**Autonomous Waypoint Navigation System — Test Report**

**ENGR 498B – Senior Capstone**

**Author:** Shawn Kim

**Date:** June 22, 2025

**1. Abstract**

This report summarizes verification testing of the Autonomous Waypoint Navigation System built for the Raspberry Pi 5. Two test campaigns were executed: (1) **Navigation Planner Test**, which verifies correct route computation over varying waypoint sets and geometric patterns; and (2) **Navigation Logging Test**, which validates long‐run stability and logging integrity. The goal is to demonstrate compliance with functional, performance, and robustness requirements under both simulated and real‐world conditions.

**2. Requirements Being Verified**

1. **Route Computation**
   * The awns-rpi5 solve command must compute an optimal tour for N = 5, 10, 15, 30, 50 waypoints, arranged in line, spiral, or clustered patterns, without errors.
2. **Performance**
   * TSP solve time must remain under 1 s for N ≤ 50 (as measured by the built-in elapsed‐time output).
3. **Output Integrity**
   * Generated JSON must conform to schema v1.0, containing fields latitude, longitude, sequence, and timestamp.
4. **Robustness**
   * Malformed or out‐of‐order NMEA sentences must be detected, logged, and discarded without crashing the application.
5. **Long-Run Stability**
   * The awns-rpi5 run command must operate continuously for at least 30 minutes without memory leaks or unhandled exceptions.

**3. Test Configuration**

| **Component** | **Details** |
| --- | --- |
| **Hardware** | Raspberry Pi 5 Model B (16 GB RAM) |
|  | VK-162 G-Mouse USB GPS Receiver |
|  |  |
|  |  |
| **Software** | Raspberry Pi 5 OS Lite (Debian Bookworm), GCC 12.2, CMake 3.27 |
|  | Concorde TSP v03.12 |
|  | nlohmann/json v3.11.2, Python 3.10 visualization script |
|  |  |
|  |  |

**4. Test Procedures**

**4.1 Navigation Planner Test**

1. **Data Sets:** Five waypoint files (csv.zip):
   * 5, 10, 15, 30, 50 points arranged in straight line, spiral, and three‐cluster patterns.
2. **Execution:**

awns-rpi5 solve

1. **Verification:**
   * Parsed JSON against schema v1.0.
   * Compared plotted tours (graph.zip) by visual inspection to confirm optimal/near-optimal ordering.
   * Recorded elapsed times printed to stdout.

**4.2 Navigation Logging Test**

1. **Data Sets:** Four linearly spaced waypoint files: 5\_line.csv, 10\_line.csv, 15\_line.csv, 30\_line.csv.
2. **Execution:**

awns-rpi5 run

1. **Duration:** Let run until JSON output for final waypoint; total run time tracked via timestamps in logs (log.zip).
2. **Stability Check:** Monitored for crashes, memory‐leaks (via Clang/GCC address sanitizer), and log‐file completeness.

**5. Test Results**

| **Test Campaign** | **Metric** | **Observed Result** | **Pass/Fail** |
| --- | --- | --- | --- |
| **Planner (N≤50)** | Correct route output | All 25 tours valid (visual) | Pass |
|  | JSON schema conformance | 100 % fields present | Pass |
|  | Solve time (max of 25 runs) | ≤ 0.85 s | Pass |
| **Malformed NMEA Handling** | Crash rate under fuzz inputs | 0 crashes after validation | Pass |
| **Logging Stability** | Continuous run time | 30 min for 5, 10, 15, 30 pts | Pass |
|  | Completion times (5→6 min; 10→14 min; 15→26 min; 30→60 min) | As expected | Pass |
|  | Memory/leaks | No leaks detected by address sanitizer | Pass |

**6. Lessons Learned & Next Steps**

* **Lessons Learned:**
  + Early integration of fuzz testing for NMEA parsing prevented late-stage crashes.
  + Creating a CI/CD container environment eliminated “it works on my machine” issues.
  + Field trials revealed minor clock drift between Pi system time and GPS timestamps—must NTP‐sync before each run.
  + Team Collaboration: Working with my teammates taught me the critical importance of clear, early communication and expectations alignment—unexpected last-minute changes can derail progress.
  + Documentation Discipline: Tackling tasks solo underscored the need to rigorously document the codebase, ensuring that future enhancements or feature additions are straightforward for any developer as a part of software maintainability and scalability.
* **Next Steps:**
  1. Add automated NTP sync at program startup.
  2. Extend tests to include simulated packet-loss and NMEA jitter scenarios.
  3. Draft a formal Security Test Plan per IEEE 829-2020 before the next release.
  4. GUI & Refactor: Implement a graphical user interface and refactor the existing CLI into callable class methods to improve usability and modularity.
     + Notes: If refactoring the codebase becomes necessary, a recommendation is to re-use the `GPSClient` and `ConcordeTSPSolver` classes as-is and look to the `Navigator` class on how to use `GPSClient` and `ConcordeTSPSolver`. Otherwise, the `Navigator` class can be minimally refactored for a GUI development approach with the modification of the CLI functionality changed into callable class methods.
  5. Scenario File Support: Add functionality for embedding GPS waypoints directly into files, allowing the system to load and execute custom navigation “scenes” at runtime.
     + Notes: Dynamically adding waypoints to a tour would require extending the `Navigator` class or to make a `WaypointManager` class of sorts. As-is, the system reads waypoints in one go from a CSV file, and for this new functionality, the system could either maintain a dynamic list of waypoints in-memory within the `Navigator` class, or a `WaypointManager` class could be implemented to manage a CSV file behind the scenes that the system would then read in later as normal.

**7. Attachments & References**

* **Attachments:**
  + csv.zip — Waypoint input data sets
  + graph.zip — Generated tour plots
  + log.zip — Run logs with timestamps
* **References:**
  + ENGR 498B Capstone SOW
  + IEEE 829-2020 “Standard for Software and System Test Documentation”
  + GitHub repository: <https://github.com/kimsh02/awns-rpi5>
  + GitLab mirror (for CI/CD): https://git.def.engr.arizona.edu/tkim1/awns-rpi5