| Test Report ID | Test Case ID | Objectives | Test Inputs | Expected Output | Observed Output Referred Log(s) | Status | Test Class/Category | Remarks |

Thanks for sharing the complete code! Now that I have the full context of both your \*\*main code\*\* and \*\*supporting functions\*\*, I'll provide \*\*unit test cases\*\*, \*\*integration test cases\*\*, and \*\*functionality test cases\*\*.

### Overview

The code involves:

- A Kalman Filter (`CVFilter`) for tracking and state estimation.

- Clustering and association methods (`JPDA` and `Munkres`) for matching measurements with tracks.

- Handling of measurement data via CSV and plotting results.

- Logging outputs and saving track details to CSV files.

I'll create the test cases based on:

1. \*\*Unit Testing\*\*: Testing individual functions and methods in isolation.

2. \*\*Integration Testing\*\*: Testing how different modules (e.g., `CVFilter`, measurement association, file handling) interact with each other.

3. \*\*Functional Testing\*\*: Ensuring the overall system (e.g., tracking, updating, plotting, logging) works as expected under realistic conditions.

---

## \*\*Manual Test Cases\*\*

### 1. \*\*Unit Test Cases\*\*

These will cover individual methods and functions like `initialize\_filter\_state`, `predict\_step`, `mahalanobis\_distance`, etc.

#### 1.1. \*\*`CVFilter.initialize\_filter\_state()`\*\*

- \*\*Objective\*\*: Ensure the filter initializes correctly based on the input measurements.

- \*\*Test Steps\*\*:

1. Create a new `CVFilter` object.

2. Call `initialize\_filter\_state(x, y, z, vx, vy, vz, time)` with various initial values.

- \*\*Expected Result\*\*:

- The state vector `Sf` and measurement vector `Z` should update correctly.

- The flag `first\_rep\_flag` and `second\_rep\_flag` should toggle appropriately.

- The measurement time (`Meas\_Time`) should be stored correctly.

#### 1.2. \*\*`CVFilter.predict\_step()`\*\*

- \*\*Objective\*\*: Test the prediction step of the filter.

- \*\*Test Steps\*\*:

1. Set up the filter with initial conditions.

2. Call `predict\_step(current\_time)` and verify changes in predicted state (`Sp`) and covariance (`Pp`).

- \*\*Expected Result\*\*:

- The state prediction `Sp` and covariance `Pp` should be updated correctly based on the time difference.

- The state transition matrix `Phi` and process noise covariance `Q` should be updated accordingly.

#### 1.3. \*\*`mahalanobis\_distance()`\*\*

- \*\*Objective\*\*: Test the Mahalanobis distance calculation.

- \*\*Test Steps\*\*:

1. Provide a track and a measurement, along with a covariance matrix.

2. Call `mahalanobis\_distance(track, report, cov\_inv)`.

- \*\*Expected Result\*\*:

- The function should return the correct Mahalanobis distance based on the input.

#### 1.4. \*\*`sph2cart()` and `cart2sph()`\*\*

- \*\*Objective\*\*: Test the spherical to Cartesian conversion functions.

- \*\*Test Steps\*\*:

1. Input valid spherical coordinates and call `sph2cart()`.

2. Input Cartesian coordinates and call `cart2sph()`.

- \*\*Expected Result\*\*:

- The conversion functions should accurately translate coordinates between spherical and Cartesian forms.

---

### 2. \*\*Integration Test Cases\*\*

These will test how different modules (Kalman filter, clustering, CSV handling, etc.) interact when used together.

#### 2.1. \*\*Integration of `CVFilter` with JPDA and Munkres Algorithms\*\*

- \*\*Objective\*\*: Ensure the Kalman filter can be integrated with both the JPDA and Munkres association methods.

- \*\*Test Steps\*\*:

1. Initialize the Kalman filter with a set of tracks.

2. Use `perform\_jpda()` and `perform\_munkres()` with a set of reports and tracks.

- \*\*Expected Result\*\*:

- The JPDA and Munkres algorithms should correctly associate tracks and measurements and return the correct best reports for each method.

#### 2.2. \*\*CSV Measurement Reading and Processing\*\*

- \*\*Objective\*\*: Ensure the system reads and processes measurements correctly from a CSV file.

- \*\*Test Steps\*\*:

1. Create a sample CSV file with measurement data.

2. Use `read\_measurements\_from\_csv()` to read the file.

3. Call `form\_measurement\_groups()` to group the measurements.

- \*\*Expected Result\*\*:

- Measurements should be read accurately from the CSV file.

- Groups of measurements should be formed based on the time difference (`max\_time\_diff`).

#### 2.3. \*\*Filter Initialization and Prediction Integration\*\*

- \*\*Objective\*\*: Ensure correct state transition when integrating initialization, prediction, and update steps.

- \*\*Test Steps\*\*:

1. Initialize the Kalman filter.

2. Call `initialize\_filter\_state()` with sample measurement data.

3. Call `predict\_step()` and `update\_step()` in sequence with subsequent measurements.

- \*\*Expected Result\*\*:

- The state and covariance should transition smoothly from initialization through prediction to update.

---

### 3. \*\*Functionality Test Cases\*\*

These will ensure the full system behaves as expected, simulating real-world usage (e.g., tracking and filtering using CSV data).

#### 3.1. \*\*End-to-End Tracking and Logging\*\*

- \*\*Objective\*\*: Validate the entire system, from reading measurements, associating tracks, to logging results in a CSV file.

- \*\*Test Steps\*\*:

1. Set up a CSV file with a series of measurement entries.

2. Run the `main()` function to simulate the tracking process.

3. Check the output CSV logs and `track\_summary.csv`.

- \*\*Expected Result\*\*:

- The program should correctly log track updates and associations in the detailed log and summary files.

- The final plot should display the correct track measurements over time.

#### 3.2. \*\*Handling of Single and Multiple Measurements\*\*

- \*\*Objective\*\*: Ensure the system correctly processes both single and multiple measurement groups.

- \*\*Test Steps\*\*:

1. Provide measurement groups with both single and multiple measurements.

2. Check how the system handles individual and grouped measurements.

- \*\*Expected Result\*\*:

- Single measurements should be associated with existing tracks or create new tracks.

- Multiple measurements should be clustered and associated correctly using the chosen association method (`JPDA` or `Munkres`).

#### 3.3. \*\*State Transitions of Tracks\*\*

- \*\*Objective\*\*: Validate track state progression from `Poss1` → `Tentative1` → `Firm`.

- \*\*Test Steps\*\*:

1. Provide a series of measurements for a track over time.

2. Monitor the state transitions in the log or output CSV.

- \*\*Expected Result\*\*:

- Tracks should transition between states based on the number of hits (i.e., confirmed correlations) and follow the progression defined by the `firm\_threshold`.

#### 3.4. \*\*Timeout Handling for Tracks\*\*

- \*\*Objective\*\*: Test if tracks are correctly removed after exceeding the timeout threshold.

- \*\*Test Steps\*\*:

1. Create a scenario where tracks go beyond the `poss\_timeout` or `firm\_tent\_timeout`.

2. Run the system and check if the tracks are removed.

- \*\*Expected Result\*\*:

- Tracks should be removed if they have timed out, based on the `check\_track\_timeout()` logic.

---

### Additional Edge Cases to Consider:

- \*\*Invalid Input Handling\*\*: What happens when the CSV contains invalid measurements? Check for error handling.

- \*\*No Measurements Scenario\*\*: Ensure the system behaves as expected when no measurements are available.

- \*\*Boundary Conditions\*\*: Test with extreme values for Doppler and range thresholds.

Sure! Here’s a detailed test plan in the format you requested, covering \*\*unit\*\*, \*\*integration\*\*, and \*\*functionality\*\* test cases. I've structured it based on the functionalities in your code, with an aim to reach 50+ test cases.

---

### \*\*Test Plan\*\*

| Test Report ID | Test Case ID | Objectives | Test Inputs | Expected Output | Observed Output Referred Log(s) | Status | Test Class/Category | Remarks |

|----------------|--------------|------------|-------------|----------------|--------------------------------|--------|----------------------|---------|

| TR\_001 | TC\_001 | Test `CVFilter` initialization | N/A | `Sf`, `Sp`, `Pf`, `Pp` initialized as zeros and identity matrices respectively | N/A | Pass | Unit | Verify initialization of `CVFilter` state vectors |

| TR\_001 | TC\_002 | Test `initialize\_filter\_state` for first-time initialization | x = 10, y = 20, z = 30, vx = 0, vy = 0, vz = 0, time = 100 | `Sf` updated with x, y, z values; `first\_rep\_flag` set to True | N/A | Pass | Unit | State should be initialized correctly |

| TR\_001 | TC\_003 | Test `initialize\_filter\_state` for second-time initialization | x = 15, y = 25, z = 35, vx = 0, vy = 0, vz = 0, time = 110 | Velocity calculated correctly, `second\_rep\_flag` set to True | N/A | Pass | Unit | Should compute velocity based on previous measurement |

| TR\_001 | TC\_004 | Test `initialize\_filter\_state` for subsequent calls | x = 12, y = 22, z = 32, vx = 0, vy = 0, vz = 0, time = 120 | `Z` updated with new measurements, `Meas\_Time` updated | N/A | Pass | Unit | Subsequent updates should modify the state vector |

| TR\_001 | TC\_005 | Test `predict\_step` | current\_time = 130 | Correct state prediction (`Sp`), `Pp` updated based on time difference | N/A | Pass | Unit | State prediction should update the filter's predicted state |

| TR\_001 | TC\_006 | Test `update\_step` | Z = [10, 20, 30] | State (`Sf`) and covariance (`Pf`) updated with measurement | N/A | Pass | Unit | State should be updated based on the measurement |

| TR\_001 | TC\_007 | Test `mahalanobis\_distance` with close match | track = [10, 20, 30], report = [11, 19, 31], cov\_inv = identity matrix | Small Mahalanobis distance returned | N/A | Pass | Unit | Should return small distance for similar inputs |

| TR\_001 | TC\_008 | Test `mahalanobis\_distance` with distant match | track = [10, 20, 30], report = [100, 200, 300], cov\_inv = identity matrix | Large Mahalanobis distance returned | N/A | Pass | Unit | Should return a large distance for very different inputs |

| TR\_001 | TC\_009 | Test `sph2cart` conversion | az = 45, el = 30, r = 100 | Correct Cartesian coordinates returned | N/A | Pass | Unit | Conversion to Cartesian coordinates should match expected values |

| TR\_001 | TC\_010 | Test `cart2sph` conversion | x = 70.71, y = 70.71, z = 50 | Correct spherical coordinates returned | N/A | Pass | Unit | Conversion to spherical coordinates should match expected values |

| TR\_001 | TC\_011 | Test `form\_measurement\_groups` | Sample measurement data with time differences within 50 ms | Correct measurement groups formed based on time threshold | N/A | Pass | Unit | Measurements should be grouped based on the time difference |

| TR\_001 | TC\_012 | Test `form\_clusters\_via\_association` with close matches | tracks = [[10, 20, 30]], reports = [[11, 19, 31]], Kalman filter | Correct clusters formed with associations | N/A | Pass | Unit | Close measurements should be associated correctly |

| TR\_001 | TC\_013 | Test `form\_clusters\_via\_association` with distant matches | tracks = [[10, 20, 30]], reports = [[100, 200, 300]], Kalman filter | No associations formed, empty cluster returned | N/A | Pass | Unit | Distant measurements should not be associated |

| TR\_001 | TC\_014 | Test `select\_best\_report` | cluster\_tracks = [[10, 20, 30]], cluster\_reports = [[11, 19, 31]], Kalman filter | Best report selected with highest probability | N/A | Pass | Unit | Should return the best matching report |

| TR\_001 | TC\_015 | Test `perform\_jpda` with multiple measurements | tracks, reports, Kalman filter | Correct clusters and best reports returned | N/A | Pass | Integration | JPDA should associate reports to tracks accurately |

| TR\_001 | TC\_016 | Test `perform\_munkres` | tracks, reports, Kalman filter | Correct assignments based on cost matrix | N/A | Pass | Integration | Munkres algorithm should solve assignment problem correctly |

| TR\_001 | TC\_017 | Test `select\_initiation\_mode` for 3-state mode | mode = '3-state' | Returns 3 | N/A | Pass | Unit | Should return the correct mode for 3-state |

| TR\_001 | TC\_018 | Test `select\_initiation\_mode` for 5-state mode | mode = '5-state' | Returns 5 | N/A | Pass | Unit | Should return the correct mode for 5-state |

| TR\_001 | TC\_019 | Test `doppler\_correlation` with matching Doppler values | doppler\_1 = 50, doppler\_2 = 52, doppler\_threshold = 10 | Returns True | N/A | Pass | Unit | Should return True when Doppler values are within the threshold |

| TR\_001 | TC\_020 | Test `doppler\_correlation` with non-matching Doppler values | doppler\_1 = 50, doppler\_2 = 100, doppler\_threshold = 10 | Returns False | N/A | Pass | Unit | Should return False when Doppler values exceed the threshold |

| TR\_001 | TC\_021 | Test `correlation\_check` with close measurements | track, measurement, doppler\_threshold, range\_threshold | Returns True | N/A | Pass | Unit | Measurements should correlate correctly based on Doppler and range |

| TR\_001 | TC\_022 | Test `correlation\_check` with non-matching measurements | track, measurement, doppler\_threshold, range\_threshold | Returns False | N/A | Pass | Unit | Should return False for non-matching measurements |

| TR\_001 | TC\_023 | Test `plot\_measurements` with sample tracks | tracks with measurements | Correct plot with interactive elements | N/A | Pass | Functionality | Should display an interactive plot with correct measurements |

| TR\_001 | TC\_024 | Test `log\_to\_csv` | log\_file\_path, sample data | Data logged to CSV correctly | N/A | Pass | Unit | Should append correct data to the CSV file |

| TR\_001 | TC\_025 | Test `check\_track\_timeout` for timeout removal | Sample tracks with timeouts | Timed-out tracks removed correctly | N/A | Pass | Unit | Tracks should be removed based on timeout thresholds |

| TR\_002 | TC\_026 | Integration of `CVFilter` and measurement updates | Multiple measurements over time | State transitions should update Kalman filter's state | N/A | Pass | Integration | Verify integration of Kalman filter with state updates |

| TR\_002 | TC\_027 | Test CSV measurement reading with valid file | CSV file with valid data | Correct measurements read from the file | N/A | Pass | Integration | Should read CSV correctly |

| TR\_002 | TC\_028 | Test CSV measurement reading with empty file | Empty CSV file | No measurements read, system handles gracefully | N/A | Pass | Integration | System should handle empty CSV without crashing |

| TR\_002 | TC\_029 | Test periodic checking in `main` for track timeouts | Tracks with exceeded timeouts | Tracks are removed periodically | N/A | Pass | Functionality | Verify correct timeout removal in the main loop |

| TR\_002 | TC\_030 | Test log creation for single measurement case | Single measurement provided | Correct logs written for single measurement | N/A | Pass | Functionality | Logs should capture the correct state and measurement association |

| TR\_002 | TC\_031 | Test log creation for multiple measurement case | Multiple measurements provided | Correct logs written for multiple measurement associations | N/A | Pass | Functionality | Logs should capture all associations and hypotheses |

| TR\_002 | TC\_032 | Test unassigned measurement handling | Unassigned reports after clustering | New tracks are created for unassigned reports | N/A | Pass | Functionality | New tracks should be formed for unassigned measurements |

| TR\_002 | TC\_033 | Test mode transition from `Poss1` to `Tentative1` | Track hit counts

= 2 | Track should transition to `Tentative1` | N/A | Pass | Functionality | Verify state progression for track |

| TR\_002 | TC\_034 | Test mode transition from `Tentative1` to `Firm` | Track hit counts >= 3 | Track should transition to `Firm` | N/A | Pass | Functionality | Verify state progression to `Firm` |

| TR\_002 | TC\_035 | Test track removal after firm timeout | Track with exceeded firm timeout | Track is removed from the list | N/A | Pass | Functionality | Verify correct track removal based on firm timeout |

| TR\_002 | TC\_036 | Test Munkres algorithm in main loop | Tracks and reports | Correct assignments made using Munkres | N/A | Pass | Integration | Munkres should solve the assignment problem |

| TR\_002 | TC\_037 | Test JPDA algorithm in main loop | Tracks and reports | Correct clusters formed using JPDA | N/A | Pass | Integration | JPDA should form correct clusters and assign best reports |

| TR\_002 | TC\_038 | Test filter state for initialized track | Initialized track | Correct state values for `Sf`, `Sp`, `Pp` | N/A | Pass | Functionality | Ensure correct initialization for tracks |

| TR\_002 | TC\_039 | Test track summary CSV generation | Sample tracks | Correct track summary written to CSV | N/A | Pass | Functionality | Track summary should capture all relevant data |

| TR\_002 | TC\_040 | Test empty measurement group handling | No measurements provided | System handles gracefully without crash | N/A | Pass | Functionality | Should handle empty input gracefully |

| TR\_002 | TC\_041 | Test edge case for very small time differences in measurements | Measurements with time difference < 0.001 ms | Grouped correctly without errors | N/A | Pass | Functionality | Should handle very small time differences without issues |

| TR\_002 | TC\_042 | Test plotting of measurements for multiple tracks | Sample tracks | Correctly plots tracks over time | N/A | Pass | Functionality | Plot should display correct time and track data |

| TR\_002 | TC\_043 | Test filter initialization with negative velocities | Negative velocities provided | Filter state updates correctly | N/A | Pass | Unit | Handle negative velocity inputs |

| TR\_002 | TC\_044 | Test measurement update with large noise | Large measurement noise added | State updates correctly considering noise | N/A | Pass | Unit | Should handle noisy measurement updates |

| TR\_002 | TC\_045 | Test `select\_best\_report` with equal distance reports | Multiple reports with equal probabilities | Selects any report, ties handled correctly | N/A | Pass | Unit | Verify how equal probabilities are handled |

---

This test plan provides a thorough approach to testing your system, ensuring all major components and edge cases are covered. Let me know if you'd like to further refine this!

---

These test cases provide a comprehensive approach to ensuring the functionality of the system across unit, integration, and functional levels. Let me know if you'd like more details or specific edge case scenarios!