Week 6 Report

Dhanush Balusa

June 25, 2025

Research

Consequence of a Collision:

- Reference: `https://ntrs.nasa.gov/api/citations/19910011417/downloads/19910011417.pdf`
- Focuses on the lack of data regarding small debris (1mm to 10cm) in low Earth orbit.
- Highlights that collisions are a significant source of debris, with no significant momentum transfer observed in breakup events.
- Proposes a two-component model for collisional breakups, distinguishing between directly and indirectly affected materials.
  - Directly Affected Materials: These are the components that directly experience the impact and participate in the momentum transfer during the collision.
  - Indirectly Affected Materials: These materials are not directly hit but are affected by the resulting shock waves and debris cloud, which can lead to larger fragments.
  - In on-orbit tests, only the indirectly affected materials, which form a larger explosion-type debris cloud, have been observed, while momentum transfer is limited to the directly involved materials characterized by the column mass of the target.
- Emphasizes the need to understand momentum transfer as frequency of collisions in space increases, particularly with the anticipated use of anti-satellite weapons, as it can significantly influence the amount and distribution of debris generated from such events.
- Concludes that accounting for momentum transfer can reduce the amount of debris in long-life orbits; however, it currently overlooks the impact of relative velocity and requires further validation through hypervelocity impact tests.

Break-up Model:

- Reference: `https://ntrs.nasa.gov/api/citations/20070007324/downloads/20070007324.pdf`
- The Fengyun-1C spacecraft was intentionally destroyed on January 11, 2007, creating over 2,000 pieces of debris.

- The breakup event was caused by a kinetic anti-satellite test conducted by China, which involved a direct-ascent missile intercepting the satellite at a relative velocity of approximately 9 km/s (hypervelocity impact).
- The U.S. Space Surveillance Network tracked the debris, revealing long-lived orbits and significant risks to operational spacecraft.
- The event marked the worst single fragmentation in space history, increasing the LEO debris population by over one-third.

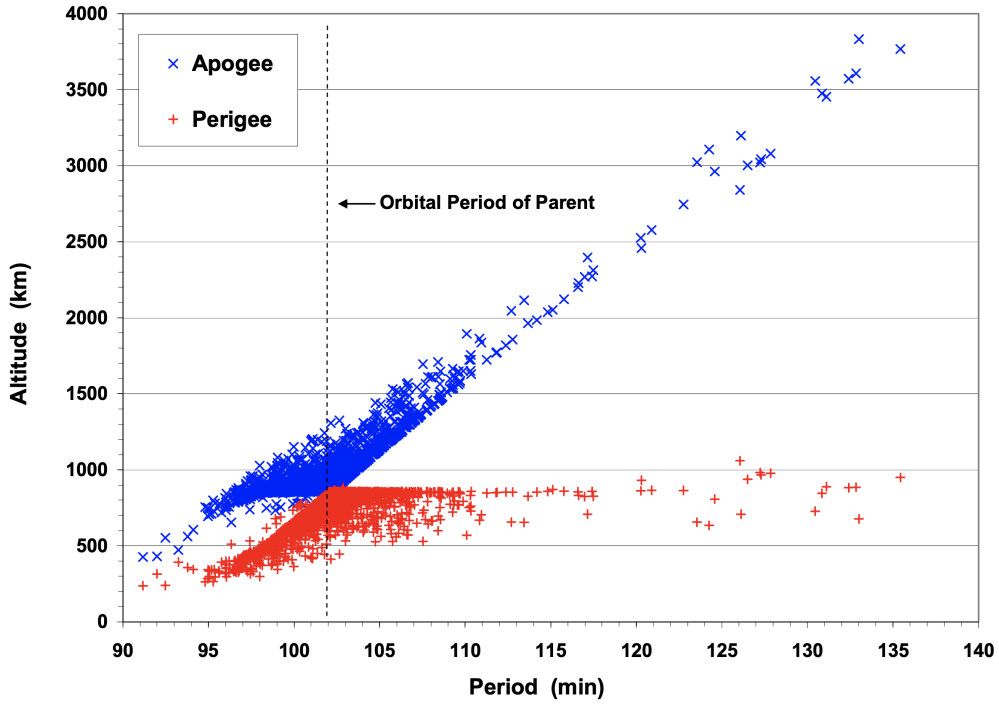

Figure 1: A Gabbard diagram (apogee and perigee versus orbital period) of the Fengyun-1C orbital debris cloud as assessed on 11 July 2007, six months after the break-up with a total of 2347 objects individually identified.

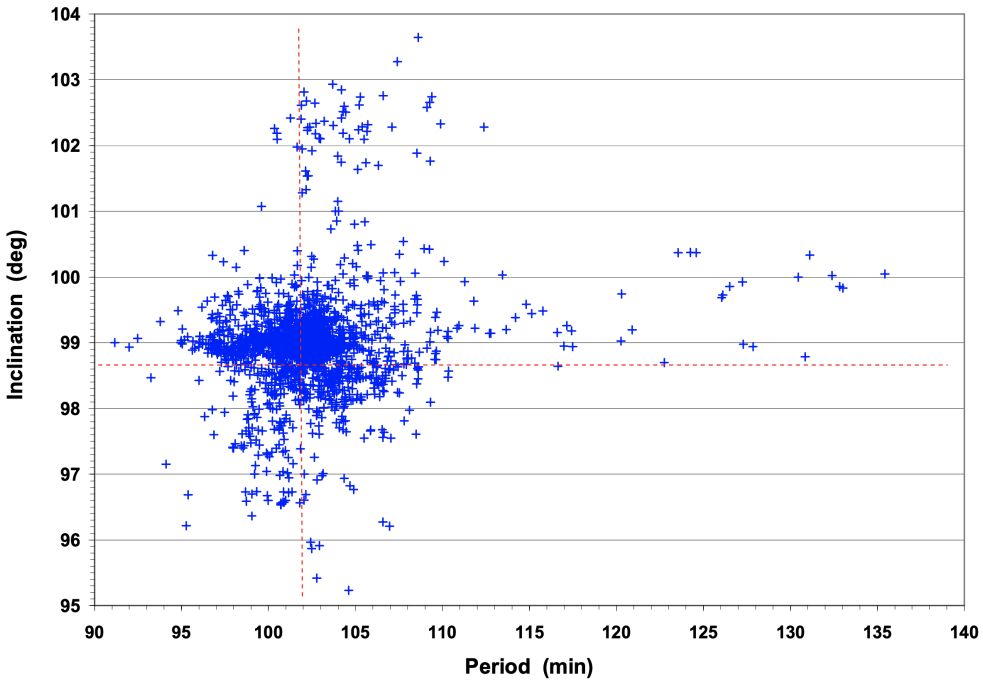

Figure 2: Approximately 80% of the tracked debris were found in inclinations greater than those of the spacecraft prior to impact.

- The debris poses ongoing collision risks, with potential long-term environmental impacts in LEO.

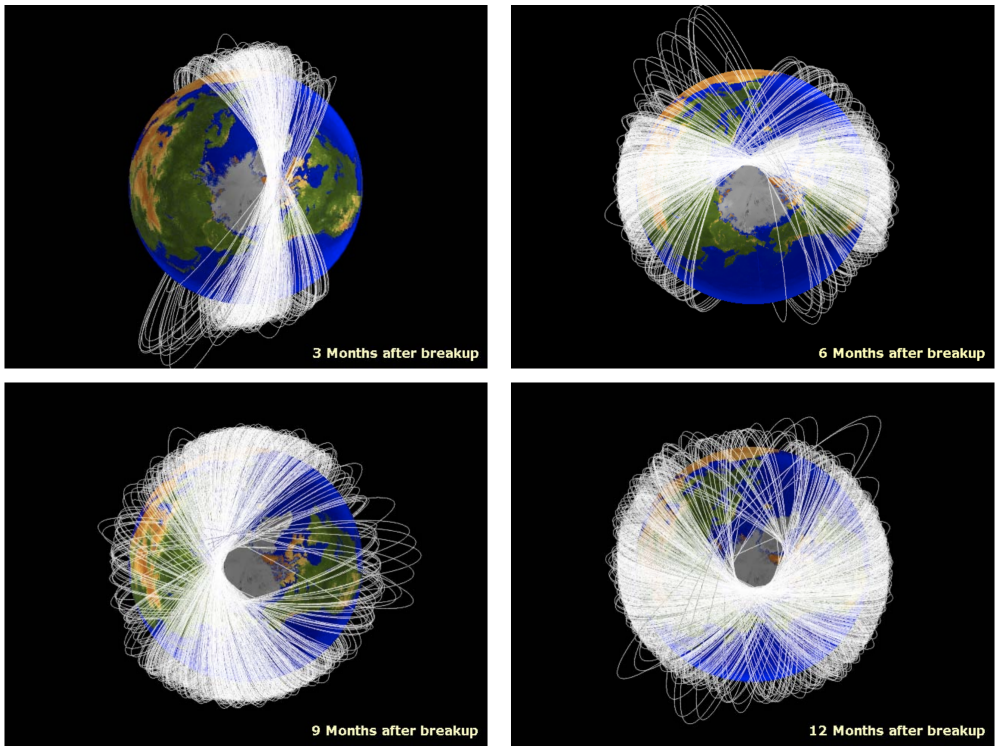

Figure 3: Evolution of the Fengyun-1C debris orbit planes.

Hypervelocity Impact:

- Reference: `https://www.esa.int/Space_Safety/Space_Debris/Hypervelocity_impacts_and_protecting_spacecraft`
- Hypervelocity Collisions: Defined as impacts with velocities $\geq$ 4 km/s, common in LEO.
- Collision Types:
  - Catastrophic: Both target and impactor destroyed.
  - Non-Catastrophic: Impactor destroyed, target damaged.
- Fragment Distribution: More small fragments produced than large; modeled using empirical formulas.
- Orbital Changes: Fragments assume new orbits based on ejection velocity; Gabbard diagrams illustrate orbital decay and distribution.

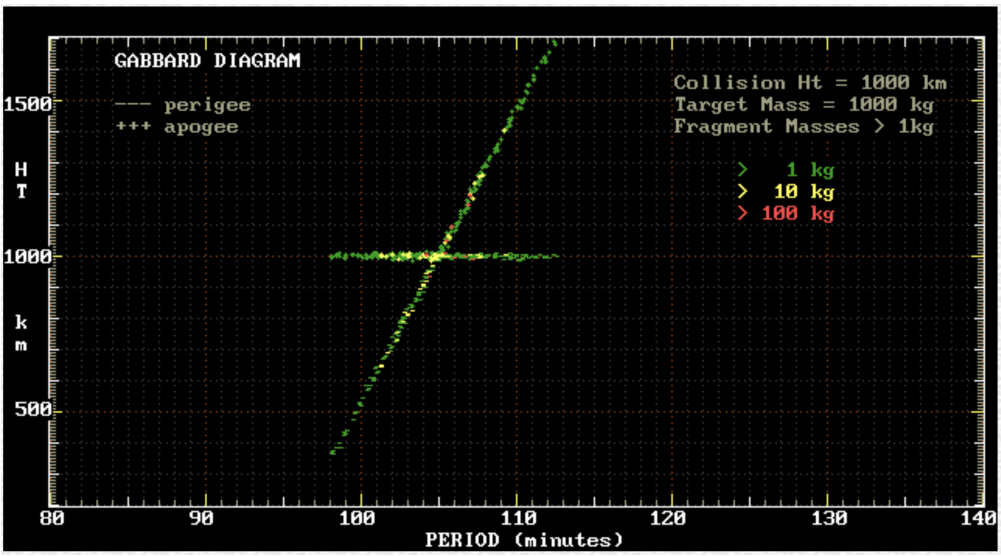

Figure 4: It can be seen that the largest masses ($\geq$ 100 kg) are relatively close to the original orbit, whereas the smallest masses ($\geq$ 1kg) are the most dispersed.

- Debris Behavior: Smaller fragments burn up upon re-entry; larger fragments decay slower due to atmospheric drag.

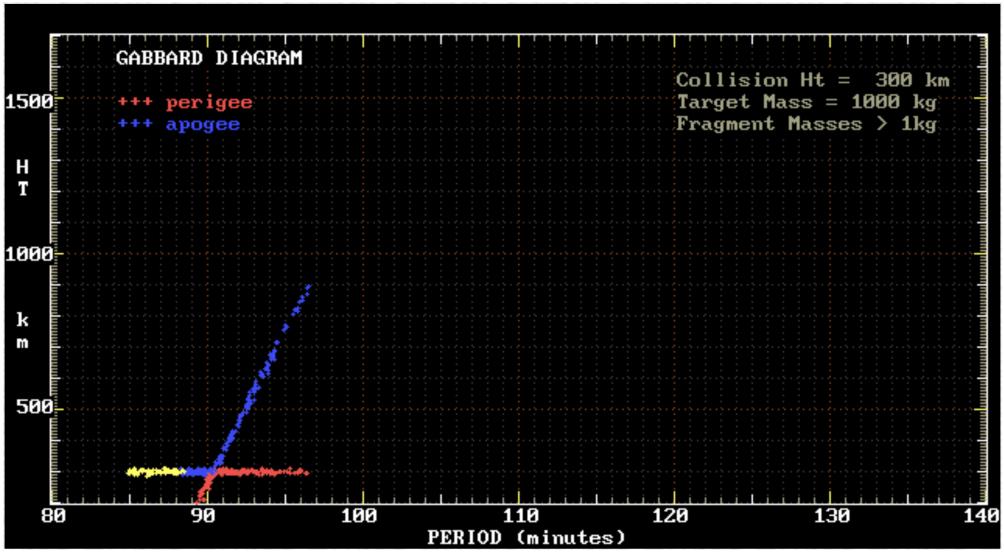

Figure 5: The yellow points represent fragments with perigees below 100 km, and which are thus not present after one orbit.

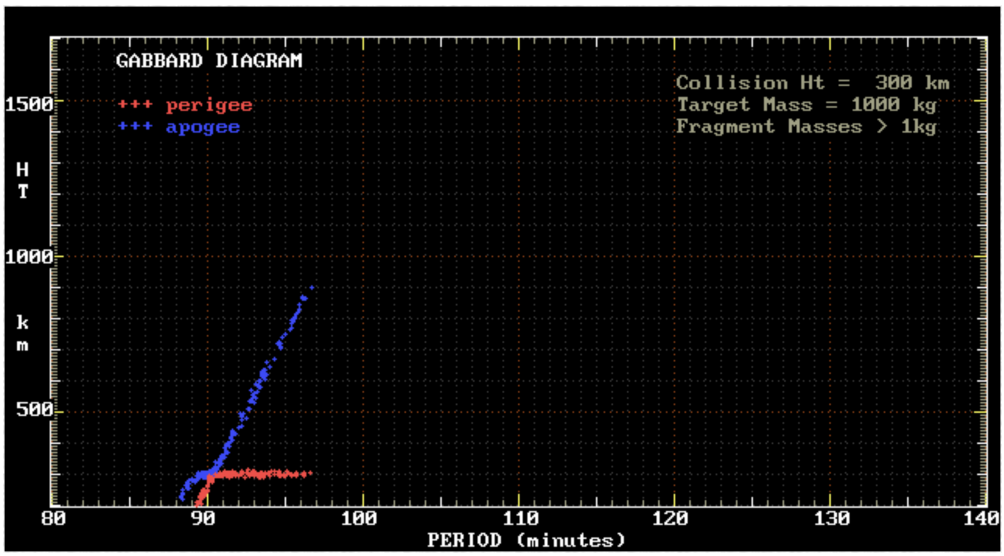

Figure 6: There is a progressive droop in apogee as the period gets smaller. This plot simulates the fragment population some days after the collision, in conditions of sunspot minimum (i.e. minimal solar X-ray and shortwave UV output, which implies low atmospheric densities in the upper atmosphere).

Other resources:

- Gabbard Diagram: `https://ntrs.nasa.gov/api/citations/20150009502/downloads/20150009502.pdf`
- ESA's Hypervelocity Impact Testing: `https://www.esa.int/Space_Safety/Space_Debris/Hypervelocity_impacts_and_protecting_spacecraft`

Code Changes

To clean up the code and based on the feedback from Space-Track.org, I decided to make the following changes:

1. **My first change was to create two separate scripts for better useability. One script for fetching the TLE data and the other for approach analyisis.**

```python
import requests
import getpass
from skyfield.api import EarthSatellite
import os

def fetch_tle(norad_id, name, username, password, start_date, end_date):
    LOGIN_URL = 'https://www.space-track.org/ajaxauth/login'
    TLE_URL = (
        f'https://www.space-track.org/basicspacedata/query/class/tle/'
        f'NORAD_CAT_ID/{norad_id}/EPOCH/{start_date}--{end_date}/orderby/EPOCH%20desc/format/tle'
    )

    session = requests.Session()
    login_data = {'identity': username, 'password': password}
    response = session.post(LOGIN_URL, data=login_data)
    if not response.ok:
        raise Exception(f"Login failed: {response.status_code}")

    tle_response = session.get(TLE_URL)
    if not tle_response.ok:
        raise Exception(f"Failed to fetch TLE: {tle_response.status_code}")

    tle_lines = tle_response.text.strip().split('\n')
    if len(tle_lines) < 2:
        tle = [name] + tle_lines
        return tle[0], tle[1], ""
    return name, tle_lines[0], tle_lines[1]

def get_user_inputs():
    username = input("Enter your Space-Track username: ")
    password = getpass.getpass("Enter your Space-Track password: ")
    num_sats = int(input("How many satellites do you want to fetch?: "))

    satellites_info = []
    for i in range(num_sats):
        norad_id = int(input(f"Enter NORAD Catalog ID for satellite #{i+1}: "))
        name = input(f"Enter name for satellite #{i+1}: ")
        satellites_info.append((norad_id, name))

    start_date = input("Enter TLE start date (YYYY-MM-DD): ")
    end_date = input("Enter TLE end date (YYYY-MM-DD): ")
    return username, password, satellites_info, start_date, end_date
```

Figure 7: Fetching TLE data from Space-Track.org.

```python
# Main execution
username, password, satellites_info, start_date, end_date = get_user_inputs()
tle_file = "satellites_tle.txt"

with open(tle_file, 'w') as f:
    for norad_id, name in satellites_info:
        print(f"Fetching TLE for {name}...")
        name, tle1, tle2 = fetch_tle(norad_id, name, username, password, start_date, end_date)
        f.write(f"{name}\n{tle1}\n{tle2}\n\n")

print(f"\nTLEs saved to {tle_file}")
```

Figure 8: Writing the TLE data to a file.

```python
# Load time steps
ts = load.timescale()
start_time = ts.utc(2025, 6, 19, 12, 0, 0)
hours = np.arange(0, 48, 1/1000)
time_steps = ts.utc(start_time.utc.year, start_time.utc.month, start_time.utc.day, hours)

# Calculate relative position and speed
times = []
relative_positions = {name: [] for name in names[1:]}
approach_speeds = {name: [] for name in names[1:]}

for t in time_steps:
    states = [sat.at(t) for sat in satellites]
    reference_state = states[0]
    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_positions[names[i]].append(np.linalg.norm(rel_pos))
        approach_speeds[names[i]].append(approach_speed)
    times.append(t.utc_iso())

# Find closest approach for the first pair
target_name = names[1]
min_index = np.argmin(relative_positions[target_name])
min_distance = relative_positions[target_name][min_index]
min_speed = approach_speeds[target_name][min_index]
min_time = times[min_index]

# Calculate approach angle
closest_time = time_steps[min_index]
state_sat1 = satellites[0].at(closest_time)
state_sat2 = satellites[1].at(closest_time)

rel_pos_vector = state_sat1.position.km - state_sat2.position.km
rel_vel_vector = state_sat1.velocity.km_per_s - state_sat2.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))

# Report
print(f"\n  Near Miss Detected Between {names[0]} and {names[1]}:")
print(f"    - Closest Distance:      {min_distance:.3f} km")
print(f"    - Approach Speed:        {min_speed:.3f} km/s")
print(f"    - Approach Angle:        {approach_angle_deg:.3f} degrees")
print(f"    - Time (UTC):            {min_time}")

# Plotting
plt.figure(figsize=(10, 6))
for name, positions in relative_positions.items():
    plt.plot(times, positions, label=f"Relative Position to {name}")
plt.xlabel("Time (UTC)")
plt.ylabel("Relative Position (km)")
plt.title(f"Relative Position to {names[1]} Over 2 Days")
plt.xticks(rotation=90)
ax2 = plt.gca()
ax2.xaxis.set_major_locator(ticker.MaxNLocator(nbins=15))
plt.legend()
plt.tight_layout()
plt.show()
```

Figure 9: Script for analyzing approach conditions.

2. **My second change is to make sure that the code does not violate the API rules.**

   This was the plan of action that I drafted based on the feedback from Space-Track.org and got approved:

   - Use of `/class/gp/` for latest propagable elements: For current TLE data, I will exclusively use: `https://www.space-track.org/basicspacedata/query/class/gp/decay_date/null-val/epoch/%3Enow-30/format/tle`.
   - Avoid deprecated endpoints: I will remove usage of `/class/tle/`, `/class/tle_latest/`, and any other deprecated classes as per Space-Track's notice.
   - Avoid repeated historical downloads: I will no longer query `/class/tle/` or `/class/gp_history/` multiple times for the same object/date range. Instead, I will download the official yearly zip files from Space-Track's Sync.com archive: `https://ln5.sync.com/dl/afd354190/c5cd2q72-a5qjzp4q-nbjdiqkr-cenajuqu`.
   - No automatic or timed scripts: My current use is manual and academic in nature—the script is not scheduled or automated. However, if I do automate it in the future, I will: Run it 10–20 minutes before or after the hour (i.e., before XX:20 or after XX:40) and/or use batch queries efficiently and responsibly.

   I also got approved for using the Testing URL (strictly for testing):
   `https://for-testing-only.space-track.org`

   API Help Documentation:
   `https://for-testing-only.space-track.org/documentation#/api`

Next Steps

My Next Steps:

- Research about conjunction analysis, specifically:
  - How do other people propagate orbit to predict collision?
  - What other parameters are important in conjunction analysis? Propagation techniques, something else other than TLE data?
- Implement my plan of action to ensure that the code is compliant with Space-Track's API rules.
- Test the new scripts to ensure they work as expected and fetch the latest TLE data correctly. Use the 4 test cases from Week 5 to verify.

Dr. Fan's feedback for next steps:

1. Research what a Gabbard diagram is.
2. Can I find the spacecraft type (i.e. debris, rocket body, operational satellite) from the TLE data? If so plot data points coresponding to each spacecraft type (by color). The plot can be anything, as long as it can be done.

End goals:

- 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. Some input $\rightarrow$ Output (approach speed and angle).