LO4: Evaluate the limitations of a given testing process, using statistical methods where appropriate, and summarise outcomes

4.a. Identifying gaps and omissions in the testing process

Although the project has a strong test suite, there are gaps between what the requirements specify and what testing can establish with certainty. Key limitations and omissions are:

1. Branch-heavy logic not fully exercised. IntelliJ shows strong line coverage overall, but branch coverage is lower (overall 64.5% branch vs 87.2% line). This indicates that considerable conditional logic still has paths that were never executed by tests. The risk is that faults hidden behind unexecuted decision paths remain undetected.

2. Limited realism of external-service testing. For FR6 and related system planning, external ILP behaviour is mostly tested using mocks or fixed local JSON. This increases repeatability but reduces realism.

3. Performance evidence is environment dependent. The MR3 timing test is useful as a regression indicator, but it is not a robust performance evaluation. Measured response time depends on the machine's performance rather than the program's efficiency; as a result, a single execution time threshold cannot guarantee production performance.

4. The current non-functional tests do not provide statistical guarantees. For example, one run of a performance test does not establish that the system meets the requirement with high probability under typical work.

5. Incomplete negative testing for planner failure modes. The planner pathfinding uses different types of fallbacks (timeouts/interruption/execution exceptions), but these failure modes are difficult to trigger in unit tests without additional scaffolding. This means resilience to interruption/timeouts is not demonstrated.

4.b. Identifying target coverage/performance levels for different testing procedures

Targets should reflect what would be realistic in a production setting, while recognising that the project is constrained by time and complexity.

Overall coverage targets. Because decision logic is significant in this system (queries, validation, and planning), the targets focus on both line and branch coverage:
Line coverage ≥ 85%
Branch coverage ≥ 70%
As line coverage ensures breadth, and branch coverage better reflects adequacy for decision-heavy logic.

Service-layer coverage target
Line coverage ≥ 90%
Branch coverage ≥ 70%
Since this package contains the highest-risk behaviour (drone selection and planning). It benefits most from branch-focused testing.

Non-functional target (MR3 response time).
Because timing varies per environment, the target should be framed statistically rather than as a single execution:

Performance target: median response time ≤ 2s for a typical payload on my machine, with an additional percentile target if possible.
Since the median is robust to outliers, using percentiles provides a clearer view of worst-case behaviour.

4.c. Comparing achieved testing outcomes with target levels

Overall Coverage achieved (from IntelliJ Coverage):
Overall: 87.2% line (780/894) and 64.5% branch (329/510)

Line coverage meets the overall target (≥85%).
Branch coverage falls short of the suggested target (≥70%).

This suggests good breadth of execution, but with remaining untested decision logic.

Service-level coverage achieved:
Service: 89.3% line (401/449) and 68.1% branch (139/204)

Line coverage is close to the 90% target but slightly below.
Branch coverage is close to the 70% target but slightly below.

This indicates the most complex logic is substantially exercised, but there are still uncovered conditional paths.

Performance targets achieved
The MR3 timing test can be made to pass with a realistic threshold, but it should be interpreted as a simple test rather than a guarantee. Without repeated runs and statistical testing, the test cannot strongly support a strict time bound under realistic work.

4.d. What would be necessary to achieve or exceed the target levels

The following improvements would most effectively be necessary to close the gap between achieved results and the targets.

1. To raise branch coverage, the most valuable work is to deliberately generate tests that cover:
   - Rare edge case combinations
   - empty lists or null handling paths
   - Multiple drone assignment cases could be supported by decision-table based test design to ensure each combination of constraint validity is covered.

2. Replace a single threshold check with a statistical measure after many runs:
   - run the endpoint 100 times
   - compute median, mean, and a percentile

   This would allow MR3 to be evaluated more credibly and would follow the course's emphasis on statistical reasoning where appropriate.

3. To cover fallback handling properly, introduce scaffolding that allows the pathfinding call to be substituted, then tests can force an exception to reliably cover fallback branches.
   This would directly increase branch coverage and confidence in robustness behaviour.

4. Consider mutation testing for residual fault estimation. This could estimate how effective the tests are at detecting likely faults in geometry logic. This would strengthen the argument about residual faults left behind beyond coverage metrics.