3D Flight Trajectory Animation & Visualization: Code Documentation and Report

Rajdeep

June 15

1 Introduction

This report documents a Python script that visualizes a 3D flight trajectory using data from an Excel file containing navigation logs. The script, developed using Pandas for data processing and Plotly for visualization, animates the flight path with a moving aircraft marker and hover tooltips displaying detailed flight parameters (time, roll, pitch, heading, longitude, latitude, altitude). The code is robust, handling large datasets (380,744 rows), parsing errors, and performance optimization through data reduction. This document explains the code's functionality, breaks down each major component, and addresses frequently asked questions (FAQs) an interviewer might ask.

The script's key features include:

- Loading and cleaning navigation data from an Excel file.
- Animating a 3D flight path with start/end markers and a moving aircraft.
- Displaying hover tooltips with all flight parameters.
- Adaptive data reduction for performance.
- Robust error handling for file loading and time parsing.

2 Code Overview

The script is written in Python and uses the following libraries:

- **Pandas**: For loading, cleaning, and manipulating Excel data.
- **Plotly**: For creating interactive 3D visualizations and animations.
- **NumPy**: For generating frame indices during animation.

The code follows a modular structure:

1. **Data Loading and Cleaning**: Reads the Excel file, standardizes column names, and parses the time column.

2. **Data Reduction**: Reduces the dataset to 10–100 points for performance.
3. **Hover Text Creation**: Generates tooltips with flight details.
4. **Visualization Setup**: Creates markers for start, end, aircraft, and trajectory.
5. **Animation Frames**: Defines frames for the animated trajectory and aircraft movement.
6. **Plot Rendering**: Configures the Plotly figure with axes, buttons, and animation controls.

3 Detailed Code Explanation

Below, each major code block is explained with its purpose, inputs, outputs, and logic.

3.1 Loading and Cleaning Data

```
try:
    df = pd.read_excel("/Users/rajdeep/Documents/test
       /2023-07-04-09-36-14-5325-NAVIGATION.xlsx")
    print(f"Loaded {len(df)} rows from Excel file.")
except FileNotFoundError:
    print("Error: File not found. Please ensure the file path is
       correct.")
    exit()
```

Purpose: Loads the Excel file containing flight data (e.g., 380,744 rows).
Inputs: File path to the Excel file.
Outputs: Pandas DataFrame (‘df‘) with raw data.
Logic: Uses ‘pd.read$_{e}xcel‘toloadthefile, withatry-exceptblocktohandlemissingfiles.Printsthenumber$

3.2 Standardizing Column Names

```
df.columns = (df.columns.str.strip()
              .str.lower()
              .str.replace(r'[()\s]', '', regex=True)
              .str.replace('deg', '')
              .str.replace('meters', ''))
df = df.rename(columns={'lattitude': 'latitude', 'trueheading': '
   heading'})
```

Purpose: Cleans and standardizes column names for consistency.
Inputs: DataFrame with raw column names (e.g., ‘TIME‘, ‘ROLL(DEG)‘, ‘LATTITUDE‘).
Outputs: DataFrame with cleaned names (e.g., ‘time‘, ‘roll‘, ‘latitude‘).
Logic: Removes spaces, parentheses, and units (‘deg‘, ‘meters‘), converts to lowercase, and renames specific columns (‘lattitude‘ to ‘latitude‘, ‘trueheading‘ to ‘heading‘).

3.3 Time Parsing

```python
time_formats = ['%H:%M:%S.%f', '%H:%M:%S', '%Y-%m-%d %H:%M:%S.%f', '%Y-%m-%d %H:%M:%S']
for fmt in time_formats:
    try:
        df['time'] = pd.to_datetime(df['time'].astype(str), format=fmt, errors='coerce')
        if df['time'].notna().sum() > 0:
            print(f"Successfully parsed time with format: {fmt}")
            break
    except Exception as e:
        print(f"Failed to parse with format {fmt}: {e}")
else:
    print("Warning: All time parsing attempts failed. Using original data without time sorting.")
    df['time'] = range(len(df))
```

Purpose: Parses the ‘time‘ column into datetime objects for sorting.
Inputs: DataFrame with ‘time‘ column (e.g., ‘07:21:15.359574‘).
Outputs: DataFrame with parsed ‘time‘ column or index-based fallback.
Logic: Tries multiple time formats, converts valid entries to datetime, and drops invalid rows. If all formats fail, uses row indices as a fallback. Prints parsing results for debugging.

3.4 Data Reduction

```python
target_points = min(max(10, len(df) // 100), 100)
step = max(1, len(df) // target_points)
df_reduced = df.iloc[::step].copy()
if len(df_reduced) < 2:
    print("Warning: Using all available points due to small dataset.")
    df_reduced = df.copy()
```

Purpose: Reduces the dataset to 10–100 points for performance.
Inputs: Cleaned Data-Frame (‘df‘).
Outputs: Reduced Data-Frame (‘df$_r$*educed*‘).
Logic : *Calculatesa‘step‘sizetosamplerows, ensuringatleast*10*points.Fallsbacktothefulldatasetiftoos*

3.5 Creating Hover Text

```python
hover_text = [
    f"Time: {row['time']}<br>"
    f"Roll: {row['roll']:.2f}  <br>"
    f"Pitch: {row['pitch']:.2f}  <br>"
    f"Heading: {row['heading']:.2f}  <br>"
    f"Longitude: {row['longitude']:.6f}<br>"
    f"Latitude: {row['latitude']:.6f}<br>"
    f"Altitude: {row['altitude']:.2f} m"
    for _, row in df_reduced.iterrows()
]
```

Purpose: Generates tooltips for hover interactions.
Inputs: Reduced Data-Frame (‘df$_reduced$‘).
Outputs : $Listofformattedstrings(‘hover_text‘).$
Logic : $IteratesoverrowstocreateHTML-formattedstringswithallflightparameters,roundedforrea$

3.6 Setting Up Markers

```python
start_marker = go.Scatter3d(
    x=[df_reduced['longitude'].iloc[0]], y=[df_reduced['latitude'].iloc[0]],
    z=[df_reduced['altitude'].iloc[0]], mode='markers+text',
    marker=dict(size=8, color='green'), text=["Start"],
    textposition="top center", hoverinfo='text', hovertext=[hover_text[0]], name='Start'
)
aircraft_marker = go.Scatter3d(
    x=[df_reduced['longitude'].iloc[0]], y=[df_reduced['latitude'].iloc[0]],
    z=[df_reduced['altitude'].iloc[0]], mode='markers',
    marker=dict(size=10, color='orange', symbol='circle', opacity=0.8),
    hoverinfo='text', hovertext=[hover_text[0]], name='Aircraft'
)
```

Purpose: Defines static and dynamic markers for the visualization.
Inputs: Reduced DataFrame and hover text.
Outputs: Plotly ‘Scatter3d‘ objects for start, end, and aircraft markers.
Logic: Creates green start and red end markers with labels, and an orange aircraft marker that moves during animation. Includes hover tooltips.

3.7 Creating Animation Frames

```python
num_frames = min(50, len(df_reduced) - 1)
frame_indices = np.linspace(1, len(df_reduced) - 1, num_frames, dtype=int)
frames = []
for i in frame_indices:
    frames.append(go.Frame(
        data=[
            go.Scatter3d(x=df_reduced['longitude'][:i+1], y=df_reduced['latitude'][:i+1],
                         z=df_reduced['altitude'][:i+1], mode='lines+markers',
                         marker=dict(size=3, color='blue'), line=dict(width=3, color='blue'),
                         hoverinfo='text', hovertext=hover_text[:i+1], name='Flight Path'),
            go.Scatter3d(x=[df_reduced['longitude'].iloc[i]], y=[df_reduced['latitude'].iloc[i]],
                         z=[df_reduced['altitude'].iloc[i]], mode='markers',
                         marker=dict(size=10, color='orange', symbol='circle', opacity=0.8),
                         hoverinfo='text', hovertext=[hover_text[i]], name='Aircraft')
        ],
        name=f'frame{i}'
    ))
```

Purpose: Defines animation frames for the trajectory and aircraft movement.
Inputs: Reduced DataFrame, hover text, frame indices.
Outputs: List of Plotly ‘Frame‘ objects.
Logic: Creates up to 50 frames, each updating the trajectory (growing line) and aircraft marker position with corresponding hover tooltips.

3.8 Rendering the Plot

```
fig = go.Figure(data=[start_marker, end_marker, go.Scatter3d(...), aircraft_marker],
                layout=go.Layout(title="3D Flight Trajectory Animation with Hover Tooltips",
                                 scene=dict(xaxis_title='Longitude', yaxis_title='Latitude',
                                            zaxis_title='Altitude (m)', aspectmode='auto'),
                                 updatemenus=[...], showlegend=True),
                frames=frames)
fig.update_scenes(xaxis=dict(range=[df_reduced['longitude'].min() - 0.0001, ...]), ...)
fig.show()
```

Purpose: Configures and displays the Plotly figure.
Inputs: Markers, frames, and layout settings.
Outputs: Interactive 3D plot in the browser.
Logic: Combines all traces, sets axis labels, adds play/pause buttons, adjusts axis ranges to fit data, and renders the plot.

4 Frequently Asked Questions (FAQs)

Below are questions an interviewer might ask about the code, along with concise answers.

4.1 Why did you use Plotly instead of Matplotlib or another library?

Answer: Plotly was chosen for its interactive 3D visualization capabilities, including animations and hover tooltips, which are ideal for exploring flight data. Matplotlib’s 3D plots are less interactive and harder to animate smoothly, while Plotly’s browser-based rendering supports large datasets with good performance.

4.2 How does the code handle large datasets like 380,744 rows?

Answer: The code reduces the dataset to 10–100 points using adaptive sampling (‘step = len(df) // target$_{p}oints‘).Thisminimizesmemoryusageandrenderingtimewhilepreservingthetrajectory's$

4.3 What error handling is implemented?

Answer: The code handles:

- FileNotFoundError for missing Excel files.
- Missing required columns with explicit checks.
- Time parsing failures by trying multiple formats and falling back to row indices.
- Insufficient data points by using the full dataset if reduction fails.

Debugging output (e.g., row counts, time samples) helps diagnose issues.

4.4 How would you optimize the code for even larger datasets?

Answer: I’d:

- Increase the sampling step size to reduce points further.
- Use chunked reading with Pandas (‘pd.read$_{e}xcel(chunksize = ...)‘)toprocesslargefilesincrementall$

- Switch to WebGL rendering in Plotly for faster 3D plots.

4.5 Why limit the animation to 50 frames?

Answer: Limiting to 50 frames balances smoothness and performance. More frames increase rendering time, especially for large datasets, while 50 provides a fluid animation with minimal lag. The frame indices are evenly spaced using ‘np.linspace‘ to cover the trajectory.

4.6 How would you add orientation to the aircraft marker?

Answer: I’d use the ‘heading‘ column to rotate a directional marker (e.g., ‘symbol=’triangle-up’‘) by mapping heading values to Plotly’s symbol orientation. Alternatively, I’d add a small line segment trace aligned with the heading angle, updated in each frame.

4.7 What challenges did you face with time parsing?

Answer: The ‘time‘ column’s format was inconsistent across datasets. I addressed this by trying multiple formats (‘

4.8 Can the code handle missing or invalid data?

Answer: Yes, it drops rows with missing values in required columns (‘time‘, ‘latitude‘, ‘longitude‘, ‘altitude‘) using ‘df.dropna‘. Invalid time entries are coerced to NaN and dropped, with a fallback to indices if all fail. Debugging output identifies data issues.

4.9 How would you test the code’s robustness?

Answer: I’d test with:

- Small datasets (¡10 rows) to check fallback logic.
- Large datasets (¿1M rows) to verify performance.
- Invalid files (wrong path, missing columns) to test error handling.
- Diverse time formats to ensure parsing flexibility.

Unit tests with ‘pytest‘ could automate these checks.

4.10 What improvements would you make for a production environment?

Answer: I’d:

- Add logging instead of print statements for debugging.
- Create a configuration file for file paths, formats, and parameters.
- Modularize the code into functions/classes for reusability.
- Add input validation for data ranges (e.g., latitude -90 to 90).
- Deploy as a web app (e.g., Dash) for broader access.

5 Conclusion

This script provides a robust, interactive visualization of flight trajectories, suitable for analyzing navigation data. Its modular design, error handling, and performance optimizations make it reliable for large datasets. The hover tooltips and animated aircraft marker enhance usability.