Limitations

Gaps and Omissions

In **Test Cases** and **Testing Document**, extensive testing was done using various testing techniques to optimise test coverage. However, despite implementing functional, structural, and integration testing, several limitations remain in the testing process.

For R2 (Flight Path Generation) and R3 (Drone Movement), the correctness for flight path generation cannot be fully validated through automated tests. Manual visual inspections of the flight paths via the GeoJson website are still required to determine the adherence to multiple of the navigational and geometrical requirements. This introduces subjectivity in the strictness of adherence to the ILP guidelines as well as limiting the repeatability of the tests. Any changes made to the flight path generation algorithm, geometric constraints, or many other interconnected parts of the system would require repeating the tests for every restaurant and boundary condition which would be a large undertaking, costing time and resources, due to the need for manual visual inspections. While this is the most crucial limitation in the testing process, it is also the one which would require the most resources to fix which as the sole developer of this project I do not have.

Another omission in the testing process is the lack of no fault-based or mutation testing which without we cannot statistically estimate the effectiveness of the test suite in finding unknown or unexpected faults. Scalability and performance testing was very limited as the ILP specification restricts the drone to a small area within which there are no-fly zones and restaurants. This means that even if 100 restaurants within the area were used to test the flight path generation it would not exhaustively test the system as many of the paths within this small area would overlap. This might also give us the mistaken impression that our algorithm for finding the shortest path is faster than it truly is as a similar flight path can be used for multiple restaurants.

Target Coverage

Target coverage for the system as a whole and each individual component is 100% even if it may not be feasible with the system and specification requirements as they are, due to the nature of this being a simplified simulation coursework. I use the IntelliJ test coverage feature to get the following results:

Component	Responsibility	Class Coverage	Method Coverage	Line Coverage	Branch Coverage	Evaluation
BasicController	Core order validation, navigation, routing logic	100%	10% (3/30)	21% (78/358)	16% (41/242)	Low structural coverage due to short-circuit validation and mutually exclusive error paths;

						functional coverage is complete
OrderValidationCode	Enumeration of all validation outcomes	100%	100%	100%	N/A	Fully exercised by integration tests
OrderValidationResult	Validation result container	100%	50%	33%	N/A	Partial coverage is acceptable as getters/setters are non-critical
RestaurantData	Static restaurant/menu data	100%	100%	100%	N/A	Fully tested indirectly by validation tests
Restaurant	Restaurant entity logic	50%	100%	100%	N/A	Constructor-only paths remain uncovered; low risk
LngLat / Geometry logic	Distance, proximity, movement primitives	~57%	~40%	84%	N/A	High line coverage for numerical logic; branch coverage not applicable
Service Layer (DefaultOrderValidator, etc.)	Delegated validation logic	50%	20%	3%	0%	Under-tested structurally; logic primarily tested via controller integration

The coverage results show that although overall structural coverage is moderate, the testing achieved strong functional adequacy for the ILP requirements. The BasicController reached full class coverage but lower method, line, and branch coverage due to its reliance on short-circuit validation and early returns, which make many branches difficult to exercise.

Order validation achieved high functional confidence, with all validation outcomes fully exercised. Lower coverage in result and data-holder classes is expected due some untested accessor methods, which are not critical to correctness. Geometric and movement components achieved high line coverage, indicating thorough testing of numerical logic.

Overall, the tests successfully identified several real defects, demonstrating that requirement-focused testing was effective despite modest structural coverage.

Target Performance Levels

Using the times given for each test in the **Test Cases** document, the performance levels of the individual endpoints as well as the entire system can measured and approximated. Performance levels were derived from test execution time by running test classes in IntelliJ and using the Junit runner riming output.

System name	Target performance level	Actual performance level
Order validation (individual tests)	<25ms	4-20ms
Order validation test suite	<3s	2.04s
Flight path generation in GeoJson format	<60ms	1st path: 1.70s Subsequent: 15-62ms
Drone movement induvial tests	<25ms	3.5-30ms
Entire system	<0.4 seconds	0.5 seconds

Comparing to Target Levels

Order validation comfortably meets its performance targets, with all individual tests executing well below 25ms and the full validation suite completing within the 3-second target. Drone movement and geometric operations are also lightweight, with most tests completing in under 10ms, indicating that low-level mathematical operations are not a performance bottleneck. However, in the test for calculating the next position due North it takes 49ms much higher than when calculating in the East direction which takes 9ms. This is expected as the North calculation comes earlier in the test run and therefore needs time to load the classes needed.

Flight path generation shows higher variance. While subsequent path calculations complete within the expected 15–62ms range, the first generated path incurs a significantly higher cost of 1.7 s. This reflects the greedy search algorithm exploring spatial constraints such as no-fly zones and the Central Area for the first time.

The measured end-to-end system runtime of 0.5s slightly exceeds the initial target of 0.4 s. This deviation is due to Spring test context start-up, JSON serialisation, and path generation overhead rather than inefficiencies in core logic.

Achieving Target Levels

In this section I will describe the efforts needed to achieve the target performance and coverage levels. These were not pursued by me due to limited resources as the sole developer on the project and the need to prioritise other requirements.

For order validation and drone movement, further optimisation is unlikely to yield meaningful gains, as measured execution times already fall well below target thresholds and are dominated by framework and test harness overhead rather than core logic.

The target of 100% requirement coverage is nearly achieved for order validation but extending this level to flight path generation and drone movement would require additional automated tests. Spatial properties such as no-fly zone avoidance and central area containment currently rely on manual visual inspection. Achieving full functional coverage would require formal geometric assertions, such as region transition detection, which would significantly increase implementation complexity. To increase white-box coverage, many of the methods I have been using in the controller class to check and validate coordinates and coordinate closeness would need to be broken down into unit tests.

For flight path generation, achieving consistently under 60 ms performance for the first computed path would require algorithmic changes. These include introducing path caching for repeated paths between the same origin and destination, adding heuristic pruning to reduce the greedy search space near no-fly zone boundaries, and reducing geometric precision checks where safety margins already exist. The most noticeable improvement would be changing the flight generation algorithm from greedy to something much faster as an A* search.

At the system level, meeting the original 0.4 s end-to-end target would require isolating performance measurements from Spring test context startup, minimising JSON serialisation overhead, and executing tests in a warmed JVM environment. Due to this mainly being an issue with the Spring/JVM environment, it may require switching to another testing software. In a professional workplace setting, this effort may be considered worthwhile if it is a specification requirement for the system otherwise it may be considered an acceptable performance level.