LO3 Support Document (Testing Evidence Document)

3.1 Range of techniques

Requirement 1: validation of drone route.

The test is divided into code test and actual scene test. We do not discuss actual scenario testing here. The document proposes three testing approaches to code testing: Functional Testing, combinatorial testing, and model-based testing. In the actual code test, first I created the "LineCrossForTest" method in Scaffolding to test whether the function of judging the intersection of line segments is correct (Functional Testing). For combinatorial testing, in order to validate the interactions between different factors that might affect the validation of the drone, NoFlyZoneTest and ConfinementAreaTest are tested separately first to ensure the functionality of identifying no-fly zones and confinement area works properly, and then a integration test are generated using different no-fly zones, confinement areas and delivery orders. For model-based testing, many test cases are generated based on different dates and different starting positions to verify the output of the algorithm.

Requirement 2: Return to charging place when battery is low.

The test is also divided into code test and real scenario test. We do not discuss real scenarios testing here. I implemented the functional testing to verify that the drone's battery checking mechanism is working correctly and it can accurately determine the remaining battery level after each move. The model-based testing is generated to simulate different drone's situation when battery is low. For instance, when drone is moving, when drone is picking up the food, when drone is delivering the food, and when drone is switching to next target position.

In summary, the actual testing was implemented strictly followed the testing plan and most of the scenarios are included in the test. Also, the stress testing is implemented to ensure the path finding algorithms can function properly for a period of time which this is not mentioned in the test document but is included in the testing code.

3.2 Evaluation criteria for the adequacy of the testing

I did not take the code coverage as the target levels for testing since I think using appropriate testing approaches for each specific requirement is more suitable than looking at the code coverage.

In R1, the **functional testing** could ensure that the pathfinding algorithm can properly judge whether going from one point to another will pass through the no-fly zone and confinement area. This can provide the most basic guarantee for later integration testing and system testing, such as testing the entire pathfinding algorithm with different confinement area and no-fly zones and different order orders. So, if there is a problem in the subsequent system testing or integration testing, then we will ignore the problem of each functional unit itself but focus on the interaction between each functional unit. By testing different combinations of inputs and conditions, **combinatorial testing** can provide a more comprehensive understanding of the system's behavior, including the interactions between different components and the effects of different environmental conditions. This can lead to the discovery of potential issues that would not be identified through traditional testing methods, such as edge cases or corner cases that involve unexpected combinations of inputs or conditions. The advantage of using **model-based testing** for drone route validation is that it allows for more comprehensive and automated testing. By using a model of the system, test cases can be generated systematically and exhaustively, covering a wide range of scenarios and edge cases. This can help to identify potential issues and ensure that the drone delivery system operates safely and reliably. For instance, verifying the software's reliability with different order dates, numbers of orders, different delays in obtaining data from web services caused by different network latencies. In summary, evaluation measurement is to using a combination of functional testing, combinatorial testing and model-based testing to make it more confident on this requirement

In R2, **the functional testing** includes the low battery detection testing and battery counting testing. This ensures the battery basic functionality works correctly and later it will be integrated with the path test returning to the charging place. If there is a problem, then there is no need to consider the problem of a separate code block, only the interaction between the focus and the code block. **The model-based testing** can ensure to improve the efficiency and speed of testing. By automating the generation of test cases, it reduces the time and effort required to manually create and execute tests. This can result in faster and more accurate testing, as well as reducing the risk of human error in the testing process. For example, consider a drone delivery system that is equipped with a battery level monitoring system and is required to return to the charging place when the battery level drops below 20%. Using model-based testing, a model of the system can be created that includes the battery level monitoring system, the charging place, and the drone's navigation system. The model can then be used to generate test cases that simulate different battery levels, distances to the charging place, and environmental conditions.

3.3 Results of testing (Test Log results)

Requirement 1: Validation of Drone Route

Path Validation Test (Line Cross Test): TRUE for cross, FLASE for not cross

Test Case	Input	Expect Output	Result

LineCrossTest1	Point1(-3.1867,55.9447) Point2(-3.1863, 55.9431)	FALSE	PASS
LineCrossTest2	Point1(3.1852,55.9457) Point2 (3.1908, 55.9431)	FALSE	PASS
LineCrossTest3	Point1(3.1870,55.9445) Point2(3.1898, 55.9449)	FALSE	PASS
LineCrossTest4	Point1(3.1902,55.9419) Point2(3.1908, 55.9431)	TRUE	PASS
LineCrossTest5	Point1(3.1902,55.9419) Point2(3.1868, 55.9466);	TRUE	PASS
LineCrossTest6	Point1(3.1907,55.9423) Point2(3.1878, 55.9468)	TRUE	PASS

Path Validation Test (No-Fly Zone Test): TRUE for cross, FLASE for not cross

Test Case	Input	Expect Output	Result
NoFlyZoneTest1	Point1(-3.1888,55.9454) Point2(-3.1881,55.9440)	TRUE	PASS
NoFlyZoneTest2	Point1(-3.1867,55.9447) Point2(-3.1863, 55.9431)	FALSE	PASS
NoFlyZoneTest3	Point1(-3.1870,55.9445) Point2(-3.1898, 55.9449)	TRUE	PASS
NoFlyZoneTest4	Point1(-3.1902,55.9419) Point2(-3.1908, 55.9431)	FALSE	PASS
NoFlyZoneTest5	Point1(-3.1902,55.9419) Point2(-3.1868, 55.9466)	TRUE	PASS
NoFlyZoneTest6	Point1(-3.1907,55.9423) Point2(-3.1878, 55.9468)	TRUE	PASS
NoFlyZoneTest7	Point1(-3.1852,55.9457) Point2(-3.1908, 55.9431)	FALSE	PASS
NoFlyZoneTest8	Point1(-3.1889,55.9451) Point2(-3.1890, 55.9442)	TRUE	PASS

Path validation Test (Confinement Area Test): TRUE for cross, FLASE for not cross

Test Case	Input	Expect Output	Result
ConfinementAreaTest1	Point1(-3.1888,55.9454) Point2(-3.1881,55.9440)	TRUE	PASS
ConfinementAreaTest2	Point1(-3.1867,55.9447) Point2(-3.1863, 55.9431)	TRUE	PASS
ConfinementAreaTest3	Point1(-3.1870,55.9445) Point2(-3.1898, 55.9449)	TRUE	PASS
ConfinementAreaTest4	Point1(-3.1902,55.9419) Point2(-3.1908, 55.9431)	TRUE	PASS
ConfinementAreaTest5	Point1(-3.1902,55.9419)	TRUE	PASS

	Point2(-3.1868, 55.9466)		
ConfinementAreaTest6	**Point1(-3.1907,55.9423)** **Point2(-3.1878, 55.9468)**	**FALSE**	**PASS**
ConfinementAreaTest7	**Point1(-3.1852,55.9457)** **Point2(-3.1908, 55.9431)**	**FALSE**	**PASS**
ConfinementAreaTest8	**Point1(-3.1889,55.9451)** **Point2(-3.1890, 55.9442)**	**TRUE**	**PASS**

Path validation Test (Integration Test)

Test Case	**Input**	**Expect Output**	**Result**
IntegrationTest1	**Point1(-3.1888,55.9454)** **Point2(-3.1881,55.9440)** **Point3(-3.1888,55.9454)** **Point4(-3.1881,55.9440)**	**TRUE**	**PASS**
IntegrationTest2	**Point1(-3.1867,55.9447)** **Point2(-3.1863, 55.9431)** **Point3(-3.1867,55.9447)** **Point4(-3.1863, 55.9431)**	**TRUE**	**PASS**
IntegrationTest3	**Point1(-3.1870,55.9445)** **Point2(-3.1898, 55.9449)** **Point3(-3.1870,55.9445)** **Point4(-3.1898, 55.9449)**	**TRUE**	**PASS**
IntegrationTest4	**Point1(-3.1902,55.9419)** **Point2(-3.1868, 55.9466)** **Point3(-3.1902,55.9419)** **Point4(-3.1868, 55.9466)**	**TRUE**	**PASS**
IntegrationTest5	**Point1(-4.1888,55.9454)** **Point2(-3.1881,55.9440)** **Point3(-4.1888,55.9454)** **Point4(-3.1881,55.9440)**	**TRUE**	**PASS**
IntegrationTest6	**Point1(-3.1888,56.9454)** **Point2(-4.1881,55.9440)** **Point3(-3.1888,56.9454)** **Point4(-4.1881,55.9440)**	**FALSE**	**PASS**

Navigation Test: FALSE for all flight path is valid

Test Case	**Input**	**Expect Output**	**Result**
navigationTest1	**Dates: 2023-06-22**	**FALSE**	**PASS**
navigationTest2	**Dates: 2023-12-22**	**FALSE**	**PASS**
navigationTest3	**Dates: 2023-12-31**	**FALSE**	**PASS**
navigationTest4	**Dates: 2022-06-27**	**FALSE**	**PASS**
navigationTest5	**Dates: 2022-02-28**	**FALSE**	**PASS**
navigationTest6	**Dates: 2023-11-03**	**FALSE**	**PASS**

Performance Test:

Test Case	Input	Expect Output	Result
performanceTest1	Dates: 2023-11-23 Battery: 2000	Time consumption<10000ms	PASS
performanceTest2	Dates: 2023-12-23 Battery: 2000	Time consumption<6000ms	PASS

Web Server Time Test:

Test Case	Input	Expect Output	Result
responseTimeTestForWebServer1	2023-11-03	Time consumption<500ms	PASS
responseTimeTestForWebServer2	2023-12-01	Time consumption<300ms	PASS
responseTimeTestForWebServer3	2023-12-31	Time consumption<100ms	FAIL

Stress Test:

Test Case	Input	Expect Output	Result
StressTesting1	Point (-3.1628,50.9177) Battery:150000	Algorithm do not crash	PASS
StressTesting2	Point (-5.2222,55.9177) Battery:800000	Algorithm do not crash	PASS
StressTesting3	Point (-0.1628,55.9177) Battery:60000	Algorithm do not crash	PASS
StressTesting4	Point (-3.1628,59.9177) Battery:80000	Algorithm do not crash	PASS

Requirement 2: Return to Charging place when battery is low.

Battey Functional Test:

Test Case	Input	Expect Output	Result
functionalTesting1	Battery: 600	Point (-3.1868, 55.9445) ±0.0015	PASS
functionalTesting2	Battery: 400	Point (-3.1868, 55.9445) ±0.0015	PASS
functionalTesting3	Battery: 300	Point (-3.1868, 55.9445) ±0.0015	PASS
functionalTesting4	Battery: 200	Point (-3.1868, 55.9445) ±0.0015	PASS
functionalTesting5	Battery: 250	Point (-3.1868, 55.9445) ±0.0015	PASS
functionalTesting6	Battery: 100	Point (-3.1868, 55.9445) ±0.0015	PASS
functionalTesting7	Battery: 50	Point (-3.1868, 55.9445) ±0.0015	PASS

Return Test:

Test Case	Input	Expect Output	Result
ReturnDetectionTest1	Low Battery: 1000	Target position changed to AT	PASS
ReturnDetectionTest2	Low Battery: 500	Target position changed to AT	PASS
ReturnDetectionTest3	Low Battery: 200	Target position changed to AT	PASS
ReturnDetectionTest4	Low Battery: 100	Target position changed to AT	PASS
ReturnDetectionTest5	Low Battery: 50	Target position changed to AT	PASS
ReturnDetectionTest6	Low Battery: 80000	Target position changed to AT	PASS

3.4 Evaluation of the results

- ### Navigation Test

  Result: Six different dates with different no-fly zones and confinement area (combinatorial testing). The software passes all six tests.

  Analysis: By testing different input combinations (combinatorial testing), we can ensure that the drone system can guarantee a valid route under different no-fly zones and within different confinement areas. While enhancing the robustness of the system to restricted areas, it also preliminarily verified the order sequence.

- ### Path Validation Test

  Results: For path validation test, 4 tests including LineCrossTest, NoFlyZoneTest, ConfinementAreaTest and IntegrationTest (combining no-fly zone test and confinement area test with different order sequence). Each test has different inputs for different cases. For instance, When the route coincides with the boundary of the no-fly zone, or the destination of a move happens to just be on the line segment of the no-fly zone or confinement area. The software passes all the sub-tests for these 4 tests with at least 6 test cases.

  Analysis: The line cross test is a unit test, and the rest 3 tests are integration test (Since it needs to interact with web server and database). All test cases passing these 4 different tests can guarantee to a certain extent that drone can guarantee the legitimacy of the route in path planning. The insufficient of the testing is that the testing cases is not enough to ensure that some edge cases would not result a problem in a real-world test.

- ### Performance Test

  Results: Two test cases are passed with different order sequences and Time limit. performanceTest1 and performanceTest2.

  Analysis: The test performance is mean to test the least time to finish all the orders given enough battery upper limit. Since for some date, the drone cannot finish all the orders given the standard battery limit (1500). This test could ensure whether the total time of the algorithm can be maintained under a certain level. 10000ms and 6000ms are tested. A better performance test is to test whether the drone algorithm can run all the orders and return the results within a given period, or whether the drone algorithm can be terminated early and return the corresponding results.

- ### Server Test

  Results: Two test cases are passed, and one test cases is failed.

  Analysis: The server test is mean to verify the time cost when the drone application is trying to get the data from web server. 3 standards are given in different dates (500, 300, 100). Because if the drone algorithm takes too long to obtain order information and delivery information from the web server, it is very likely to lead to a decline in overall performance, so it is necessary to fetch information with less time cost.

- ### Stress Test

  Results: Four Stress Test are given to test the durability of the drone system and all test are passed.

  Analysis: The stress test only tests the durability path finding algorithms by setting the starting point to a place where it is very far away from the order target positions. This stress test is inadequate since only durability is tested. To fully cover every unfunctional requirements, we need to test the reliability, security, Scalability and Cost-effectiveness, etc.

- ### Battery Functional Test

  Results: Seven test cases for testing the low battery detection functionality are generated and all tests are passed.

  Analysis: Test whether low battery power can be detected in different situations by using the battery to drain power with the shipping order at different initial power levels.

- ### Return Test

  Results: Six test cases for testing the return to AT when battery is low. All six test cases are passed.

  Analysist: Inputs to return tests have different dates, different battery capacities. The return test is lack of considering some edge cases. For instance, the position of the aircraft is at the farthest distance from the low battery of the aircraft, and the aircraft needs an extra battery when it finally lands. This situation has not been taken into account.