Discussion Summary – Orientation & Position Tracking

This document summarizes the work done today regarding orientation tracking using VPython and initial attempts at position tracking using accelerometer data. The goal was to visualize both the orientation and trajectory of an object in 3D space based on sensor readings.

1. Orientation Tracking

We successfully implemented orientation tracking using VPython. The orientation of a 3D box was updated in real-time based on gyroscope readings (angular velocity in radians/sec). The following key points were covered:

- A 3D box model was created in VPython with a clear understanding of its axis alignment:
    - X-axis → length (red)
    - Y-axis → height (green)
    - Z-axis → width (blue)
- Coordinate axes were added to the scene for easier visualization.
- Gyroscope data was integrated over time to estimate roll, pitch, and yaw.
- Rotations were applied to the 3D box, matching the device's orientation.

2. Position Tracking Attempt

We attempted to implement position tracking using accelerometer data (in the device frame). The process involved integrating acceleration to obtain velocity and then integrating velocity to obtain position. The following aspects were discussed:

- Reading acceleration and rotation rate data from a CSV file with headers: `time, ax, ay, az, wx, wy, wz`.
- Adjusting for the fact that acceleration data was in the device frame, requiring conversion to the world frame using orientation data.
- Creating a separate object in VPython to track position independently from orientation.
- Encountering visibility issues when the tracked position moved in certain axes, making it hard to interpret motion in 3D.

3. Alternative Visualization

To better visualize the position tracking results, we discussed plotting the computed positions in a separate matplotlib window. The plan was to display the trajectory in real-time with X and Y axes visible to gain insight into movement patterns. However, the live plotting implementation had significant lag, which will need optimization in future iterations.

4. Next Steps

- Optimize position tracking calculations to run smoothly in real-time.
- Improve live plotting performance for matplotlib (or switch to a faster visualization tool).

- Implement proper orientation-to-world-frame acceleration transformation.
- Test position tracking with cleaned, calibrated sensor data to reduce integration drift.
- Possibly combine VPython and live 2D/3D plotting for better clarity.