MICHIGAN STATE
UNIVERSITY

Simulation and Classification of Objects in Orbit Using Scattering Patterns

ME 490: Independent Study

Student
Philipp Waeltermann
(waelterm@msu.edu)

Faculty Mentor
Dr. Firas A. Khasawneh
(khaswn3@egr.msu.edu)

Department of Mechanical Engineering
Michigan State University
East Lansing, MI 48824

May 4, 2018

Acknowledgements

First and foremost, I would like to thank my research mentor Dr. Firas A. Khasawneh for offering me this incredible opportunity to work with him on this project. Without his ideas, his continuous support and guidance, as well as his technical expertise, this project would have not been possible. Furthermore, I would like to thank Dr. Geoffrey Recktenwald for connecting me with Dr. Firas Khasawneh and making the cooperation possible.

I would also like to thank Dr. Patton Allison and Dr. John Luginsland for sharing their expertise and knowledge. I am especially grateful for Dr. Allison for lending me a couple of his books on astrodynamics and space mechanics [1, 2] for the duration of the project.

Contents

1 Introduction

Space object identification and classification is important in many fields including astronomy, astrodynamics, and celestial mechanics. This ability to understand space objects leads to an improved Space Situational Awareness (SSA) which enables the accurate characterization and tracking of space objects [3]. In many cases, the observations of the space objects are performed using earth-based or orbit-based sensors. These sensors typically collect scattered light information from the objects moving in space which includes asteroids and resident space objects (RSOs).

One of the important space objects to study and understand are the earth orbiting objects which include active/inactive satellites and space debris. The signals used to study these objects often include infrared-sensors [4], which do not require solar illumination, and cameras that measure the intensity of scattered light [5]. More information can be inferred about the nature of the earth-orbiting objects by applying machine learning to a collection of signals such as the orbit information and the light curves [5]. However, there are two main difficulties in applying machine learning to classify space objects: 1) the size of training data is limited to historic observations which may not be sufficient for training a classifier, and 2) historic data does not help us adapt to scenarios that have not been previously recorded.

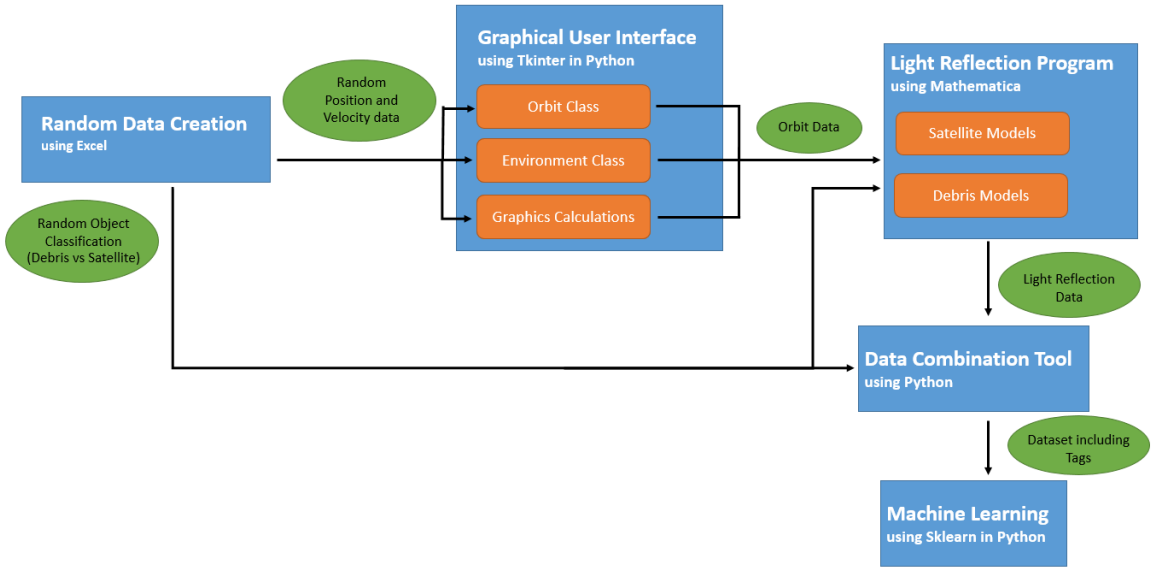

Figure 1: Satellite Simulation and Classification Toolchain

Therefore, the objective of this work is to improve space situation awareness by simulating earth-based light intensity measurements of earth-orbiting objects. Specifically, we build a pipeline that involves three main components (see Fig.1):

1. **Orbit tool**: This tool uses Python code to calculate the orbit parameters for the RSO and it includes a user interface to enable the easy usage of the orbit class and its features as well as a Graphical User Interface (GUI) with the necessary graphing capabilities.
2. **Light reflection tool**: This tool combines the simulated orbit data with the base Mathematica ray tracing abilities to compute the solar light intensity that is scattered from the

RSO. A camera location on earth is then simulated to capture the perceived light intensity at the surface of earth within the camera's filed of view. This enables simulating an arbitrary number of time series that represent the light intensity of a large number of RSOs.

3. **RSO classification tool**: We also show how the framework can be incorporated into a machine learning framework by implementing an elementary classifier using k-NN classifier (we set $k = 1$), and two similarity measures: 1) the discrete Fréchet distance [6], and 2) the discrete time warping measure with an ℓ_2 Euclidean metric [7].

The simulation was designed using a set of reasonable assumptions (see Sections 2.2 and 4) and it was tested against several textbook cases [1].

A list of the main activities accomplished during this project, as well as a reference to the corresponding section within this report, include:

1. **Characterizing Satellites' orbits** (Section 2)
   (a) Describe all possible satellite's orbits
   (b) Write `Python` Program that calculates all variables needed to define a satellite's orbit based on position and velocity, three positions, or two positions and the time of flight.
   (c) Validate the calculated orbits parameters with example problems from Ref. [1]
2. **Animation of Satellite's orbit** (Section 7.2)
   (a) Use an appropriate `Python` animation library (`matplotlib`)
   (b) Plot/Animate the orbit of a satellite in a 3D plot
   (c) Plot/Animate the orbit of a satellite in a 2D plot simulating a camera perspective from earth's surface)
3. **Animation of Satellite reflection** (Section 5.2)
   (a) Find resources that give a model of the reflection patterns of a satellite
   (b) Synthesize and program a model for the reflection pattern of satellites and debris
   (c) Visualize the reflection pattern using the previous animation and the reflection model
4. **Automate reflection data creation** (Section 8)
   (a) Automate the orbit creation model in Python to create bulk data
   (b) Create Mathematica Program to create bulk data of light reflections
   (c) Combine the results of the two tools to create data used for machine learning
5. **Machine Learning Framework** (Section 6)
   (a) Train a classifier using `Python's sci-kit`
   (b) Simulate and tag a large number of time series
   (c) split, train, and test the simulated time series

2 Orbit Characterization

Satellites move in predetermined paths which are called orbits. Figure 2 shows the different types of orbits where each orbit is completely characterized by the corresponding position and velocity vectors with respect to earth. The type of the orbit often reveals information about the detected object. For example, hyperbolic or parabolic orbits are non-repeating trajectories. This means that the object will leave earth's gravitational field. Examples of objects with non-repeating orbits include space-shuttles and interplanetary rockets.

In contrast, an elliptic or circular orbit is a repeating trajectory, meaning that it will return to its initial position after a full rotation. If an object is found to have this type of orbit, it is likely a satellite. These earth-orbiting satellites can range from large communication satellites to

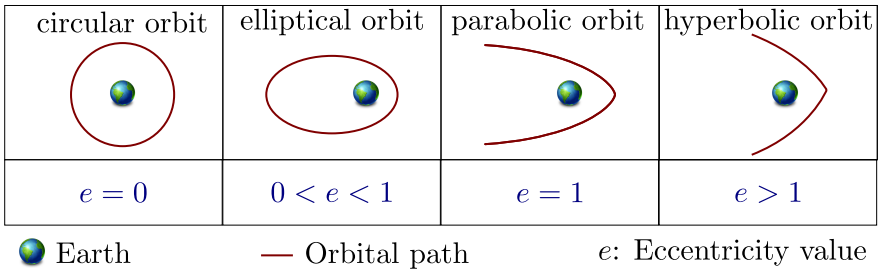

Figure 2: Types of Orbits.

small camera satellites. However, not all elliptic or circular orbits result in infinite repetition. If the distance between the object in orbit and earth continues declining, it becomes on a collision path with earth. This scenario can occur, for instance, when a satellite's placement into orbit malfunctions, or when a ballistic missile is launched. All these examples show that correct orbit determination can lead to important conclusions about the detected objects.

The goal of this section is to discuss the methods used for determining orbit types, defining the orbit in relation to earth, and predicting the path of objects in orbit. We start by discussing the variables that describe an orbit in Section 2.1. Section 2.3 discusses the model assumptions that we use and comments on the consequences of neglecting some effects on the results.

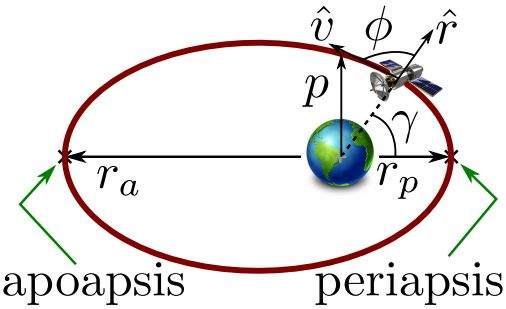

Figure 3: Elliptical Orbit Characteristics

2.1 Descriptive Variables of an Orbit

There are nine important variables describing an orbit: Eccentricity (e), the semi-latus rectum (p), and the angular momentum vector h, determine the shape and size of the orbit. Six angles will then be used to describe the orientation of the orbit to earth. Some of these angles may be undetermined. This is a result of one of the vectors (e.g angular momentum vector) having a magnitude of zero. This leads to a vector without direction; as a consequence, every angle that references that vector will be undetermined. There are three different measurements that can be used to determine the orbit of an object [1]. Any of these sets of data are sufficient to completely define the orbit's parameters. These methods for determining the orbit of an object are:

1. Using a position and a velocity vector relative to earth. This is the easiest method.
2. The object's orbit can also be determined from three different positions. The positional data is then used to calculate the velocity vector for one of the positions. Once this is done, the algorithm for case 1 can be used.

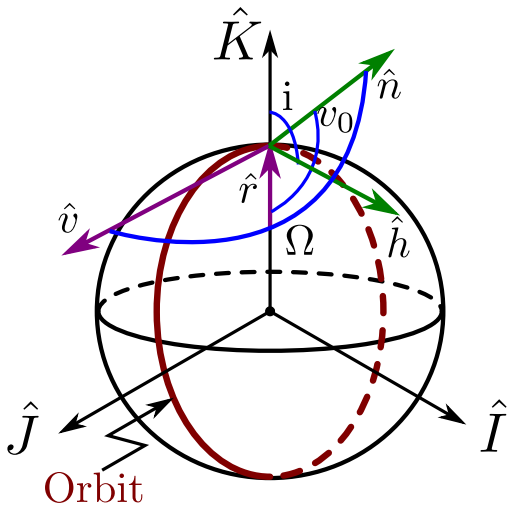

Figure 4: Example Orbit in geocentric-equatorial reference frame ($e = 0$; $p = 1$; $l_0 = 90°$; $u_0 = 90°$; $\Omega = 180°$; $i = 90°$)

3. The final option to calculate the orbit of an object is from two positions and the time of flight between the two locations.

The last method is the most difficult method because it involves solving what is known as the "prediction problem." This problem indicates that there is no analytical approach for determining the position of a satellite based on its time of flight even when its orbit is known thus necessitating a numerical solution. Some of the numerical methods for solving the prediction problem include using the p-iteration method, the universal value method, the f and g series method, or the original Gauss method. Using these methods a velocity vector for one of the positions can be found, and based on this velocity the orbit elements can be found as in case 1. Section 2.2 outlines our numerical implementation using Python for the first two methods for computing the orbit.

2.2 Numerical Computation of the Orbit

The different algorithms described in the introduction to Section 2 for computing the orbit of an object were implemented using Python. These algorithms compute the orbit elements starting from a set of known orbit observables. The first algorithm that uses position and velocity information was found to be reliable when tested with several practice problems.

A second algorithm was implemented to find the velocity vector based on three known orbit positions. We also found this algorithm to reliably yield the correct orbit parameters. By combining the second algorithm with the first one described above, the orbit elements can be found.

The third algorithm for computing the orbit relies on having information on two orbit positions as well as the time to travel between these positions. This is called the time of flight. In this algorithm it is necessary to estimate a velocity vector based on these two positions and the time of flight. Other than in the second algorithm, this velocity vector cannot be found analytically. There are four numerical methods that can be used to calculate the velocity vector:

1. p-Iteration: Using an initial guess for the semi-latus rectum and run an optimization algorithm until the calculated time of flight for the semi-latus rectum converges to the desired value.

2. Universal Variable: Using an initial guess for the universal variable z and run an optimization algorithm until the calculated time of flight for the z converges to the desired value.
3. Gauss Method: Uses Gaussian Iteration to find velocity vector
4. f and g series: F and G can be used as terms of eccentric anomaly and they can help defining the velocity vector

In this work we only implemented the first two methods for estimating the velocity. However, the current implementation of the first velocity estimation algorithm is not reliable and it only gives the correct results a single test case out of 3. Further, the implementation of the universal variable method does not converge if the two positions are too far apart while the p-iteration method gave significant errors in the final answer.

In conclusion, whereas the algorithms that determine the orbit using a position and a velocity vector relative to earth or using three different positions on the orbit were found to be reliable, algorithms that determine the orbit based on two positions and the time of flight are not sufficiently tested. The difficulties in determining the orbit based on the time of flight stems from the prediction problem mentioned in Section 2.1. Therefore, for the remainder of this project we focus on the former two and we defer the latter for future work.

Further, while the described numerical methods have worked in some scenarios, it is not clear when each method should be used. Without a set of rules that determine whether a method converges and is accurate for a certain type of problems, it is very difficult to analyze the capabilities of the different methods.

2.3 Perturbations

Two main assumptions have been made to create the model of the satellite:

1. Both, the satellite and earth are spherically symmetric.
2. There are no forces other than the gravitational force between satellite and earth.

The effect of the unconsidered perturbations will be discussed in the following sections.

2.3.1 Multiple Objects (gravitational fields)

As there are more objects in space than earth and satellites. There are obviously more gravitational fields than the ones considered in the created model. Due to its immense mass the sun has the strongest gravitational field on a satellite followed by the moon and planets of our solar system. However, both of those gravitational constants have a magnitude of less than .07 percent of the earth's gravitational field. These small gravitational effects cannot be neglected for long-time path predictions of satellites because the effects of the gravitational constants accumulate. However, since the purpose of this project is to simulate earth-based observations of the the satellite within a predefined field of view, which a very short portion of the overall trajectory of the satellite, these effects can safely be neglected.

2.3.2 Oblateness of Earth

A bigger effect than the existence of additional gravitational fields is the imperfection that earth is not completely round but slightly oblate. This leads to a gravitational field whose strength is not only dependent on the distance between the satellite and earth, but also changes depending on the exact orientation of earth. The effect can be incorporated into the round earth model by correcting

the gravitational constant with a position dependent adjustment of up to 10^{-3}. This corresponds to around 0.1% of earth's gravitational field. Neveretheless, similar to the perturbation through multiple objects in

Table 1: Effects of Different Perturbations

Object/Perturbation	**Acceleration in G's on 200nm Earth Satellite**	**Percentage Error Compared Earth's Gravitational Field**
Earth	0.89	n/a
Earth Oblateness	1.0×10^{-3}	0.1123
Sun	6.0×10^{-4}	0.0674
Moon	3.3×10^{-6}	0.0037
Jupiter	3.2×10^{-8}	0.0000

2.3.3 Atmospheric Drag

The atmospheric drag is extremely hard to predict because it depends on the fluctuating atmospheric pressure. It can, however, be assumed that for a high altitude satellite, the atmospheric drag can be neglected for short-term path predictions. Since drag forces oppose the velocity direction, the velocity will decrease over time and the satellite's orbit, if not corrected, will ultimately become smaller. Eventually, the satellite might leave its stationary orbit.

2.3.4 Radiation Pressure

Radiation Pressure—the pressure introduced by photons hitting a surface— has generally a very small effect even on satellites with big surface area. It can therefore be neglected in the model.

2.3.5 Thrust

We assume that the satellite remains in its orbit during the period of observation without any thrust adjustments. Therefore, thrust is irrelevant in our model.

2.4 Conclusion on Perturbations

The model makes reasonable assumptions for the purpose of the project. However, it must be noted that long-term predictions of a satellite's path will not be possible with this model. The prediction of fractions of a full revolution of the satellite's orbit should be realistic.

3 Space Mechanics

4 Light scattering overview

The scattering of light and other electromagnetic radiation is a topic that has been widely researched [8]. It has applications in astronomy, lens and telescope design, aerosol optics [9], x-rays, and magnetic resonance imaging (MRI). Two types of broad problems are related to the scattering

of light: (1) **the direct scattering problem** and (2) **the inverse scattering problem**. The former is related to the determination of how the radiation or particles are scattered based on the properties of the scatterer, while the latter is related to inferring the properties of a scatterer based on how it scatters incoming radiation or particles.

The general mathematical formulation for electromagnetic waves absorption and scattering is given by Maxwell's equations [10]. Exact solutions for these equations exist only for very limited cases while numerical solutions are necessary in the most general case. The resulting exact or approximate formulations depend on the physical characteristics of particles such as size, shape, and refractive index. Specifically, the ratio of the scatterer's size to the wavelength provides guidance as to which theory to use for studying scattering and absorption, see Table 2.

In addition, the shape of the scatterer is also important. For example, scattering and absorption by spheres, both homogeneous or layered, of any size requires no approximation theory since the Mie (or Mie-Lorenz) theory provides an exact treatment. A Python suite for mie scattering can be found at `www.t-matrix.de`. However, for small spherical particles it is possible to obtain accurate answers using faster computations with approximate methods such as the Rayleigh approximation. An overview of several practical numerical methods for computing electromagnetic scattering of non-spherical particles can be found in [11].

The analysis of electromagnetic scattering gives the surface field, i.e., the light scattering on the field of the scatterer. To obtain the propagation of the scattered wave away from the scatterer, then we need to consider scattering in the far-field zone. In this range the scattered field is polarized transverse to the propagation direction and decays inversely with distance from the scatterer. The scattered field is also different when considering a single particle versus a group of particles.

Particle size	$\frac{\text{particle size}}{\text{wave length}} << 1$	$\frac{\text{particle size}}{\text{wave length}} \approx 1$	$\frac{\text{particle size}}{\text{wave length}} >> 1$
Analysis method	Rayleigh approximation	Mie scattering [9]	Geometric optics [12]

Table 2: Scattering models classification. The region where $\frac{\text{particle size}}{\text{wave length}} \approx 1$ is called the resonance region. The geometric optics approximation is also known as the ray-tracing or ray optics approximation.

Since this project is concerned with the scattering of light from artificial active or inactive satellites as well as large space debris, we will be using the geometrical optics approximation. Section 4.1 discusses this approximation in more detail. Note that the geometric optics is the zero wavelength limit of wave optics and that wave optics can also be derived from geometric optics [13].

4.1 Geometric optics

The geometric optics approximation is also known as the ray-tracing or ray optics approximation. There are two types of ray tracing: sequential and non-sequential. In the sequential ray tracing, the ray is traced in a pre-defined order of scatterers and rays hit each surface once according to their order. In non-sequential ray tracing, rays can intersect multiple objects in any order and they can intersect the same object more than once. We will be focusing on the scattering from a single celestial object, so the distinction be tween sequential and non-sequential scattering is not relevant. Further, since we are simulating observing the scattered light from the surface of earth, we are interested in far-field scattering, i.e., scattered light away from the surface of the scatterer. Section 4.2 discusses our criteria and choice of an optics software that will perform the desired

computations.

4.2 Optics Analysis Software

There are several choices for a software that will assist in our direct scattering problem. Ideally, this software will be able to

1. Calculate the far-field light intensity using the geometric optics approximation for objects that are far apart, i.e., the sun as a light source, the satellite as the scatterer, and a point on earth as the far-field zone for the light intensity.
2. Load a CAD file as the scatterer.
3. Be able to interface with a Python script to allow the calculation of the light intensity as a function of time for different orientations and positions of the scatterer.
4. Be reasonably priced ($\ll$ \$5000).

To accurately simulate the reflection of a satellite, several available optics programs were evaluated. Specifically, six commercial products have been considered for the project.

- **Wolfram Mathematica:** After receiving a free Mathematica student license, we found that the optics libraries only works with specific 3D objects like spheres, boxes, and cylinders. Therefore, only compound shapes composed of theses basic shapes can be studied.
- **OSLO:** After testing the free trial, the program was found to be inappropriate for long distance ray tracing. The program is mainly used to design lenses and other optics equipment.
- **ZeMax:** There exists no student or academic license for the software, and the regular price is $\$5,600$, which is outside of our price point.
- **FRED:** Similar to OSLO, the product specializes in designing optics equipment using micro-optics. This is not necessarily applicable to our long distance reflection.
- **CodeV:** The product's regular price of $\$7,000$ is too high. However, free student licenses are available, and we obtained one. Upon evaluating CodeV, we discovered that it is geared more towards lens design and that Lightools, another product by the same company Synapsys performs the ray tracing that we need. The latter option was not pursued because of its high cost.
- **Wolfram Mathematica & Optica add-on:** Base Mathematica has some ray-tracing capabilities that are extended by the Optica add-on, which is sold separately. This is a library for Mathematica that gives additional functionality. Its academic license costs \$998.

Based on our investigation Mathematica seemed to satisfy most of the desired features for ray tracing. Mathematica has light and reflection tools including a lighting function which simulates different lighting on basic objects. These objects include spheres, cylinders, boxes, and any other objects that can be obtained by combining these simple shapes. Unfortunately, importing CAD models is not possible using the base Mathematica installation but we were able to mimic the shape of a satellite by combining a cylinder and a box. The result can be seen in Fig. 8. Further, it seems that there is no easy integration of Mathematica functions into Python because direct communication between Python and Mathematica is not possible. Consequently, indirect communication via terminal is necessary and the only way to interface the two software is through this tedious series of steps:

1. Write the input parameters for the Mathematica script into a comma separated values (CSV) file using Python.
2. Give the Python file permission to open other programs from the terminal.

3. Run a terminal command from Python to the execution of the Mathematica script using the CSV file as input.
4. Write an output file using Mathematica.
5. Import Mathematica's output file back into the Python program.

We decided to bypass these steps and instead focus on a two-step program execution whereby we run Python to compute the orbit and generate the corresponding CSV file of the orbit, see Fig. 1 for an outline of the pipeline. We then load the generated CSV file into a Mathematica script and perform the needed light intensity computations. The resulting light intensity is then read back into Python for further analysis and plotting.

5 Constructing the Camera Image from Orbit Data

We now switch focus to creating a camera picture simulation from the orbit data. This process involves two steps: (1) projecting the 3D orbit of the satellite onto the a 2D plane that represents the field of view of the camera, and (2) computing the light intensity in this 2D plane to construct the simulated image of the satellite. We describe how we successfully designed an algorithm to perform both of these tasks using Mathematica in Sections 5.1 and 5.2, respectively.

5.1 Perspective Transformation

When a camera takes a picture, it transforms a three-dimensional scene into a two-dimensional image. In order to simulate a camera picture of the orbit, a similar transformation must be performed. One method to accomplish this transformation is perspective projection which can be used to turn the three-dimensional orbit coordinates into a two dimensional plane.

In this work we used the perspective projection method and our input data was the 3D orbit points, the distance, the orientation, and the field of view of the camera. These inputs were used to create a transformation matrix resulting in a 2D projection $\mathbf{b}_{x,y}$ for each point in the orbit [14].

In order to obtain the 2D projection we first need to account for the perspective of the camera which results in farther points showing up smaller than closer ones. This requires what is called a camera transform given by

$$\begin{bmatrix}\mathbf{d}_x\\ \mathbf{d}_y\\ \mathbf{d}_z\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\\ 0 & \cos\theta_x & \sin\theta_x\\ 0 & -\sin\theta_x & \cos\theta_x\end{bmatrix}\begin{bmatrix}\cos\theta_y & 0 & -\sin\theta_y\\ 0 & 1 & 0\\ \sin\theta_y & 0 & \cos\theta_y\end{bmatrix}\begin{bmatrix}\cos\theta_z & \sin\theta_z 0\\ -\sin\theta_z & \cos\theta_z & 0\\ 0 & 01\end{bmatrix}\left(\begin{bmatrix}\mathbf{a}_x\\ \mathbf{a}_y\\ \mathbf{a}_z\end{bmatrix} - \begin{bmatrix}\mathbf{c}_x\\ \mathbf{c}_y\\ \mathbf{c}_z\end{bmatrix}\right), \tag{1}$$

where $\mathbf{a}_{x,y,z}$ and $\mathbf{a}_{x,y,z}$ are the 3D positions of the points A and C (C refers to the camera), respectively, that we are projecting, $\theta_{x,y,z}$ is the orientation of the camera in Tait-Bryan angles, $\mathbf{e}_{x,y,z}$ is the viewers position relative to the display surface that goes through point C.

After obtaining the transformed coordinates $\mathbf{d}_{x,y,z}$ we projected these points onto the 2D plane using

$$\begin{aligned}\mathbf{b}_x &= \frac{e_z}{d_z}\mathbf{d}_x - \mathbf{e}_x,\\ \mathbf{b}_y &= \frac{e_z}{d_z}\mathbf{d}_y - \mathbf{e}_y.\end{aligned} \tag{2}$$

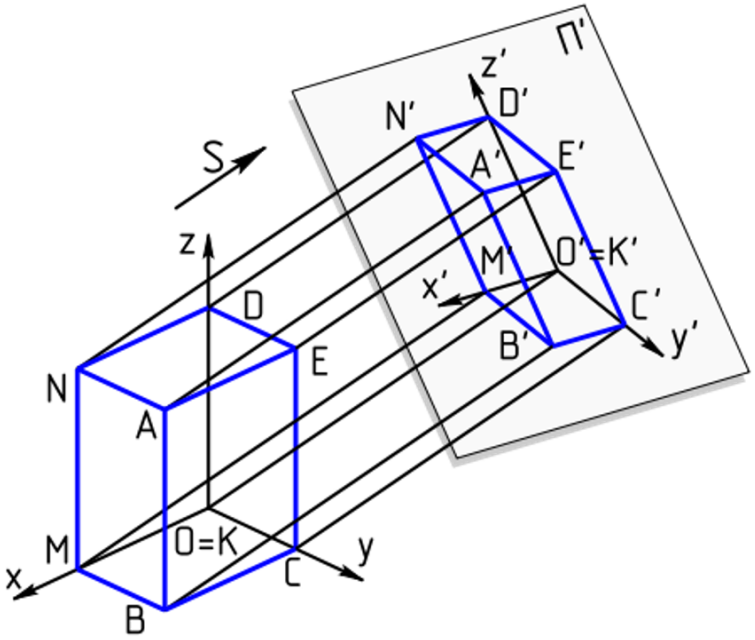

Figure 5: A pictorial example of the perspective projection algorithm [14]. It shows the projection of a 3D object into a 2D plane.

The resulting plot would be an accurate representation of the satellite seen from earth if its light intensity was always constant. However, the light reflection intensity will generally vary and this variation must be considered for a realistic camera plot. A light intensity array will be used to define the thickness of the scatter-points in the camera plot. This gives a more realistic two-dimensional camera picture as seen in Figure 6. To accurately create the camera plot, points outside of the field of view had to be found and clipped. This was accomplished by first constructing the vectors 1) from the center of earth to the all the points in the orbit, and 2) from the center of earth to the camera position. The angles between these two vectors were then calculated and we deleted all orbit points for which the angle was greater than half the field of view thus retaining only the points that the camera would record. However, the computed values did not necessarily have the correct time ordering which is necessary to record a continuous trajectory of the earth-orbiting object. In order to obtain the correct ordering we ensured that the first orbit point was the left-most right-most orbit point. Specifically, the position with the greatest angle relative to the camera position was found, and the array of orbit points was then shifted so that it would begin with the left-most orbit point. This step provided continuous trajectories as inputs to the light intensity computations, see Section 5.2.

5.2 Light Reflection Algorithm

The second step for simulating the light intensity of a satellite is an algorithm for obtaining its light reflection based on the satellite's position and orientation. The sun was assumed to be the sole light source, sun rays were assumed to parallel, and we ignored atmospheric diffraction in the model. Our parallel light source assumption is practical because although the sun is theoretically

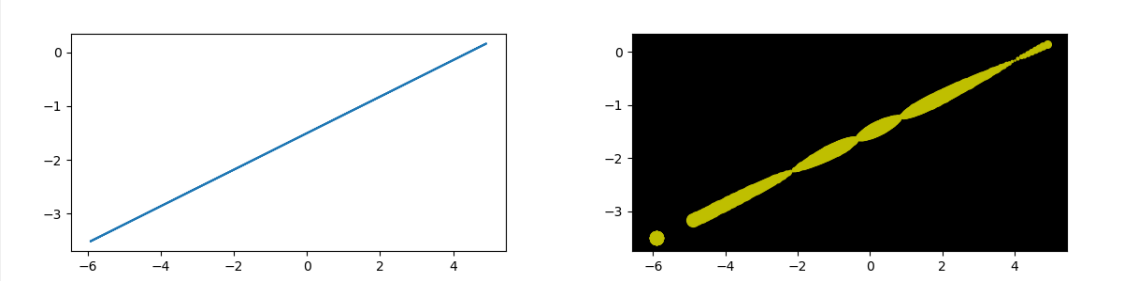

Figure 6: Camera Plot as result of Perspective Transformation for an orbit with sinusoidally varying light intensity.

a point light source, by the time sunlight reaches earth the light array can be assumed parallel.

The satellite is modeled using a cylinder for the satellite body and two plates for its wing, see Figs. 8 and 9. Five different debris models have also been implemented. All of these consist of spheres with different sizes at different positions. Thes can be seen in Figure 7 As the satellite

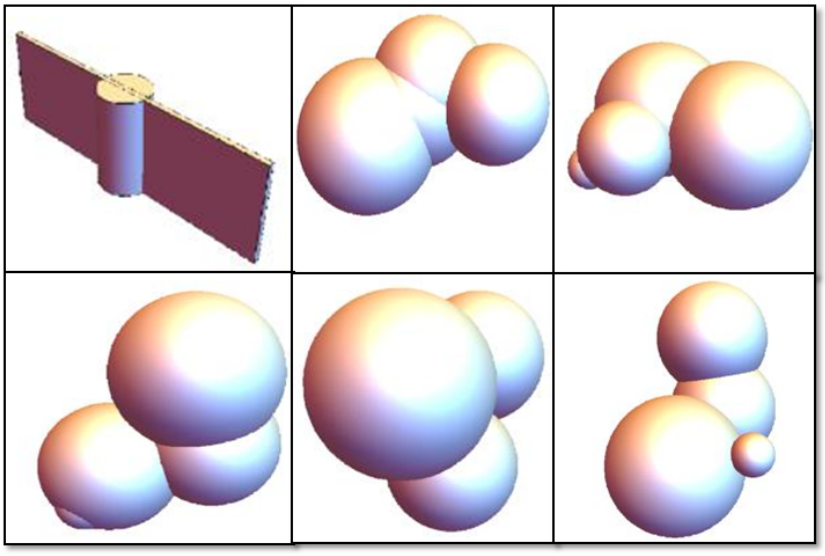

Figure 7: Five three dimensional models for debris and one for a satellite

travels along its orbit, its perspective projection varies and occupies a different area at each point in time. When using Mathematica for capturing the light intensity, the program zooms in to the size of the projected image which creates a time series of nonuniform images. We solved this problem by encasing the satellite model with a completely transparent sphere whose diameter is equal to the satellite's wing span. This sphere prevented the undesirable auto-scaling effect caused by the rotation of the satellite while avoiding any interference with the satellite's light reflection. We implemented the corresponding algorithm in Mathematica according to the following sequence:

1. Build a 3D model of a satellite using a cylinder as the core and a rectangular box as the wings.
2. Add a lighting source using the directional parameter. This results in a parallel light source.

3. Set the relative orientation of satellite and camera.
4. Set the background of the plot to black.
5. Save the created plot as a 2D image.
6. Acquire the average pixel brightness of the image.
7. Use the average brightness in combination with the distance of the satellite to calculate the light reflection intensity.

The resulting light reflection on the surface of the satellite is shown in Figs. 8 and 9.

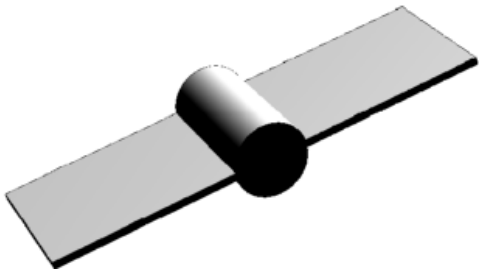

Figure 8: Light reflection on white background.

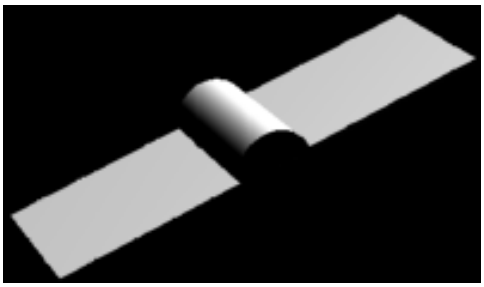

Figure 9: Light reflection on black background.

5.3 Combinatorial Explosion: The Effect of Numerous Design Variables

A realistic determination of the light reflection from earth-orbiting objects involves a high level of complexity due to the large number of possible factors. Examples of these factors include the camera position or the satellite orientation. This resulted in a very large number possibilities for the combinations of input variables into the simulation, a situation referred to as combinatorial explosion. Handling this amount of unknown input arguments remains a challenge for any light intensity simulation of RSOs. Some examples of the inputs used for the light reflection determination include:

- Position of camera
- Position of satellite
- Position of Sun (time of the year)
- Position of Earth
- Orientation of camera
- Orientation of satellite
- Orientation of Earth
- Field of view of camera

All of the parameters above influence the light reflection pattern algorithm. Some of the inputs, like the position of the sun, can be assumed to be steady. Others, like the orientation of the satellite, have to be defined by a time-varying model.

If all of the new input variables are combined with the infinite amount of possible elliptical satellite orbits around earth, we can see an exponential increase in the possible combinations of the parameters. This makes it increasingly difficult to develop a program that can account for every single combination of these parameters. While there are still issues with the integration of Mathematica into Python, for our project, it is crucial to create reasonable data for main scenarios; this data will then be used as ground truth data for Machine Learning in Section 6. Therefore, accounting for all possible inputs becomes a secondary goal.

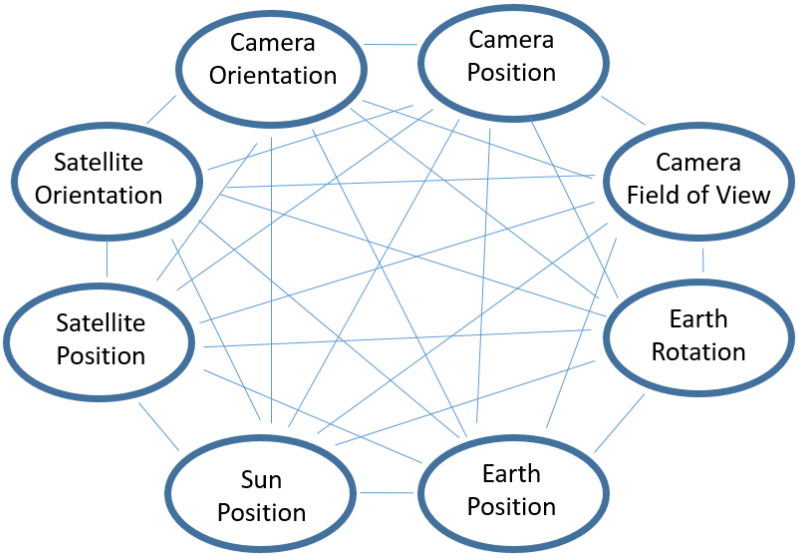

Figure 10: Combinatorial explosion due to the large number of inter-dependent parameters.

Several simplifications were introduced to ensure that enough ground truth data could be created before the end of the project. These simplifications include:

1. That the camera orientation is pointed in the direction of the starting position of the satellite.
2. That the camera is placed where the surface of the earth intersects an imaginary line between the center of earth and the satellite's starting position.

More simplifications will be made regarding the position and orientation of earth, sun, and satellite, and they will be introduced when the orbit model in Python is combined with the light reflection model in Mathematica.

6 Machine Learning

The simulation generates orbit and light intensity data in the form of time series, i.e., ordered sequences of (real) values indexed by time. We define a time series T according to

$$T : ((p_0, t_0), \ldots, (p_{n-1}, t_{n-1})),$$

where $p_k \in \mathbb{R}^2$, $t_k \in \mathbb{R}$ $\forall k \in [1, \ldots, n]$, $\forall n \in \mathbb{N}$, and n is the length of the time series T. When we refer to different time series we will include a superscript T^i to denote the ith time series.

It is important to normalize the raw time series before any meaningful classification can occur [15]. In this work we mean normalize the time series and any sub-sequence of the time series using the Z-normalization

$$T = \frac{T - \mu_T}{\sigma_T},$$

where μ and σ are the mean and standard deviation of the time series, respectively. The resulting normalized time series has zero mean with a standard deviation of one.

After normalization, we choose a supervised machine learning approach that uses a k-nearest neighbor classifier. The main ingredients for this classifier are (1) the k-nearest neighbor algorithm (kNN), and (2) a similarity measure. For the similarity measure we have a few options that we can obtain from two broad categories for distances on time series [16]

- Shape-based distances such as Hudsdorff and Fréchet distances.
- Warp-based distances such as discrete time warping distance (DTW) [7, 6], Longest Common Subsequence (LCSS) [17], Edit Distance on Real sequence (EDR) [18] and Edit distance with Real Penalty (ERP) [19].

In this work We utilized the standard k-nearest neighbor algorithm implemented in Python's `scikit-learn` package (`sk-learn`) with $k = 1$. We also used two different distance metrics: the discrete Fréchet distance (DFD) [6], and the discrete time warping distance (DTW). However before we apply the classifier to the simulated light intensities in Section 6.2, we apply the same classifier to a time series from the UCR archive [20] in Section 6.1.

6.1 Classifier Test 1: Synthetic Control Time Series

In order to verify our classification pipeline before testing it on the light intensity data, we tested the 1-NN classifier on the Synthetic Control time series from the UCR Time Series Classification Archive [20]. This set has 600 time series tagged using 6 different classes. We randomly split the data into 80/20 training/testing sets and applied the 1-NN classifier using DTW and DFD where we for the DTW we chose the euclidean distance as the cost measure. The process was repeated 10 times and each time the success rate for each distance measure was computed. The resulting success rates are shown in Tab. 3.

Distance metric	average success rate	standard deviation of success rate
DTW	99.2%	0.75%
DFD	98.8%	0.75%

Table 3: The average and the standard deviation of the percent of correctly classified cases for the synthetic control example [20]. Ten randomly selected 80/20 train/test sets were used to compute the statistics for each distance measure.

6.2 Classifier Test 2: Simulated Light Intensity Time Series

Using the tool-chain we designed and illustrated in Fig. 1, we created 250 data-sets with up to 1000 reflection points per data-set. This means that for 250 elliptical orbits, we generated the light reflection at 250 positions. Half of these were generated using the satellite model while the other half used a debris model. We randomly split the data into 80/20 training/testing sets and applied the 1-NN classifier using DFD. The process was repeated 10 times and each time the success rate for each distance measure was computed. No data could be generated with the 1-NN classifier using DTW because the algorithm exceeded the computational resources available. Once better computational resources are available, we will generate data using DTW.
No data could be generated with the 1-NN classifier using DTW because the algorithm exceeded the available computational resources.

It can be seen that the trained network cannot accurately predict whether the object is a satellite or debris. A possible reason for this might be that the distance values dominate the light

Distance metric	average success rate	standard deviation of success rate
DTW	not available	not available
DFD	51.2%	7.13%

Table 4: The average and the standard deviation of the percent of correctly classified cases for the simulated light intensity of the space objects. Ten randomly selected 80/20 train/test sets were used to compute the statistics for each distance measure.

reflection values to an extend that the differences due to different shapes are negligible.

Distance metric	average success rate	standard deviation of success rate
DFD	58.9%	10.8%

Table 5: The average and the standard deviation of the percent of correctly classified cases for the simulated light intensity with distance correction of the space objects. 10 randomly selected 80/20 train/test sets were used to compute the statistics for each distance measure.

Therefore, we created data that disregards the distance of the object. This would correspond to collected light intensity values that would then be multiplied by the square of the measured distance. In reality, this could be done by tracking position and light intensity of the object. We split 105 data-sets randomly into 80/20 training/testing sets and applied the 1-NN classifier using DFD. The result can be seen in Table 5. We can see that the success rate has significantly incresed even though the number of datasets was lower. The classifier can still not reliably predict if the object is a satellite or debris. However, using light reflection with a distance correction has improved results.

Other options to improve the detection algorithms are:

- Include different motion pattern for satellite and debris
- Use different Machine Learning algorithms
- Generate different input data (e.g. light intensity plots as in Fig. 6

It can be seen that the classifier cannot accurately predict whether the object is a satellite or debris. A possible reason for this might be that the distance values dominate the light reflection values to the extent that the differences due to different shapes are negligible. For future purposes, data should be created that disregards the distance of the object. This would correspond to collected light intensity values that would then be multiplied by the square of the measured distance. In reality, this could be done by tracking both the position and the light intensity of the object [5]. Other suggestions to improve the detection algorithms include:

- Incorporating different motion patterns for RSOs.
- Using different Machine Learning algorithms or features.
- Generating different input data (e.g. light intensity disregarding distance, light intensity plots similar to the one shown in Fig. 6).

7 Graphical User Interface Design (GUI)

This section describes the user interface that we designed to enable path prediction and visualization of the orbit. It also discusses the imperfections and unconsidered effects of the created orbit models.

The documentation of the orbit library is referenced in order to provide more in-depth information about the usage of the library.

The structure of the graphical user interface (GUI) was guided by the three-layer design shown in Fig. 11 [21]. The three elements in this design are (1) the calculation Layer which does the necessary background calculations and includes the necessary logic and numerical algorithms, (2) the communication Layer which sends information between the calculation and the front panel layer, and (3) the front Panel Layer which includes the interactive user interface also knows as the graphical user interface (GUI).

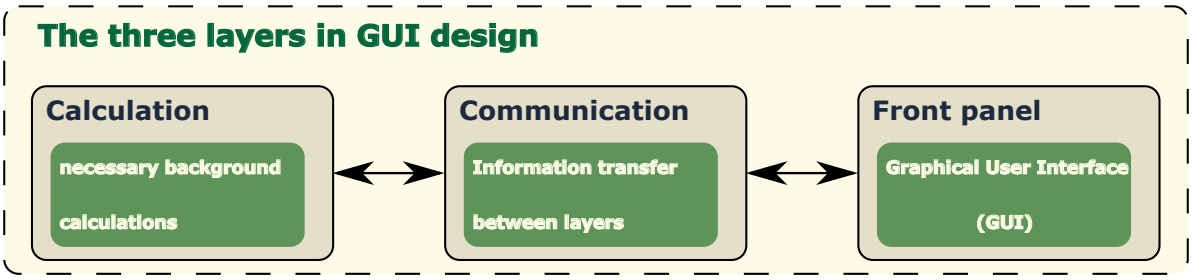

Figure 11: The three recommended design layers for a graphical user interface (GUI) [21].

7.1 Elements of the User Interface

The created orbit model can be accessed via two User Interfaces. One interface depends on manual data, the other reads a CSV file to gain the needed information. Both interfaces allow the three different input methods described in Section 2.2.

Problem Type	Environment	r1	r2	r3	v1	v2	v3	TOF	Method	p_0	z_0
1	Earth	0 0 1			1 0 0						
2	Normalized	1.414 0 1.414	1.811 1.0607 0.311	1.3535 1.414 -0.6464							
3	Normalized	1 0 0	1 1 1					1.0922	p_iteration	2	
3	Normalized	1 0 0	1 1 1					1.0922	universal_var		0
3	Normalized	1 0 0	1 1 1					1.0922	g_f		

Figure 12: A sample input Template for the Automated User Interface.

Advantages of the first user interface include its simplicity: the user only needs to respond the prompted questions for data entry. However, the manual input cannot be replicated and only one problem can be solved at a time. The created template for the automated user interface allows easily solving several problems in one run, and repeat those many times without the need for inputting new data every time. However, the template still needs to be appropriately filled out in order to function correctly.

7.2 Time Dependent Orbit and Visualization

An additional feature was added to the orbit determination library to allow the calculation of a satellite's position after any given time of flight. This also allows a large number of discrete position values to be concurrently plotted to display the orbit of a satellite. As shown in Fig. 13, a big blue

sphere and a small red sphere have been added to the plot to show the current location of earth and the satellite.

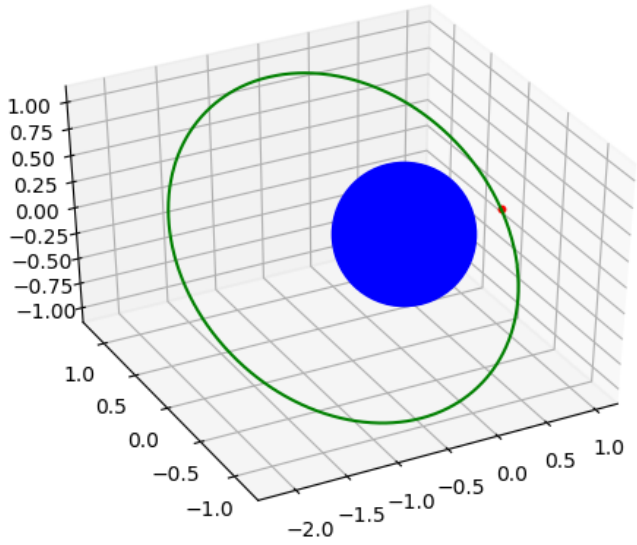

Figure 13: Orbit Visualization Example.

7.3 Graphical Representation

A Graphic User Interface was crated to enhance the ease of implementing the user data and displaying the results in more user-friendly fashion. The three main programs that were identified for this task included `PyQT`, `Tkinter`, and `WxPython`. After evaluating the different tools, we decided to use Tkinter. The advantage of `Tkinter` over `PyQt` is that it is native to Python and does not require additional commercial software. In contrast `PyQt` requires `QTDesigner`. In addition, `PyQt` uses additional commercial GUI creation software like `QTDesigner`. `WxPython`was eliminated because of `Tkinter` has better documentation and is easier to use. Figure 14 shows an example of a simple practice interface that I created using Tkinter.

7.4 GUI Update and Documentation

The first attempt to build a GUI mainly used commands and options to create pages. In the second attempt an entire application infrastructure was built to allow the fast creation of new pages and the easy navigation between them. Figure 15 shows the created main page which lets the user choose one of three input options. These options were described in Section 2. A third option exists allowing the user to upload a CSV file including satellite data like positions or velocity. These values are used as the input for orbit determination problems; the results are then stored in an additional CSV file. This is similar to the automated User Interface described in 7.1. In addition to displaying the three dimensional graph of the orbit, a simple animation was built to show the

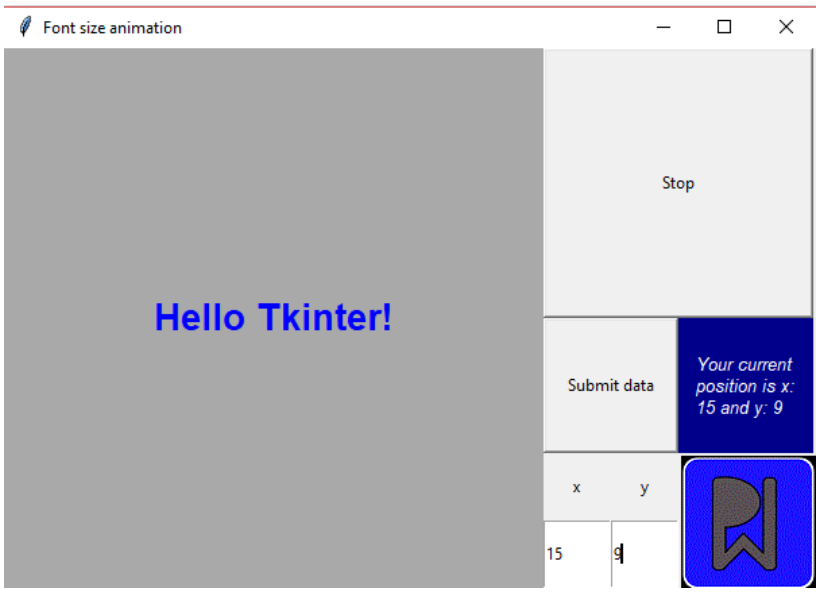

Figure 14: First Trial of GUI Creation

position of the satellite in time. This gives the user an indication of the relative speed of the satellite at various positions.

Currently, the input which uses three positions or one position and velocity is available. Alternatively, the input can also be an uploaded CSV file. The option to input two positions and the measured time of flight was postponed for future work.

8 Bulk Data Generation and Automation

For the final goal of using generated data for machine learning, a lot of data has to be generated. In order to do so, large parts of the data generation had to be automated to avoid large amounts of manual labor. The entire toolchain of data creation can be seen in Fig. 1. The process starts with creation of random satellite object. These observations include a velocity and a position vector of the object. For this purpose, 10,000 x, y, and z coordinates between 0 and 2 have been created in excel for both the position and the velocity. Furthermore, the object definition has been created in the same excel sheet. The satellite model is given a probability of 50%, while each of the five debris models is given a probability of 10%.

The position and velocity data is then used in the automation tab of the graphical user interface. We calculate the orbit elements for all 10,000 input data-sets, and select all inputs that result in elliptical orbits. For each of these, 1000 orbit points are generated and saved in a csv file. More than 700 sets of orbit points were generated in this way.

The orbit data in combination with the object classification is then loaded into a Mathematica script which calculates the light intensity at every point of every orbit for the specific object. Only 250 of the 700 orbit point sets have been used for this because the intensity calculation are very time consuming at 9 hours per 100 orbit point sets. The light intensity values, and object classification were then imported into the machine learning script,

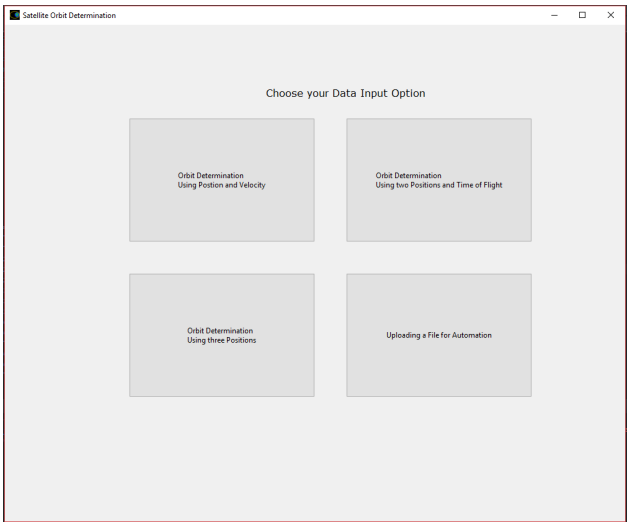

Figure 15: GUI Start Page

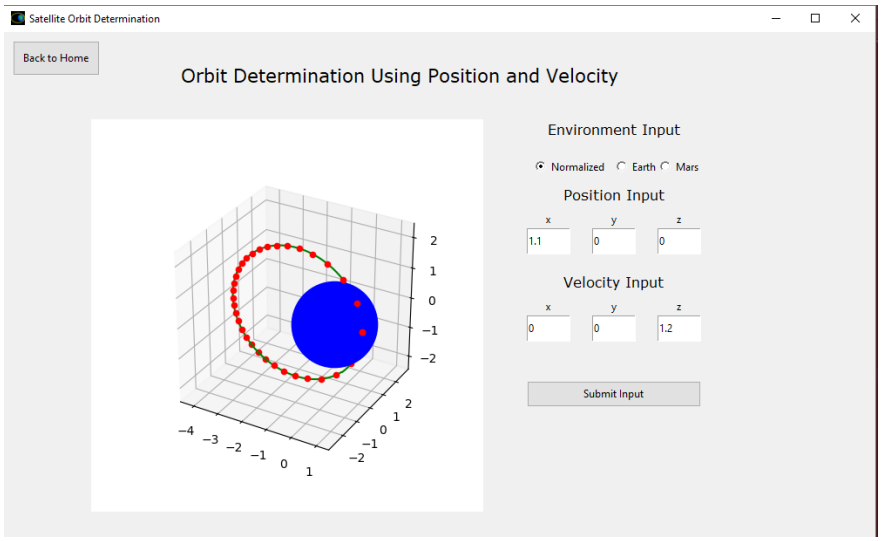

Figure 16: GUI Animation Example

9 Code Documentation

To document the code library, the tool “Sphinx” has been used. This automates the documentation process by turning the python script comments and annotations into an html file system.

Figure 18 shows three main boxes in the code documentation: The first box displays the list of contents currently consisting of the Orbit Determination library, the second box presents a search bar which allows users to quickly find methods or classes they want to acquire information about, while the third box shows the main content.

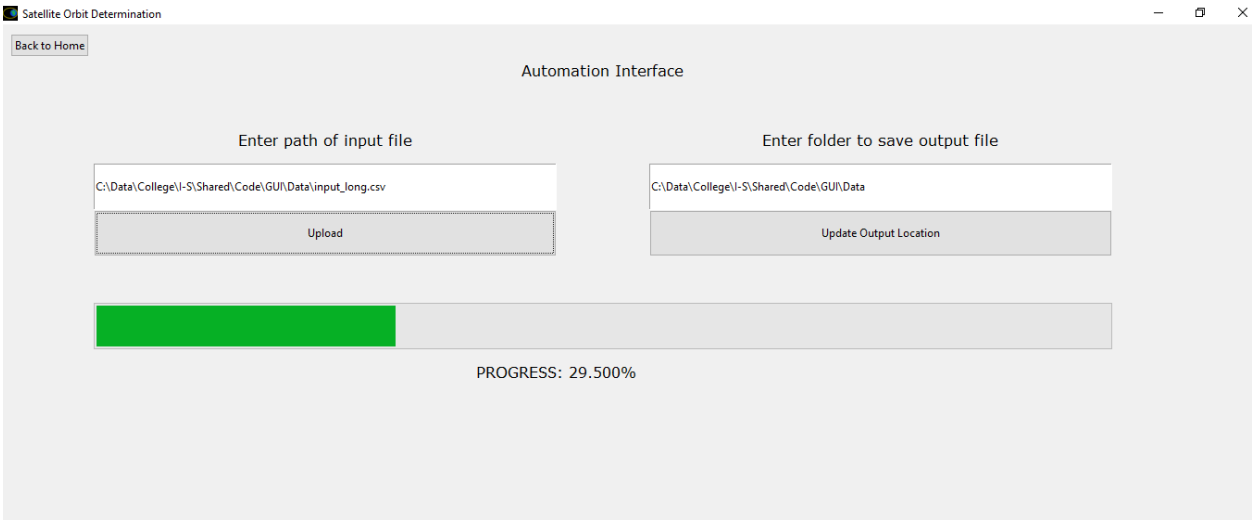

Figure 17: Automation Tab in Graphical User Interface

10 Future Work

We have successfully built a tool-chain that creates simulated satellite and debris light patterns. These can then be implemented into machine learning models. Although we were not able to train an accurate classifier for delineating satellites from space debris, we now have a pipeline that can be used to train better classifier in our future work. This can be achieved by improving the simulation model or the machine learning model. We expect an improvement in the classification if both the simulation and the machine learning are significantly improved.

10.1 Improvements to the Simulation

The current status of the simulation is based on several simplifying assumptions. These include:

- The rotation of the earth is neglected for the light reflection values
- The camera is always positioned perpendicular to earth's surface
- Perturbations in the orbit model have been neglected
- Both the satellite and debris do not rotate around themselves.

Taking these assumptions into consideration can improve the simulation model and lead to significant improvements in object classification. If different motion models for satellite and debris are implemented, this will give the machine learning algorithm more training data to distinguish between debris and satellite. Currently, the only difference between the two object is the shape.

In addition to increasing how realistic the simulation model is, another option is to change the output that the machine learning algorithm is trained on. It might be possible to create light reflection data that is multiplied by the square of the distance to disregard the significant decrease of light intensity with distance. Furthermore, a combination of position data and light reflection data, similar to the one shown in Fig. 6, could be used as the input for the machine learning algorithm.

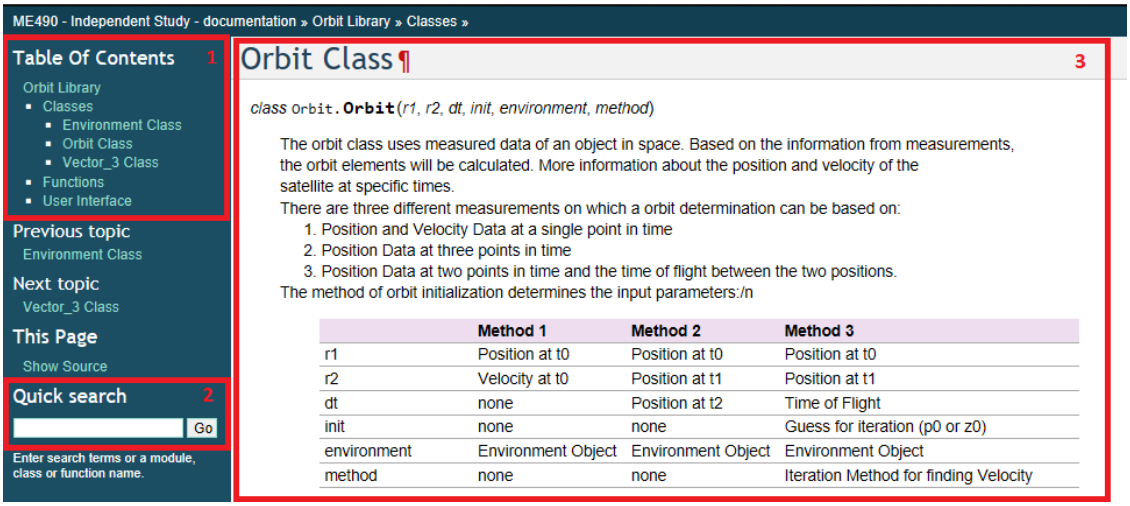

Figure 18: Documentation Excerpt

10.2 Improvements to the Machine Learning Models

Only a single machine learning algorithm has been used to train and test the created data. It is likely that more appropriate algorithms exist for our project. After improving our understanding of machine learning and all of its different methods, we believe that it will be possible to choose better machine learning algorithms for our data.

Appendices

A Useful tools

A.1 File Sharing

To ensure easy communication with Professor Khasawneh, I set up a file sharing system using Bitbucket as a repository host, and Sourcetree as the user interface. This helped monitor the project's status and made sure that it remained on-time.

A.2 Usage of VIM

Learning to use the text editor VIM was a side project to increase the typesetting and programming efficiency. VIM is a text editor that allows the modification of text files within a Powershell and it has an optimized text file control using the keyboard. This allows for faster coding as well as faster modification in text files. I needed a tutorial and some practice to get used to the editor.

A.3 Usage of Python

Python was used for the majority of all programming. Main components used include matplotlib to create graphs and visualization, tkinter to build the GUI, and sklearn for machine learning. The majority of the created data was stored in CSV files.

A.4 Usage of Mathematica

Mathermatica's three dimensional lighting tools were very useful when calculating light intensities for debris and satellites. Mathematica imported the Python-created CSV files of the orbit trajctory as the input for its calculations, and created other CSV files to export the results of the light intensity computations.

A.5 Usage of LaTeX

The text editor LaTeXwas used to create all of the bi-weekly reports and the final report. Its implemented library functionality makes it easy to work with large files with many sections and figures.

A.6 Usage of Powerpoint and Inkscape

For the creation of figures and illustrations powerpint and Inkscape have been used.

A.7 Usage of Sphinx

For the creation of figures and illustrations powerpint and Inkscape have been used. Sphinx has been used for Code Documentation in Python. A html document has been created that can help understand and use the Orbit libraries.

References

[1] R. R. Bate, *Fundamentals of Astrodynamics*. Dover Publications, 1971.

[2] W. T. Thomson, *Introduction to Space Dynamics*. Dover Publication, 1986.

[3] A. P. Cox, C. K. Nebelecky, R. Rudnicki, W. A. Tagliaferri, J. L. Crassidis, and B. Smith, "The space object ontology," in *2016 19th International Conference on Information Fusion (FUSION)*, pp. 146–153, July 2016.

[4] P. D. McCall, *Modeling, simulation, and characterization of space debris in low-earth orbit*. PhD thesis, Florida International University, 2013.

[5] B. Jia, K. D. Pham, E. Blasch, D. Shen, Z. Wang, and G. Chen, "Space object classification using fused features of time series data," in *Advanced Maui Optical and Space Surveillance Technologies Conference (AMOS)*, 2017.

[6] T. Eiter and H. Mannila, "Computing discrete frchet distance," tech. rep., 1994.

[7] D. J. Berndt and J. Clifford, "Using dynamic time warping to find patterns in time series," in *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, AAAIWS'94, pp. 359–370, AAAI Press, 1994.

[8] M. Kerker and E. M. Loebl, *The scattering of light and other electromagnetic radiation*, vol. 16 of *Physical Chemistry: A Series of Monographs*. Academic Academic Press Inc., 1969.

[9] B. J. Sumlin, W. R. Heinson, and R. K. Chakrabarty, "Retrieving the aerosol complex refractive index using pymiescatt: A mie computational package with visualization capabilities," *Journal of Quantitative Spectroscopy and Radiative Transfer*, vol. 205, pp. 127 – 134, 2018.

[10] C. F. Bohren and D. R. Huffman, *Absorption and Scattering of Light by Small Particles*. Wiley, 1998.

[11] M. Mishchenko, J. W. Hovenier, and L. D. E. Travis, *Light Scattering by Nonspherical Particles: Theory, Measurements, and Applications*. Academic Academic Press Inc., 2000.

[12] J. Newman, *Geometrical Optics*, pp. 1–20. New York, NY: Springer New York, 2008.

[13] T. Pradhan, "Maxwell's equations from geometrical optics," *Physics Letters A*, vol. 122, no. 8, pp. 397 – 398, 1987.

[14] "3d projection," Mar. 2018.

[15] T. Rakthanmanon, B. Campana, A. Mueen, G. Batista, B. Westover, Q. Zhu, J. Zakaria, and E. Keogh, "Addressing big data time series: Mining trillions of time series subsequences under dynamic time warping," *ACM Trans. Knowl. Discov. Data*, vol. 7, pp. 10:1–10:31, Sept. 2013.

[16] P. C. Besse, B. Guillouet, J. M. Loubes, and F. Royer, "Review and perspective for distance-based clustering of vehicle trajectories," *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, pp. 3306–3317, Nov 2016.

[17] M. Vlachos, G. Kollios, and D. Gunopulos, "Discovering similar multidimensional trajectories," in *Proceedings 18th International Conference on Data Engineering*, pp. 673–684, 2002.

[18] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, (New York, NY, USA), pp. 491–502, ACM, 2005.

[19] L. Chen and R. Ng, "On the marriage of lp-norms and edit distance," in *Proceedings of the Thirtieth International Conference on Very Large Data Bases - Volume 30*, VLDB '04, pp. 792–803, VLDB Endowment, 2004.

[20] Y. Chen, E. Keogh, B. Hu, N. Begum, A. Bagnall, A. Mueen, and G. Batista. The UCR Time Series Classification Archive, 2015.

[21] M. Feathers, "The humble dialog box," tech. rep., 2002.