Sensor Fusion Spoofing Attack Analysis

Research Goals

- **Objective:** Test the limits of Kalman filter-based GPS+IMU sensor fusion against four GPS spoofing attacks in the CARLA simulator.
- **Attacks Tested:**
  1. Gradual Drift
  2. Sudden Jump
  3. Random Walk
  4. Replay
- **Key Question:** Can sensor fusion alone (without explicit spoofing detection) mitigate these attacks?

File Integrations and Pipeline Overview

Key Files and Their Roles

File	Purpose/Role
`scene.py`	Spawns and navigates the vehicle using CARLA's autopilot.
`fpv_ghost.py`	Real-time dual camera (FPV and ghost) visualization using Pygame.
`sync.py`	Real-time Kalman filter and position/trajectory visualization using Pygame.
`integration_files/advanced_kalman_filter.py`	Advanced Kalman filter implementation for GPS+IMU fusion.
`integration_files/sensor_fusion.py`	Sensor fusion logic using the advanced Kalman filter; accepts spoofed GPS and IMU data.
`integration_files/gps_spoofer.py`	Implements the four GPS spoofing attacks.
`integration_files/sequential_attack_test.py`	Orchestrates the sequential execution of all four attacks, collects results, and analyzes.
`integration_files/data_processor.py`	Processes and analyzes collected data for ML and statistical analysis.
`run_all.py`	Launches all real-time visualizations and scene setup in separate terminals.

Integration Flow

1. **Vehicle is spawned and navigated** using `scene.py` (autopilot).
2. **Sensor fusion** is performed in real time using the advanced Kalman filter (`advanced_kalman_filter.py`) via `sensor_fusion.py`.

3. **GPS spoofing attacks** are applied using `gps_spoofer.py`.
4. **Sequential attack testing** is managed by `sequential_attack_test.py`, which runs all attacks, collects results, and prints analysis.
5. **Real-time visualizations** are provided by `sync.py` (Kalman/trajectory) and `fpv_ghost.py` (camera views), launched via `run_all.py` or manually.
6. **Post-run analysis** can be performed using `data_processor.py`.

What We Are Testing and Why

- **Purpose:**
    - To determine if sensor fusion (Kalman filter with GPS+IMU) can mitigate or detect various GPS spoofing attacks without explicit spoofing detection logic.
- **Why:**
    - Sensor fusion is a common defense in autonomous vehicles, but its limits against sophisticated spoofing are not always clear.
    - Understanding these limits informs whether additional detection/correction logic is needed.

Test Methodology

- **Each attack is run for a fixed duration** using `sequential_attack_test.py`.
- **Sensor fusion** is performed in real time, fusing GPS (possibly spoofed) and IMU data.
- **Real-time visualizations** allow for live monitoring of vehicle state and filter performance.
- **Results** (position/velocity errors) are collected and analyzed for each attack.

Results Summary

Attack	Mean Pos Error	Max Pos Error	Mean Vel Error	Max Vel Error	Notes
Gradual Drift	9.29 m	273.05 m	21.53 m/s	1463.05 m/s	Large errors, high variance
Sudden Jump	17.40 m	111.23 m	11.47 m/s	787.73 m/s	Large errors, high variance
Random Walk	0.23 m	0.97 m	2.76 m/s	37.99 m/s	Small errors, low variance
Replay	0.001 m	0.042 m	0.004 m/s	0.35 m/s	Negligible errors

Interpretation

Gradual Drift

- **Kalman filter is not able to fully mitigate the attack.**
- Position error grows large over time as the spoofed GPS drags the estimate away from the true position.
- High max error indicates the filter can be completely misled.

Sudden Jump

- **Kalman filter is not robust to sudden jumps.**
- Large, sudden GPS changes pull the filter far from the true position, resulting in large errors.
- The filter may recover between jumps, but the attack is effective.

Random Walk

- **Kalman filter is very effective.**
- The filter smooths out random noise, keeping the estimate close to the true position.
- Occasional spikes in velocity error, but overall robust.

Replay

- **Kalman filter completely mitigates the attack.**
- The filter's prediction and IMU data allow it to ignore repeated GPS values, maintaining an accurate estimate.

How is the Kalman Filter Working?

Strengths:

The filter is very good at rejecting high-frequency, random noise (random walk, replay). When the attack is “unrealistic” (e.g., GPS values that don’t match IMU predictions), the filter can rely on the IMU to maintain a good estimate.

Weaknesses:

Slow, consistent drift (gradual drift) and large, sudden jumps (sudden jump) can “fool” the filter, especially if the filter trusts the GPS too much or the IMU integration drifts over time. The filter’s ability to reject attacks depends on the relative trust (covariance) assigned to GPS vs. IMU, and the nature of the attack.

Is Sensor Fusion Catching the Attacks?

Random Walk & Replay:

Yes. The fusion is robust, and the filter “catches” and corrects the attack.

Gradual Drift & Sudden Jump:

No. The filter is not able to detect or correct these attacks on its own. The position error grows large, indicating the attack is successful.

Overall Conclusions

- **Sensor fusion with a Kalman filter is highly effective against random walk and replay attacks, but vulnerable to gradual drift and sudden jump attacks.**
- **For full spoofing resilience, additional detection/correction logic is needed beyond basic sensor fusion.**

- **Real-time visualizations and data collection confirm these findings and provide insight into filter performance.**

Recommendations

- Tune Kalman filter parameters for improved robustness.
- Add explicit spoofing detection (e.g., innovation monitoring, statistical checks).
- Consider fusing additional sensors for greater resilience.

How to Run the Pipeline

1. **Start the CARLA Simulator (CarlaUE4).**
2. **Activate your virtual environment.**
3. **From your project root, run:**

   ```
   python run_all.py
   ```

   - This opens three terminals for scene setup, Kalman/trajectory visualization, and camera visualization.
4. **In a fourth terminal, run:**

   ```
   python sensor_fusion_testing/integration_files/sequential_attack_test.py
   ```

   - This runs the attack test and prints results.
5. **For post-run analysis, run:**

   ```
   python sensor_fusion_testing/integration_files/data_processor.py
   ```