Take out the Trash

Deck 36, Main Engineering

A cloud of space junk is in your constellation's orbital plane. Use the space lasers on your satellites to vaporize it! Destroy at least 51 pieces of space junk to get the flag.

The lasers have a range of 100 km and must be provided range and attitude to lock onto the space junk. Don't allow any space junk to approach closer than 10 km.

Command format:

[Time_UTC] [Sat_ID] FIRE [Qx] [Qy] [Qz] [Qw] [Range_km]

Command example:

2021177.014500 SAT1 FIRE -0.7993071278793108 0.2569145028089314 0.0 0.5432338847750264 47.85760531563315

This command fires the laser from Sat1 on June 26, 2021 (day 177 of the year) at 01:45:00 UTC and expects the target to be approximately 48 km away. The direction would be a [0,0,1] vector in the J2000 frame rotated by the provided quaternion [-0.7993071278793108 0.2569145028089314 0.0 0.5432338847750264] in the form [Qx Qy Qz Qw].

One successful laser command is provided for you (note: there are many possible combinations of time, attitude, range, and spacecraft to destroy the same piece of space junk):

2021177.002200 SAT1 FIRE -0.6254112512084177 -0.10281341941423379 0.0 0.773492189779751 84.9530354564239

General Information

We were given 2 sets of TLE, one describing our fleet of satellites, the other a pile of debris heading our way. As the description stated our satellites are equipped with some magic LAZOR-cannon powerful enough to vaporize any incoming debris. The mission was simple: hit'em and don't get hit.

Solution

I've started by just loading the TLEs into skyfield, which is internally using sgp4 for propagating the orbits. Since our weapons had a range of 100km, we just stepped numerically from the start time (June 26, 2021 00:00:00 UTC) in 15s intervals a few days into the future, testing each time if any undestroyed debris got near one of our spacecrafts.

For each encounter, targeting data was then calculated by subtracting the position of the spacecraft from the debris, leaving us with a distance vector \vec{v} in our reference coordinate system.

 \vec{v} was now decomposed into its length $||\vec{v}||$ (the target range), as well as a normalized direction vector

$$\vec{v_d} = \frac{\vec{v}}{||\vec{v}||}$$

describing the direction in which to point. The Quaternion was now found by recyling our code of the quaternionchallenge matching the [0,0,1] - vector onto $\vec{v_d}$. After every encounter we just directly assumed the affected debris to be destroyed, limiting the amount of future encounters.

Generating a sufficient amount of commands based on this crude algorithm, yielded 52 encounters quite quickly. However the server responded in an unwanted, although honestly expected way. Most of our commands were invalid, since the lazers had cooldown. By looking at the closest 2 Commands that were executed we determined he shortest time between shots to be a minute. Thus a few improvements had to be made. 1. We've implementened a "cooldown" in calculating the encounters, by just filtering out all satellites that shot within the last minute. (It would've probably been sufficient to step in 1 min intervals and just limiting each satellite to one shot per timestep) 2. As a bit of extra caution, the satellites were trimmed to have a bit more self-presevation engrained by always targeting the closest target, if multiple were available.

This "improved" algorithm generated a list of commands looking like this:

2021177.002200 SAT1 FIRE -0.6418303193313932 -0.02640936170027976 0.0 0.7663917971909313 79.94429440423356 2021177.002300 SAT1 FIRE -0.5859898454097088 -0.1129372532788008 0.0 0.8024095449946652 69.69552282591509 2021177.002400 SAT1 FIRE -0.5568843306787012 -0.3446936681041382 0.0 0.7556891671933964 59.40604998754104

```
2021177.002500 SAT1 FIRE -0.5929175931636828 -0.0014828807047325661 0.0 0.8052617765557988 49.210536263655 2021177.002600 SAT1 FIRE -0.5112044273176659 -0.31738946719542466 0.0 0.7987076809473049 39.10452676621887 2021177.002700 SAT1 FIRE -0.2223377142518798 -0.4516107396680691 0.0 0.8640680995371307 29.10587624403453 ...
```

This was then just sent to the service, yielding us the flag.

Script

```
from skyfield.api import EarthSatellite, load
from skyfield.timelib import Time
from skyfield.framelib import ecliptic_J2000_frame
from astropy import units as u
from pyquaternion import Quaternion
import numpy as np
from IPython import embed
from scipy.spatial.transform import Rotation
ts = load.timescale()
def import_tles(filename):
    sats = []
    with open(filename) as f:
        while True:
            name = f.readline()
            if name == "":
                break
            11 = f.readline()
            12 = f.readline()
            sat = EarthSatellite(l1.strip(),l2.strip(),name.strip(), ts)
            # print(sat)
            sats.append(sat)
    return sats
sats = import_tles("sats.tle")
debris = import_tles("spacejunk.tle")
vaporized = [None for _ in debris]
last_lazor_shot = [0 for _ in debris]
def propagate(t):
    sats_t = [sat.at(t) for sat in sats ]
    debris_t = [deb.at(t) for deb in debris ]
    return sats_t, debris_t
def find_intersections(t, thresh):
    sats_t, debris_t = propagate(t)
    for s_i, s in enumerate(sats_t):
        dist_list = []
        idx_list = []
        for d_i, d in enumerate(debris_t):
            if vaporized[d_i] is not None:
            dist_list.append(np.linalg.norm(s.position.km - d.position.km))
```

```
idx_list.append(d_i)
        d_i = idx_list[np.argmin(dist_list)]
        test_encounter(t, s, s_i, debris_t[d_i], d_i, thresh)
def test_encounter(t, s, s_i, d, d_i, thresh):
    if np.linalg.norm(s.position.km - d.position.km) < thresh:</pre>
        if time.utc_datetime().timestamp() - last_lazor_shot[s_i] < 60:</pre>
            return
        vaporized[d_i] = (t, s_i, d_i)
        last_lazor_shot[s_i] = t.utc_datetime().timestamp()
        target_vector = d.position.km - s.position.km
        dist_km = np.linalg.norm(target_vector)
        v1 = np.array([0,0,1])
        target_dir = target_vector / dist_km
        # v1, target_dir = target_dir, v1
        R = np.cross(v1, target_dir)
        q1 = R[0]
        q2 = R[1]
        q3 = R[2]
        q4 = np.dot(v1, target_dir) + np.sqrt(np.linalg.norm(v1) ** 2 * np.linalg.norm(target_dir) ** 2)
        q = Quaternion(q1, q2, q3, q4).normalised
        day_frac = 177 - 26 \setminus
            + time.utc.day \
            + 1000 * time.utc.year \
            + time.utc.hour / 100 \
            + time.utc.minute / 10000 \
            + time.utc.second / 1000000
        print(f"{day_frac:.6f} SAT{s_i + 1} FIRE {q[0]} {q[1]} {q[2]} {q[3]} {dist_km}")
        if np.linalg.norm(s.position.km - d.position.km) < 30:</pre>
            print("DANGER:", t.utc, sats[s_i].name, debris[d_i].name)
# for thresh in [50, 60, 80, 95]:
for day in range(5):
    for hour in range(24):
        for m in range(0, 60):
            for s in [0, 30]:
                time = ts.utc(2021, 6, 26+day, hour, m, s)
                find_intersections(time, 100)
\# time = ts.utc(2021, 6, 26, 0, 22, 0)
\# sats t, debris t = propagate(time)
# test_encounter(time, sats_t[0], 0, debris_t[0], 0)
# print(sats)
# print(debris)
```