Week 4 Report

Dhanush Balusa

June 11, 2025

Research

Error in TLE & SGP4

- **Resource 1:** `https://conference.sdo.esoc.esa.int/proceedings/sdc6/paper/41/SDC6-paper41.pdf`
- TLEs are not suitable for precise orbit estimation because of innacuracies in modeling, timeliness, and inconsistency of the available orbital information.
- TLEs only accurate to about 2-3 days after the epoch. A week later, the error grows significantly.

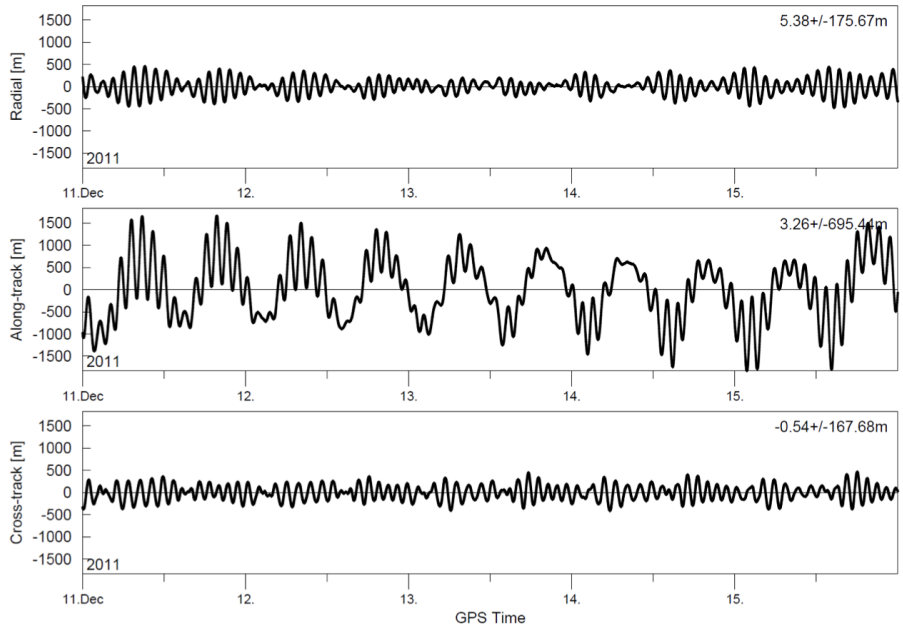

Figure 1 Orbit Error of the Fitted TLE during OD Period (5d Fit)

Figure 1: Different types of error in SGP4 over a period of 5 days.

- **Resource 2:** `https://www.researchgate.net/profile/Eberhard-Gill-2/publication/224781886_Real-Time_Estimation_of_SGP4_Orbital_Elements_from_GPS_Navigation_`

Data/links/54ddb8d80cf28a3d93fa2e31/Real-Time-Estimation-of-SGP4-Orbital-Elements-f
pdf

- Using SGP4, the initial position and speed will have a small error. These small errors at the start (called "at epoch") usually stay small, unless they affect the semi-major axis (SMA).
- If the SMA is off by a small amount (10 meters), it slowly causes the satellite to get ahead or fall behind in its orbit (called along-track error). This error grows about 1 to 1.5 km per day.
- Good news? Using the filter (called RTSGP4), the SMA gets corrected quickly - within 1 day, the error becomes less than 10 meters.

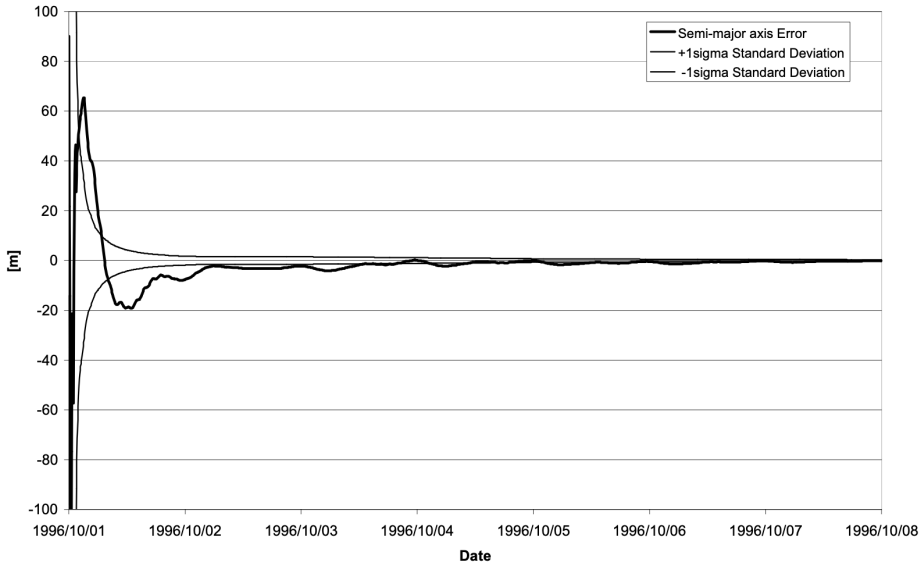

Fig. 2: GPS/MET: Error of the mean semi-major as obtained from the RTSGP4 filter with respect to the result of a 7 days batch orbit determination

- Summary: SGP4, when coupled with real-time mean-element estimation and drag-coefficient updating, can deliver sub-kilometre prediction accuracy over 12–24 h—good enough to feed the ISS's higher-precision conjunction assessment pipeline. Left unrefined, its along-track drift grows too large for reliable collision modeling.

Other resources:

- https://ntrs.nasa.gov/api/citations/20160005313/downloads/20160005313.pdf

Data Analysis from last week's collision event:

The key takeaways: The regular sinusoid makes sense because satellites in LEO orbit Earth every 90 minutes, so the position appears to oscillate rapidly within 2 days. That would be around 32 orbits so the graph would have 32 periods, which it does. I counted

an there were 29 periods in the graph so that is close enough. The sinusoid within the sinusoid is likely due to the error when using SGP4 to predict the position of the satellite. Although I didn't properly understand the error analysis for SPG4, it does seem like the error is periodic.

Previous Code

My code from the previous week calculated the approach conditions between the collision of Iridium 33 and Cosmos 2251.

```python
# Fetch and create satellites for collision analysis
satellites2 = []
for norad_id, name in satellites_info2:
    print(f"Fetching TLE for {name}...")
    sat = fetch_and_create_satellite(norad_id, name, username, password)
    satellites2.append(sat)

# Load timescale for collision analysis
ts2 = load.timescale()
start_time2 = ts2.utc(2009, 2, 10, 12, 0, 0)
hours2 = np.arange(0, 48, 1/60)
time_steps2 = ts2.utc(start_time2.utc.year, start_time2.utc.month, start_time2.utc.day, hours2)

# Store results for collision analysis
times2 = []
relative_positions2 = {name: [] for _, name in satellites_info2[1:]}
approach_speeds2 = {name: [] for _, name in satellites_info2[1:]}
for t in time_steps2:
    states = [sat.at(t) for sat in satellites2]
    iss_state = states[0]
    reference_state = states[1]  # Iridium 33 set as reference
    for i, state in enumerate(states[1:], start=1):
        rel_pos = reference_state.position.km - state.position.km
        rel_vel = reference_state.velocity.km_per_s - state.velocity.km_per_s
        approach_speed = np.linalg.norm(rel_vel)
        relative_positions2[satellites_info2[i][1]].append(np.linalg.norm(rel_pos))
        approach_speeds2[satellites_info2[i][1]].append(approach_speed)
    times2.append(t.utc_iso())

# Find time of closest approach between Iridium 33 and COSMOS 2251
min_index2 = np.argmin(relative_positions2["COSMOS 2251"])
min_distance2 = relative_positions2["COSMOS 2251"][min_index2]
min_speed2 = approach_speeds2["COSMOS 2251"][min_index2]
min_time2 = times2[min_index2]

# Print results for collision analysis
print("\n  Near Miss Detected Between Iridium 33 and COSMOS 2251:")
print(f"    - Closest Distance:      {min_distance2:.3f} km")
print(f"    - Approach Speed:        {min_speed2:.3f} km/s")
print(f"    - Time (UTC):            {min_time2}")
```

Figure 2: Code from week 4 calculating the approach conditions between the collision of Iridium 33 and Cosmos 2251

Code Changes

Based on the feedback from last week's team meeting, I decided to make the following changes:

1. **My first change in the code was to check when I am pulling the data.** Here is the snippet of the code that I added to check the epoch of the satellites's TLEs I am pulling:

   ```
   for sat in satellites2:
     print(f"{sat.name} Epoch: {sat.epoch.utc_iso()}")
   ```

   Here is the output of the code:

   ```
   ISS (ZARYA), Epoch: 2009-02-10T21:26:33Z
   IRIDIUM 33, Epoch: 2009-02-10T18:16:11Z
   COSMOS 2251, Epoch: 2009-02-10T18:09:29Z
   ```

   Observation: The epoch of the TLEs I am pulling are from February 10, 2009 but they are at past 18 UTC. I wonder how my code was accurately able to calculate the collision was exactly at 16:56 UTC on February 10, 2009. It's good to know that I am able to pull historical TLEs from CelesTrak, but I wonder how I'm still able to calculate the collision time accurately. I'm not sure if I can find a TLE closser to the collision time than this so 2 hours of a difference is the best I can do, and plus the data looks accurate.

2. **My second change in the code was to add the calculation of the approach angle.**

   ```
   # Get satellite states at closest approach
   closest_time = time_steps2[min_index2]
   state_iridium = satellites2[1].at(closest_time)
   state_cosmos = satellites2[2].at(closest_time)

   rel_pos_vector = state_iridium.position.km - state_cosmos.position.km
   rel_vel_vector = state_iridium.velocity.km_per_s - state_cosmos.velocity.km_per_s
   cos_theta = np.dot(rel_pos_vector, rel_vel_vector) / (np.linalg.norm(rel_pos_vector) * np.linalg.norm(rel_vel_vector))
   approach_angle_deg = np.degrees(np.arccos(cos_theta))

   ```

   Figure 3: Approach angle logic for Iridium 33 collision with Cosmos 2251.

The code calculates the **approach angle** between two satellites by measuring the angle between their *relative position vector* and the *velocity vector* with respect to the first satellite (Iridium 33 in this case).

This is computed using the dot product formula:

$$\theta = \arccos\left(\frac{\vec{r} \cdot \vec{v}}{|\vec{r}||\vec{v}|}\right) \tag{1}$$

where:

- $\vec{r}$ is the relative position vector between the two satellites (from Cosmos 2251 to Iridium 33),
- $\vec{v}$ is the relative velocity vector (how fast and in what direction Iridium 33 is moving relative to Cosmos 2251),
- θ is the angle between the two vectors.

The result is initially in radians and is then converted to degrees for easier interpretation.

This approach angle provides insight into the geometry of the encounter:

- An angle near 0° indicates a *head-on approach*.
- An angle near 90° suggests a *tangential or glancing pass*.
- An angle near 180° implies the satellites are moving in *opposite directions*.

In this case, the code outputs an approach angle of **20.592°** for the Iridium 33 and Cosmos 2251 event. I have to understand how to interpret the angle correctly, by the looks of it from visual plots, the two satellites didn't really have a head-on collision, I would think the output would be closer to +90°.

I tried to use GMAT to visualize the collision but I was unable to get GMAT to work on my Mac and the school's virtual machine was not working either. I will try to get it working next week. For now, I have visual confirmation from NASA's resource about the collision.

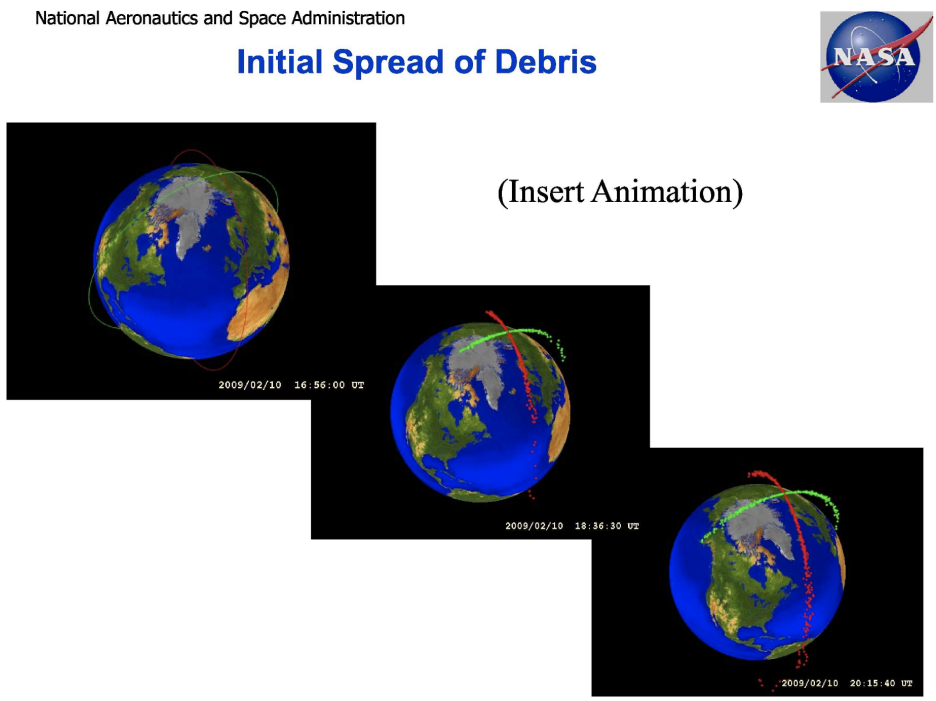

Figure 4: Visualization of the Iridium 33 collision with Cosmos 2251.

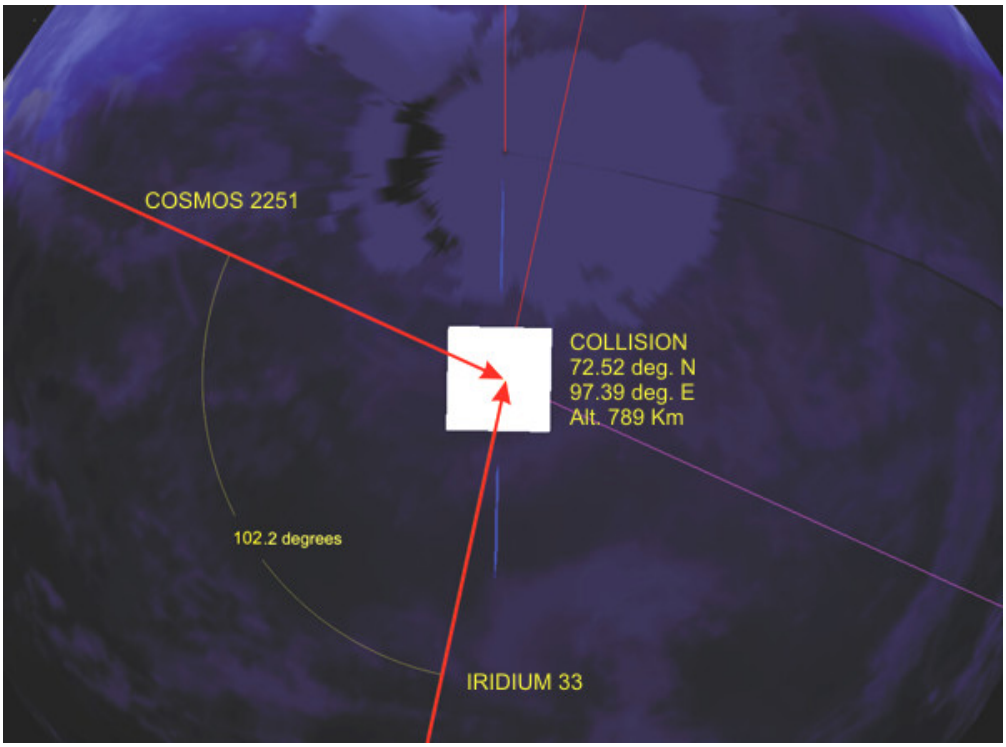

Figure 5: Visualization of the collision on Wikipedia.

Next Steps

- Further investigate the approach angle calculation and how to interpret it correctly.
- Try opening GMAT again to visualize the collision and the approach angle.
- Try other new senerios of collisions. Consider making my own senerios with TLEs that I can create.

Incomplete Goals from last week:

- Loop through all the TLEs in the LEO category and calculate the approach conditions for each satellite if it is within a certain distance (e.g., 100 km) of the ISS.
- Make my code a usable function so Catherine can just call it. Input? $\rightarrow$ Output (angle and approach speed)