Y-Achse nach oben zeigt. Zudem wird aus dem Winkel zur X-Achse φ der **Längengrad** λ und aus dem Winkel zur Z-Achse θ der Gegenwinkel zum **Breitengrad** φ . Dazu kommt noch die geographische **Höhe** h . Dadurch lässt sich nun die Ursprungsposition einer beliebigen geographischen Position so berechnen:

$$\begin{pmatrix} x \\ y \\ z \end{pmatrix} = r * \begin{pmatrix} \cos \lambda * \cos \varphi \\ -\sin \lambda \\ \cos \lambda * \sin \varphi \end{pmatrix}$$

$$r = r_{Erde} + h$$

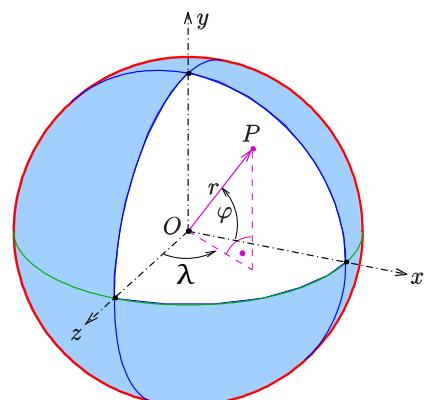


Abbildung 42: Kugelkoordinaten in Koordinatensystem des Programmes [13] (Bearbeitet: selbst)

Nun nutzt das Programm das von der «Erde-Klasse» bereits erzeugte Array der Erdrotation (siehe Abbildung 38) um den Ort mit dem gleichen Winkel um die Y-Achse zu drehen, wie es im Array für die Erdrotation gespeichert ist.

²⁴ Siehe Quelle [13]

5.3.5 Satellit

Der Satellit ist bei weitem das komplizierteste «Programm-Organ». Es beinhaltet mehrere Funktionen, welche nacheinander aufgerufen werden müssen; Als erstes wird die Position zum Zeitpunkt der Beobachtung festgelegt. Danach muss der Satellit die möglichen Positionen ausrechnen und in einem Array speichern. Diese Aufgabe besteht aus den Unteraufgaben der Bestimmung der Rotationsachse sowie der Umlaufszeit.

Als erstes erstellt es der Vektor²⁵ $v = \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$,

welcher kollinear zur X-Achse steht. (Siehe Abbildung 43)

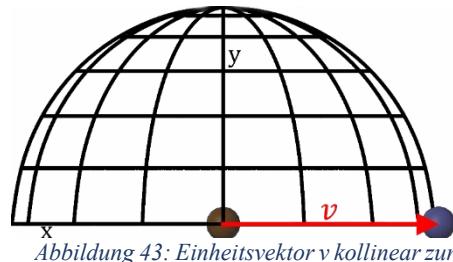


Abbildung 43: Einheitsvektor v kollinear zur X-Achse

Der Vektor v wird nun um den Elevationswinkel h um die Z-Achse gedreht. Damit erhält man den Vektor v' . (Siehe Abbildung 44)

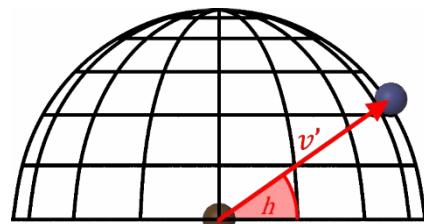


Abbildung 44: Einheitsvektor v' , gedreht gemäss Elevation h

Dieser Vektor v' wird nun zum Vektor v'' , indem er gemäss dem Azimut a um die Y-Achse rotiert wird. (Siehe Abbildung 44) Zusätzlich wird der Vektor im gleichen Schritt auch um den Breitengrad λ und um die Erdrotation (welches aus dem Array der Erdrotation gelesen wird) gedreht, da er für diese Zwecke um die gleiche Achse gedreht werden muss

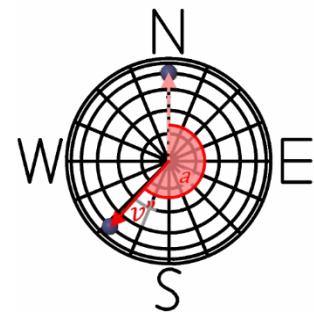


Abbildung 45: Einheitsvektor v'' , gedreht gemäss Azimut a und den Längengrad λ

Schliesslich wird der Vektor noch gemäss dem Breitengrad rotiert. Als Rotationsachse dient das Kreuzprodukt²⁶ der Y-Achse und des Ortsvektors

²⁵ Siehe Anhang: Lexikon; Seite 52

²⁶ Siehe Anhang: Lexikon; Seite 52

5.3.5.1 momentane Position berechnen

Wenn der Nutzer in Abbildung 30 (Seite 28) «SAT to LOC» drückt, bewegt sich der Satellit eingegebenen Angaben die horizontalen Wenn der Nutzer in Abbildung 30 (Seite 28) «SAT to LOC» drückt, bewegt sich der Satellit eingegebenen Angaben die horizontalen Koordinatensysteme gemäss an dementsprechenden Ort. Dies erreicht «Tarovs» auf folgende Weise:

5.3.5.2 zukünftige Positionen ausrechnen

5.3.5.2.1 Rotationsachsen bestimmen

Damit der Satellit nicht nur an der gleichen Stelle bleibt, muss sich der Satellit um die Erde drehen. Damit dies jedoch geht, braucht man eine Achse, um welche er sich dreht. Da diese Achse dem Normalvektor²⁷ der Umlaufbahnebene entspricht, weiss man folgendes über die Rotationsachse:

- Der Winkel der Rotationsachse zur XZ-Ebene entspricht der Bahninklination.
- Die momentane Position des Satelliten liegt in der Bahnebene

Demnach lassen sich die Rotationsachsen mittels präziderender Vektoren (Siehe Abbildung 46) herausfinden wovon zwei Zustände die momentane Satellitenposition in der Ebene liegen haben. Ob die momentane Satellitenposition in der Normalebene liegt, wird folgendermassen bestimmt:

Die Distanz D zur Ebene, zu der ein gegebener Normalvektor \vec{n} senkrecht steht und welche durch den Ursprung geht, lässt sich folgendermassen darstellen.

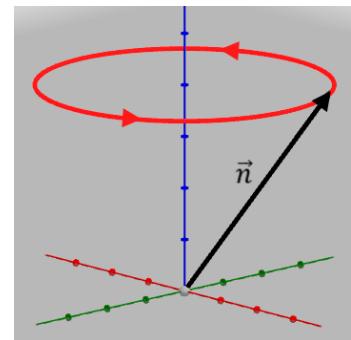


Abbildung 46: Präzession einer potentiellen Rotationsachse \vec{n}

$$\vec{n} = \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix}, \quad D = \frac{n_x * x + n_y * y + n_z * z}{|\vec{n}|}$$

Nun ist die Rotationsachse jedoch ein Einheitsvektor und hat somit den Betrag 1. Zudem kann man den Punkt (x, y, z) auch als Vektor \vec{p} darstellen. Dann bleibt lediglich noch:

$$D = \vec{n} \cdot \vec{p}$$

²⁷ Siehe Anhang: Lexikon; Seite 48

Wobei « \cdot » das Skalarprodukt der Vektoren darstellt.

Mathematisch sucht man den \vec{n} für $D = 0$ als Rotationsachse, jedoch geht das Programm nach dem Schema von Abbildung 46 numerisch vor, weswegen «Tarovos» dafür einen sehr kleinen Toleranzbereich beinhaltet. $\rightarrow D = \pm 0.01$.

5.3.5.2.2 Umlaufszeit bestimmen

Für Erdumlaufbahnen gilt folgende Gleichung²⁸, welche sich vom dritten Keplerschen Gesetz ableitet, bei Berechnen der Periode²⁹ T der Umlaufbahn mit Radius r :

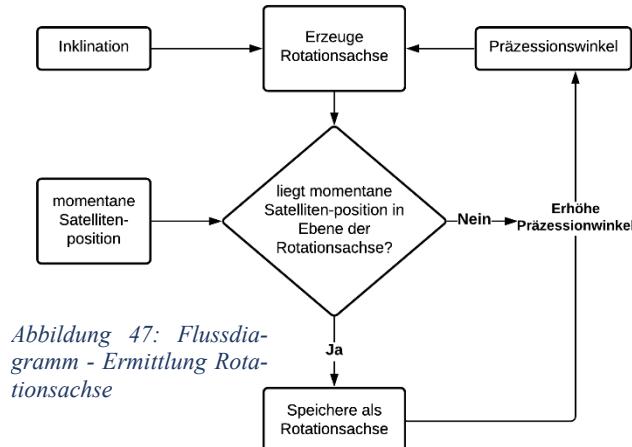


Abbildung 47: Flussdiagramm - Ermittlung Rotationsachse

$$r = r_{Erde} + h_{Satellit}, \quad T = 2 * \pi * \sqrt{\frac{r^3}{G * M_{Erde}}}$$

Tatsächlich ändert sich die Periode nicht wenn man die Exzentrizität³⁰ verändert. Demnach stimmen die Berechnungen der Umlaufszeit auch für elliptische Bahnen. Da das «Tarovos» jedoch den Satelliten mit gleichmässiger Höhe und Geschwindigkeit darstellt, ist die Darstellung von elliptischen Bahnen nicht realitätsgetreu.

5.3.5.2.3 Umrechnen in das Array

«Tarovos» benutzt nun diese Rotationsachsen und berechnet mit ihnen das Array der Satelliten-Positionen. Dies erreicht es, indem es zu jedem Zeitschritt den Startpunkt, welcher bei Kapitel 5.2.5.1 «momentane Position berechnen» auf Seite 37 um die Rotationsachse dreht. Der Winkel, mit dem diese Drehung erfolgt, wird entspricht dem Zeitunterschied zwischen dem zu verrechnenden Moment und dem Zeitpunkt der Beobachtung in Sekunden multipliziert mit der Anzahl Grad, welche der Satellit pro Sekunden zurücklegen muss. Diese Zahl kann mittel $\frac{360^\circ}{T}$ ermittelt werden.

Demnach entspricht der Winkel dieser Gleichung

²⁸ Siehe Quelle [13]

²⁹ Siehe Anhang: Lexikon; Seite 48

³⁰ Siehe Anhang: Lexikon; Seite 48

$$\phi(t)[deg] = (t - t_{start}) * \frac{360^\circ}{T}$$

Die Bibliothek, welche für das Programm benutzt wurde, beinhaltet eine Funktion, welche solche Drehungen durchführt. Mathematisch kann man es als eine multiplizieren einer Drehmatrix³¹ mit dem Ortsvektor des Startpunktes multipliziert: (\vec{n} repräsentiert eine der Rotationsachsen aus Kapitel 5.2.5.2.1 «Rotationsachsen bestimmen» auf Seite 38)

$$\vec{r}(\vec{n}, \phi) =$$

$$\begin{pmatrix} n_x^2(1 - \cos \phi) + \cos \phi & n_x n_y(1 - \cos \phi) - n_z \sin \phi & n_x n_z(1 - \cos \phi) + n_y \sin \phi \\ n_y n_x(1 - \cos \phi) + n_z \sin \phi & n_y^2(1 - \cos \phi) + \cos \phi & n_y n_z(1 - \cos \phi) - n_x \sin \phi \\ n_z n_x(1 - \cos \phi) - n_y \sin \phi & n_z n_y(1 - \cos \phi) - n_x \sin \phi & n_z^2(1 - \cos \phi) + \cos \phi \end{pmatrix} * \vec{r}_{start}$$

Dies rechnet «Tarovos» nun für jede Sekunde in den fünf Tagen nach der Beobachtung aus und notiert sich dies für beide möglichen Rotationsachsen in einem Array. Dieses Array wird in Abbildung 38 auf Seite 33 als «Array Satellitenpositionen» bezeichnet.

5.3.6 Sichtbarkeits-Rechner

In jedem Zeitschritt berechnet «Tarovos» nun, ob der Satellit vom geographischen Ort aus sichtbar ist. Dies geschieht folgendermassen:

In jedem Zeitschritt berechnet «Tarovos» nun, ob der Satellit vom geographischen Ort aus sichtbar ist. Dies geschieht folgendermassen:

- Der Sichtbarkeits-Rechner besorgt sich die aktuellen Positionen des Satelliten, des Ortes und der Sonne. Diese werden nun als Ortsvektoren benutzt.

$$\vec{r}_{Satellit} = \begin{pmatrix} x_{Satellit} \\ y_{satellit} \\ z_{satellit} \end{pmatrix}, \quad \vec{r}_{Ort} = \begin{pmatrix} x_{Ort} \\ y_{Ort} \\ z_{Ort} \end{pmatrix}, \quad \vec{r}_{Sonne} = \begin{pmatrix} x_{Sonne} \\ y_{Sonne} \\ z_{Sonne} \end{pmatrix}$$

- Aus diesen Vektoren werden die Vektoren vom Ort zum Satelliten \vec{w} und vom Ort zu der Sonne \vec{s} gebildet:

$$\text{Vom Ort zum Satelliten: } \vec{w} = \vec{r}_{Ort} - \vec{r}_{Satellit}$$

$$\text{Vom Ort zum zur Sonne: } \vec{s} = \vec{r}_{Ort} - \vec{r}_{Sonne}$$

- Nun kann das Skalarprodukt angewandt werden, um die Elevation des Satelliten herauszufinden. Die Elevation $h_{Satellit}$, lässt sich damit folgendermassen beschreiben:

$$h_{Satellit}[deg] = 90 - \cos^{-1} \left(\frac{\vec{w} \cdot \vec{r}_{Ort}}{|\vec{w}| * |\vec{r}_{Ort}|} \right)$$

³¹ Siehe Quelle [22]

Wenn $h_{Satellit} > 0^\circ$ ist, befindet sich der Satellit über dem Horizont, aber da solche Überflüge durch die Atmosphäre und mögliche Hindernisse der Landschaft selten gesehen werden können, wird es nur als Überflug gewertet, wenn $h_{Satellit} > 10^\circ$ ist.

4. Liegt ein Überflug vor, wird dasselbe auf die Elevation der Sonne angewandt:

$$h_{Sonne}[deg] = 90 - \cos^{-1}\left(\frac{\vec{s} \cdot \vec{r}_{Ort}}{|\vec{s}| * |\vec{r}_{Ort}|}\right)$$

Satelliten sind am Abend nur etwa 2 Stunden nach der Dämmerung und am Morgen etwa zwei Stunden vor der Dämmerung gut sichtbar. Demnach könnte man behaupten so lange $-30^\circ < h_{Sonne} < 0^\circ$ ($30^\circ = 2\text{h}$ in Zeitmass) sind die Satelliten sichtbar. Tatsächlich sind die Sonnenpositionen von «Tarovos» vereinfacht (elliptische Eigenschaft der Erdumlaufbahn wurde vernachlässigt), weswegen man am besten eine Toleranz von 5° ($=20$ Minuten) einfügt. Trifft nun $-35^\circ < h_{Sonne} < 5^\circ$ zu, wird der Überflug als «sichtbar» gespeichert.

Dieser Vorgang wird wie in Abbildung 47 für jeden Zeitschritt wiederholt.

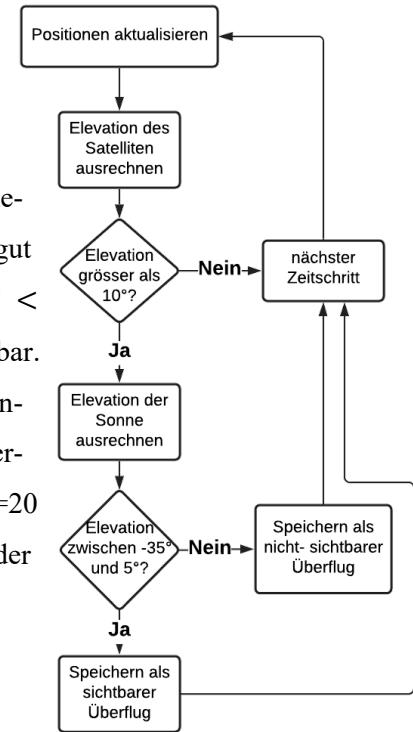


Abbildung 48: Flussdiagramm - Sichtbarkeitsrechner

6 Ergebnisse

6.1 Ergebnisse des Verfahrens

Nach dem Einsetzen der Daten aus Tabelle 2 (Seite 22) in «Tarovos» und auswerten der erhaltenen Output-Dateien wie in «5.1.3 Datei auswerten» (Seite 31) erhält man eine Liste mit sichtbaren Überflügen. Als Beispiel behandelt dieses Kapitel die Ergebnisse der Daten des 18.09.2021. Wie man in Tabelle 5 (folgende Seite) sieht, kann man damit den Beginn, den Hochpunkt und das Ende als Punkt im horizontalen Koordinatensystem angeben. Bei der Messung gibt es keine Angaben zu Start und Ende, da dort jeweils nur der Hochpunkt ausgewertet werden muss.

Um die Überflüge zu veranschaulichen, wurden die Überflüge gemäss der Messung während zwei Tagen mittels Excel als Projektionen analog zu einer Sternenkarte abgebildet. Damit die Spur der Überflüge auf der Karte richtig dargestellt werden können, muss die Blick-richtung vom Boden richtung Himmel aus geschehen, dh. Osten und Westen getauscht werden. Dabei ist zu beachten, dass lediglich die beschrifteten Punkte dem Programm entnommen worden sind und der Rest

Ergebnisse

mittels quadratischer Näherung für den Zweck der Anschaulichkeit interpoliert wurden. Diese Überflüge sind zudem in Tabelle 5 gelb markiert. Siehe Seite 43, Abbildung 50, Abbildung 49.

Tabelle 5: Auswertung «Tarovos»-Vorhersagen mit Daten des 18.09.2021

Hr = Himmelsrichtung

h: Elevation

a: Azimut

Messung	Beginn			Hochpunkt			Ende		
	Datum	Zeit (UTC)	Hr	h	a	Zeit (UTC)	Hr	h	a
18.09.2021	-	-	-	-	-	18:57:35	SE	57	148
19.09.2021	18:06:45	SSW	11	202.5	18:09:38	S0	28	135	18:12:30
19.09.2021	19:42:50	W	11	270	19:45:15	NNW	44	337.5	19:45:15
20.09.2021	17:20:15	S	11	180	17:21:45	S0	13	135	17:23:25
20.09.2021	18:54:45	SW	11	225	18:58:05	S0	66	135	19:01:25
Vorhersage	20.09.2021	20:31:45	W	11	270	20:32:05	N	13	0
	21.09.2021	18:07:05	SSW	11	202.5	18:13:05	S0	32	135
	21.09.2021	19:43:20	WSW	11	247.5	19:45:30	NW	36	315
	22.09.2021	18:55:10	SW	11	225	18:58:33	S0	75	135
	22.09.2021	17:20:15	S	11	180	17:22:15	S0	16	135
								17:24:15	0
								11	90

Ergebnisse

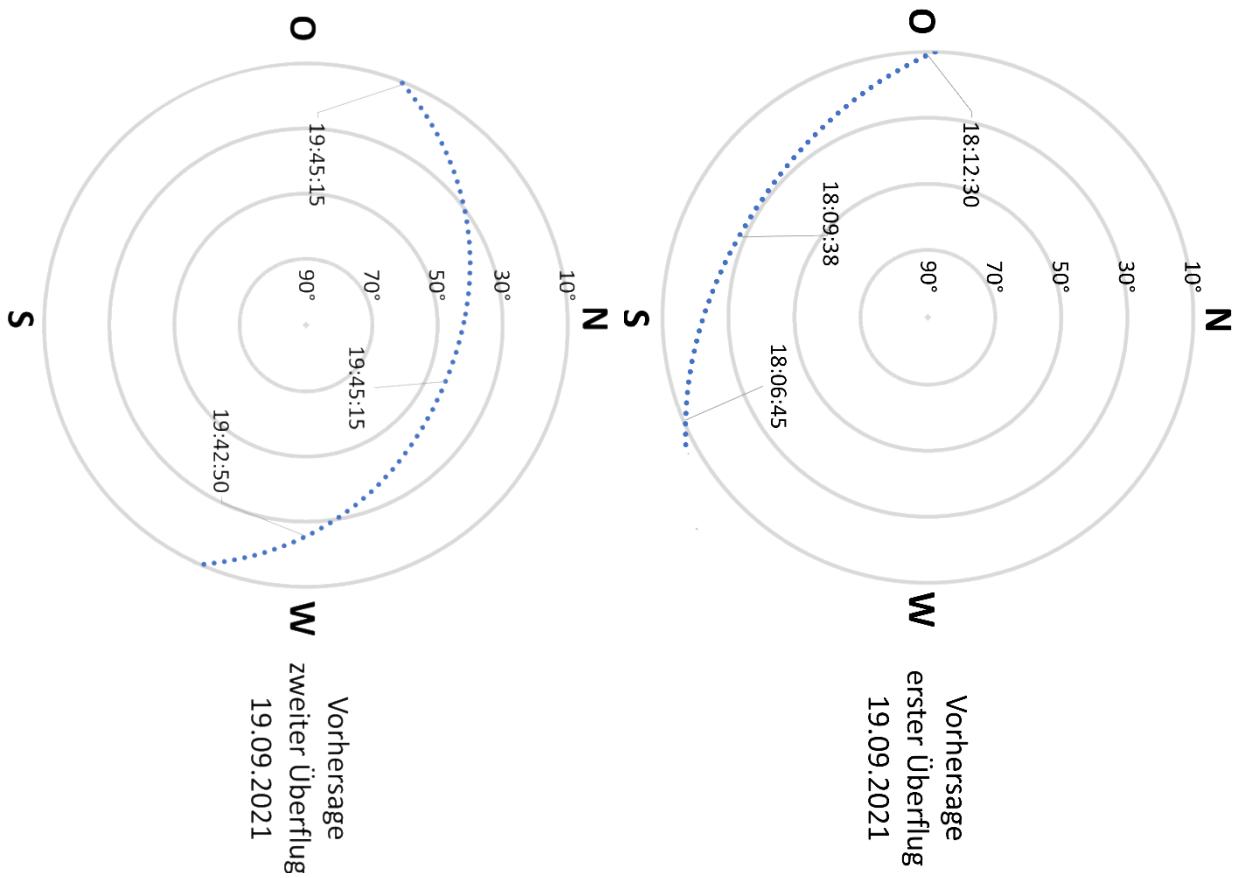


Abbildung 49: Überflugs-Vorhersage für den 19.09.2021 mit den Daten des 18.09.2021

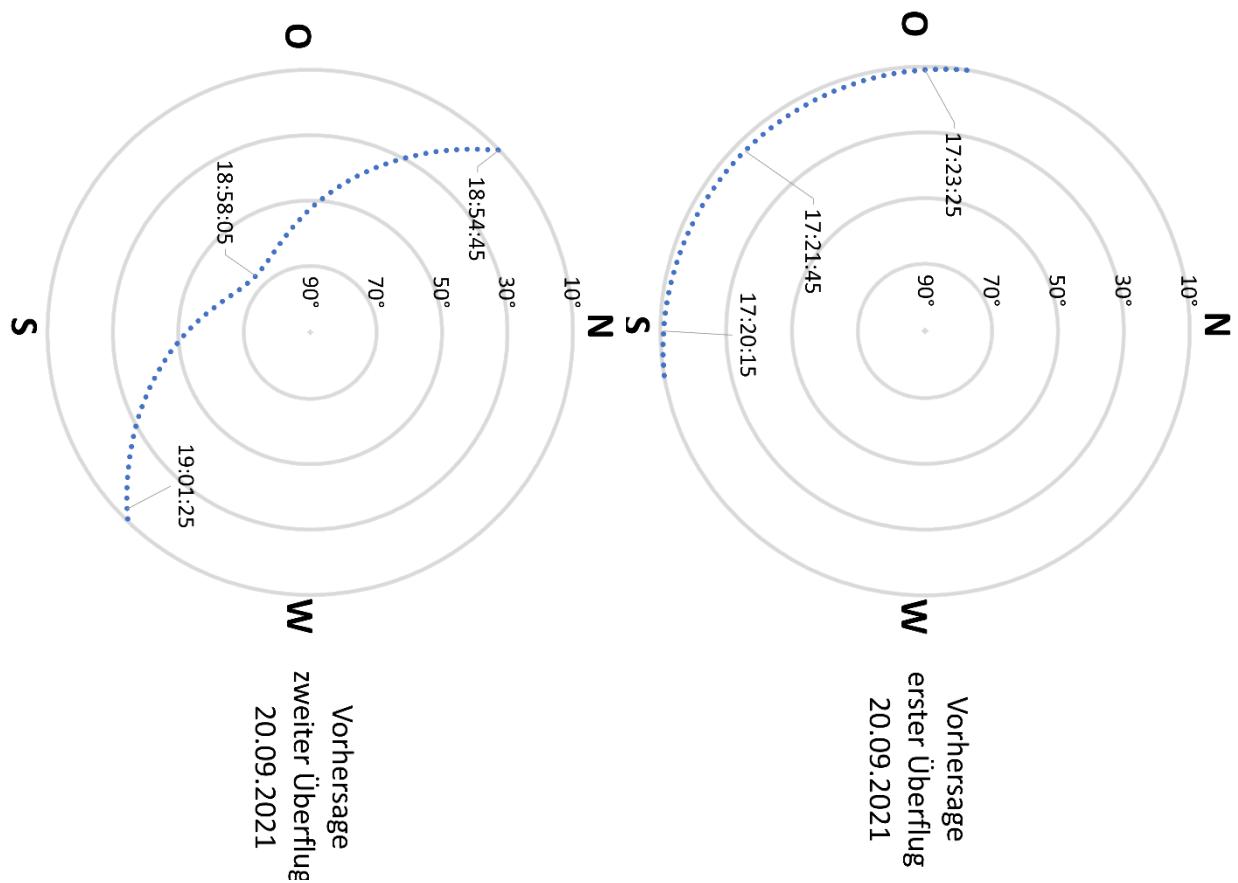


Abbildung 50: Überflugs-Vorhersage für den 20.09.2021 mit den Daten des 18.09.2021

6.2 Vergleich mit Literaturwerten

Um diese erhaltenen Überflüge auf die Genauigkeit zu prüfen, werden sie folgend mit archivierten Überflugs-Daten verglichen. Dafür habe ich «astro-viewer.net»³² verwendet, Nach Angaben des Erstellers entnimmt es von NASA die TLE³³ des gewünschten Datums und gibt aufgrund dessen vergangene Überflüge an.

Wie in Abbildung 50 zu sehen ist, stimmen die Anzahl der vorhergesagten Überflüge nicht genau mit den tatsächlichen Überflügen überein. Tatsächlich ist es so dass es weniger Überflüge gibt, als «Tarvos» vorhersagt.

In Abbildung 51 wird sichtbar, dass der Betrag des Zeitunterschiedes mit steigender Zeit zunimmt. (ein Punkt auf 0 würde bedeuten, dass die Vorhersage zeitlich exakt mit den Daten von «astroviewer.net» übereinstimmt. Über null bedeutet, dass die Vorhersagen später als der tatsächliche Überflug geschieht und unter null beschreibt das Gegenteil davon.). Wobei jedoch die Überflüge stets früher vorhergesagt werden als sie stattfinden. Besondere Aufmerksamkeit muss dem ersten

Punkt im Diagramm geschenkt werden, da dieser die Messung repräsentiert und somit zeigt, dass die Messung um etwa 2 Minuten und 14 Sekunden von den Werten aus «astroviewer.net» abweichen und somit im Vergleich als «zu spät» erscheint.

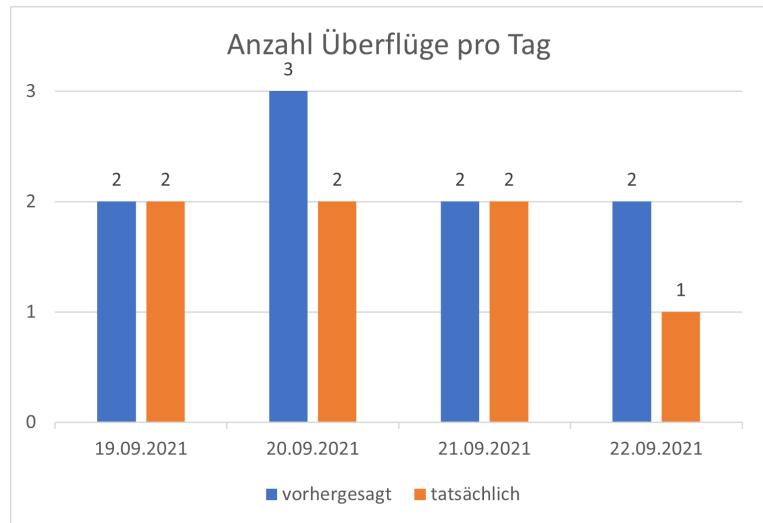


Abbildung 51: Anzahl Überflüge pro Tag für fünf Tage nach dem 18.09

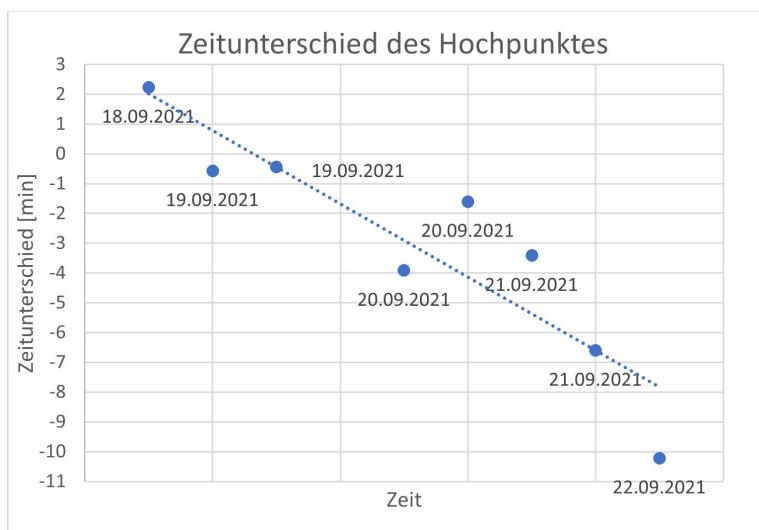


Abbildung 52: Zeitunterschied des Hochpunktes mit Vorhersagen aus den Daten des 18.09.2021

³² Siehe Quelle [16]

³³ Siehe Anhang: Lexikon; Seite 52

Diskussion

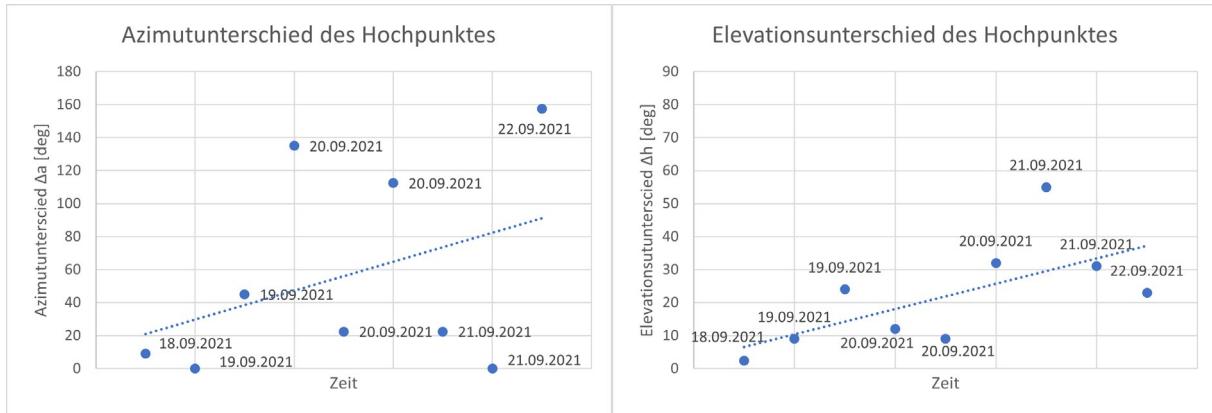


Abbildung 53: Azimut- und Elevationsunterschied des Hochpunktes für Vorhersagen aus Daten den 18.09.2021

Was man aus Abbildung 52 entnehmen kann, ist, dass mit zunehmender Zeit der Unterschied zwischen vorhergesagtem Azimut sowie Elevation und den tatsächlichen Werten ebenfalls zunimmt.

Wenn man in Abbildung 51 und Abbildung 52 jeweils den ersten Punkt nach der Messung betrachtet (19.09.2021) fällt auf, dass dieser Punkt eine sehr Geringe Abweichung mit von dem Wert von «astroviewer.net» aufweist. Wenn man dies mit weiteren Datensätzen wie in Kapitel 13.4.2 «Literaturvergleiche weiterer Überflüge» auf Seite 56 vergleicht, so kann man erkennen, dass sich das Muster, das der erste Überflug nach der Messung vergleichsweise gut mit den Daten aus «astroviewer.net» übereinstimmt, sich durch alle Vorhersagen zieht.

7 Diskussion

Die aus Kapitel 6.2 erlernten Einsichten kann man folgendermassen verstehen. Die Unterschiede zwischen der Anzahl Überflüge pro Tag, den Zeit des Hochpunkts sowie den Azimut- und Elevationsunterschied lassen sich als Resultat einer Summe von Ungenauigkeiten erklären. Je weiter die Vorhersagen zeitlich von der Messung entfernt ist, desto merkbarer ist deren Einfluss. Diese lassen sich einteilen in Ungenauigkeiten im Programm und in äussere Ungenauigkeiten.

7.1 Ungenauigkeiten in der Simulation

Einer der kleineren Faktoren ist die Art der Bestimmung der Rotationsachse wie in Kapitel 5.3.5.2.1 «Rotationsachsen bestimmen» auf Seite 38 numerisch und mit einem Toleranzbereich verläuft. Dadurch ist die Lösung leicht anders, als wenn man die Rotationsachse rechnerisch bestimmen könnte.

Diskussion

Danach sollte man die Präzession von Satelliten³⁴, welche von der Erdfigur ausgelöst wird betrachten, welche von der Inklination des Satelliten abhängig ist. Man kann diese auch gut berechnen, solange man jedoch nicht von Sonnen-Synchronen Satelliten ausgeht, kann deren Einfluss auf den Zeitskalen, welche für diese Arbeit gewählt wurden (fünf Tage) nicht ausschlaggebend für die Abweichungen sein.

Weiter betrachtet man den Zerfall³⁵ der Umlaufbahnen. Vor allem, dadurch, dass LEO-Satelliten in Kontakt mit teilen der Atmosphäre sind, welche zwar eine kleine Dichte aufweisen, welche aber nicht vernachlässigbar ist und somit von der Atmosphäre langsam «gebremst» werden und durch diverse andere Faktoren wie gezeitenbedingte Beschleunigung sowie Licht- und Wärmestrahlung zerfällt die Umlaufbahn und wird somit stehts Energie-ärmer, nimmt also an Höhe ab, was dafür die Geschwindigkeit erhöht und somit die Umlaufszeit beeinflusst. Dieser Faktor wurde nicht berücksichtigt und ist mit grosser Wahrscheinlichkeit einer der Hauptauslöser für den Unterschied zwischen Literaturwert und Simulation

Eine weitere Ungenauigkeit ist der vernachlässigte Ellipsencharakter der Umlaufbahnen. Er allein ändert, bei genügend kleiner Exzentrizität (was bei der ISS gegeben ist), an der Umlaufszeit des Satelliten nichts, sondern beeinflusst nur deren Form, welche bei geringer Exzentrizität jedoch auf das Ergebnis auch keinen grossen Einfluss hat. Aber dafür sie ist ausschlaggebend für eine Ungleiche Verteilung des oben erwähnten Zerfalls der Umlaufbahn und für die Präzession der Umlaufbahn. Man kann demnach behaupten, dass der elliptische Charakter es erschwert die anderen Faktoren zu berücksichtigen.

7.2 äussere Ungenauigkeiten

Als kleinere Ungenauigkeit ausserhalb des Programmes kann man die Ungenauigkeiten, welche beim Messen entstehen ansehen. Hierbei ist vor allem die Messung mit «AstroImageJ» gemeint, bei welchem das exakte beim Bestimmen der Start und Endpunkte nur selten auf den Pixel genau geschehen kann. Der Grössere Einfluss hat aber wahrscheinlich, dass man den Durchschnitt von Start und Endpunkt berechnet, da die Satellitenspur bei längeren Belichtungszeiten bereits keine Gerade mehr ist, sondern eine Kurve und damit der Mittelpunkt von Start- und Endpunkt nicht exakt auf dieser Kurve liegen.

Abgesehen davon kann man die von aussen angenommener Höhe (welche Jeweils über das «Orbit»-Fenster von «heavens-above.com» der grossen Halbachse entnommen worden ist) als

³⁴ Siehe Quelle [27]

³⁵ Siehe Quelle [28]

Ungenauigkeit ansehen, da es nicht auf einer eigenen Messung basiert. Dasselbe kann man über die Inklination behaupten.

Dadurch kann die Fragestellung nicht mit einem definitiven «Ja» beantwortet werden genau diese Grössen nicht von der Messung selbst bestimmt werden können.

7.3 Verbesserungsmöglichkeiten

Ein Weg die Vorhersagen zu verbessern wäre nicht nur mehr Beobachtungen durchzuführen, sondern auch gleichzeitige Beobachtungen desselben Satelliten von wesentlich unterschiedlichen Perspektiven. Dies würde es erlauben, die Höhe das Satelliten auszurechnen, anstatt sich auf einen Literaturwert verlassen zu müssen.

Zudem lässt die Nutzerfreundlichkeit von «Tarovos» zu wünschen übrig, da es die Daten nicht präsentieren kann, sondern nur als lange Datei ausspuckt. Tatsächlich ist das Navigieren mit der Zeit auch keine einfache Sache und das Umstellen des Jahres muss man bisher in den Programmcode greifen und ihr verändern, was keine Lösung ist, falls man das Programm auch noch in einigen Monaten mit aktuellen Daten verwenden können will.

Tatsächlich wird im Rahmen des Nationalen Wettbewerbs von «Schweizer Jugend Forscht» diese Arbeit weitergeführt, weswegen ich mir erhoffe manche der Ungenauigkeiten beheben zu können sowie mehr Messdaten zu sammeln und andere Satelliten als nur die ISS auszuwerten.

8 Literaturverzeichnis

- [1] „Wikipedia - Geo Synchronous Orbit,“ [Online]. Available: https://en.wikipedia.org/wiki/Geosynchronous_orbit. [Zugriff am 23 11 2021].
- [2] „Wikipedia - Präzession - Präzession der Erdachse,“ [Online]. Available: https://de.wikipedia.org/wiki/Pr%C3%A4zession#Pr%C3%A4zession_der_Erdachse. [Zugriff am 02 12 2021].
- [3] „Converalot - Celestial Coordinate Converter,“ 2004. [Online]. Available: http://www.converalot.com/celestial_horizon_co-ordinates_calculator.html. [Zugriff am 12 11 2021].
- [4] „Heavens Above,“ [Online]. Available: <https://www.heavens-above.com/orbit.aspx?satid=25544>. [Zugriff am 12 11 2021].

Literaturverzeichnis

- [5] „Nasa - ISS Galerie.“ 4 10 2018. [Online]. Available: https://www.nasa.gov/mission_pages/station/images/station_post_construction/index.html. [Zugriff am 12 11 2021].
- [6] „Nasa - ISS Steckbrief.“ [Online]. Available: <https://www.nasa.gov/content/international-space-station-length-and-width/#.YY6-NGDMJaQ>. [Zugriff am 12 11 2021].
- [7] „Nasa - ISS Steckbrief.“ [Online]. Available: https://www.nasa.gov/mission_pages/station/multimedia/config.html. [Zugriff am 12 11 2021].
- [8] „Öffentliche Universität Louisville - AstroImageJ.“ 5 8 2021. [Online]. Available: <https://www.astro.louisville.edu/software/astroimagej/>. [Zugriff am 12 11 2021].
- [9] „Scheinbare Helligkeit - Wikipedia.“ [Online]. Available: https://de.wikipedia.org/wiki/Scheinbare_Helligkeit. [Zugriff am 31 10 2021].
- [10] „Wikipedia - Siderischer Tag.“ [Online]. Available: https://de.wikipedia.org/wiki/Siderischer_Tag. [Zugriff am 19 11 2021].
- [11] „Wikipedia - astronomische Koordinatensysteme.“ [Online]. Available: https://de.wikipedia.org/wiki/Astronomische_Koordinatensysteme#Relative_Koordinatensysteme. [Zugriff am 6 11 2021].
- [12] „Wikipedia - Julianisches Datum.“ [Online]. Available: https://de.wikipedia.org/wiki/Julianisches_Datum. [Zugriff am 12 11 2021].
- [13] „Wikipedia - Kugelkoordinaten.“ [Online]. Available: <https://de.wikipedia.org/wiki/Kugelkoordinaten#Umrechnungen>. [Zugriff am 21 11 2021].
- [14] A. Hanselmeier, «Einführung in die Astronomie und Astrophysik,» Spektrum Akademischer Verlag, 2007, p. 85.
- [15] „Wikipedia - Low Earth Orbit.“ [Online]. Available: https://en.wikipedia.org/wiki/Low_Earth_orbit. [Zugriff am 23 11 2021].
- [16] TWCarlson, „Wikimedia Commons.“ [Online]. Available: https://commons.wikimedia.org/wiki/File:Azimuth-Altitude_schematic.svg. [Zugriff am 24 11 2021].
- [17] D. Matussek, „AstroViewer.“ [Online]. Available: <https://www.astrovieviewer.net/iss/en/observation.php>. [Zugriff am 28 11 2021].
- [18] „Wikipedia - Newtonsches Gravitationsgesetz.“ [Online]. Available: https://de.wikipedia.org/wiki/Newtonssches_Gravitationsgesetz. [Zugriff am 28 11 2021].
- [19] „Wikipedia - Satellitenorbit, Teilkapitel "Vereinfachter Überblick der Umlaufbahnen".“ [Online]. Available: https://de.wikipedia.org/wiki/Satellitenorbit#Vereinfachter_%C3%9Cberblick_der_Umlaufbahnen. [Zugriff am 28 11 2021].
- [20] „Wikipedia - Internationale Raumstation - Einleitung.“ [Online]. Available: https://de.wikipedia.org/wiki/Internationale_Raumstation. [Zugriff am 29 11 2021].
- [21] „Wikipedia - Chinesische Raumstation - Einleitung.“ [Online]. Available: https://de.wikipedia.org/wiki/Chinesische_Raumstation. [Zugriff am 29 11 2021].
- [22] Tubas, „Wikimedia Commons - Inclined Geosynchronous Orbit (IGSO).“ [Online]. Available: <https://commons.wikimedia.org/wiki/File:IGso3063.jpg>. [Zugriff am 29 11 2021].
- [23] „Wikipedia - Drehmatrix.“ [Online]. Available: https://de.wikipedia.org/wiki/Drehmatrix#Drehmatrizen_des_Raumes_%E2%84%9D%E2%81%BF. [Zugriff am 29 11 2021].
- [24] „Wikimedia Commons.“ [Online]. Available: https://commons.wikimedia.org/wiki/File:Form_der_Erde.jpg. [Zugriff am 30 11 2021].

Abbildungsverzeichnis:

- [25] „Gluon - Scene Builder,“ [Online]. Available: <https://gluonhq.com/products/scene-builder/>. [Zugriff am 01 12 2021].
- [26] „JetBrains - IntelliJ IDEA,“ [Online]. Available: <https://www.jetbrains.com/de-de/idea/>. [Zugriff am 01 12 2021].
- [27] „Blender,“ [Online]. Available: [Blender.org](https://www.blender.org). [Zugriff am 01 12 2021].
- [28] „Wikipedia - Sonnensynchrone Umlaufbahn, Berechnung,“ [Online]. Available: https://de.wikipedia.org/wiki/Sonnensynchrone_Umlaufbahn. [Zugriff am 01 12 2021].
- [29] „Wikipedia - Orbital Decay - Modelling Orbital Decay,“ [Online]. Available: https://en.wikipedia.org/wiki/Orbital_decay. [Zugriff am 01 12 2021].

9 Abbildungsverzeichnis:

Abbildung 1: Kreisbewegung.....	5
Abbildung 2: relevante Bahnelemente	6
Abbildung 3: Bodenspur zweier IGSO-Satelliten [21]	7
Abbildung 4: Mondaufnahme des 18.09.2021 zwischen Satellitenbeobachtungen	7
Abbildung 5: Darstellung horizontales Koordinatensystem mit Elevation h und Azimut a. ursprüngliches Bild: [15], Bearbeitet: selbst.....	8
Abbildung 6: das Äquatoriale Koordinatensystem.....	9
Abbildung 7: ISS am 4. Okt. 2018 aus Sicht der Soyuz-Kapsel der «Expedition 56» [5].....	10
Abbildung 8: Kamera (Canon EOS 250D) mit Objektiv (SIGMA 20mm F1,4 DG HSM Art)	11
Abbildung 9: Stativ eingefahren, 60 cm (Hama Star 62)	11
Abbildung 10: Stativ ausgefahren, 160 cm (Hama Star 62).....	11
Abbildung 11: negatives Beispiel des Wetterradars der App "MeteoSwiss"	13
Abbildung 12: Logo von AstroImageJ [8]	14
Abbildung 13: Anleitung zum Importieren mehrerer Bilder in AstroImageJ	14
Abbildung 14: Eintragen des API-Keys	14
Abbildung 15: Plate-Solving Einstellungen in AstroImageJ.....	15
Abbildung 16: Auffindungsort des Astrometry Tools.....	15
Abbildung 17: Benutzung des Astrometry Tools.....	16
Abbildung 18: Ausschnitt des Daten-Blatts der Excel-Tabelle.....	17
Abbildung 19: ISS durch Orion (07.11.2021).....	21
Abbildung 20: ISS über Sonnenuntergang (31.05.2021)	21
Abbildung 21: 18.09.2021 im äquatorialen Koordinatensystem.....	23
Abbildung 22: 31.05.2021 im äquatorialen Koordinatensystem.....	23

Tabellenverzeichnis

Abbildung 23: 07.11.2021 im äquatorialen Koordinatensystem.....	23
Abbildung 24: 18.09.2021 im horizontalen Koordinatensystem.....	24
Abbildung 25: 31.05.2021 im horizontalen Koordinatensystem.....	24
Abbildung 26: 07.11.2021 im horizontalen Koordinatensystem.....	25
Abbildung 27: GUI des Programmes in Gluon Scene Builder geladen	26
Abbildung 28 Logo von «Gluon Scene Builder» [24]	27
Abbildung 29 Logo von «Blender» [26]	27
Abbildung 30: Logo von «IntelliJ IDEA» [25]	27
Abbildung 31: beschrifteter Screenshot von «Tarfos»	28
Abbildung 32: Logo von «Tarfos», erstellt mit Blender 2.93.6.....	28
Abbildung 33: Screenshot von «Tarfos» mit ISS-Umlaufbahn (nach Schritt 4)	30
Abbildung 34: negatives (links) und positives (rechts) Beispiel für die Auswahl der Umlaufbahn)....	31
Abbildung 35: Ausschnitt einer Output-Datei.....	31
Abbildung 36: Anleitung zur Auswertung einer Output-Datei	32
Abbildung 37: Himmelsrichtung eines Überfluges bestimmen	32
Abbildung 38: Flussdiagramm - Programmstruktur.....	33
Abbildung 39: Koordinatensystem des Programmes	34
Abbildung 40: Sonnentag vs. siderischer Tag [10]	34
Abbildung 41: Kugelkoordinaten in XYZ-Achsenystem [13].....	36
Abbildung 42: Kugelkoordinaten in Koordinatensystem des Programmes [13] (Bearbeitet: selbst)....	36
Abbildung 43: Einheitsvektor v kollinear zur X-Achse	37
Abbildung 44: Einheitsvektor v', gedreht gemäss Elevation h.....	37
Abbildung 45: Einheitsvektor v'', gedreht gemäss Azimut a und den Längengrad λ	37
Abbildung 46: Präzession einer potenziellen Rotationsachse n	38
Abbildung 47: Flussdiagramm - Ermittlung Rotationsachse	39
Abbildung 48: Flussdiagramm - Sichtbarkeitsrechner	41
Abbildung 49: Überflugs-Vorhersage für den 19.09.2021 mit den Daten des 18.09.2021	43
Abbildung 50: Überflugs-Vorhersage für den 20.09.2021 mit den Daten des 18.09.2021	43
Abbildung 51: Anzahl Überflüge pro Tag für fünf Tage nach dem 18.09	44
Abbildung 52: Zeitunterschied des Hochpunktes mit Vorhersagen aus den Daten des 18.09.2021	44
Abbildung 53: Azimut- und Elevationsunterschied des Hochpunktes für Vorhersagen aus Daten den 18.09.2021	45
Wenn nicht anders vermerkt ist, wurde die Abbildung selbst erstellt.	

10 Tabellenverzeichnis

Tabelle 1: Umrechnung in Horizontales Koordinatensystem Anhand von Daten des 18.09.2021.....	22
Tabelle 2: Koordinaten der Hochpunkte (ISS Beobachtungen)	22

Dank

Tabelle 3: Legende zu Abbildung Abbildung 27	28
Tabelle 4: Ergebnisse von Abbildung 32	32
Tabelle 5: Auswertung «Tarvos»-Vorhersagen mit Daten des 18.09.2021.....	42

11 Dank

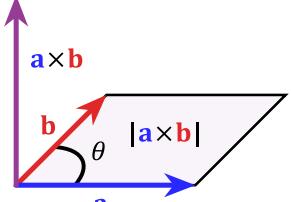
Zum Schluss möchte ich mich herzlichst bei Herrn Götz bedanken, welcher diese Arbeit stets offen für Fragen und hilfsbereit mentoriert hat. Zudem danke ich Maria Jäckle, welche in dieser Arbeit durch das Korrekturlesen geholfen hat. Zudem möchte ich meiner Familie und meinen Bekannten danken, welche mich während der Arbeit unterstützt haben.

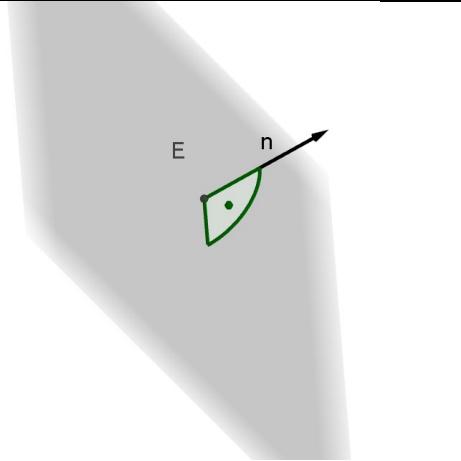
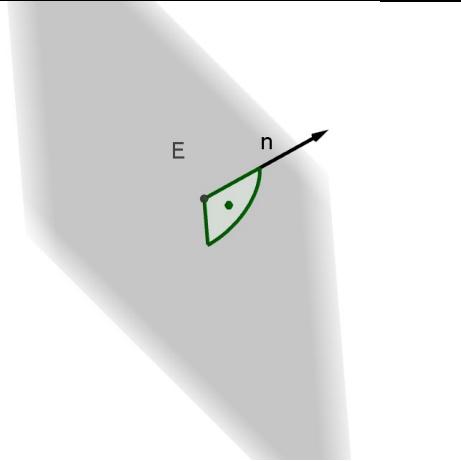
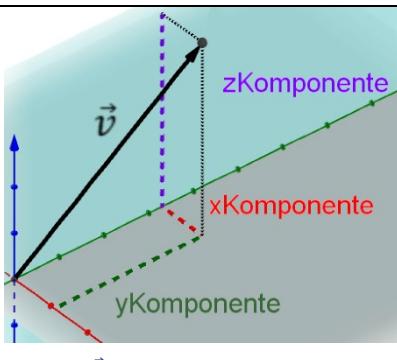
12 Selbstständigkeitserklärung

Der Unterzeichnete bestätigt mit seiner/ihrer Unterschrift, dass die Arbeit selbstständig verfasst und in schriftliche Form gebracht worden ist, dass sich die Mitwirkung anderer Personen auf Beratung und Korrekturlesen beschränkt hat und dass alle verwendeten Unterlagen und Gewährspersonen aufgeführt sind. Er gibt das Einverständnis, dass diese Arbeit zur Abklärung künftiger möglicher Plagiate Dritter beigezogen werden darf. Er weiss, dass die erstellte Arbeit erst nach Abschluss der Maturaarbeit veröffentlicht und frühestens nach Ablauf der Rekursfrist verwertet werden darf.

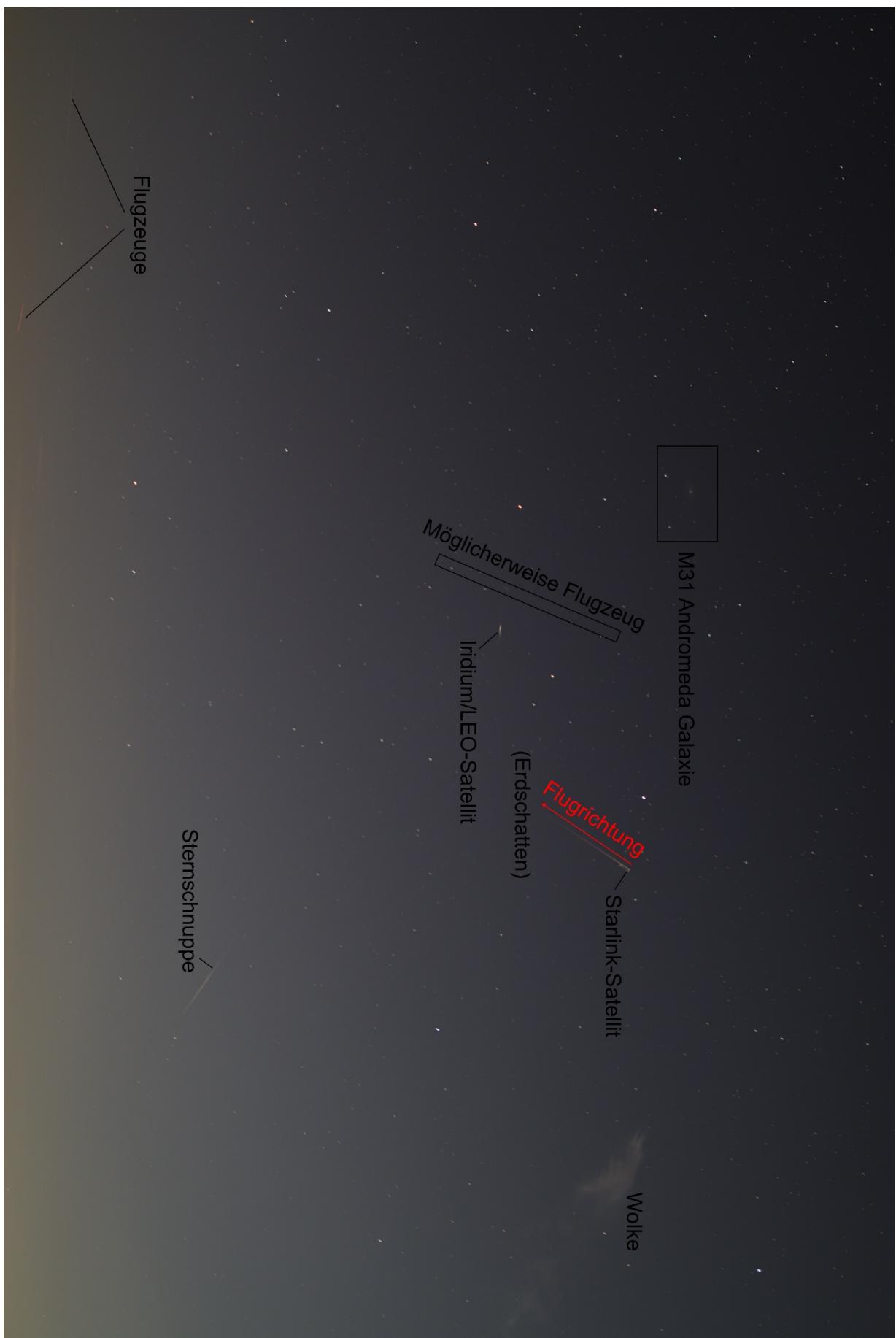
13 Anhang

13.1 Lexikon

Begriff	Erklärung
Array	Ein Array ist eine digital gespeicherte geordnete Liste an Werten
Azimut	Der Azimut h ist die Winkeldistanz einer Position am Himmel zur nördlichen Himmelsrichtung. Siehe Kapitel «3.3.1 Das horizontale Koordinatensystem» Seite 8
Deklination	Die vertikale Komponente im rotierenden äquatorialen Koordinatensystem. Siehe Kapitel 3.3.2 «Das äquatoriale Koordinatensystem» auf Seite 9.
Elevation	Die Elevation h ist der Winkel zwischen einer Position am Himmel und dem Horizont (Negativer Wert: Position befindet sich unter dem Horizont) Siehe Kapitel «3.3.1 Das horizontale Koordinatensystem» Seite 8
Exzentrizität	Die Exzentrizität ist das Mass, welches angibt, wie stark elliptisch eine Umlaufbahn ist. Im Kontext gilt je grösser die Exzentrizität, desto grösser ist der Unterschied zwischen dem erdnächsten und erdffernsten Punkt auf der Umlaufbahn.
GSO	Geo Synchronous Orbit , Umlaufbahn, dessen Periode 23h 56 min und 4 sec (Höhe: 35,786 km) beträgt. Satellit scheint sich im Nachthimmel dadurch nicht stark zu bewegen [1]
GUI	Graphical User Interface : Die Nutzeroberfläche des Programmes.
Inklination	Die Inklination i oder auch Bahnneigung gibt den Winkel zwischen der Bahn-ebene und der Äquatorialeben an Siehe Kapitel « » auf Seite 5 für Abbildung
Kreuzprodukt	Ein Vektor, welcher senkrecht zu zwei gegebenen Vektoren senkrecht steht: $\vec{a} \times \vec{b} = \begin{pmatrix} a_y * b_z - a_z * b_y \\ a_z * b_x - a_x * b_z \\ a_x * b_y - a_y * b_x \end{pmatrix}$ 
LEO	Low Earth Orbit , Erdnahe Umlaufbahn (zwischen 200 und 2000km Höhe) Siehe Kapitel 3.2.2 «Höhe» auf Seite 6
Library	(Auf Deutsch: Bibliothek) Ansammlung an öffentlichem Code, welche man bezieht um eine Auswahl an funktionen zu haben.

Normalebene	Eine Normalebene ist eine Ebene (In Abbildung rechts: E) welche Senkrecht zu einem Vektor steht (In Abbildung rechts: n)	
Normalvektor	Ein Normalvektor ist ein Vektor (In Abbildung rechts: n) welche Senkrecht zu einer Ebene steht (In Abbildung rechts: n)	 <i>Normalvektor und Normalebene in GeoGebra</i>
Orbit	Ist lateinisch für «Bahn» und In der Astronomie synonym zu Umlaufbahn.	
Ortsvektor	Ein Ortsvektor ist der Vektor, welcher vom Ursprung zu einem gegebenen Punkt geht.	
Periode	Die Periode T ist die Zeit, welche ein Satellit benötigt, um einen Umlauf zu beenden.	
Rektaszension	Die horizontale Komponente im rotierenden äquatorialen Koordinatensystem. Siehe Kapitel 3.3.2 «Das äquatoriale Koordinatensystem» auf Seite 9.	
Skalarprodukt	Das Skalarprodukt zweier vektoren ist die Summe der Produkte der korrespondierenden Komponenten. Dies ist gleich dem Produkt den Kosinus des Winkels zwischen den zwei Vektoren und den Beträgen der Vektoren zum Beispiel: $\vec{a} = \begin{pmatrix} a_1 \\ a_2 \\ a_3 \end{pmatrix}, \vec{b} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}, \vec{a} \cdot \vec{b} = a_1 * b_1 + a_2 * b_2 + a_3 * b_3 = \cos \varphi * \vec{a} * \vec{b} $	
Stundenwinkel	Die horizontale Komponente im ruhenden äquatorialen Koordinatensystem. Siehe Kapitel 3.3.2 «Das äquatoriale Koordinatensystem» auf Seite 9.	
TLE	Two Line Element: eine standardisierte Methode Satellitenlaufbahnen anzugeben	
Vektor	Ein Vektor ist eine Gerichtete Strecke in der Mathematik. Im dreidimensionalen Raum besteht er aus drei Komponenten und wird häufig mit einem Pfeil über dem Buchstaben und einer 1×3 Matrix dargestellt: $\vec{v} = \begin{pmatrix} x\text{Komponente} \\ y\text{Komponente} \\ z\text{Komponente} \end{pmatrix} = \begin{pmatrix} 0.11 \\ 0.33 \\ 0.42 \end{pmatrix}$	 <i>Vektor \vec{v} in GeoGebra dargestellt</i>

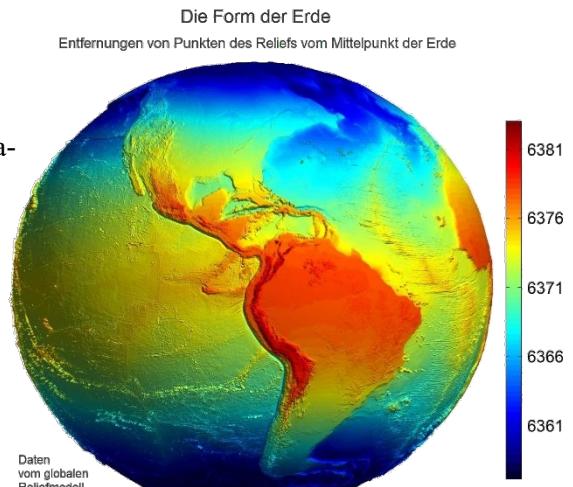
13.2 Ungezielte Beobachtungen



13.3 Erläuterungen

13.3.1 Erdfigur

Die Form der Erde ist nicht genau eine Kugel, da der Radius aufgrund der Scheinkraft der Zentrifugalkraft, welche durch die Erdrotation entsteht, am Äquator rund 21 Kilometer grösser ist als an den Polen. Dadurch entsteht tatsächlich eine stärkere Kraft zum Erdäquator hin als zu den Polen, was die Umlaufbahnen von Satelliten zum Präzidieren bringt. bei Satelliten eine Präzession der Umlaufbahn, welche vor allem bei LEO-Satelliten spürbar wird. Man kann dies auch nutzen, um Satelliten in eine Sonnensynchrone Umlaufbahn zu bringen, bei welcher diese Präzession in genau einem Sonnenjahr 360° beträgt und die Umlaufbahnebene somit immer zur Sonne gerichtet bleibt.



Relief der Ozeanböden und Landmassen [23]

13.3.2 Erdrotationswinkel

Die Sternzeit kann auch mit dem moderneren «Erdrotationswinkel» beschrieben werden. Um die Sternzeit trotzdem genau zu halten führt man eine Verschiebung von etwa 73 sec ein.

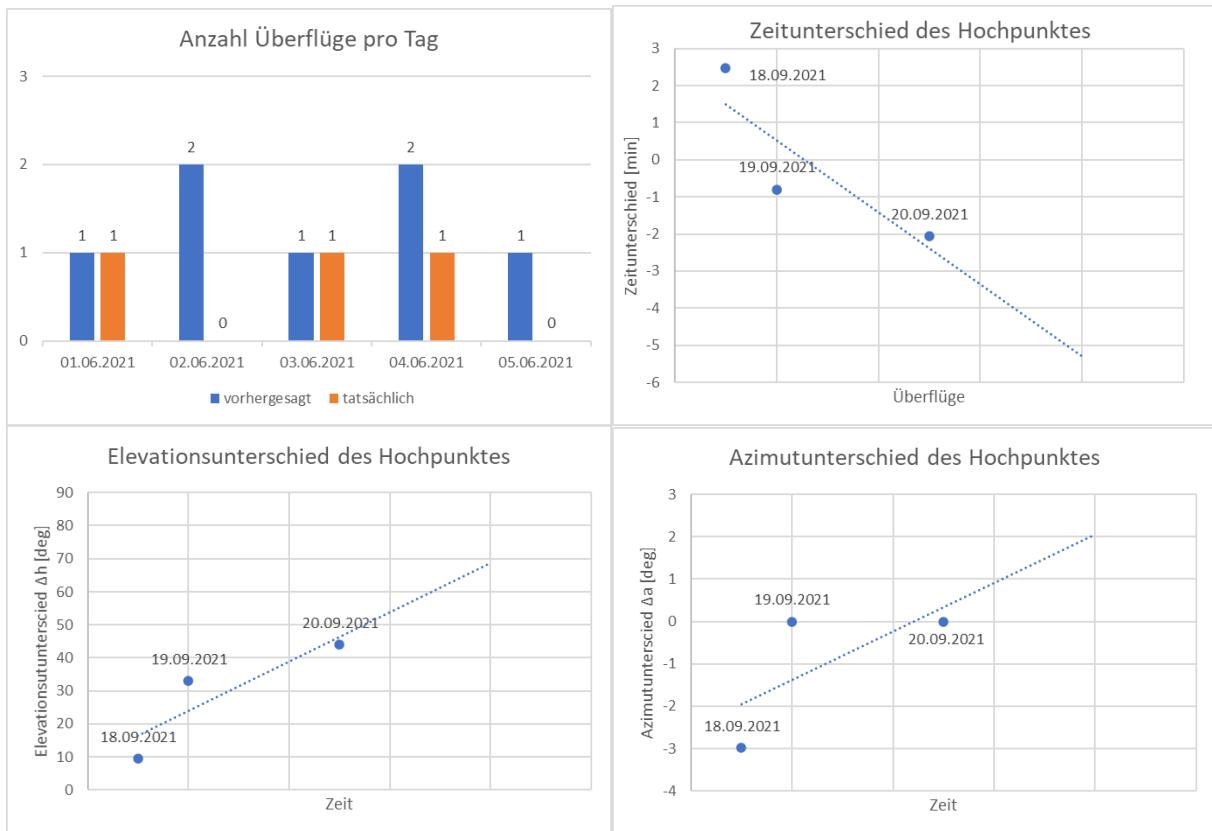
13.3.3 Achsenpräzession

Dieser Begriff beschreibt das Phänomen, dass sich die Rotationsachse der Erde ändert. Es geht etwa 25'800 Jahre, bis sich die Rotationsachse einmal vollständig gedreht hat. [2]

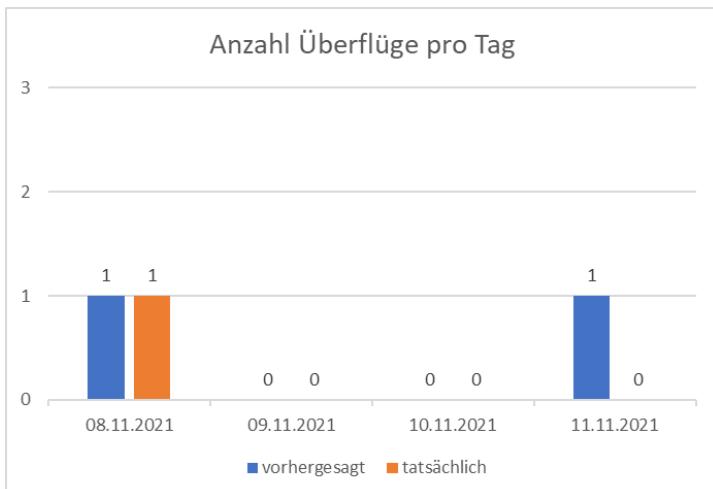
13.3.4 Literaturvergleiche weiterer Überflüge

Die Daten für den Vergleich der Vorhersagen wurde wie in Kapitel 6.2 «Vergleich mit Literaturwerten» auf Seite 44 mit «astroviewer.net» durchgeführt.

Vergleich der Vorhersagen aus Daten des 31.05.2021



Vergleich der Vorhersagen aus Daten des 07.11.2021



Nur ein «gemeinsamer» Überflug, deswegen in Tabellenform

	Messung	08.11.2021
ΔT [min]	2.98333333	28.26666667
ΔP [min]	4.90	1.97
ΔALT [deg]	1.2347619	5
ΔAZ [deg]	6.9047619	0

13.4 Ausdruck des Programm Codes

Falls das Laden des Programmes nicht funktioniert, kann man es in einer IDE öffnen und starten.

13.4.1 Programmstruktur

```
|src
|javaCode
-Axes.java
-Controller.java
-Earth.java
-Ecliptic.java
-EquatorialPlane.java
-Firmament.java
-Main.java
-ModelLoader.java
-MouseController.java
-Orbit.java
-Satellite.java
-SunLight.java
-SurfaceCamera.java
-ValueSet.java
|resources
|axis
  axes.obj
  axes.mtl
|earth
  -8081_earthnmap4k.jpg
  -8081_earthspe4k.jpg
  -earth.obj
  -earth.mtl
|firmament
  -firmament.obj
  -firmament.mtl
  -skyChart.png
|fxml
  -GUI_gluon.fxml
|icon
  -appIcon_big.png
  -appIcon_small.png
|orbitTrack
  -circularOrbit.mtl
  -circularOrbit.obj
|selfIlluminationMaps
  -eclipticScale.png
  -equatorialPlaneScale.png
  -selfIlluminationLocation.png
  -selfIlluminationOrbitalPlane.png
  -selfIlluminationSatellite.png
```

Erklärung
«|» → Order
«-» → Datei

Beispiel:

```
|order1
|ordner2
  -datei1
  -datei2
|ordner3
  -datei3
```

Für 3D-Modelle:
Wenn nicht anders deklariert, wurde es mit Blender erstellt

13.4.2 Axes.java

```
package javaCode;

import javafx.scene.Group;

class Axes extends Group {

    Axes(double size){

        //3D-model
        this.getChildren().add( ModelLoader.loadModel(
            "programm/src/resources/axis/axes.obj" ) );

        //Scale to visual Size
        this.getTransforms().add( ValueSet.uniformScale(size) );
    }
}
```

13.4.3 Controller.java

```
package javaCode;

import javafx.animation.AnimationTimer;
import javafx.scene.*;
import javafx.scene.control.Button;
import javafx.scene.control.Slider;
import javafx.scene.control.TextField;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.paint.Color;
import javafx.scene.text.Text;
import javafx.scene.transform.Rotate;
import javafx.scene.transform.Translate;
import javafx.util.StringConverter;

import static java.lang.Double.parseDouble;
import static java.lang.Math.*;
import static java.lang.String.*;
import static javaCode.ValueSet.*;

public class Controller {

    //declare FXML objects
    public SubScene mainScene;
    public ImageView imageIconContainer;
    public Slider speedSlider;
    public Slider scrollSlider;
    public Slider timeSlider;
    public Text timeDisplay;
    public TextField timeScaleFactorTextField;
    public TextField altitudeTextField;
    public TextField azimuthTextField;
    public TextField heightTextField;
    public TextField inclinationTextField;
    public TextField longitudeTextField;
    public TextField latitudeTextField;
    public TextField elevationTextField;
    public Button oneXButton;
    public Button toCoordsButton;
    public Button timeResetButton;
```

```
public Button switchCameraButton;
public Button pauseButton;
public Button satToLocButton;
public Button recalculateButton;
public Button timeToObservationButton;
public Button switchCalcButton;

//declare optional program-options buttons
public Button axesButton;
public Button equatorialPlaneButton;
public Button eclipticButton;
public Button orbitTrackButton;
public Button orbitalPlaneButton;
public Button firmamentButton;
public Button writeToFileButton;

//declare Non-FXML objects
Earth earth;
CoordinateLocation location;
Satellite satellite;
Orbit orbit;
IsVisibleCalculator isVisibleCalculator;
Camera flyingCamera;
SurfaceCamera surfaceCamera;
SunLight sun;
Axes axes;
EquatorialPlane equatorialPlane;
Ecliptic ecliptic;
Firmament firmament;
AmbientLight ambientLight;

//Global Variables
public double programTime;
public double observationTime;
public byte simMode = 0;

//Standard Values
public boolean enableAxes = false;
public boolean enableEquatorialPlane = false;
public boolean enableEcliptic = false;
public boolean enableOrbitTrack = true;
public boolean enableOrbitalPlane = false;
public boolean enableFirmament = false;
public boolean enableWriteToFile = false;

public void initialize() {
    //initialize calculated objects
    earth      = new Earth();
    location   = new CoordinateLocation( earth );
    satellite  = new Satellite();
    sun        = new SunLight();

    //initial GUI objects
    ambientLight      = new AmbientLight(
        Color.grayRgb( BACKGROUND_LIGHT_INTENSITY ) );
    flyingCamera     = new PerspectiveCamera( true );
    surfaceCamera    = new SurfaceCamera(
        earth, location, latitudeTextField, longitudeTextField );
    isVisibleCalculator = new IsVisibleCalculator(
        earth, location, satellite, sun );
    //flying Camera
    Translate flyingCameraPosition = new Translate();
    flyingCameraPosition.zProperty().set(
```

```

    EARTH_RADIUS * VISUALISATION_SCALE_FACTOR * - 5 );
flyingCamera.getTransforms().add( flyingCameraPosition );
flyingCamera.setNearClip( 10 );
flyingCamera.setFarClip( 1E8 );
//surface Camera
Rotate surfaceCameraRotate = new Rotate( 0, Rotate.Y_AXIS );
surfaceCamera.getTransforms().add( surfaceCameraRotate );

//initiate optional program-settings
axes = new Axes( 100 );
ecliptic = new Ecliptic(sun, PLANE_SIZE);
equatorialPlane = new EquatorialPlane( PLANE_SIZE );
firmament = new Firmament( 1000 );
orbit = new Orbit();

//enable/disable optional program-settings
axes.setVisible( enableAxes );
ecliptic.setVisible( enableEcliptic );
equatorialPlane.setVisible( enableEquatorialPlane );
firmament.setVisible( enableFirmament );
orbit.setVisible( enableOrbitTrack );
orbit.switchOrbitalPlane( enableOrbitalPlane );

//add objects to visible group
Group visibleGroup = new Group();
visibleGroup.getChildren().addAll(
    earth, location, orbit, satellite, sun,
    ambientLight, firmament,
    axes, equatorialPlane, ecliptic, surfaceCamera
);
mainScene.setRoot( visibleGroup );
mainScene.setCamera( flyingCamera );
mainScene.setVisible( true );

//initiate mouse control (panning)
MouseController mouseController = new MouseController();
mouseController.initMouseControl(
    visibleGroup, scrollSlider, mainScene, 0 );

//start animation
ProgramTimer timer = new ProgramTimer();
timer.start();

pauseAndPlay();

//GUI-CONTROLS:
//Time Slider
timeSlider.setMax(
    SLOW_CALC_PRE_CALCULATION_PERIOD * SLOW_CALC_MULTIPLIER);
timeSlider.valueProperty().addListener(
(observableValue, oldValue, newValue) -> {
//Whenever Slider is used
    programTime = newValue.doubleValue();
//update programTime
    updateNodesToTime( programTime );
} );

//Speed slider
speedSlider.setMax( log( 360_000 ) / log( 60 ) );
// Max=log_60(360'000) -> Max to actual 360'000
speedSlider.valueProperty().addListener(
(observableValue, oldValue, newValue) -> {

```

Anhang

```
//Whenever Slider is used
    timeScaleFactor = pow( 60, newValue.doubleValue() );
//timeScaleFactor
    timeScaleFactorTextField.setText( Integer.toString( (int) time-
ScaleFactor ) ); //is updated
} );
speedSlider.setLabelFormatter( new StringConverter<>() { //Change
slider-scale to logarithmic
    @Override
    public String toString(Double d) {
        return valueOf( (int) (pow( 60, d )) );
    }
    @Override
    public Double fromString(String s) {
        return null;
    }
} );

//timeScaleFactor Text Field
timeScaleFactorTextField.textProperty().addListener(
observable, oldValue, newValue) -> {
    if ( ! newValue.matches( "\\\d*" ) ) {
        // force the field to be numeric only
        timeScaleFactorTextField.setText( newValue.replaceAll(
"[^\\d]", "" ) );
    }
} );

//Scroll Slider
scrollSlider.setLabelFormatter( new StringConverter<>() {
//Slider-Scale to log with base 2 and
// 2^0.5 through 2^28
    @Override
    public String toString(Double d) {
        return valueOf( (int) (pow( 2, d )) );
    }
    @Override
    public Double fromString(String s) {
        return null;
    }
} );
scrollSlider.valueProperty().addListener(
observableValue, oldValue, newValue) -> {
//Whenever Slider is used
flyingCameraPosition.zProperty().set(
- 1 * EARTH_RADIUS * VISUALISATION_SCALE_FACTOR
* pow( 2, newValue.doubleValue() ) );
// flyingCameraPosition is recalculated
    surfaceCameraRotate.setAngle( 45 * newValue.doubleValue() );
// surfaceCamera is rotated
} );

//set icon
imageIconContainer.setImage(
new Image( "resources/icon/appIcon_small.png" ) );

//fill default values
latitudeTextField .setText( valueOf( LATITUDE ) );
longitudeTextField .setText( valueOf( LONGITUDE ) );
elevationTextField .setText( valueOf( ELEVATION ) );
altitudeTextField .setText( "90" );
azimuthTextField .setText( "0" );
heightTextField .setText( valueOf( STANDARD_ORBIT_HEIGHT ) );
```

```

        inclinationTextField.setText( valueOf( STANDARD_INCLINATION ) );
        recalculateButton.setDisable( true );
    }

    public double FRAME_LENGTH = 1E9 / FRAMES_PER_SECOND;
    public double timeScaleFactor = 1;
    public double timeScaleFactorS = 1;
    public boolean timeRunning = true;

    class ProgramTimer extends AnimationTimer {

        long last = System.nanoTime();

        { resetTime(); }

        @Override
        public void handle(long now) { //now is in Nanoseconds

            if (abs( now - last ) > FRAME_LENGTH) {

                programTime += (now - last) / 1E9 * timeScaleFactor;
                timeSlider.setValue( programTime );

                //update nodes
                if (timeRunning) updateNodesToTime( programTime );

                //update time display
                timeDisplay.setText( dateToString( programTime ) );

                //check visibility
                if (timeRunning)
                    System.out.print(
                        isVisibleCalculator.isVisibleNow(programTime,
                            simMode, timeDisplay.getText())
                    );
                last = now;
            }
        }
    }
    public void updateNodesToTime(double time) {
        try {
            earth.rotateToTime( time );
            location.rotateToTime( time );
            satellite.orbitToTime( simMode, time );
            sun.rotateToTime( time );
        } catch (Exception e) {
            System.out.println( "calcError" );
            if (programTime > timeSlider.getMax()) {
                programTime -= 86400; //Sets programTime back 1 day back;
                pauseAndPlay(); // and pauses
                System.out.println( "calculation has reached it's end" );
            }
        }
    }
    //GUI-methods
    public void pauseAndPlay() {
        if (timeRunning) {
            timeScaleFactorS = timeScaleFactor;
            timeScaleFactor = 0;
        } else {
            timeScaleFactor = timeScaleFactorS;
        }
        timeScaleFactorTextField.disableProperty().set( timeRunning );
    }
}

```

```

speedSlider           .disableProperty().set( timeRunning );
timeSlider           .disableProperty().set( ! timeRunning );
satToLocButton       .disableProperty().set( ! timeRunning );
recalculateButton    .disableProperty().set( ! timeRunning );

timeRunning ^= true;
}

public void recalculate() {
    simMode = 0;
    observationTime = programTime;
    timeToObservationButton.disableProperty().set( false );
    timeSlider.setMin( programTime );
    timeSlider.setMax( programTime + SATELLITE_PRE_CALCULATION_PERIOD );
    satellite.calculatePositions(
        (int) programTime,
        parseDouble( inclinationTextField.getText() ),
        orbit,
        simMode
    );
    satellite.updateScale( parseDouble( heightTextField.getText() ) );
    if (enableWriteToFile)
        isVisibleCalculator.isVisibleInPreCalculationPeriod(
            observationTime );
}

public void resetTime(){programTime = System.currentTimeMillis() * 1E-3
- DELTA_UNIX_YEAR - DELTA_UNIX_VERNAL_EQUINOX; }

public void satToLoc() {
    try {
        satellite.positionToLocation(
            parseDouble( altitudeTextField.getText() ),
            parseDouble( azimuthTextField.getText() ),
            parseDouble( heightTextField.getText() ),
            parseDouble( elevationTextField.getText() ),
            location,
            earth,
            programTime );
        recalculateButton.setDisable( false );
    }catch(Exception e){
        System.out.println(
            "Enter Valid Values (ALT: 0 - 90, AZ: 0 - 360, height > 0," +
            " latitude: -90 - 90, longitude: 0 - 360, elevation > 0)");
    }
}

public void speedToOne() {
    timeScaleFactor = 1; speedSlider.setValue( 0 );
}

public void switchCalculation(){
    simMode ^= 1;
    updateNodesToTime( programTime ); orbit.switchStates( simMode );
}

public void switchCamera() {
    if (flyingCamera.equals( mainScene.getCamera() )) {
        mainScene.setCamera( surfaceCamera );
    } else if (surfaceCamera.equals( mainScene.getCamera() )) {
        mainScene.setCamera( flyingCamera );
    } else {
        System.out.println( "Camera Error" );
    }
}

public void timeToObservation(){programTime = observationTime; }

public void toCoords(){
    double lat = parseDouble( latitudeTextField.getText() );
    double lon = parseDouble( longitudeTextField.getText() );
}

```

```
        double elv = parseDouble( elevationTextField.getText());
        location.calculatePositions(earth, lat, lon, elv);
        surfaceCamera.updateCameraRotation(earth, lat, lon);
        updateNodesToTime( programTime );
    }
    public void updateTimeScaleFactorTextField(){
        double value = log(
            parseDouble( timeScaleFactorTextField.getText() ) ) / log( 60 );
        speedSlider.setValue(
            Double.min( value, speedSlider.getMax() )
        //set SliderValue to log_60 of entered Value
        );
    }
    //optional program-option functions
    public void switchAxes(){
        enableAxes ^= true;
        axes.setVisible( enableAxes );
        axesButton.setText(switchEnableDisable(axesButton.getText(),enableAxes));
    }
    public void switchEquatorialPlane(){
        enableEquatorialPlane ^= true;
        equatorialPlane.setVisible( enableEquatorialPlane );
        equatorialPlaneButton.setText(switchEnableDisable(equatorialPlane-
Button.getText(),enableEquatorialPlane));
    }
    public void switchEcliptic(){
        enableEcliptic ^= true;
        ecliptic.setVisible( enableEcliptic );
        eclipticButton.setText(switchEnableDisable(eclipticBut-
ton.getText(),enableEcliptic));
    }
    public void switchFirmament(){
        enableFirmament ^= true;
        firmament.setVisible( enableFirmament );
        firmamentButton.setText( switchEnableDisable( firmamentBut-
ton.getText(), enableFirmament ) );
    }
    public void switchOrbitTrack(){
        enableOrbitTrack ^= true;
        orbit.setVisible( enableOrbitTrack );
        orbitTrackButton.setText(switchEnableDisable(orbitTrackBut-
ton.getText(),enableOrbitTrack));
    };
    public void switchOrbitalPlane(){
        enableOrbitalPlane ^= true;
        orbit.switchOrbitalPlane( enableOrbitalPlane );
        orbitalPlaneButton.setText(switchEnableDisable(orbitalPlaneBut-
ton.getText(),enableOrbitalPlane));
    }
    public void switchWriteToFile(){
        enableWriteToFile ^= true;
        writeToFileButton.setText( switchEnableDisable( writeToFileBut-
ton.getText(), enableWriteToFile ) );
    }
    public String switchEnableDisable(String s, Boolean mode){
        if (!mode) {
            return s.replace( "Disable", "Enable" );
        } else {
            return s.replace( "Enable", "Disable" );
        }
    }
}
```

13.4.4 CoordinateLocation.java

```
package javaCode;

import javafx.geometry.Point3D;
import javafx.scene.image.Image;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Sphere;

import javafx.scene.transform.Translate;

import static java.lang.Math.*;
import static javaCode.ValueSet.*;
import static javafx.scene.transform.Rotate.*;

public class CoordinateLocation extends Sphere {

    Translate modifiableTranslate;
    Point3D[] position;
    double currentLatitude, currentLongitude, currentElevation;

    public CoordinateLocation(Earth earth) {
        //set radius
        super( 50 );

        //set material
        PhongMaterial material = new PhongMaterial( Color.ORANGERED );
        material.setSelfIlluminationMap(new Image("resources/selfIlluminationMaps/selfIlluminationLocation.png"));
        this.setMaterial( material );

        currentLatitude = LATITUDE;
        currentLongitude = LONGITUDE;
        currentElevation = ELEVATION;

        calculatePositions( earth, currentLatitude, currentLongitude, currentElevation );

        modifiableTranslate = new Translate(0,0,0);
        this.getTransforms().add( modifiableTranslate );
    }

    public void calculatePositions(Earth earth, double inputLatitude, double inputLongitude, double elevation){
        //calculates coordinate locations for given latitude, longitude and elevation
        currentLatitude = inputLatitude;
        currentLongitude = inputLongitude;

        Point3D initialPosition = new Point3D(
            cos( toRadians( inputLatitude ) ) * cos( toRadians( inputLongitude ) ),
            -sin( toRadians( inputLatitude ) ),
            cos( toRadians( inputLatitude ) ) * sin( toRadians( inputLongitude ) )
        ).multiply( (EARTH_RADIUS + elevation) * VISUALISATION_SCALE_FACTOR );
        double[] er = earth.getRotationArray();

        position = new Point3D[SLOW_CALC_PRE_CALCULATION_PERIOD];
        for (int i = 0; i < position.length; i++)
```

```
//rotates location according to earth rotation
position[i] = rotatePoint( initialPosition, Y_AXIS.multiply( -1 ), er[i] );
}
public void rotateToTime(double time) {
modifiableTranslate.setX( position[(int) time/SLOW_CALC_MULTIPLIER].getX() );
modifiableTranslate.setY(position[(int) time/SLOW_CALC_MULTIPLIER].getY());
modifiableTranslate.setZ(position[(int) time/SLOW_CALC_MULTIPLIER].getZ());
}
}
```

13.4.5 Earth.java

```
package javaCode;

import javafx.scene.Group;
import javafx.scene.transform.Rotate;

import static javaCode.ValueSet.*;
import static javafx.scene.transform.Rotate.*;

public class Earth extends Group {

    public Rotate modifiableRotate;
    public double[] preCalcRotation;
    public double degreesPerSecond;

    Earth() {
        //load 3D-model
        Group model = ModelLoader.loadModel( "programm/src/resources/earth2/earth.obj" );
        this.getChildren().add( model );

        //scale model
        model.getTransforms().add(
            uniformScale( EARTH_RADIUS * VISUALISATION_SCALE_FACTOR ) );

        //pre-calculation
        degreesPerSecond = 360/ SIDEREAL_DAY_LENGTH;
        preCalcRotation = new double[SLOW_CALC_PRE_CALCULATION_PERIOD];
        for (int i = 0; i < preCalcRotation.length; i++)
            preCalcRotation[i] =
                (i * SLOW_CALC_MULTIPLIER * degreesPerSecond );

        //add modifiable rotation
        modifiableRotate = new Rotate( 0, Rotate.Y_AXIS.multiply( -1 ) );
        this.getTransforms().add( modifiableRotate );
    }

    public void rotateToTime(double time){modifiableRotate.setAngle(preCalcRotation[(int) time/SLOW_CALC_MULTIPLIER]);}

    public double[] getRotationArray(){ return preCalcRotation; }
}
```

13.4.6 Ecliptic.java

```
package javaCode;

import javafx.scene.image.Image;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Cylinder;
import javafx.scene.transform.Rotate;

import static javaCode.ValueSet.*;
import static javafx.scene.transform.Rotate.*;

public class Ecliptic extends Cylinder {
```

```
Ecliptic(SunLight sun, double size){  
    //set dimensions  
    super( VISUALISATION_SCALE_FACTOR * EARTH_RADIUS * size,1,32 );  
  
    //set Material  
    PhongMaterial material = new PhongMaterial();  
    Image image = new Image(  
        "resources/selfIlluminationMaps/eclipticScale.png");  
    material.setSelfIlluminationMap( image );  
    material.setDiffuseMap( image );  
    this.setMaterial( material );  
  
    //rotate to actual ecliptic  
    Rotate eclipticRotate = new Rotate(EARTH_AXIAL_TILT,  
        sun.rotationAxis.crossProduct( Y_AXIS ));  
    this.getTransforms().add( eclipticRotate );  
}  
}
```

13.4.7 EquatorialPlane.java

```
package javaCode;  
  
import javafx.scene.image.Image;  
import javafx.scene.paint.PhongMaterial;  
import javafx.scene.shape.Cylinder;  
  
import static javaCode.ValueSet.*;  
  
public class EquatorialPlane extends Cylinder {  
  
    EquatorialPlane(double size){  
        //set dimensions  
        super( VISUALISATION_SCALE_FACTOR * EARTH_RADIUS * size,1,32 );  
  
        //set material  
        PhongMaterial material = new PhongMaterial();  
        Image image = new Image(  
            "resources/selfIlluminationMaps/equatorialPlaneScale.png");  
        material.setSelfIlluminationMap( image );  
        material.setDiffuseMap( image );  
        this.setMaterial( material );  
    }  
}
```

13.4.8 Firmament.java

```
package javaCode;  
  
import javafx.scene.Group;  
import javafx.scene.transform.Rotate;  
  
import static javaCode.ValueSet.*;  
import static javafx.scene.transform.Rotate.*;  
  
public class Firmament extends Group {  
  
    Firmament(double scaleFactor){  
        //load Model  
        Group model = ModelLoader.loadModel(  
            "programm/src/resources/firmament/firmament.obj" );  
        this.getChildren().add( model );  
    }  
}
```

```
//scale Model
double scale = scaleFactor *
EARTH_RADIUS * VISUALISATION_SCALE_FACTOR;
model.getTransforms().add( uniformScale( scale ) );

//rotate according to sidereal time
Rotate siderealTimeRotate = new Rotate( GMST_VERNAL_EQUINOX, AXIS );
this.getTransforms().add( siderealTimeRotate );
}
```

13.4.9 IsVisibleCalculator.java

```
package javaCode;

import javafx.geometry.Point3D;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.nio.file.StandardOpenOption;

import static java.lang.Math.*;
import static javaCode.ValueSet.*;
import static javafx.geometry.Point3D.*;

public class IsVisibleCalculator {
    Earth referenceObject;
    Satellite satellite;
    SunLight sun;
    CoordinateLocation location;
    double angleToSatellite, horizonAngleToSun;
    Point3D referenceObjectLocation, coordinateLocation,
               satelliteLocation, sunLocation;
    Point3D vector_planetToSurfaceLocation,
               vector_surfaceLocationToSatellite, vector_surfaceLocationToSun;
    Path output;

    IsVisibleCalculator(Earth inputReferenceObject,
                       CoordinateLocation inputLocation,
                       Satellite inputSatellite,
                       SunLight inputSun) {

        satellite = inputSatellite;
        referenceObject = inputReferenceObject;
        location = inputLocation;
        sun = inputSun;

        output = Paths.get("output.txt");
    }
    public String isVisibleNow(double time, byte simMode, String timeText) {
        String output;
        updatePositions(time, simMode);

        //visible angle of the Satellite
        angleToSatellite = vector_planetToSurfaceLocation
            .angle( vector_surfaceLocationToSatellite );

        //visible angle of the Sun
        horizonAngleToSun = vector_planetToSurfaceLocation
```

Anhang

```

        .angle( vector_surfaceLocationToSun ) - 90 ;

    if (angleToSatellite < MINIMUM_SATELLITE_VISIBILITY_ANGLE) {
        //= is satellite over horizon?
        output = timeText + " ALT:" + (90-floor( angleToSatellite ));
        if (horizonAngleToSun < MAX_SUNLIGHT_ANGLE) //= is evening?
            output = "VISIBLE: " + output;
        return output + System.lineSeparator();
    } else return "";
}

public void isVisibleInPreCalculationPeriod(double startTime) {
    try {
        //calculates all visible passes for the next PRE_CALCULATION_DAYS
        Files.deleteIfExists( output );
        Files.createFile( output );
        for (byte simMode = 0; simMode <= 1; simMode++) {
            Files.write(output, ("SimMode: " + (simMode) +
        System.lineSeparator()).getBytes(), StandardOpenOption.APPEND );
            for (int t = 0;
t < SATELLITE_PRE_CALCULATION_PERIOD/VISIBILITY_CHECK_MULTIPLIER;
t++) {
                Files.write(output,
                    isVisibleNow(
                        startTime + t*VISIBILITY_CHECK_MULTIPLIER, simMode,
                        (dateToString( startTime + t*VISIBILITY_CHECK_MULTIPLIER
                    )).getBytes(),
                        StandardOpenOption.APPEND );
                }
            }
        } catch (IOException e) {
            System.out.println(
"unable to write to file " + output.toString()
);
            e.printStackTrace();
        }
    }
}

public void updatePositions(double time, byte simMode) {
    //Updates Point3Ds of referenceObject (Earth),
    //satellite and the coordinates in relation to Parent Group
    referenceObjectLocation = ZERO;
    coordinateLocation =
    location.position[(int) time/SLOW_CALC_MULTIPLIER];
    try {
        satelliteLocation = satellite.position[simMode][(int) time];
    }catch (NullPointerException n){
        satelliteLocation = ZERO;
    }
    sunLocation = sun.sunPosition( time );

    //Vector from Planet to Location
    vector_planetToSurfaceLocation =
    coordinateLocation.subtract( referenceObjectLocation );
    //Vector from Location to Satellite
    vector_surfaceLocationToSatellite =
    satelliteLocation.subtract( coordinateLocation );
    //Vector from Location to Sun
    vector_surfaceLocationToSun =
    sunLocation.subtract( coordinateLocation );
}
}

```

13.4.10 Main.java

```
package javaCode;

import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.*;
import javafx.scene.image.Image;
import javafx.stage.Stage;

public class Main extends Application {

    @Override
    public void start(Stage primaryStage){
        try {
            //load GUI
            FXMLLoader loader = new FXMLLoader(
                getClass().getClassLoader().getResource(
                    "resources/fxml/GUI_gluon.fxml" ) );
            Parent root = loader.load();
            Scene fxmlScene = new Scene( root );
            primaryStage.setScene( fxmlScene );

            //set title, icon & resizable
            primaryStage.setTitle( "Tarvos" );
            primaryStage.getIcons().add( new Image(
                "resources/icon/appIcon_big.png" ) );
            primaryStage.setResizable( false );

            primaryStage.show();
        }catch (Exception e){
            System.out.println("error");
            e.printStackTrace();
            System.exit( 1 );
        }
    }
    public static void main(String[] args) {
        launch(args);
    }
}
```

13.4.11 ModelLoader.java

```
package javaCode;

import com.interactivemesh.jfx.importer.obj.ObjModelImporter;
import javafx.scene.Group;
import javafx.scene.shape.MeshView;

import java.nio.file.Paths;

public class ModelLoader {
    static Group loadModel(String relativePath){
        Group modelRoot = new Group();

        String url = Paths.get(relativePath).toAbsolute-
Path().toString().replace("/", "\\\\");

        ObjModelImporter importer = new ObjModelImporter();
        try {
            importer.read(url);

            for (MeshView view : importer.getImport()) {
                modelRoot.getChildren().add( view );
            }
        }
```

```
        }
    }catch (Exception e){
        System.out.println("could not load " + url);
    }
    return modelRoot;
}
}
```

13.4.12 MouseController.java

```
package javaCode;

import javafx.beans.property.DoubleProperty;
import javafx.beans.property.SimpleDoubleProperty;

import javafx.scene.Group;
import javafx.scene.SubScene;
import javafx.scene.control.Slider;
import javafx.scene.input.ScrollEvent;
import javafx.scene.transform.Rotate;

public class MouseController {
    Rotate xRotate;
    Rotate yRotate;
    private double anchorX, anchorY;
    private double anchorAngleX = 0;
    private double anchorAngleY = 0;
    private final DoubleProperty angleX = new SimpleDoubleProperty(0);
    private final DoubleProperty angleY = new SimpleDoubleProperty(0);
    private double sensitivity = 0.25;

    public void initMouseControl(Group group, Slider scrollSlider, SubScene scene, double tilt){

        group.getTransforms().addAll(
            xRotate = new Rotate(0, Rotate.X_AXIS), //connects rotation
with angleX und Y
            yRotate = new Rotate(0, Rotate.Y_AXIS)
        );
        xRotate.angleProperty().bind( angleX );
        yRotate.angleProperty().bind( angleY );

        scene.setOnMousePressed( event -> {
            anchorX = event.getSceneX();
            // anchorX, anchorY = mouse position when pressing
            anchorY = event.getSceneY();
            anchorAngleX = angleX.get();
            // AnchorAngleX and Y = rotation when pressing
            anchorAngleY = angleY.get();
        } );
        scene.setOnMouseDragged( event -> {
            angleX.set( anchorAngleX - sensitivity *
            (anchorY - event.getSceneY())
            /*y-distance when pressing and dragging */ - tilt );
            //subtracts the y-distance from he x-angle
            angleY.set( anchorAngleY + sensitivity *
            (anchorX - event.getSceneX()) );
            //adds the x-distance to the y-angle
        } );
        scene.addEventHandler( ScrollEvent.SCROLL, event -> {
            double scrollDelta = event.getDeltaY();
        } );
    }
}
```

```
        scrollSlider.setValue( scrollSlider.getValue()+
            scrollDelta * 0.006
        );
    }
}
```

13.4.13 Orbit.java

```
package javaCode;

import javafx.geometry.Point3D;
import javafx.scene.Group;
import javafx.scene.image.Image;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Cylinder;
import javafx.scene.transform.Rotate;
import javafx.scene.transform.Scale;

import static javaCode.ValueSet.*;
import static javafx.geometry.Point3D.*;
import static javafx.scene.transform.Rotate.Y_AXIS;

class Orbit extends Group {

    public Rotate[] rotations;
    public Scale modifiableScale;
    public Rotate modifiableRotate;
    public Cylinder orbitalPlane;

    Orbit() {
        //load torus 3D-model
        this.getChildren().add( ModelLoader.loadModel( "programm/src/resources/orbitTrack/circularOrbit.obj" ) );

        //create orbital Plane
        orbitalPlane = new Cylinder(1, 0.001, 128);
        PhongMaterial orbitalPlaneMaterial = new PhongMaterial();
        Image image = new Image("resources/selfIlluminationMaps/selfIlluminationOrbitalPlane.png");
        orbitalPlaneMaterial.setSelfIlluminationMap( image );
        orbitalPlaneMaterial.setDiffuseMap( image );

        //set orbital plane visibility
        orbitalPlane.setMaterial( orbitalPlaneMaterial );
        this.getChildren().add( orbitalPlane );

        rotations = new Rotate[]{new Rotate(0, ZERO), new Rotate(0, ZERO)};
        modifiableRotate = new Rotate(0, ZERO);
        modifiableScale = uniformScale(0);
        this.getTransforms().addAll( modifiableRotate, modifiableScale );
    }

    public void recalculate(Point3D[] satelliteRotationAxis,
                           double inclination,
                           double semiMajorAxis,
                           int simMode) {
        //re-rotate
        for (int mode = 0; mode < 2; mode++)
            rotations[mode] = new Rotate(inclination,
                satelliteRotationAxis[mode].crossProduct( Y_AXIS ));
        this.getTransforms().set( 1,
            uniformScale( semiMajorAxis*VISUALISATION_SCALE_FACTOR ) );
        switchStates( simMode );
    }
}
```

```
        }
        public void switchStates(int simMode){
            this.getTransforms().set( 0,  rotations[simMode] );
        }
        public void switchOrbitalPlane(boolean enableOrbitalPlane){
            orbitalPlane.setVisible( enableOrbitalPlane );
        }
    }
```

13.4.14 Satellite.java

```
package javaCode;

import javafx.geometry.Point3D;
import javafx.scene.image.Image;
import javafx.scene.paint.Color;
import javafx.scene.paint.PhongMaterial;
import javafx.scene.shape.Sphere;
import javafx.scene.transform.Translate;

import java.util.ArrayList;
import java.util.Objects;

import static java.lang.Math.*;
import static java.lang.Math.sin;
import static javaCode.ValueSet.*;
import static javafx.geometry.Point3D.*;
import static javafx.scene.transform.Rotate.*;

public class Satellite extends Sphere{

    Point3D[] rotationAxis;
    Point3D[][] position;
    Translate modifiableTranslate;
    Point3D orbitAnchor;
    Boolean calculated;

    Satellite(){
        calculated = false;

        //Modifiable Transform
        modifiableTranslate = new Translate(0,0,0);
        this.getTransforms().add( modifiableTranslate );

        //MATERIAL
        PhongMaterial satelliteMaterial = new PhongMaterial(Color.BLUE);
        satelliteMaterial.setSelfIlluminationMap(
            new Image("resources/selfIlluminationMaps/selfIllumina-
tionSatellite.png"));
        this.setMaterial(satelliteMaterial);
    }

    public void orbitToTime(byte simMode, double time){
        if (calculated) {
            modifiableTranslate.setX( position[simMode][(int) time].getX());
            modifiableTranslate.setY( position[simMode][(int) time].getY());
            modifiableTranslate.setZ( position[simMode][(int) time].getZ());
        }
    }

    public void calculatePositions(int initialTime,
                                  double inclination,
                                  Orbit orbit,
                                  byte simMode){}
```

```

orbitAnchor = this.localToParent( ZERO );
double semiMajorAxis = orbitAnchor.magnitude()/
                      VISUALISATION_SCALE_FACTOR;
rotationAxis = iterateRotationAxis( inclination, orbitAnchor );

//for first to possible rotation-axes,
//    calculate positions for the next PRE-CALCULATION_DAYS
position =
new Point3D[2][initialTime + SATELLITE_PRE_CALCULATION_PERIOD];
double u = orbitalPeriod( semiMajorAxis );
System.out.println(u/60);
for (int dl = 0; dl < position.length; dl++) {
    for (int i = 0; i < SATELLITE_PRE_CALCULATION_PERIOD; i++) {
        position[dl][i + initialTime] = rotatePoint(
            orbitAnchor, rotationAxis[dl], i / ( u / 360 ) );
    }
}
orbit.recalculate( rotationAxis, inclination,
                   semiMajorAxis, simMode );
calculated = true;
}

public void updateScale(double h){
    this.setRadius( 75 * (h + EARTH_RADIUS) / EARTH_RADIUS );
}

public void positionToLocation(double alt,
                               double az,
                               double h,
                               double elevation,
                               CoordinateLocation loc,
                               Earth earth,
                               double t) {
    Point3D locationPosition, earthToSurfaceVector, altRotated,
    azLonRotated, latRotated, p;

    locationPosition = loc.position[(int) t/ SLOW_CALC_MULTIPLIER];
    double earthRotation = earth
    .getRotationArray()[(int) t/ SLOW_CALC_MULTIPLIER];

    earthToSurfaceVector = locationPosition
    .normalize().multiply((EARTH_RADIUS + elevation)
    * VISUALISATION_SCALE_FACTOR);
    //Vector from origin to CoordinateLocation

    altRotated = rotatePoint( new Point3D( -h/sin(toRadians( alt )) )
    * VISUALISATION_SCALE_FACTOR, 0, 0 ),
    // initial Point on the X-Axis distance to origin equal
    // to the distance from the observer
    Z_AXIS, alt);
    // Creates a Vector According to the Altitude-angle
    azLonRotated = rotatePoint( altRotated, Y_AXIS,
    azLoc.currentLongitude-earthRotation);
    // Rotates the alt-angle-rotated vector according to
    the Azimuth-Angle, Longitude and Earth's Rotation
    latRotated = rotatePoint( azLonRotated, locationPosition
    .crossProduct( Y_AXIS ), 90-loc.currentLatitude );
    // Rotates the Vector according to Latitude, by using
    // the crossproduct of the coordinateLocation and Y-Axis as
    // the Rotation-axis

    p = earthToSurfaceVector.add( latRotated );
    //Adds the vectors together

    p.normalize()
}

```

```

.multiply( (EARTH_RADIUS + h) * VISUALISATION_SCALE_FACTOR );
//Set Height back to entered Height

modifiableTranslate.setX( p.getX());
modifiableTranslate.setY( p.getY());
modifiableTranslate.setZ( p.getZ());
updateScale(h);
}

public Point3D[] iterateRotationAxis(double inclination,
                                         Point3D initialPosition){
    Point3D ra;
    ArrayList<Object> deltaLambda = new ArrayList<>();
    for (int degree = 0; degree < 360; degree++) {
        for (int i = 0; i < RA_ITERATIONS_PD; i++) {
            ra = createRotationAxis(
                inclination, degree + i/ RA_ITERATIONS_PD );
            if(distanceToNormalPlane( ra, initialPosition )
            < RA_ITERATIONS_TOLERANCE) {
                deltaLambda.add( degree + i/ RA_ITERATIONS_PD );
                break;
            }
        }
    }
    System.out.println(deltaLambda);
    if (!deltaLambda.isEmpty()){
        return new Point3D[]{
            createRotationAxis( inclination, (Double) deltaLambda.get( 0 ) ),
            createRotationAxis( inclination, (Double) deltaLambda.get( 1 ) )
        };
    }
    else {
        System.out.println("RA iteration failed");
        return new Point3D[]{ZERO, ZERO};
    }
}
public Point3D createRotationAxis(double inclination,
                                   double deltaLambda) {
    return rotatePoint( new Point3D(
        sin( toRadians( inclination)),
        cos( toRadians( inclination)),
        0 ).multiply( -1 ), Y_AXIS, deltaLambda );
}
public static double orbitalPeriod(double radius) {
    return 2 * PI
        * sqrt(
            pow( radius, 3 )
                / (GRAVITATIONAL_CONSTANT*
                    EARTH_MASS)
        );
}
}

```

13.4.15 SunLight.java

```

package javaCode;

import javafx.geometry.Point3D;
import javafx.scene.PointLight;

import static java.lang.StrictMath.*;
import static javaCode.ValueSet.*;

```

```
public class SunLight extends PointLight {

    public Point3D p;
    public Point3D p0;
    public Point3D rotationAxis;

    SunLight() {
        p0 = new Point3D( SUN_DISTANCE*VISUALISATION_SCALE_FACTOR, 0, 0 );
        rotationAxis = new Point3D(
            0,
            cos( toRadians( EARTH_AXIAL_TILT ) ),
            sin( toRadians( EARTH_AXIAL_TILT ) ) )
            .multiply( -1 );
    }

    public Point3D sunPosition(double programTime) {
        double angle = (programTime)/(SIDEREAL_YEAR_LENGTH)*360;
        return rotatePoint(p0, rotationAxis, angle);
    }

    public void rotateToTime(double time) {
        p = sunPosition( time );
        this.setTranslateX( p.getX() );
        this.setTranslateY( p.getY() );
        this.setTranslateZ( p.getZ() );
    }

}
```

13.4.16 SurfaceCamera.java

```
package javaCode;

import javafx.scene.PerspectiveCamera;
import javafx.scene.control.TextField;
import javafx.scene.transform.Rotate;
import javafx.scene.transform.Translate;

import static javafx.scene.transform.Rotate.*;
import static javaCode.ValueSet.*;

public class SurfaceCamera extends PerspectiveCamera {

    public Rotate latRotate;
    public Rotate lonRotate;

    public SurfaceCamera(Earth earth, CoordinateLocation location, TextField latTF, TextField lonTF) {

        this.setFieldOfView( 120 );
        this.setNearClip( 1 );
        this.setFarClip( 5E4 );

        latRotate = new Rotate(LATITUDE, X_AXIS);
        lonRotate = new Rotate(LONGITUDE, Y_AXIS);

        Translate locationTranslate = new Translate();
        locationTranslate.xProperty().bind(location.modifiableTranslate.xProperty());
        locationTranslate.yProperty().bind(location.modifiableTranslate.yProperty());
        locationTranslate.zProperty().bind(location.modifiableTranslate.zProperty());

        this.getTransforms().addAll( locationTranslate, latRotate,
```

```
lonRotate);  
  
        updateCameraRotation( earth, location.currentLatitude, location.currentLongitude );  
    }  
    public void updateCameraRotation(Earth earth, double latitude, double longitude) {  
        latRotate.setAngle( latitude );  
        lonRotate.setAngle( earth.modifiableRotate.getAngle() + longitude  
);  
    }  
}
```

13.4.17 ValueSet.java

Beinhaltet Globale Konstanten und Funktionen

```
package javaCode;  
  
import javafx.geometry.Point3D;  
import javafx.scene.shape.Sphere;  
import javafx.scene.transform.Rotate;  
import javafx.scene.transform.Scale;  
import javafx.scene.transform.Translate;  
  
import static java.lang.Math.floor;  
import static java.lang.StrictMath.*;  
import static java.lang.String.format;  
import static javafx.geometry.Point3D.*;  
import static javafx.scene.transform.Rotate.*;  
  
class ValueSet {  
    //Physical Constants  
    public static final double GRAVITATIONAL_CONSTANT = 6.67408E-11;  
    public static final double EARTH_MASS = 5.9722E24;  
    public static final double EARTH_RADIUS = 6371E3;  
    public static final double EARTH_AXIAL_TILT = 25.5;  
    public static final double SUN_DISTANCE = 149_597_870_700D;  
  
    //Default Values  
    public static double STANDARD_INCLINATION = 51.664;  
    public static double STANDARD_ORBIT_HEIGHT = 424E3;  
  
    public static double LATITUDE = 47.447839910007204;  
    public static double LONGITUDE = 9.640037238574251;  
    public static double ELEVATION = 402;  
  
    public static final double VISUALISATION_SCALE_FACTOR = 1E-3;  
    public static final double FRAMES_PER_SECOND = 60;  
    public static final double RA_ITERATIONS_PD = 1E4; //Rotation-Axis  
search Iterations per degree  
    public static final double RA_ITERATIONS_TOLERANCE = 1E-2; //Rotation-  
Axis search Iterations per degree  
    public static final short BACKGROUND_LIGHT_INTENSITY = 25;  
    public static final double PLANE_SIZE = 7;  
  
    public static final int MINIMUM_SATELLITE_VISIBILITY_ANGLE = 80;  
    public static final int MAX_SUNLIGHT_ANGLE = 15;  
  
    public static final int DELTA_UNIX_YEAR = 0; //Seconds  
between Jan 1 1970 and Jan 1 2021  
    public static final int DELTA_UNIX_VERNAL_EQUINOX = 1616241600;  
    public static final double SIDEREAL_DAY_LENGTH = 86164.098912; // IN  
SEC
```

```

public static final double SIDEREAL_YEAR_LENGTH= 31558149.504;
public static final double GMST_VERNAL_EQUINOX = 180 ; ;

public static final int SATELLITE_PRE_CALCULATION_DAYS = 5 ; // Amount
of Days to Calculate for the Satellites
public static final int SATELLITE_PRE_CALCULATION_PERIOD = SATEL-
LITE_PRE_CALCULATION_DAYS * 24 * 60 * 60; // PRE_CALCULATION_DAYS in Sec-
onds

public static final byte VISIBILITY_CHECK_MULTIPLIER = 5;

public static final byte SLOW_CALC_MULTIPLIER = 5;
public static final int SLOW_CALC_PRE_CALCULATION_DAYS = 370 ; // 
Amount of Days to Calculate for Earth and CoordinateLocation
public static final int SLOW_CALC_PRE_CALCULATION_PERIOD =
SLOW_CALC_PRE_CALCULATION_DAYS * 24 * 60 * (60/SLOW_CALC_MULTIPLIER); // 
PRE_CALCULATION_DAYS in 5 second increments

public static Scale uniformScale(double factor){
    return new Scale(factor, factor, factor);
}
public static double distanceToNormalPlane(Point3D n, Point3D p){
    return abs(n.dotProduct( p ))/(n.magnitude());
}
public static Point3D rotatePoint(Point3D point, Point3D rotationAxis,
double angle ){
    Sphere calcSphere = new Sphere();
    calcSphere.getTransforms().add( new Translate( point.getX(),
point.getY(), point.getZ() ) );
    calcSphere.setRotationAxis( rotationAxis );
    calcSphere.setRotate( angle );
    return calcSphere.localToScene( ZERO );
}
public static Rotate[] rotateTowards(Point3D target){
    Rotate xRotate, yRotate;
    xRotate = new Rotate(asin( target.getY()/target.magnitude() ),
X_AXIS);
    yRotate = new Rotate(atan2( target.getZ(), target.getX()), Y_AXIS);
    return new Rotate[]{xRotate, yRotate};
}
public static double unixToJulian(double unixTime) {return unixTime /
86_400 + 2_440_587.5;}
public static int[] julianToDate(double jd) {
    int z, c, d, e;
    int year, month, day, hour, min, sec;
    double alpha, a, f, b;
    double hour_dec, min_dec;

    z = (int) floor( jd + 0.5 );
    alpha = floor( (z - 1_867_216.25) / 36_524.25 );
    a = z + 1 + alpha - floor( alpha / 4 );

    f = jd + 0.5 - z;
    b = a + 1524;
    c = (int) floor( (b - 122.1) / 365.25 );
    d = (int) floor( 365.25 * c );
    e = (int) floor( (b - d) / 30.6001 );

    double day_dec = (int) b - d - floor( 30.6001 * e ) + f;
    if (e <= 13) {
        month = e - 1;
        year = c - 4716;
    } else {
}

```

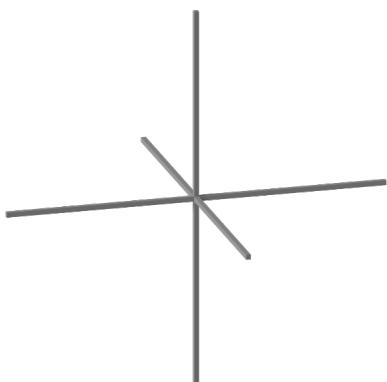
```
month = e - 13;
year = c - 4715;
}
day = (int) floor( day_dec );
hour_dec = (day_dec % 1) * 24;
hour = (int) floor( (day_dec % 1) * 24 );
min_dec = ((hour_dec % 1) * 60);
min = (int) floor( (hour_dec % 1) * 60 );
sec = (int) floor( (min_dec % 1) * 60 );

return new int[]{year, month, day, hour, min, sec};
}
public static String dateToString(double time) {
    int[] t = julianToDate(
        unixToJulian( time + DELTA_UNIX_VERNAL_EQUINOX + DELTA_UNIX_YEAR ) );
    String[] months = {
        "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul",
        "Aug", "Sep", "Okt", "Nov", "Dec"};
    return format( "%02d %s %02d %02d:%02d:%02d ",
        t[0], months[t[1] - 1], t[2], t[3], t[4], t[5] );
}
}
```

13.4.18 Axes.obj

Kann man mit «Enable Axes» einschalten.

Sieht, wenn man es rendert, folgendermassen aus:

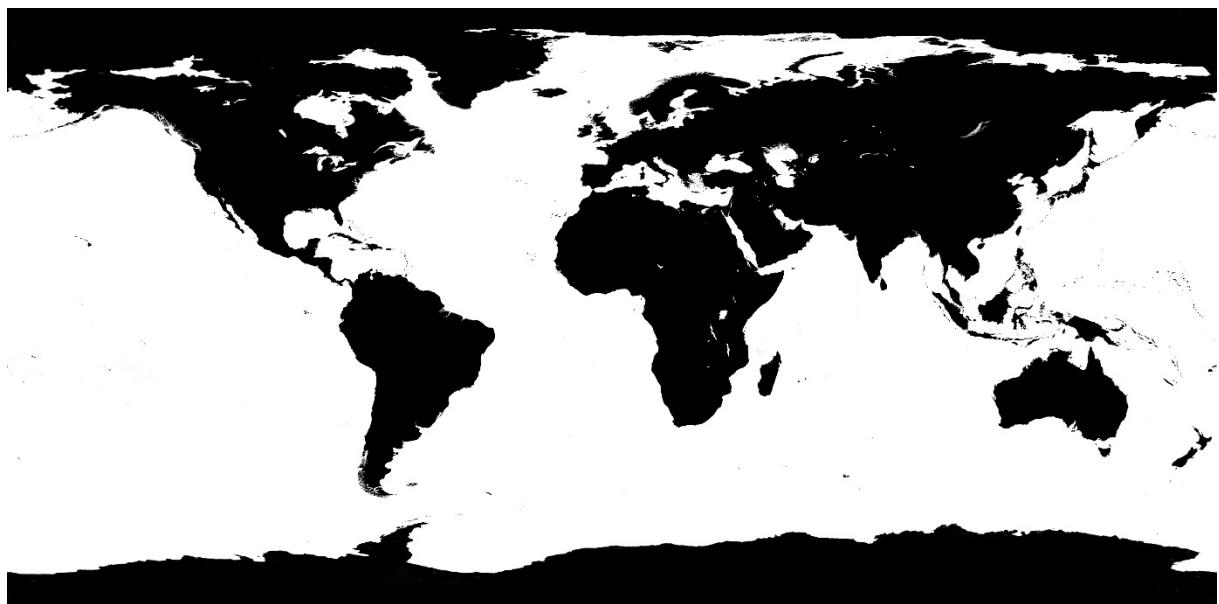


13.4.19 8081_earthnmap4k.jpg



Karte, welche für die Oberflächenfarbe im Programm verwendet wird. Mit Attribuierung:
<http://planetpixelemporium.com/earth8081.html>

13.4.20 8081_earthspecl4k.jpg



Karte welche für die Reflektivität der Erde verwendet wird. Attribuierung dieselbe wie oben.

13.4.21 Earth.obj

Modell einer perfekten Kugel mit passender UV-Projektion und so gedreht, dass er im Programm der Nullmeridian auf der XY-Ebene liegt.



Auf dem USB unter gleichen Namen und als «.blend»-datei zu finden

13.4.22 firmament.obj

Modell des Sternenhimmels auf einer Kugel, welche die Textur «skychart.png» verwendet.

Wurde so rotiert, dass der Frühlingspunkt auf der Z-Achse liegt.

Kann in «Tarflos» mittels «Enable Firmament» eingeschaltet werden.

13.4.23 skychart.png



Wird für firmament.obj verwendet und wurde mit dem Program «Cartes du Ciel erstellt»
<https://www.ap-i.net/skychart/en/start>

13.4.24 GUI_Gluon.fxml

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.scene.SceneAntialiasing?>
<?import javafx.scene.SubScene?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Slider?>
<?import javafx.scene.control.TextField?>
<?import javafx.scene.image.ImageView?>
<?import javafx.scene.layout.AnchorPane?>
<?import javafx.scene.layout.Region?>
```

```
<?import javafx.scene.shape.Rectangle?>
<?import javafx.scene.text.Font?>
<?import javafx.scene.text.Text?>

<!-- depthBuffer="true" -->

<AnchorPane prefHeight="900.0" prefWidth="1130.0" xmlns="http://javafx.com/javafx/15.0.1" xmlns:fx="http://javafx.com/fxml/1" fx:controller="javaCode.Controller">
    <SubScene fx:id="mainScene" depthBuffer="true" depthTest="ENABLE" fill="BLACK" height="900.0" layoutX="230.0" width="900.0">
        <antiAliasing>
            <SceneAntialiasing fx:constant="BALANCED" />
        </antiAliasing>
        <root>
            <Region />
        </root>
    </SubScene>
    <Slider fx:id="speedSlider" blockIncrement="0.1666666666666666" layoutX="546.0" layoutY="822.0" majorTickUnit="1.0" max="3.0" minorTickCount="60" prefHeight="65.0" prefWidth="540.0" showTickLabels="true" snapToTicks="true" />
        <Text fx:id="timeDisplay" fill="WHITE" layoutX="351.0" layoutY="64.0" strokeType="OUTSIDE" strokeWidth="0.0" text="00:00:00 0 Days" textAlign="CENTER" wrappingWidth="757.9830722808838">
            <font>
                <Font name="Consolas" size="42.0" />
            </font>
        </Text>
        <Button fx:id="oneXButton" layoutX="323.0" layoutY="810.0" mnemonicParsing="false" onAction="#speedToOne" prefHeight="64.0" prefWidth="64.0" text="1x">
            <font>
                <Font name="Consolas" size="20.0" />
            </font>
        </Button>
        <TextField fx:id="timeScaleFactorTextField" layoutX="393.0" layoutY="810.0" onAction="#updateTimeScaleFactorTextField" onInputMethodTextChanged="#updateTimeScaleFactorTextField" prefHeight="64.0" prefWidth="142.0" text="1">
            <font>
                <Font name="Consolas" size="24.0" />
            </font>
        </TextField>
        <Slider fx:id="scrollSlider" blockIncrement="0.01" layoutX="1080.0" layoutY="93.0" majorTickUnit="1.0" max="7.0" min="0.25" minorTickCount="60" orientation="VERTICAL" prefHeight="696.0" prefWidth="29.0" showTickLabels="true" />
        <TextField fx:id="altitudeTextField" layoutX="255.0" layoutY="756.0" prefHeight="44.0" prefWidth="150.0" promptText="altitude">
            <font>
                <Font name="Consolas" size="24.0" />
            </font>
        </TextField>
        <TextField fx:id="azimuthTextField" layoutX="415.0" layoutY="756.0" prefHeight="42.0" prefWidth="150.0" promptText="azimuth">
            <font>
                <Font name="Consolas" size="24.0" />
            </font>
        </TextField>
        <TextField fx:id="heightTextField" layoutX="575.0" layoutY="756.0" prefHeight="42.0" prefWidth="150.0" promptText="height">
            <font>
```

Anhang

```
<Font name="Consolas" size="24.0" />
</font>
</TextField>
<TextField fx:id="inclinationTextField" layoutX="735.0" layoutY="756.0"
prefHeight="42.0" prefWidth="150.0" promptText="inc">
<font>
<Font name="Consolas" size="24.0" />
</font>
</TextField>
<Button fx:id="timeResetButton" layoutX="261.0" layoutY="120.0" mnemonic-
icParsing="false" onAction="#resetTime" text="t = now">
<font>
<Font name="Consolas" size="24.0" />
</font>
</Button>
<Button fx:id="switchCameraButton" layoutX="859.0" layoutY="120.0" mne-
monicParsing="false" onAction="#switchCamera" text="Switch Camera">
<font>
<Font name="Consolas" size="24.0" />
</font>
</Button>
<Slider fx:id="timeSlider" blockIncrement="0.0" layoutX="261.0" lay-
outY="61.0" majorTickUnit="10.0" minorTickCount="0" prefHeight="65.0" pref-
Width="804.0" />
<Button fx:id="pauseButton" layoutX="255.0" layoutY="810.0" mnemon-
icParsing="false" onAction="#pauseAndPlay" prefHeight="64.0" pref-
Width="64.0" text="P">
<font>
<Font name="Consolas" size="20.0" />
</font>
</Button>
<Button fx:id="satToLocButton" layoutX="457.0" layoutY="120.0" mnemon-
icParsing="false" onAction="#satToLoc" text="SAT to LOC">
<font>
<Font name="Consolas" size="24.0" />
</font>
</Button>
<Button fx:id="recalculateButton" disable="true" layoutX="623.0" lay-
outY="120.0" mnemonicParsing="false" onAction="#recalculate" text="recalcu-
late">
<font>
<Font name="Consolas" size="24.0" />
</font>
</Button>
<Button fx:id="timeToObservationButton" layoutX="261.0" layoutY="164.0"
mnemonicParsing="false" onAction="#timeToObservation" text="t = obs">
<font>
<Font name="Consolas" size="24.0" />
</font>
</Button>
<Button fx:id="switchCalcButton" layoutX="895.0" layoutY="756.0" mnemon-
icParsing="false" onAction="#switchCalculation" prefHeight="44.0" pref-
Width="150.0" text="switch">
<font>
<Font name="Consolas" size="24.0" />
</font>
</Button>
<TextField fx:id="latitudeTextField" layoutX="255.0" layoutY="706.0"
prefHeight="44.0" prefWidth="200.0" promptText="latitude">
<font>
<Font name="Consolas" size="24.0" />
</font>
</TextField>
```

Anhang

```
<TextField fx:id="longitudeTextField" layoutX="465.0" layoutY="706.0" prefHeight="44.0" prefWidth="200.0" promptText="longitude">
    <font>
        <Font name="Consolas" size="24.0" />
    </font>
</TextField>
<Button fx:id="toCoordsButton" layoutX="895.0" layoutY="706.0" mnemonicParsing="false" onAction="#toCoords" prefHeight="44.0" prefWidth="150.0" text="toCoords">
    <font>
        <Font name="Consolas" size="24.0" />
    </font>
</Button>
<TextField fx:id="elevationTextField" layoutX="675.0" layoutY="706.0" prefHeight="44.0" prefWidth="210.0" promptText="elevation">
    <font>
        <Font name="Consolas" size="24.0" />
    </font>
</TextField>
<Rectangle arcHeight="5.0" arcWidth="5.0" fill="#3c3c3c" height="900.0" stroke="BLACK" strokeType="INSIDE" width="230.0" />
    <Button fx:id="axesButton" layoutX="26.0" layoutY="165.0" mnemonicParsing="false" onAction="#switchAxes" prefHeight="79.0" prefWidth="178.0" text="Enable Axes">
        <font>
            <Font name="Consolas" size="24.0" />
        </font>
    </Button>
    <Button fx:id="equatorialPlaneButton" layoutX="26.0" layoutY="254.0" mnemonicParsing="false" onAction="#switchEquatorialPlane" prefHeight="111.0" prefWidth="178.0" text="Enable equatorial Plane" textAlignment="CENTER" wrapText="true">
        <font>
            <Font name="Consolas" size="24.0" />
        </font>
    </Button>
    <Button fx:id="eclipticButton" layoutX="26.0" layoutY="375.0" mnemonicParsing="false" onAction="#switchEcliptic" prefHeight="79.0" prefWidth="178.0" text="Enable Ecliptic" textAlignment="CENTER" wrapText="true">
        <font>
            <Font name="Consolas" size="24.0" />
        </font>
    </Button>
    <Button fx:id="orbitTrackButton" layoutX="26.0" layoutY="464.0" mnemonicParsing="false" onAction="#switchOrbitTrack" prefHeight="79.0" prefWidth="178.0" text="Disable Orbit-Track" textAlignment="CENTER" wrapText="true">
        <font>
            <Font name="Consolas" size="24.0" />
        </font>
    </Button>
    <Button fx:id="orbitalPlaneButton" layoutX="26.0" layoutY="553.0" mnemonicParsing="false" onAction="#switchOrbitalPlane" prefHeight="111.0" prefWidth="178.0" text="Enable orbital Plane" textAlignment="CENTER" wrapText="true">
        <font>
            <Font name="Consolas" size="24.0" />
        </font>
    </Button>
    <Button fx:id="firmamentButton" layoutX="26.0" layoutY="674.0" mnemonicParsing="false" onAction="#switchFirmament" prefHeight="79.0" prefWidth="178.0" text="Enable Firmament" textAlignment="CENTER" />
```

```
wrapText="true">
    <font>
        <Font name="Consolas" size="24.0" />
    </font>
</Button>
<Button fx:id="writeToFileButton" layoutX="26.0" layoutY="763.0" mnemonicParsing="false" onAction="#switchWriteToFile" prefHeight="111.0" prefWidth="178.0" text="Enable write to file" textAlignment="CENTER" wrapText="true">
    <font>
        <Font name="Consolas" size="24.0" />
    </font>
</Button>
<ImageIcon fx:id="imageIconContainer" fitHeight="143.0" fitWidth="142.0" layoutX="44.0" layoutY="14.0" pickOnBounds="true" preserveRatio="true" />
</AnchorPane>
```

13.4.25 appicon_big.png

Wird benutzt für das Logo im Programm oben links. Siehe Abbildung 32 auf Seite 28.

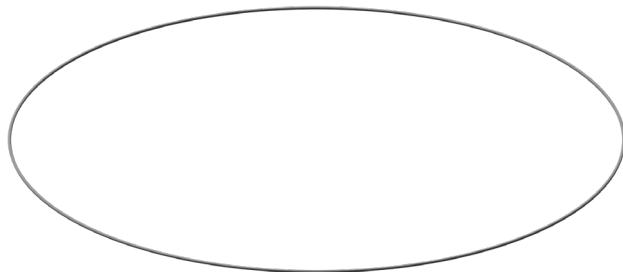
13.4.26 appicon_small.png

Wird benutzt für das Fenster-Logo und das Logo in der Taskbar:



13.4.27 circularOrbit.obj

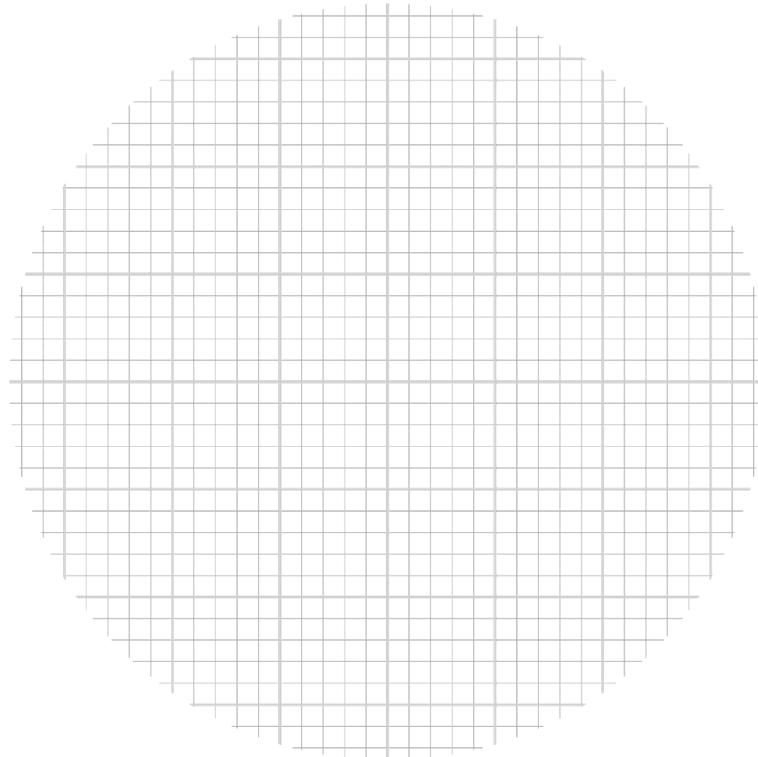
Wird für das Anzeigen des Umlaufbahn Pfades verwendet.



Kann man mit «Enable Orbit Track» ein- und ausschalten.

13.4.28 **eclipticScale.png**

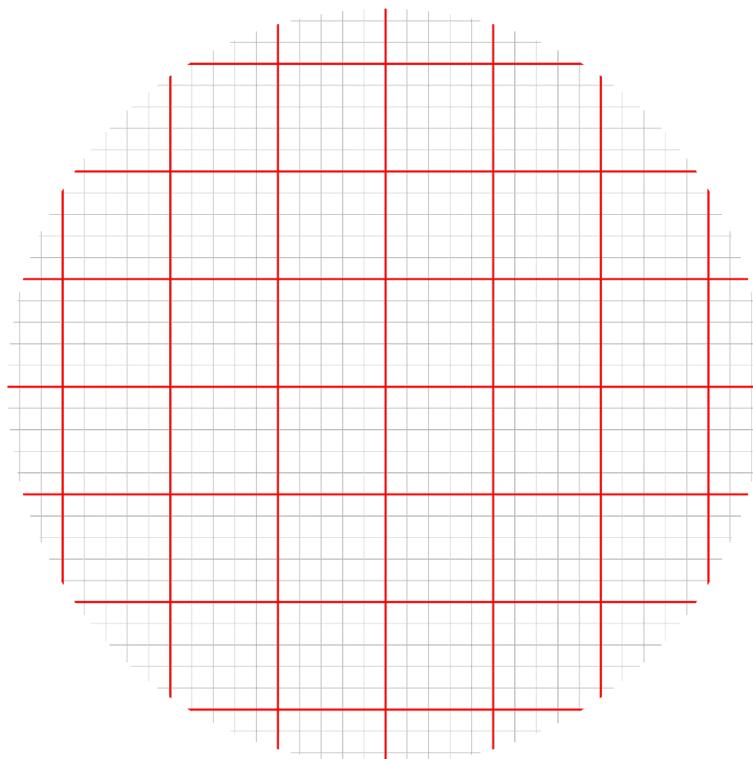
Gitter, welches für die Ekliptikebene verwendet wird



Kann mit «Enable Ecliptic ein- und ausgeschaltet werden

13.4.29 **equatorialPlaneScale.png**

Gitter, welches für die Ekliptikebene verwendet wird



Kann mit «Enable Ecliptic ein- und ausgeschaltet werden

13.4.30 selfIlluminationLocation.png

1x1 Pixel grosse orangerote Bilddatei, welche existiert, damit der Ort in dieser Farbe leuchtet



HEX: ff4f4f, RGB: 255, 79 ,79

13.4.31 selfIlluminationOrbitalPlane.png

1x1 Pixel grosse weisse Bilddatei, welche existiert, damit die Umlaufbahn in dieser Farbe leuchtet



HEX: ffffff, RGB: 255, 255 ,255

13.4.32 selfIlluminationSatellite.png

1x1 Pixel grosse blaue Bilddatei, welche existiert, damit der Satellit in dieser Farbe leuchtet



HEX: c8c8ff, RGB: 200, 200 ,255