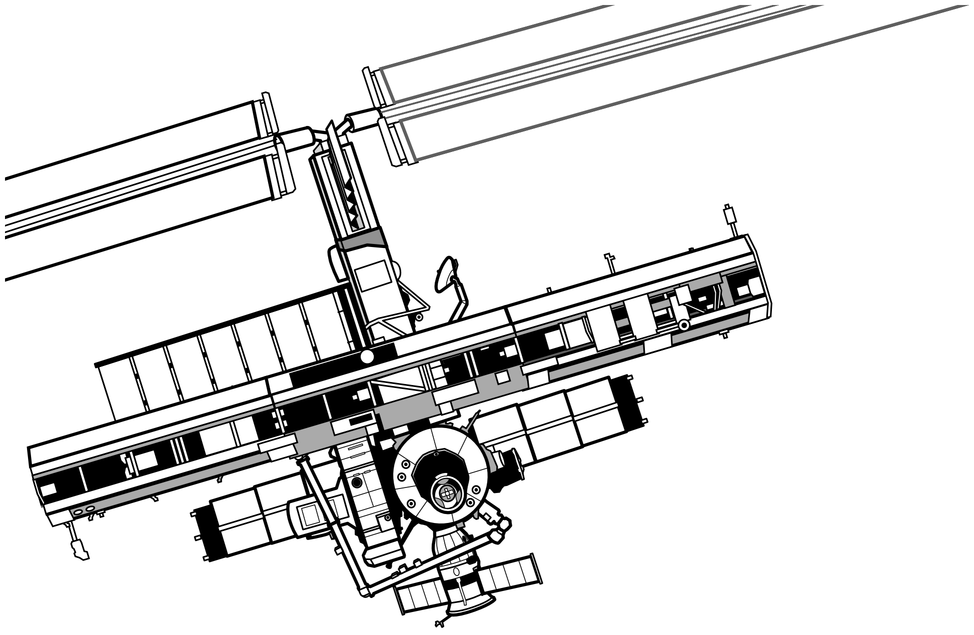
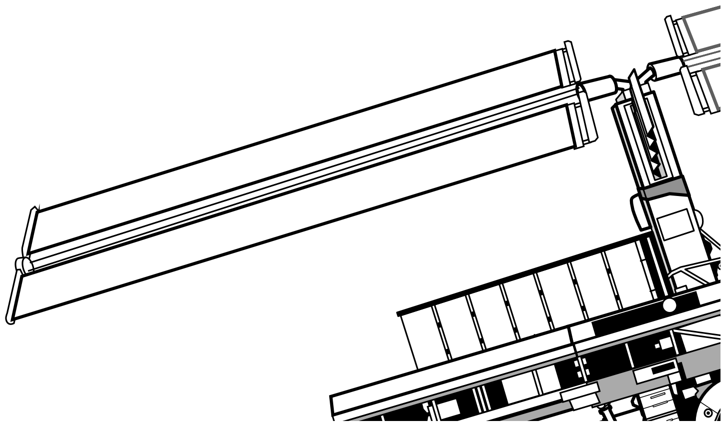


TRACKING THE ISS

Aubrey L. Baker

COE332: Software Engineering and Design



High-Level Project Description

Due to the sparse atmosphere at the altitude of the International Space Station (ISS), there is resistance that can gradually cause errors in its expected trajectory. To address this, orbital mechanists update the projected trajectory to obtain the most accurate estimation possible. Accurate trajectory calculations are crucial for establishing communication links, scheduling rendezvous with visiting spacecraft, and ensuring the ISS's path remains unobstructed by potential collisions. The data that the orbital mechanists collect is public information and can be accessed online from NASA's ISS dataset in XML or TXT format. This application retrieves the data in XML format and returns it in a useful and readable way for the user. This web app was built using Flask and Docker. Both of these tools allow for easy accessibility and repeatability for any user at any time.

Data

The data source for this web application is NASA's Orbit Ephemeris Message data set. The raw data can be found at this link. The website offers the same data set that can be downloaded in XML or TXT format. These files contain, but are not limited to, the orbit ephemeris message, state vectors, and a list of future predictions for the ISS's trajectory. This data set can be used for scientific research, educational purposes, or simply for tracking the ISS's movements in real-time.

The web application makes a request to the URL that contains the XML file with the data. Once the data is loaded into the web application, the XML file is then parsed as a list of dictionaries for accessibility. The dictionaries contain a lot of useful information that can be accessed through the API route endpoints, but the state vectors are used most often. The state vectors are displayed in many of the route endpoints as well as used to calculate the speed, position, longitude, and latitude of the ISS.

Flask

Flask is known for its simplicity, flexibility, and ease of use. It is used for building web applications, particularly for creating web APIs. Flask provides a number of features that make it a popular choice for web development.

One of the key benefits of Flask is its lightweight nature. It does not require any particular tools or libraries to be installed, making it easy to set up and use. Flask is also highly customizable, allowing developers like me, to add or remove features as needed.

Another advantage of Flask is its extensive documentation and large community of developers. Since this is my first web application, the abundance of documentation allowed me to easily debug any errors that I ran into and quickly enhance my knowledge in using the Flask framework.

Docker

Docker is a containerization platform that allows developers to package and deploy their applications in a self-contained environment, known as a container. A container is a standalone executable package that contains everything an application needs to run, including code, libraries, dependencies, and more.

Even though this project does not have any dependencies nor is very complex, I was able to solidify my Docker knowledge. Implementing Docker in this project allowed me to understand the scalability and potential of containerization for more advanced projects.

Route Endpoints

There are a total of 12 decorators in iss_tracker.py. The decorators and their description are listed below:

Route	What it Returns
`/`	The entire data set
`/epochs?limit=int&offset=int`	Return modified list of Epochs given query parameters
`/epochs/<epoch>`	State vectors for a specific Epoch from the data set
`/epochs/<epoch>/speed`	Instantaneous speed for a specific Epoch in the data set
`/epochs/<epoch>/location`	Return latitude, longitude, altitude, and geoposition for given Epoch
`/now`	Return latitude, longitude, altidue, and geoposition for Epoch that is nearest in time
`/help`	Return help text (as a string) that briefly describes each route
`/delete-data`	Delete all data from the dictionary object
`/post-data`	Reload the dictionary object with data from the web
`/comment`	Return 'comment' list object from ISS data
`/header`	Return 'header' dict object from ISS data
`/metadata`	Return 'metadata' dict object from ISS data

Each of these decorators returns different information or edits the data. The user must append the proper decorator to return what they are looking for.

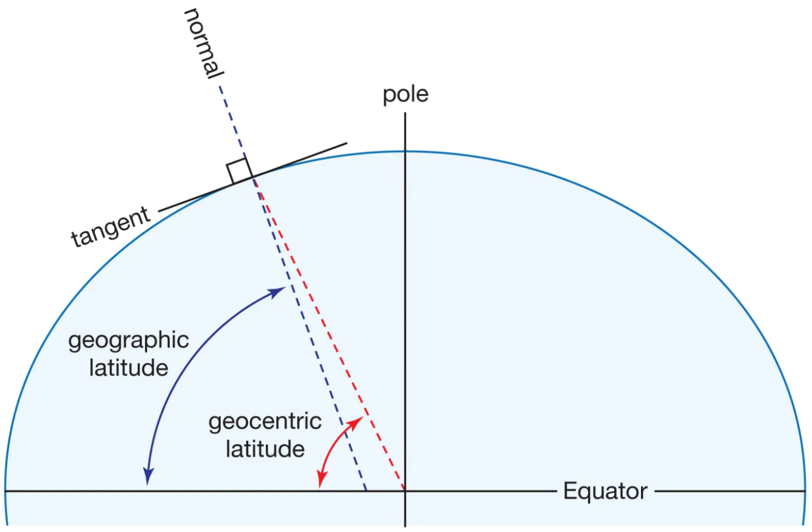
Variable Calculations

The math for the latitude, longitude, and altitude calculations are a bit involved. The following are explanations for each equation that was derived for each variable:

Latitude, φ: The angular distance of a place north or south of the earth's equator expressed in degrees.

$$\varphi = \tan^{-1}\left(\frac{z}{\sqrt{x^2 + y^2}}\right)$$

To get the distance of the x-y vector, it must be normalized. The acrtan of z to the normalized x-y vector identifies the angle between the z position and the normalized x-y vector. Normalizing the x-y vector will result in a vector with the same direction as the original vector but a magnitude of 1 (no longer in three dimensions but rather two like shown in the image below). This allows us to find the angle between the z-position and the x and y positions.

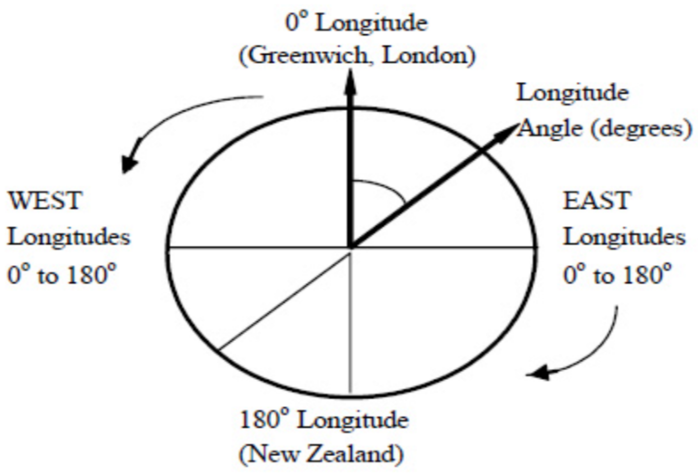


Latitude diagram from Britannica.com

Longitude, λ: The angular distance of a place east or west of the meridian at Greenwich, England.

$$\lambda = \tan^{-1}\left(\frac{y}{x}\right) - \left[(hrs - 12) + \left(\frac{mins}{60}\right)\right] \cdot \frac{360}{24} + 32$$

Arctan is used to get the angular distance between the y and x coordinates. Then, you have to subtract the angle of time in degrees that is representative of that specific epoch to transform the angle to start at the meridian at Greenwich, England. Lastly, the addition of $32°$ is an adjustment made to account for the irregular rotation of the Earth on its axis. The rotation of the Earth is not perfectly regular, and its speed can vary due to various factors. As a result, the difference can accumulate over time, leading to a discrepancy between the calculated value for longitude with and without the correction. This correction is important because it ensures that the longitude calculated remains closely aligned with Earth’s rotation.



Longitude diagram from energy-models.com/earth-and-sun

Altitude: The height of an object or point in relation to sea level.

$$alt = \sqrt{x^2 + y^2 + z^2} - R_{earth}$$

Finding the magnitude of the 3D vector determines the total length of it. This tells us how far away the ISS is from the center of the earth at a given epoch. To find the altitude, you must subtract the radius of the earth.

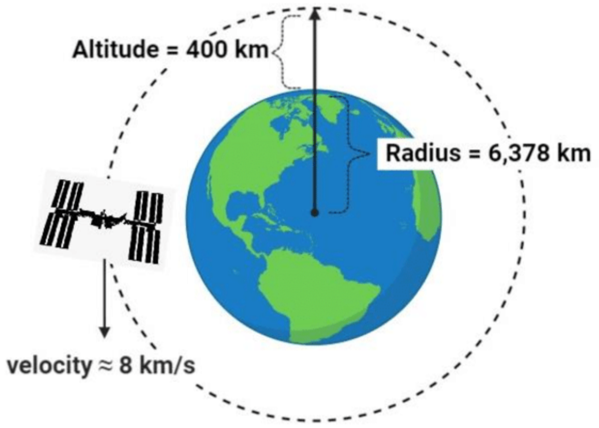


Diagram of altitude with respect to Earth's radius from Karami

File Contents and Usage

The contents of this project includes `iss_tracker.py`, `Dockerfile` , and `docker-compose.yml`. An explanation of each file and how to use it is detailed below:

`iss_tracker.py` - This file contains the web application. There are a total of 12 decorators that return and execute different commands. Each decorator has its own function and is tied to a URL. Each URL is described in the Route Endpoints section.

`Dockerfile` - This file utilizes Docker to allow the user to execute the web application with the same exact environment as I have on my computer without requiring them to install any additional packages that are not already included within Python.

`docker-compose.yml` - Docker Compose is a powerful tool that simplifies the management of multi-container applications. It enables you to define and document all the options you need to start the containers in a single YAML file. This `docker-compose.yml` defines all the application containers and allows for the use of a few straightforward commands to start or stop all of the services at once. If this file was not included, the commands needed to start up the Flask application could become very long and cumbersome.

Using the Files

Run *My* Docker Image

When the user clones my repository for this project, they will have access to the `Dockerfile` and `docker-compose.yml`. The user can pull `Dockerfile` using a simple command as shown: `docker pull alb6443/iss_tracker:midterm1`. Once the docker image is pulled from Docker Hub, the user can then run the `docker run -p 5000:5000 -rm -it alb6443/iss_tracker:midterm1` command to launch the docker image.

Build *Your Own* Docker Image

If the user wants to build their own image, they can build one using my `Dockerfile`. The user must call the `docker build -t <dockerhubusername>/<code>:<version> .` command. Another way the user can build their own Docker image is with `docker-compose --build`. This command will build a new docker image using the context / `Dockerfile` / image name listed in the `docker-compose.yml`.

Launching a Container

When the user is ready to launch the container, there are multiple ways to go about it. One way the user can launch the container is by using the `docker run -p 5000:5000 --rm -it <dockerhubusername>/<code>:<version` command. When that command gets cumbersome and complex, it is much simpler to call `docker-compose up`. This command utilizes the `docker-compose.yml` file that the user has access to within my GitHub. If the user makes changes to the `Dockerfile`, it would be best to run `docker-compose up --build` to force a Docker image rebuild and put the service up.

Using the API

To use the API, the user must have the container up and running by using one of the commands previously mentioned. The user must enter a valid route with the `curl` command. The user must type `curl '<some_base_url>:<some_port>/<some_url_path>`. For example when I execute `curl 'localhost:5000/'` , the following block of code is the output:

```
… {
    "EPOCH": "2023-080T15:02:07.856Z",
    "X": {
      "@units": "km",
      "#text": "5621.1084877163303"
    },
    "Y": {
      "@units": "km",
      "#text": "3544.7448574938098"
    },
    "Z": {
      "@units": "km",
      "#text": "-1444.4760791656199"
    },
    "X_DOT": {
      "@units": "km/s",
      "#text": "-1.5310068154662599"
    },
    "Y_DOT": {
      "@units": "km/s",
      "#text": "4.7704102331297102"
    },
    "Z_DOT": {
      "@units": "km/s",
      "#text": "5.7899431758210804"
    } …
}
```

Ethical and Professional Responsibilities

It is important to acknowledge and give credit to the source that the data was from in this web application. The professionals responsible for collecting and analyzing complex data sets such as this play a crucial role in the success of the ISS. Giving credit to these professionals not only recognizes their contributions but also promotes collaboration and transparency in the space industry. It is a fundamental principle of professional and ethical conduct to acknowledge and give credit to the sources of information and expertise that we rely on.

Moreover, citing the data source allows for individuals to traceback information to the origin. This is important because it ensures transparency and accountability in the collection and analysis of data and promotes trust and credibility in the information used for the ISS orbit. By citing the source, users can verify the accuracy and reliability of the data and understand the methods used to derive the trajectory predictions. This not only helps to prevent errors or misunderstandings but also enables users to make informed decisions about the use and interpretation of the data.

Works Cited

Britannica, The Editors of Encyclopaedia. "latitude and longitude". Encyclopedia Britannica, 31 Aug. 2022, https://www.britannica.com/science/latitude. Accessed 8 March 2023.

Karami, Danial. (2021). Generalizing the Physical Time Impact on the Astronauts Living on the International Space Station to the Theory of Relativity. 10.20944/preprints202104.0123.v1.

“Models.com.” Energy Models, https://energy-models.com/earth-and-sun.

“Spot the Station.” NASA, NASA, https://spotthestation.nasa.gov/trajectory_data.cfm.