Test Cases

R1: Order Validation

Categories	Requirement	Test cases	ID
CVV Length CVV Value	3 numeric	Valid: 3 numbers (CVV = 123) Invalid: length 2, numeric (CVV = 12) length 3, all letters (CVV = abc) Boundary (Invalid): length 3, 2 numbers followed by a letter (CVV = 12a)	FR-OV-03
Expiry Date	Not before 01/26, MM/YY format	Valid: Expiry date = 01/27 Boundary (Valid): Expiry date = 01/26 Invalid: Expiry date = 01/25 (expired) Expiry date = 01 (wrong format)	FR-OV-01

Card Number	Length 16	Valid:
		CN = 1234567890123456
		Invalid:
		CN = 12345678901234567 (16 digits)
		CN = 123456789012345a
		(Letters included)
			FR-OV-02

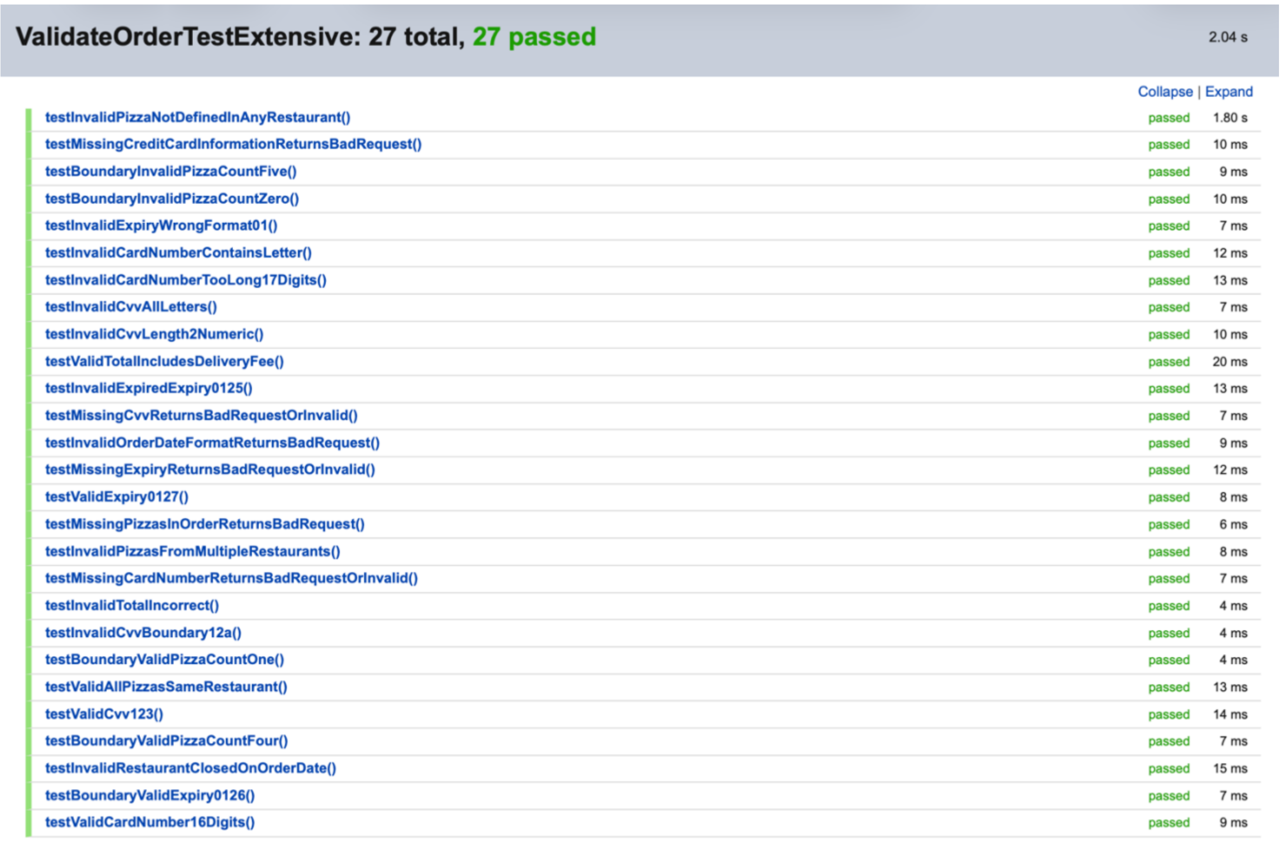
Order cost	Added pizza price plus delivery fee	Valid: Correct total plus delivery fee Invalid: Incorrect total	FR-OV-08
Order date	Date before 01-01-2026, Restaurant must be open	Valid order date Invalid: Order date invalid Restaurant closed	FR-OV-09

Pizza count	Between 1 and 4	Boundary (Valid):
		C = 1
		C = 4
		Boundary (Invalid):
		C = 0
		C = 5
			FR-OV-04
			FR-OV-05

Source of pizza	All pizzas come from the same restaurant	Valid: All pizzas from one restaurant Invalid:	

		Pizza not defined in any restaurants Pizzas from multiple restaurants	FR-OV-06 FR-OV-07

Results and Evaluation



ValidateOrderTestExtensive: 27 total, 27 passed — 2.04 s

testInvalidPizzaNotDefinedInAnyRestaurant()	passed	1.80 s
testMissingCreditCardInformationReturnsBadRequest()	passed	10 ms
testBoundaryInvalidPizzaCountFive()	passed	9 ms
testBoundaryInvalidPizzaCountZero()	passed	10 ms
testInvalidExpiryWrongFormat01()	passed	7 ms
testInvalidCardNumberContainsLetter()	passed	12 ms
testInvalidCardNumberTooLong17Digits()	passed	13 ms
testInvalidCvvAllLetters()	passed	7 ms
testInvalidCvvLength2Numeric()	passed	10 ms
testValidTotalIncludesDeliveryFee()	passed	20 ms
testInvalidExpiredExpiry0125()	passed	13 ms
testMissingCvvReturnsBadRequestOrInvalid()	passed	7 ms
testInvalidOrderDateFormatReturnsBadRequest()	passed	9 ms
testMissingExpiryReturnsBadRequestOrInvalid()	passed	12 ms
testValidExpiry0127()	passed	8 ms
testMissingPizzasInOrderReturnsBadRequest()	passed	6 ms
testInvalidPizzasFromMultipleRestaurants()	passed	8 ms
testMissingCardNumberReturnsBadRequestOrInvalid()	passed	7 ms
testInvalidTotalIncorrect()	passed	4 ms
testInvalidCvvBoundary12a()	passed	4 ms
testBoundaryValidPizzaCountOne()	passed	4 ms
testValidAllPizzasSameRestaurant()	passed	13 ms
testValidCvv123()	passed	14 ms
testBoundaryValidPizzaCountFour()	passed	7 ms
testInvalidRestaurantClosedOnOrderDate()	passed	15 ms
testBoundaryValidExpiry0126()	passed	7 ms
testValidCardNumber16Digits()	passed	9 ms

The tests uncovered several errors in the logic and implementation of the code. The delivery charge wasn't being added correctly to the total of the pizzas resulting in several false classifications of valid orders and misclassifications of the true cause of an invalid order. This error was easy to fix once found. Testing the border case for when the expiry date is in the same month of the order revealed that the system considered that invalid when it should actually be valid as the card won't expire till the end of the month. Once identified, the issue was resolved by validating expiry dates at month granularity rather than using a raw date comparison.

Overall, the results indicate high effectiveness of functional and boundary testing for this requirement. Each validation rule was exercised independently, mirroring the ILP auto-marker's behaviour and ensuring strong fault localisation. After fixes were applied, all validation codes were triggered exactly once by their corresponding test cases, providing confidence that the implementation aligns with the specification.

Residual risk remains low for order validation, as the logic is deterministic and fully exercised across all specified equivalence classes. Any remaining defects are likely to relate to malformed input beyond the specification rather than missing validation rules.

R2: Flight Path Generation

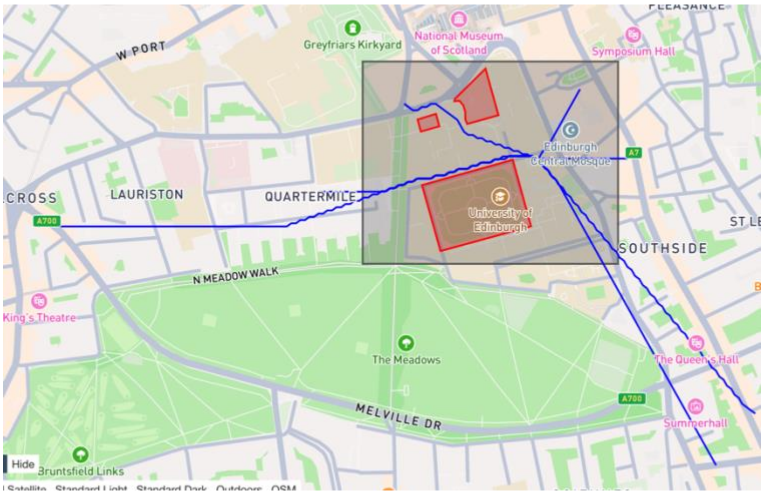
For the following requirements, they will all be tested together in one test which will generate the flight path of 10 unique valid orders.

FR-N-04: The system will avoid entering no-fly zones
FR-N-05: Once the drone enters the Central Area, it will not leave it again

FR-N-07: For valid orders, the system will return a sequence of positions representing the delivery path

This requirement was tested separately as well as in the tests above indirectly. In the test class CalcDeliveryPathTest, I test that given a valid order the system returns a path for each of the 7 restaurants given in the ILP coursework.

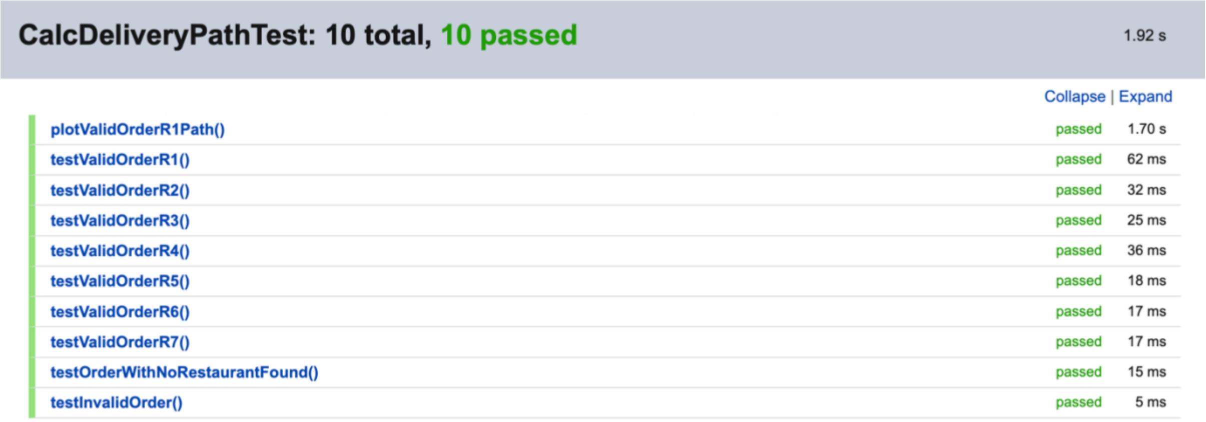
Results and Evaluation
FR-N-04 and FR-N-05



Through the inspection of this image generated by the geojson.io website, it can be seen that none of the paths enter the no-fly zones specified by the ILP specification nor does it leave the Central Area once it has entered it. FR-N-07 is satisfied as the sequence of positions returned were then put into geojson.io to create this image.

Tests were executed using the Spring test context to ensure realistic request handling and response generation.

FR-N-07



CalcDeliveryPathTest: 10 total, 10 passed 1.92 s

Collapse | Expand

plotValidOrderR1Path()	passed	1.70 s
testValidOrderR1()	passed	62 ms
testValidOrderR2()	passed	32 ms
testValidOrderR3()	passed	25 ms
testValidOrderR4()	passed	36 ms
testValidOrderR5()	passed	18 ms
testValidOrderR6()	passed	17 ms
testValidOrderR7()	passed	17 ms
testOrderWithNoRestaurantFound()	passed	15 ms
testInvalidOrder()	passed	5 ms

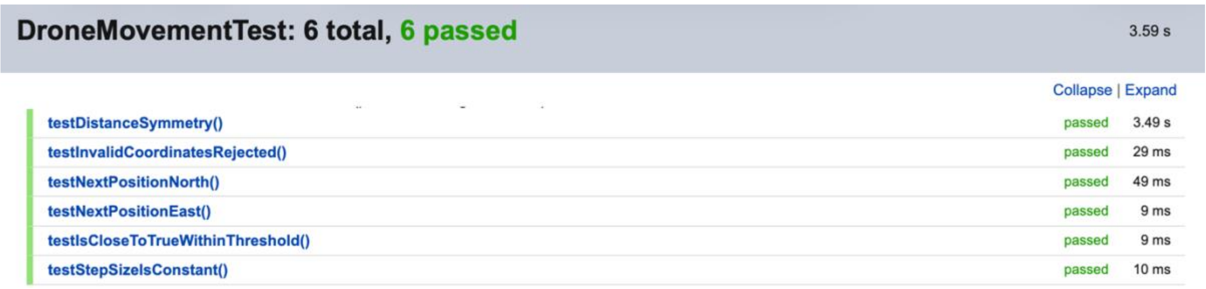
During testing I realised that my method for detecting no-fly zones didn't account for if the path between two points outside of the no-fly zones cuts across one which resulted in the drone cutting through the corners of the zones. To fix this issue I increased the proximity threshold so that the path would be further from the zones.

Valid orders consistently produced delivery paths, indicating correct system-level behaviour. Confidence is high, with residual risk primarily related to floating-point precision and geometric edge cases.

R3: Drone Movement

Requirement	Test Level
FR-N-01	Controller unit test
FR-N-02	Controller unit test
UT-GEO-01	Controller unit test
UT-GEO-02	Controller unit test
UT-GEO-03	Controller unit test
FR-N-03	Integration / system

Most of the tests for drone movement are implemented as methods within my controller class as they are low level and deterministic. For this reason, they can't be tested explicitly as unit tests very effectively. I did however test for consistency in the DroneMovementTest test class.



DroneMovementTest: 6 total, 6 passed 3.59 s

testDistanceSymmetry()	passed	3.49 s
testInvalidCoordinatesRejected()	passed	29 ms
testNextPositionNorth()	passed	49 ms
testNextPositionEast()	passed	9 ms
testIsCloseToTrueWithinThreshold()	passed	9 ms
testStepSizeIsConstant()	passed	10 ms

Results and Evaluation
As the requirements for drone movement are not difficult to implement and are straightforward to understand, further testing did not raise any errors or gaps in the code. These tests serve to check and assure us that the fundamental mathematical considerations of the system are correctly implemented.

Hover behaviour was validated indirectly through integration tests and manual inspection of generated paths. Overall confidence is high, with minimal residual risk, as errors in this area would likely surface immediately during higher-level testing.