

Project Title: System Verification and Validation Plan for Mechatronics Engineering

Team # 34, ParkingLotHawk

Fady Zekry Hanna

Winnie Trandinh

Muhammad Ali

Muhammad Khan

November 3, 2022

1 Revision History

Table 1: Revision History

Date	Version	Notes
November 3, 2022	1.0	Initial Revision

Contents

1	Revision History	i
2	Symbols, Abbreviations and Acronyms	iv
3	General Information	1
3.1	Summary	1
3.2	Objectives	1
3.3	Relevant Documentation	2
4	Plan	2
4.1	Verification and Validation Team	2
4.2	SRS Verification Plan	3
4.3	Design Verification Plan	4
4.4	Verification and Validation Plan Verification Plan	7
4.5	Implementation Verification Plan	8
4.6	Automated Testing and Verification Tools	8
4.7	Software Validation Plan	8
5	System Test Description	11
5.1	Measuring Outputs Testing	11
5.2	Assumptions Regarding Test Cases	11
5.3	Tests for Functional Requirements	11
5.3.1	Flight Dynamics	11
5.3.2	Battery	16
5.3.3	Communication	19
5.3.4	User Error States	21
5.3.5	Visual Perception and Path Planning	24
5.4	Traceability between Test Cases and Functional Requirements	28
5.5	Tests for Nonfunctional Requirements	29
5.5.1	User Manual	29
5.5.2	Usability	31
5.6	Traceability between Test Cases and Non-Functional Requirements	33
6	Unit Test Description	35
6.1	Unit Testing Scope	35
6.2	Tests for Functional Requirements	35
6.2.1	Module 1	35
6.2.2	Module 2	36
6.3	Tests for Nonfunctional Requirements	36
6.3.1	Module ?	36
6.4	Traceability Between Test Cases and Modules	36

7	Appendix	37
7.1	Symbolic Parameters	37
7.2	Reflection	38
7.2.1	Knowledge & Skills	38

List of Tables

1	Revision History	i
2	Verification and Validation Team	2
3	STC_001	12
4	STC_002	13
5	STC_003	14
6	STC_004	15
7	STC_005	16
8	STC_006	17
9	STC_007	18
10	STC_008	19
11	STC_009	19
12	STC_010	20
13	STC_011	22
14	STC_012	23
15	STC_013	24
16	STC_014	25
17	STC_015	26
18	STC_016	27
19	FR Traceability Table	28
20	STC_017	29
21	STC_018	30
22	STC_019	31
23	STC_020	31
24	STC_021	32
25	STC_022	32
26	NFR Traceability Table	33
27	Symbolic Constants	37
28	Required Testing	38
29	Knowledge Acquisition	39

List of Figures

1	Blanderbuss Dataset Sample	9
2	CNRPark Dataset Sample	10

2 Symbols, Abbreviations and Acronyms

See where [Key terms, acronyms, and abbreviations](#) are defined.

3 General Information

The Verification and Validation (VnV) Plan outlines the various methods that the team will conduct to verify and validate the ParkingLotHawk. The general overview of the plan can be found in [Plan](#), with detailed tests outlined within [System Test Description](#) and [Unit Test Description](#).

3.1 Summary

The ParkingLotHawk is an autonomous aerial drone that helps parking lot operators understand the state of their parking lot. The drone shall be able of full autonomy, where it automatically explores the parking lot, or by semi-autonomy, where the operator specifies the locations that the drone should go to. During flight, the drone shall transmit live information about the parking lot sections it detects, and display this information to the operator.

3.2 Objectives

There are multiple objectives to be accomplished for the proper operation of the drone. The following objectives will mainly focus on the most important qualities of the objectives:

- **Software Algorithm Correctness:** The drone should be able to operate according to its specifications, and the application to be used by the operators should be able to accurately relay the commands to the drone.
- **Hardware Correctness:** The components of the drone functions properly according to their intended purpose and with minimal to no errors.
- **State Machine Implementation:** Control events exist and are correct for the various components of the drone in order to operate those specific parts individually.
- **Operator's PC and Drone Communication:** The components on the drone can accurately communicate with the operator application and the communication is functional during the entire time that the drone is functioning within the parking lot.
- **Safety Features:** The drone has a backup or failsafe code in the situation where any of its parts are malfunctioning or prior to the malfunction.
- **Ease of Use:** The operator should be able to properly understand how to operate the drone from the application, under the assumption that they only have basic knowledge of operating computers.

3.3 Relevant Documentation

In the remaining portion of the document, there will be various sources of information that will help verify the process for the product. The major sources will include the reference of the [SRS](#) and [HA](#) documents for the requirements, and the [MG](#) and [MIS](#) documents for the design.

4 Plan

This section outlines the various methods that the team will use to verify and validate the components of the system. This includes verifying the SRS, design, VnV plan, and implementation. Automated testing and verification tools will also be presented to aid in the iterative verification process. Finally, methods for validating that the system solves the problem will be presented.

4.1 Verification and Validation Team

The members of the team are assigned an area of testing, where the area of testing is not their area of expertise. This ensures that the creator and tester are not the same person, which eliminates the bias that occurs when testing their own components. The assigned person to the testing area are then responsible for managing all tests in that domain. In the cases where a test covers multiple domains, the test will be conducted in a joint manner between the different domains. The assignment of the testing areas are indicated within [Verification and Validation Team](#).

Table 2: Verification and Validation Team

Role	Name	Description
Visual Perception and Path Planning	Fady	Verifies that the visual perception and autonomous exploration algorithm is performing within specifications.
Drone Finite State Machine (FSM) and Communication	Ali	Verifies that all communication between the drone components and the drone to the Operator's application are working correctly.
Mechanical Testing	Zaid	Verifies that all physical components and the dynamics are working within specifications.
Operator's Application	Winnie	Verifies that the Operator's application and user manual meet the specifications outlined.

4.2 SRS Verification Plan

To verify the SRS, both formal and informal processes will be conducted. To test the completeness of the FSM, a formal decision table shall be created and maintained. This process ensures that all transitions and external stimuli are accounted for, in addition to identifying any states that cannot be reached or cannot be exited. The initial decision table has already been created within the SRS, located in [Appendix B: Formal Transition Table](#). To verify that the decision table is accurate as the project progresses, this table will be updated with any changes to the FSM.

To informally verify the other components of the SRS, guided reviews and a checklist shall be conducted. The guided review will consist of the team explaining the Functional Requirements to a technical external party, and the external party will list any potential NFRs that are related to the FR. The team shall then ensure that the NFR is present within the SRS. Furthermore, the team shall have other capstone groups review the SRS using the provided rubric as guidelines for the review. The last method is to conduct a checklist, conducted either internally within the team or by an external party. The checklist is as follows:

- ☐ For each requirement within the SRS, are the following met?
 - ☐ Does the requirement address a specific goal?
 - ☐ Is the requirement unique?
 - ☐ Is the requirement abstract?
 - ☐ Is the requirement traceable?
 - ☐ Is the requirement complete?
 - ☐ Is the requirement measurable?
 - ☐ Are there no inputs that are not used in the determination of the output?
 - ☐ Does each output use at least one input, and are all required inputs listed?
- ☐ Does the Finite State Machine comply with the following?
 - ☐ Are all states unique?
 - ☐ Are all states complete, and should not be combined or split into multiple states?
 - ☐ Are all error states included?
 - ☐ Are all data exceptions accounted for?
 - ☐ Are error logging and recovery mechanisms present?
 - ☐ Are all transitions mutually exclusive?
 - ☐ Are all state names meaningful and representative of the state?

4.3 Design Verification Plan

The team plans to verify the design through structured review processes and a checklist. The review shall consist of the team explaining the product and design to an external party, and providing the external party with the FRs. They will then ensure that all the FRs are met, by using the checklist as a reference. The checklist is as follows, with additional detail on the requirements available in [Functional Requirements](#):

- ☐ Does the design implement all the general function requirements listed below?
 - ☐ The product shall be able to recognize Clear Boundaries.
 - ☐ The product shall provide live update of `c_CurrentLoc`, `c_CurrentView` and `c_OccupancyMap` during all normal and non-configurational operation states.
 - ☐ The product shall allow the operator to configure the `i_MinHoverHeight`, `i_MaxHoverHeight`, and `i_DesiredHoverHeight`.
 - ☐ The condition `i_MinHoverHeight <= i_DesiredHoverHeight <= i_MaxHoverHeight` shall always be true.
 - ☐ The product shall be able to identify non-occupied parking spots.
 - ☐ The product shall highlight non-occupied parking slots on the operator's display (update `c_CurrentView`).
- ☐ Does the design implement all the state implementation requirements listed below?
 - ☐ The product shall implement an Idle state.
 - ☐ The product shall implement a Hover State.
 - ☐ The product shall implement a Manual Move State.
 - ☐ The product shall implement an Autonomous Explore State.
 - ☐ The product shall implement a Configure state.
 - ☐ The product shall implement an Off state.
 - ☐ The product shall implement a Land state.
 - ☐ The product shall implement a Desired Location Error state.
 - ☐ The product shall implement a No Parking Lot Detected Error state.
 - ☐ The product shall implement a Malfunction state.
 - ☐ The product shall implement a Communication Lost state.
 - ☐ The product shall implement a Compulsive Move State.
- ☐ Does the design implement all the state transition requirements listed below?
 - ☐ Upon the `m_PowerOn` becoming false, the drone shall enter the Off state.

- ☐ Upon the `m_PowerOn` becoming true, the drone shall enter the Idle state.
- ☐ Upon the `m_Launch` becoming true, the drone shall enter the Hover state if the `i_Mode` was set to normal, and enters the Configure state if the `i_Mode` was set to configure.
- ☐ If in the Hover state, and `c_ParkingLotDetected` is equal to true, the product shall enter the Autonomous Explore state and explore the detected lot.
- ☐ Once the user enters or changes `m_DesiredUserLoc` and `m_CompulsiveMove` is asserted as false, the drone shall automatically enter the Manual Move state.
- ☐ If while in the Manual Move state and the product determines that `m_DesiredUserLoc` is outside parking lot boundaries, the product shall enter the Desired Location Error state.
- ☐ When `m_AutonomousExplore` is set to true and `c_ParkingLotDetected` is equal to true, the product shall enter the Autonomous Explore state.
- ☐ When `m_AutonomousExplore` is set to true but `c_ParkingLotDetected` is equal to false, the product shall enter the No Parking Lot Detected Error state.
- ☐ Upon `m_Land` being true, the product shall enter the Land state.
- ☐ If `c_Connected` becomes false for more than 5 seconds, or signal strength (dBm) has lost 80% of its typical value at any point during operation, then the product shall enter the Communication Lost state.
- ☐ If while in the Communication Lost State, `c_Connected` becomes true for more than 5 seconds, or signal strength (dBm) has returned to 50% of its typical value at any point during operation, then the product shall enter the Hover state.
- ☐ Once the user enters or changes `m_DesiredUserLoc` and `m_CompulsiveMove` is asserted as true, the drone shall automatically enter the Compulsive Move state.

Internally within the team, a more detailed checklist shall be conducted that includes all the requirements listed within the SRS. To exclude any biases from the creator of the modules, the creator and tester will not be the same when conducting this review. Therefore, the members listed within section [xx] will be responsible for reviewing the requirements within their section. This ensures that all the requirements are accounted for within the design. The full checklist consists of the checklist above, in addition to the following:

- ☐ Does the design implement all the performance requirements listed below?
 - ☐ The product shall explore up to 1400 m^2 of the detected parking lot during the Autonomous Explore State.
 - ☐ The product shall takeoff to `i_MaxHoverHeight` and land from `i_MaxHoverHeight` within 25 seconds.
 - ☐ The product shall move to a specified location with an average speed exceeding 4km/hour.

- ☐ The product shall transmit all data to the operator at a rate exceeding 0.5 frames per second.
- ☐ The product shall maintain a longitudinal and lateral position within a 1.5m radius during the Hover State.
- ☐ While the product is not hovering (moving from one location to another), it shall always maintain an altitude between `i_MaxHoverHeight` and `i_MinHoverHeight`, within a tolerance of $\pm 5\%$.
- ☐ The product shall be operable within requirements within non-inclement weather.
- ☐ The product shall maintain a longitudinal and lateral position within a 1.5m radius once the product has reached `m_DesiredUserLoc` while in the Manual Move State or Compulsive Move State.
- ☐ Does the design implement all the design constraints listed below?
 - ☐ The product shall cost less than \$750 to manufacture.
- ☐ Does the design implement all the standards and compliance requirements listed below?
 - ☐ The product shall weigh a total of less than 25kg.
 - ☐ The product shall use radio communication only within the 2.4 GHz or 900 MHz range.
- ☐ Does the design implement all the security requirements listed below?
 - ☐ The operator's application shall only be launched by a user with authorized access.
 - ☐ The product shall not upload any gathered data to any external parties.
- ☐ Does the design implement all the maintainability requirements listed below?
 - ☐ The product shall be fully recharged within 1 hour.
 - ☐ The product shall be able to sustain a fall of greater than 1m without sustaining damage that affects operation performance.
 - ☐ The product shall be mechanically waterproof, to the point it can sustain a light drizzle for 1 minute of operation while still performing within the requirements.
- ☐ Does the design implement all the safety requirements listed below?
 - ☐ The product shall not influence or interact with dynamic actors positioned in the parking lot.
 - ☐ The product shall not allow the operator to set `i_MaxHoverHeight`, `i_MinHoverHeight`, or `i_DesiredHoverHeight` to be below 7m.
 - ☐ The product shall not require the operator to physically manipulate the product in any way in any state outside of Off State.

- ☐ The product shall not cause distractions or negatively impact greater than 2% of the visitors in the parking lot.
- ☐ The product shall include a mechanical Off switch to the product.
- ☐ Does the design implement all the usability requirements listed below?
 - ☐ The product shall provide a visual trace of its location for the past 60 seconds +- 1 second.
 - ☐ The product shall allow the operator to save the current visual and raw data into a folder.
 - ☐ The product shall be able to operate and provide data to the operator for > 5 minutes without the need to recharge.
 - ☐ The product shall require less than 2 hours of training for the operator to use.
 - ☐ The product shall display the current state to the Operator's PC Application.

4.4 Verification and Validation Plan Verification Plan

The Verification and Validation Plan shall be verified through an informal peer review process and a checklist. The peer review shall be conducted by another technical Capstone group and will use the provided rubric as a guideline. The checklist can then be conducted either internally within the team, or by an external party. The checklist is as follows, and ensures that all components of the VnV Plan are present:

- ☐ Are methods outlined to verify the SRS and the requirements within it?
 - ☐ Are all requirements covered by system tests?
- ☐ Are methods outlined to verify the design?
- ☐ Are methods outlined to verify the implementation of the design?
- ☐ Are automated testing and verification tools clearly outlined and feasible?
- ☐ Are methods outlined to validate the design and ensure that it solves the problem statement?
- ☐ Are all methods outlined in the VnV Plan feasible, given the current resources available for VnV?
- ☐ Is there traceability between the test plans and the requirements?
- ☐ Are tests outlined specifically and clearly, such that they can be reproduced by an external party?

4.5 Implementation Verification Plan

The implementation shall be verified primarily by the tests outlined within sections [xx] and [xx]. These sections include a combination of static, dynamic, and stress tests to ensure that the system is meeting the requirements of the system. Furthermore, the team shall conduct a code walkthrough for the FSM implementation. Conducted within the team, the implementor of the FSM shall provide the code used, and explain the state and transitions implemented. Using just the code, the team shall then recreate the entire FSM model as shown within the SRS at [here]. This process ensures that the implementation of the FSM matches the required FSM exactly.

The team shall also conduct code reviews of the changes before the changes are merged into the master branch within GitHub. This ensures that a peer review process is implemented as part of the GitHub management system and that the changes are correct. In addition to the code walkthroughs, automated testing and verification tools shall also be used to verify the implementation of the system, as outlined within [Automated Testing and Verification Tools](#).

4.6 Automated Testing and Verification Tools

The team shall implement a variety of tools to automate the testing and verification process. Discussed upon within the [Coding Standard](#), the team shall use a common IDE within the team, with integrated linters for both Python and C++. This ensures that static analyzers and coding standards are implemented automatically by the IDE and linter. A unit testing framework shall also be used to aid in the testing of the modules. Due to the usage of ROS within the product, the team shall use GTest for C++ modules, unittest for Python modules, and RosTest for ROS specific modules and communication. These three frameworks directly integrate within the ROS workflow, allowing for automatic execution of these tests during the compilation of the ROS packages. To further aid in the compilation of the various ROS packages, CMake shall be used to automate and simplify the compilation process. In addition to these tools, the team shall also produce a custom testing framework to test the visual perception algorithms. The custom tool shall automate the process of feeding in hardcoded input images into the algorithm, and record the algorithm outputs within a CSV file for easy manipulation and analysis of the outputs.

4.7 Software Validation Plan

The system shall be verified by two main methods. A guided review process shall be conducted by an external party, where the team provides and explains the FRs to the external party. They will then determine what the problem statement is, without the actual problem statement being provided by the team. This ensures that the SRS solves the given problem. To verify that the SRS has been met, the [SRS Verification Plan](#) will be used. To further test the functionality of the visual perception, open source datasets shall be used, in addition to the testing methods outlined within [Automated Testing and Verification Tools](#). Two open

source datasets will be used: the [Blanderbuss dataset](#) and the [CNRPark dataset](#). Sample images of these datasets are shown within Figures 1 and 2. These datasets will then verify the performance of the system with regards to the parking lot detection and occupancy in a variety of different conditions.

Figure 1: Blanderbuss Dataset Sample



Figure 2: CNRPark Dataset Sample



5 System Test Description

The proceeding section outlines detailed system tests that verify the implementation of the system.

5.1 Measuring Outputs Testing

There are several ways to measure the output variables. Some suggestions are given in the following section.

In general, given that all output variables can be viewed using the Operator's Application, this is perhaps the easiest source to measure the drone's response. There are special systems and unit tests to verify if the Operator's Application is correctly displaying/gathering the output variable.

Another method of viewing output variables is to print and store the output variables in log files within the onboard drone computer's file system. They can then be collected or sent to the Operator's PC to analyze the logs later.

To collect time statistics, either the timer on the embedded computer can be used and printed in log files, or one can use an external manual timer.

For all test cases, to measure location and height, the GPS location and height displayed on the Operator's PC Application can be used, except for the Flight Dynamics Tests [5.3.1](#), in these scenarios an external location and height measurement tool must be used. Examples to measure lateral location include adding an external and light GPS to the drone or walking underneath the drone with an external GPS (like the ones on smart phones). To measure height one may use marks on a wall, or even attach a rope to the drone.

5.2 Assumptions Regarding Test Cases

Unless otherwise stated, the test case requires that there is enough battery to complete the test case. Unless the creation of special stubs is specified, the drone is assumed to be complete mechanically, electrically, and software-wise.

5.3 Tests for Functional Requirements

The test cases were designed to for coverage over all the requirements that needed testing for verification. Requirements are grouped into the system component that is most difficult or most central to the task. However there is overlap between the test case categories, for example, the User Error Test Cases [5.3.4](#) utilize the Visual Perception component to make decisions/transitions.

5.3.1 Flight Dynamics

Table 3: STC_001

ID	STC_001
Control	Manual
Initial State	The product is in its Off state.
Input	Enter the configure state and set Indoor_Hover_Params . Afterwards, launch the drone in normal operation and wait for 1 min 25 sec.
Output	The drone should take less than 25 seconds to reach and hover within 1 +- 0.5 m from the moment the drone launches. While hovering, the drone should laterally stay within a 1.5 m radius of the launch location.
How test will be performed	<p>A stub must be created in the code to suppress the requirement of the hover parameters being at least Min_Hover_Params, as indoor conditions permit only a much lower flight height.</p> <p>The first step of the test, configuring height parameters, is accomplished by:</p> <ol style="list-style-type: none"> 1. Setting the input i_Mode as Configure and turning the power switch of the drone to On. 2. Assert m_Launch, so the drone enters the Configure state. 3. Setting the height parameters in the Operator's PC Application to Indoor_Hover_Params. <p>The second step of the drone, having the drone enter and stay within the hover state, is accomplished by:</p> <ol style="list-style-type: none"> 1. Turning the power switch on the drone off. Then set i_Mode as Normal and turn the power switch on, so the drone enters Idle state. 2. Placing the drone in a non-parking lot area. 3. Asserting m_Launch as true, then wait for 25 sec. 4. Wait for 1 min. During this wait measure the output variables. <p>Once the output variables have been measured and the test is complete, assert m_Land to true so the drone transitions to the land state.</p>
Test case derivation	<p>As per SRS, while in the Hover state, the drone will enter the Autonomous Explore state if it sees a parking lot. The purpose of placing the drone in a non-parking lot area was to keep the drone in the Hover state rather than entering the Autonomous Explore state.</p> <p>While hovering the drone should stay within a 1.5 m radius laterally, without the presence of any external forces such as wind.</p> <p>This test is an extension of STC_001, where this test is conducted within a controlled environment, and STC_001 is conducted within the actual environment with external forces.</p>
Purpose of test and/or relationship to other tests	<ul style="list-style-type: none"> • Assesses and verifies the flight dynamics in an indoor scenario, which won't have the gusts and winds of outdoor test cases. • Verifies the states Off, Idle, Hover, Configure and Land as well as the transitions between them (STA_000, STA_001, STA_004, STA_005, STA_006, TRANS_002, TRANS_003, TRANS_009). • Verifies a corner case in the parking lot detection algorithm, that it correctly detects no parking lot when there is no parking lot (GEN_001). • Elucidates how the drone operates in a non-parking lot area.

Table 4: STC_002

ID	STC_002
Control	Manual
Initial State	The product is in its Off state.
Input	Enter the configure state and set the hover height parameters to be the smallest values possible value Min_Hover_Params . Afterward, launch the drone in normal operation and wait for 1 min 25 sec.
Output	From the moment of launch, it should take less than 25 seconds to reach and hover within 7+-1.5 m. While hovering, the drone should laterally stay within a 1.5 m radius of the launch location, and it should stay within 5.5m and 8.5m above the ground at all times.
How the test will be performed	<p>The first step of the test, configuring height parameters, is accomplished by:</p> <ol style="list-style-type: none"> 1. Setting the input i_Mode as Configure and turning the power switch of the drone to On. 2. Assert m_Launch, so the drone enters the Configure state. 3. Setting the height parameters in the Operator's PC Application to Min_Hover_Params. <p>The second step of the drone, having the drone enter and stay within the hover state, is accomplished by:</p> <ol style="list-style-type: none"> 1. Turning the power switch on the drone off. Then Setting i_Mode as Normal and turning the power switch on so the drone enters the Idle state. 2. Placing the drone in a non-parking lot area. 3. Asserting m_Launch as true, then wait for 25 sec. 4. Wait for 1 min. During this wait, measure the output variables. <p>Once the output variables have been measured and the test is complete: Assert m_Land to true so that drone will enter the land state.</p>
Test case derivation	<p>As per SRS, the smallest possible height parameters are Min_Hover_Params. As per SRS, the drone should take less than 25 seconds to reach i_MaxHoverHeight (7+-1.5 m) from the moment the drone launches. This is why the input steps specify a 25-second wait after launch.</p> <p>Also, if the drone sees a parking lot in the Hover state, it must enter the Autonomous Explore state. To prevent this transition, a non-parking lot area is selected.</p> <p>While hovering, the drone should stay within a 1.5m lateral radius, with a height within $i_MaxHoverHeight - 1.5m = 7 - 1.5 = 5.5m$ and $i_MaxHoverHeight + 1.5m = 7 + 1.5 = 8.5m$ above the ground.</p>
Purpose of test and/or relationship to other tests	<p>This verifies that the configure state can change flight parameters in accordance with the restrictions specified in the SRS (GEN_003, GEN_004). Elucidates how the drone operates in a non-parking lot area. Verifies that the drone's flight dynamics are accurate, to the point that it can fly at the minimum height while maintaining minimum height safety requirements. Verifies the states Off, Idle, Hover, Configure, and Land as well as the transitions between them. In particular, it verifies the NFR requiring the drone to stay within a 1.5m radius within the hover state (STA_000, STA_001, STA_004, STA_005, STA_006, TRANS_002, TRANS_003, TRANS_009). Verifies a corner case in the parking lot detection algorithm, where it correctly detects no parking lot when there is no parking lot. Verifies the NFR requiring the product to take off to i_MaxHoverHeight within 25 seconds (PERF_002).</p>

Table 5: STC_003

ID	STC_003
Control	Manual
Initial State	The product is in its Off state.
Input	Enter the configure state and set the hover height parameters to be Med_Hover_Params . Afterward, launch the drone in normal operation and wait for 1 min 25 sec.
Output	The drone should take less than 25 seconds to reach and hover within 7+-1.5 m from the moment the drone launches. While hovering, the drone should laterally stay within a 1.5 m radius of the launch location, and it should stay within 18.5m and 21.5m above the ground at all times.
How test will be performed	<p>The first step of the test, configuring height parameters, is accomplished by:</p> <ol style="list-style-type: none"> 1. Setting the input i_Mode as Configure and turning the power switch of the drone to On. 2. Assert m_Launch, so the drone enters the Configure state. 3. Setting the height parameters in the Operator's PC Application to Med_Hover_Params. <p>The second step of the drone, having the drone enter and stay within the hover state, is accomplished by:</p> <ol style="list-style-type: none"> 1. Turning the power switch on the drone off. Then set i_Mode as Normal and turn the power switch on, so the drone enters Idle state. 2. Placing the drone in a non-parking lot area. 3. Asserting m_Launch as true, then wait for 25 sec. 4. Wait for 1 min. During this wait measure the output variables. <p>Once the output variables have been measured and the test is complete: Assert m_Land to true, so that drone will enter the land state.</p>
Test case derivation	<p>As per SRS, the drone should take less than 25 seconds to reach i_MaxHoverHeight (20+-1.5 m) from the moment the drone launches. This is why the input steps specify a 25 sec wait after launch.</p> <p>As per SRS, while in the Hover state, the drone will enter the Autonomous Explore state if it sees a parking lot. The purpose of placing the drone in a non-parking lot area is to keep the drone in the Hover state.</p> <p>While hovering the drone should stay a 1.5 m radius. In terms of height, it means that the drone should hover within $i_MaxHoverHeight-1.5m=20-1.5=18.5m$ and $i_MaxHoverHeight+1.5m=20+1.5=21.5m$ above the ground.</p>
Purpose of test and/or relationship to other tests	<p>This test is very similar to STC_001, except that the height parameters are configured to be much higher. The purpose of configuring the height parameters differently is to verify that the drone can hover accurately at different heights, that the drone can land safely from different heights, and that the Configure state is actually capable of configuring the height variables (GEN_003, GEN_004).</p> <p>Verifies the states Off, Idle, Hover, Configure, and Land as well as the transitions between them (STA_000, STA_001, STA_004, STA_005, STA_006, TRANS_002, TRANS_003, TRANS_009).</p> <p>Verifies a corner case in the parking lot detection algorithm where it correctly detects no parking lot when there is no parking lot.</p> <p>Verifies the NFR requiring the product to take off to i_MaxHoverHeight within 25 seconds (PERF_002).</p>

Table 6: STC_004

ID	STC_004
Control	Manual
Initial State	The product is in any of its flying states.
Input	Set m_CompulsiveMove as false. Change m_DesiredUserLoc to a location within the parking lot and to the diagonal front right of the drone at least 20m away.
Output	The drone should enter the Manual Move state upon a change to m_DesiredUserLoc. The drone should move to and stay within 1.5m radially of the specified location. The user should also see a visual trace of the drone's movement; it should be a roughly straight line (shortest path). Measure the time takes to reach the specified GPS location and calculate the average speed. As per SRS, ensure it is more than 4 m/sec.
How test will be performed	Input section contains enough detail.
Test case derivation	<p>As per SRS, the drone should enter Manual Move state whenever m_DesiredUserLoc is changed and m_CompulsiveMove is false. In this state the drone should travel to m_DesiredUserLoc and hover with an accuracy of 1.5m.</p> <p>A visual trace of the drone's movement in the past 60 seconds should be displayed on the Operator's PC Application. It should appear as a relatively straight diagonal line toward the hover location, as the drone's path planning should make it take the shortest path.</p> <p>Using the time taken, the average speed of the drone can be calculated as distance/time. As per the SRS, it should be at least 4 m/sec.</p>
Purpose of test and/or relationship to other tests	<ul style="list-style-type: none"> • Assesses the ability of the drone to move to forward as well as move rightward in a stable and efficient manner toward a specified GPS location. • Helps verify the Manual Move state and transitions related to proceeding to a given location when the location is within the parking lot (STA_002, TRANS_005). • Verifies the NFRs related to lateral accuracy (PERF_008). • Verifies the NFRs related to height accuracy during movement (PERF_006). • Verifies the visual trace requirement (USE_001). • Verifies the average speed requirement (PERF_003).

5.3.2 Battery

Table 7: STC_005

ID	STC_005
Control	Manual
Initial State	The product is in the Idle state.
Input	Record the remaining battery stated on the Operator's PC Application. Disconnect the battery from the drone and connect it to the battery charger and record the remaining battery levels it detects.
Output	Battery levels stated on the Operator's Application should match the battery levels stated on the battery charger.
How test will be performed	Input section contains enough detail.
Test case derivation	As per HA, the drone should display the remaining battery on the Operator's PC Application.
Purpose of test and/or relationship to other tests	Verification that the display of the remaining battery on the Operator's PC Application is accurate (SR_003).

Table 8: STC_006

ID	STC_006
Control	Manual
Initial State	The product is in its Idle state. The drone battery is fully charged.
Input	To launch the drone, assert m_Launch as true. Let the drone operate in any flying state.
Output	Observe the battery levels displayed on the Operator's Application fall with time. It should fly in normal operation for at least 3.5 minutes. At some point, the battery will be running very low (less than 1.5 minutes of flight left) and the drone will automatically enter the Malfunction state. In this state an error message will be logged to c_Log, the c_HealthStatus is set to Unhealthy, and the drone returns to the launch location.
How test will be performed	Regarding the initial state, verifying that the battery capacity is full can be accomplished by using the battery's charger. Input steps are self-explanatory.
Test case derivation	As per SRS, the Malfunction state must send an error message to c_Log, change the c_HealthStatus to Unhealthy, and the drone returns to the launch location. As per HA, the drone must enter the Malfunction state and land itself when the remaining battery is detected to be low (less then 1.5 minutes). As per SRS, the drone must have a total battery capacity of lasting at least 5 minutes, leading to at least 3.5 minutes of flight. As per HA, the drone must display the amount of battery remaining to the operator.
Purpose of test and/or relationship to other tests	<ul style="list-style-type: none"> • Verification of the low battery transition to the malfunction state (SR_011). • Verification of the operation of the malfunction state and transition into it (STA_009). • Verification of the NFR requiring flight time to be at least 5 minutes (USE_003). • Verification of the FR requiring the drone to display remaining battery life to the operator (SR_003).

Table 9: STC_007

ID	STC_007
Control	Manual
Initial State	The product is in its Idle state. The drone has less than 3 minutes of battery remaining.
Input	Attempt to launch the drone via setting m_Launch as true.
Output	Drone should not launch, and instead, a descriptive error should be logged into the variable c_Log.
How test will be performed	Input section contains enough detail.
Test case derivation	As per HA, the drone should not fly unless there is more the 3 minutes of battery remaining.
Purpose of test and/or relationship to other tests	Verification of the FR requiring the drone to not fly unless their sufficient battery is available (SR_012).

5.3.3 Communication

Table 10: STC_008

ID	STC_008
Control	Manual
Initial State	The drone is flying in any of its flight states.
Input	The operator turns off the communication network to the drone.
Output	The drone will enter the Communication Lost state, in which it will log an error to c_Log, set c_HealthStatus to Unhealthy, and move back toward the launch location.
How test will be performed	For example, if Wifi is used as the communication mechanism, turn off Wifi on the Operator's PC.
Test case derivation	As per SRS, the drone should enter the Communication Lost state when communication is lost for more than 5 sec. The Communication Lost state must log an error to c_Log, set c_HealthStatus to Unhealthy, and move back toward the launch location.
Purpose of test and/or relationship to other tests	<ul style="list-style-type: none"> • Assess drone operation when communication is lost abruptly. • Verifies the Communication Lost state as well as transitions related to entering it (STA_010, TRANS_010).

Table 11: STC_009

ID	STC_009
Control	Manual
Initial State	The drone is flying in any of its flight states.
Input	The operator closes the Operator's PC Application.
Output	The drone will enter the Communication Lost state, in which it will log an error to c_Log, set c_HealthStatus to Unhealthy, and move back toward the launch location.
How test will be performed	Close the PC Application's main window or close the process in the task manager.
Test case derivation	As per SRS the drone should enter the Communication Lost state when communication is lost for more than 5 sec. The Communication Lost state must log an error to c_Log, set c_HealthStatus to Unhealthy, and move back toward the launch location.
Purpose of test and/or relationship to other tests	<ul style="list-style-type: none"> • Assess drone operation when communication is lost abruptly. • Verifies the Communication Lost state as well as transitions related to entering it (STA_010, TRANS_010).

Table 12: STC_010

ID	STC_010
Control	Manual
Initial State	The drone is on the ground of a non-parking lot and in its idle state.
Input	Launch the drone by setting m_Launch to true. After which wait 25 sec. Assert m_CompulsiveMove as true. Change the value of m_DesiredUserLoc to a location at least 2km forward.
Output	<p>Once the value of m_DesiredUserLoc is changed, the drone will enter the Compulsive Move state. The drone will continue to move forward.</p> <p>When the drone loses connection, the drone will enter the Communication Lost state, in which an error will be logged to c_Log, c_HealthStatus changes to Unhealthy and the drone will move back toward the launch location.</p> <p>Record the last GPS location received to determine the distance traveled before was communication lost.</p> <p>While flying toward the launch location the drone regains communication, it should reenter the Hover state.</p>
How test will be performed	Input steps are self-explanatory.
Test case derivation	<p>As per SRS, the drone should hover within 25 seconds of being launched, so the operator must wait 25 seconds before they can ensure the drone is hovering.</p> <p>As per SRS, in the Compulsive Move state, the drone will move toward the desired location regardless of if it is within the parking lot.</p> <p>As per SRS, the drone should enter the Communication Lost state when communication is lost for more than 5 sec. The Communication Lost state must log an error to c_Log, set c_HealthStatus to Unhealthy, and move back toward the launch location. Communication is likely to be lost as the m_DesiredUserLoc is very far (2km).</p> <p>As per SRS, if the drone regains connection in the Communication Lost state, it should transition to the hover state.</p>
Purpose of test and/or relationship to other tests	<ul style="list-style-type: none"> • Estimate the drone's flying range. • Stress tests the drone's range. • Elucidates how the drone's functionality decreases with greater distances. • Assesses the ability of the drone to regain connection (SR_007). • Verifies the Communication Lost state as well as related to exiting it (STA_010, TRANS_010).

5.3.4 User Error States

Table 13: STC_011

ID	STC_011
Control	Manual
Initial State	The product is in any of its flying states.
Input	Set m_CompulsiveMove as false. Change m_DesiredUserLoc to a location outside the parking lot and in front of the drone.
Output	<p>The drone should enter the Manual Move State upon a change to m_DesiredUserLoc. The drone should move forward towards the location until it is close to the edge of the parking lot after which the drone should enter the Desired Location Error State. Upon entry to this state, it logs an error to the user in c_Log and sets c_UserError to Desired_Location_Out_Of_Bounds.</p> <p>There should be a straight line in the visual trace of the past GPS location if the path planning algorithm is correctly optimized to take the shortest path toward the GPS location.</p> <p>Monitor the FPS of the c_CameraView output; it should be more than 0.5 FPS.</p>
How test will be performed	Input section contains enough detail.
Test case derivation	<p>As per SRS, the drone should enter Manual Move whenever m_DesiredUserLoc is changed and m_CompulsiveMove is false. Given the GPS location is in front of the drone, it should fly forward, but when it recognizes that the requested GPS location is outside the parking lot, it should enter the Desired Location Error State. Within this state, the SRS specifies that the drone should inform the user that their movement command is not possible (through a message in c_Log), and begin hovering in place.</p> <p>A visual trace of the drone's movement in the past 60 seconds should be displayed on the Operator's PC Application. It should appear as a relatively straight line toward the hover location, as the drone's path planning should make it take the shortest path. As per SRS, the FPS should be more than 0.5.</p>
Purpose of test and/or relationship to other tests	<ul style="list-style-type: none"> • Assesses the ability of the drone to fly to forward. • Verifies the drone's ability to identify Parking Lot boundaries (GEN_001). • Helps verify the Manual Move state and transitions related to it, specifically its ability to detect and report the error when the requested location is invalid (STA_002, TRANS_005). • Verifies the Desired Location Error state and related transitions (STA_007, TRANS_006). • Verifies the visual trace requirement (USE_001). • Verifies the NFR specifying the minimum required FPS of c_CurrentView to be at least 0.5 (PERF_004).

Table 14: STC_012

ID	STC_012
Control	Manual
Initial State	The height parameters are set as Min_Hover_Params . The drone is in the Hover state, on the surface of a non-parking lot area but at least 50m away from a parking lot.
Input	Attempt to enter the Autonomous Explore state. Now move the drone to a location within the nearby parking lot utilizing the Compulsive Move state. Once the drone is in the parking lot attempt to enter the Autonomous Explore State again.
Output	Upon the first attempt to enter the Autonomous Explore State, the attempt should fail, and instead, the drone should enter the No Parking Lot Detected State. Upon entrance to this state, c_UserError is set to No_Lot_Detected_State and an error message is logged to c_Log. Afterward, when the drone enters the Compulsive move state, c_UserError should be None, and the drone should travel to m_DesiredUserLoc. Upon the second attempt to enter the Autonomous Explore State, the attempt should succeed.
How test will be performed	<p>The test case can be restated in terms of the input and output variables:</p> <ol style="list-style-type: none"> 1. Assert m_AutonomousExplore as true to request the Autonomous Explore state. Measure/analyze behavior. 2. Now assert and hold m_CompulsiveMove as true, and change m_DesiredUserLoc to a location within the nearby parking lot. 3. Wait until the drone is within 1.5m of the m_DesiredUserLoc 4. Try setting m_AutonomousExplore to true again to request entrance in the parking lot. Measure/analyze behavior.
Test case derivation	<p>As per SRS the Autonomous Explore state should only be entered if c_ParkingLotDetected is true. The Initial State behavior specified above (height parameters and distance to nearby parking lot) is set in a way to ensure that the nearby parking lot is not in the field of view of the drone. This is why the first attempt to enter the Autonomous Explore fails while the second attempt succeeds. When an attempt to enter Autonomous Explore fails, the drone should enter the No Parking Lot Detected State.</p> <p>As per SRS, upon entrance into the No Parking Lot Detected State c_UserError should be set to No_Lot_Detected_State and an error message is should be logged to c_Log. Upon exit _UserError should be set to None.</p> <p>As per SRS, the Compulsive Move state should be entered if the m_DesiredUserLoc is changed and m_CompulsiveMove is true.</p>
Purpose of test and/or relationship to other tests	<ul style="list-style-type: none"> • This test case is not designed to test the accuracy of flight dynamics, as the drone moves too quickly and too far to accurately measure GPS. • Assess the accuracy of the visual perception algorithm in recognizing parking lot area (GEN_001). • Helps to verify the Autonomous explore state as well as the transition to it from the input m_AutonomousExplore (STA_003, TRANS_004). • Verifies the Compulsive Move state as well as transitions related to it (STA_011, TRANS_012). • Verifies the No Parking Lot Detected Error state as well as transitions related to it (STA_008, TRANS_008).

5.3.5 Visual Perception and Path Planning

Table 15: STC_013

ID	STC_013
Control	Manual
Initial State	The drone is on the ground of a rectangular-shaped parking lot with less than 30 parking spots, is fully charged, and is in its idle state.
Input	Assert m_Launch.
Output	The drone should first enter the Hover state, but once it begins hovering and it detects the parking lot, the drone should enter the Autonomous Explore state. Within the 3.5 minutes of guaranteed operations, the drone should have explored the full parking and completed its c_OccupancyMap with reasonable accuracy. Monitor the FPS and output of c_CameraView output.
How test will be performed	Input section contains enough detail.
Test case derivation	As per SRS, while in the hover state, the drone should enter the Autonomous Explore state automatically once it detects a parking lot. The Autonomous Explore state specifies that the drone should explore the parking lot during this state. Assuming that the drone moves at roughly 4m/sec and sees at least 1 parking spot every frame, it is reasonable to assume that 3.5 minutes is more than enough time for the drone to explore an entire parking lot of this size.
Purpose of test and/or relationship to other tests	<p>Assesses the accuracy of the visual perception algorithm to segment the parking lot, accuracy in recognizing unoccupied parking spots, and accuracy in its generated occupancy map (c_OccupancyMap) (GEN_005, GEN_006).</p> <ul style="list-style-type: none"> • Assesses the accuracy of the path planning algorithm (does it ever explore the same area twice, does it explore the parking lot in a systematic and predictable way, etc.) (STA_003). • Verifies the drone's ability to identify Parking Lot boundaries (GEN_001). • Helps to verify the Autonomous explore state as well as the transition to it from the Hover state (STA_003, TRANS_004). • Verifies the NFR specifying the minimum required FPS of c_CurrentView to be 0.5 (PERF_004). • Verifies the NFR requiring that the drone explores up to 1400m² of the parking lot assuming enough time and a small enough size of the parking lot (PERF_001).

Table 16: STC_014

ID	STC_014
Control	Manual
Initial State	The drone is on the ground in its idle state, on a street with yellow and white lines but over 50m away from a parking lot.
Input	Launching is accomplished by asserting m_Launch to true, wait for 1 min 25 sec.
Output	The drone should remain in its hover state.
How test will be performed	Input steps are self-explanatory.
Test case derivation	<p>As per SRS, the drone should hover within 25 seconds of being launched. By waiting for another 1 minute the tester ensures the drone has sufficient time to enter the hover state and process the input images for many frames.</p> <p>As per SRS, the drone should remain in the hover state as it shouldn't detect any parking lots.</p>
Purpose of test and/or relationship to other tests	<ul style="list-style-type: none"> • Ensures that the drone does not detect streets as parking lots. This is an important use case as the drone will likely see streets in its normal operation as streets are often found near parking lots. • Verifies the ability of the drone to identify parking lots (GEN_001, SR_009).

Table 17: STC_015

ID	STC_015
Control	Automatic
Initial State	The drone is in the Idle state, and visual perception features and camera technology is functional.
Input	Hard code c_CurrentView as an image from open source datasets. See 4.6 for the datasets used. i_DesiredHoverHeight is set to 0m.
Output	The drone should output c_CurrentView, c_OccupancyMap, and c_ParkingLotDetected. c_CurrentView should feature the input image with overlays for the parking lot slots and the boundaries of the parking lot. c_OccupancyMap should indicate all visible parking lots, and whether each parking lot is occupied or not. c_ParkingLotDetected should be true. The specific outputs will be determined by which input image is being passed in.
How test will be performed	The recreation of the initial state may require modifying code. The idle state has no requirements related to the usage of the visual perception features. If the Idle state does not enable the usage of the camera and visual perception features already, modify the Idle state's code (creating a stub) to do this. Furthermore, in order to enhance safety, keep the drone on the ground several meters away, remove the propellers and modify the transition code such that the Idle state is never exited. Another stub will be produced to feed in a hard-coded input image to c_CurrentView rather than the live camera feed. The outputs of the system will be observed through the Operator's application. An external script will be used to automate the input image passing and will export the outputs to an external CSV (detailed in 4.6).
Test case derivation	As per SRS, the drone should process the visual input from the camera to determine the parking lot information and display that information to the Operator's application. This includes the parking lot slot identifications, the occupancy map, and the parking lot bounds. To ensure repeatability in this test, hard-coded inputs are used instead of the real environment. This prevents any changes to the environment from affecting the results of this test.
Purpose of test and/or relationship to other tests	<ul style="list-style-type: none"> • Verifies the pipeline that communicates outputs from the visual perception feature to the Operator's PC application. • Helps to gauge the accuracy and performance of the visual perception on datasets of varying weather, number of vehicles, types of vehicles, camera height, etc. By using hardcoded datasets, the perception algorithms create consistency between tests, as different versions of the algorithm can be compared on the same inputs. • Verifies the ability to recognize clear boundaries (GEN_001). • Verifies the live updating of the c_OccupancyMap (GEN_002). • Verifies the ability to identify non-occupied parking spots (??, SR_009). • Verifies display of non-occupied parking spots on the Operator's application (GEN_006).

Table 18: STC_016

ID	STC_016
Control	Automatic
Initial State	The drone is in the Automatic Explore state within the 3D Software in the Loop (SITL) environment.
Input	Position the drone within a custom-made 3D SITL environment. Send the drone to the Hover state and transition to the Automatic Explore state.
Output	<p>The drone should output <code>c_CurrentView</code>, <code>c_OccupancyMap</code>, and <code>c_ParkingLotDetected</code>. <code>c_CurrentView</code> should feature the input image with overlays for the parking lot slots and the boundaries of the parking lot. <code>c_OccupancyMap</code> should indicate all visible parking lots, and whether each parking lot is occupied or not. <code>c_ParkingLotDetected</code> should be true. The specific outputs will be determined by which input image is being passed in.</p> <p>Furthermore, the current positions of the drone should be outputted to the Operator's application: <code>c_CurrentLoc</code> and <code>c_PastLoc</code>.</p>
How test will be performed	A custom-made SITL environment consisting of 6 rows of parking slots, with 10 slots per row, will be created with Gazebo. 25% of the parking slots will be occupied by vehicles. An external script will be used to start the SITL setup with the premade scenario and execute the required sequence of inputs to bring the drone to the Autonomous Explore state. <code>c_CurrentView</code> for the entire test will be saved as a .avi file, and the remaining outputs will be exported to a CSV for analysis. The test ends when either the entire parking lot has been explored, or until 3.5 minutes has passed since the drone's initial launch.
Test case derivation	As per SRS, the drone should process the visual input from the camera to determine the parking lot information and display that information to the Operator's application. This test is similar to STC_014, but this test is within a closed loop environment, whereas STC_014 is with an open loop system. Furthermore, this tests the path planning algorithm specified within the SRS.
Purpose of test and/or relationship to other tests	<ul style="list-style-type: none"> • Usage of the 3D SITL allows the verification of the path planning algorithm, visual perception algorithm, and the integration between them on consistent test cases. • Verifies the ability to recognize clear boundaries (GEN_001). • Verifies the live updating of the <code>c_OccupancyMap</code>, <code>c_CurrentLoc</code>, and <code>c_CurrentView</code> (GEN_002). • Verifies ability to identify non-occupied parking spots (GEN_005). • Verifies display of non-occupied parking spots onto the Operator's application (GEN_006). • Verifies the Autonomous Explore state (STA_003, TRANS_003).

5.4 Traceability between Test Cases and Functional Requirements

Table 19: FR Traceability Table

Functional Requirement	Test Case to Verify
GEN_001	STC_001, STC_011, STC_012, STC_013, STC_014, STC_015, STC_016
GEN_002	STC_015, STC_016
GEN_003	STC_001, STC_002
GEN_004	STC_001, STC_002
GEN_005	STC_013, STC_016, STC_015
GEN_006	STC_013, STC_016, STC_015
STA_000	STC_001, STC_002, STC_003
STA_001	STC_001, STC_002, STC_003
STA_002	STC_005, STC_011
STA_003	STC_016, STC_012, STC_013
STA_004	STC_001, STC_002, STC_003
STA_005	STC_001, STC_002, STC_003
STA_006	STC_001, STC_002, STC_003
STA_007	STC_011
STA_008	STC_012
STA_009	STC_006
STA_010	STC_008, STC_009, STC_010
STA_011	STC_012
TRANS_001	STC_001, STC_002, STC_003
TRANS_002	STC_001, STC_002, STC_003
TRANS_003	STC_001, STC_002, STC_003
TRANS_004	STC_013, STC_012
TRANS_005	STC_004, STC_011
TRANS_006	STC_011
TRANS_007	STC_012, STC_013
TRANS_008	STC_011
TRANS_009	STC_001, STC_002, STC_003
TRANS_010	STC_008, STC_009, STC_010
TRANS_011	STC_008, STC_009, STC_010
TRANS_012	STC_012, STC_010

5.5 Tests for Nonfunctional Requirements

5.5.1 User Manual

Table 20: STC_017

ID	STC_017
Control	Manual
Initial State	Two volunteers are available for two hours. The drone is in its Off state.
Input	Volunteers read the user manual, attempt to install the Operator's Application, and then attempt to conduct STC_009 without any support from developers.
Output	All volunteers read the user manual, install the necessary software and successfully conduct STC_009 within 2 hours.
How test will be performed	Input section is self-explanatory.
Test case derivation	As per SRS, a new non-technical user shall be able to operate the drone within 2 hours.
Purpose of test and/or relationship to other tests	<ul style="list-style-type: none">• Build confidence that the user manual is readable and that the product is well documented.• STC_009 is one of the most complicated states, as it involves multiple inputs, the ability to operate the drone in two flight states, and the operation of an error state.• Verifies NFR requiring a new user to be able to operate the drone within 2 hours (USE_004).

Table 21: STC_018

ID	STC_018
Control	Static
Initial State	Two participants have read the manual in its entirety.
Input	<p>Ask each participant to highlight the sentence(s) that specify the</p> <ol style="list-style-type: none"> 1. Weather conditions of when not to fly. 2. Steps/inspection to conduct prior to flight and after flight. 3. Orientation to hold the drone. 4. States in which drone can be held. 5. Whether or not the password can be shared.
Output	<p>The answer to each question is listed below in corresponding order. A successful test is one in which the participants are able to identify the appropriate sentence(s) corresponding to each question:</p> <ol style="list-style-type: none"> 1. Weather with rain, snow, fog, and/or winds over 50 km/hour is considered inclement weather. 2. Inspect drone for damage pre-flight and post-flight. Wait for the drone to cool down post-flight. 3. Correct orientation is specific to the frame design, and thus cannot be specified at this time. 4. Drone can be held in Off, Idle, and Configure states. 5. Password must be kept private.
How test will be performed	Input section is self-explanatory.
Test case derivation	As per SRS, the user manual must contain certain safety specifications regarding non-inclement weather, pre-flight damage inspection, postflight wait for cooldown, orientations to hold drone, holdable states, and secrecy of password (SR_002 , SR_006 , SR_010 , SAFE_003).

5.5.2 Usability

Table 22: STC_019

ID	STC_019
Control	Static
Initial State	-
Input	Place the drone on a weighing scale.
Output	Drone weighs less than 25 kg.
How test will be performed	Input section is self-explanatory.
Test case derivation	As per SRS, the drone must weigh less than 25 kg (STD_001).

Table 23: STC_020

ID	STC_020
Control	Static
Initial State	Drone battery has discharged completely.
Input	Connect the drone to the charger, and disconnect once it is fully charged.
Output	Battery is fully charged in less than an hour.
How test will be performed	To complete the Initial State, discharge the battery through the discharge setting on the charger.
Test case derivation	As per SRS, the drone battery should recharge in one hour (MTNC_001).

Table 24: STC_021

ID	STC_021
Control	Static
Initial State	Drone is any flight state, and 10 people in their cars are stationed around the parking lot.
Input	Survey the 10 people as to whether the drone (sight or noise) would negatively influence their driving in a serious way.
Output	No participants respond to the survey question with yes to the question.
How test will be performed	Input section is self-explanatory.
Test case derivation	As per SRS, the drone should not disturb more then 2% of people (SAFE_004 , SAFE_001).

Table 25: STC_022

ID	STC_022
Control	Manual
Initial State	Drone is any flight state.
Input	Assert m_SaveOutput. Attempt to save the raw visuals (c_OccupancyMap and c_CurrentView) in any folder.
Output	Both visuals are saved in the specified folders with specified names and reasonable similarity to what was seen on the Operator's Application.
How test will be performed	Input section is self-explanatory.
Test case derivation	As per SRS, the drone should allow the user to have the current output into a folder (USE_002).

5.6 Traceability between Test Cases and Non-Functional Requirements

Certain requirements do not have a corresponding test case. A reason is specified if this is the case. For example, if a requirement is a pass or fails determined by the parts purchased, then testing will not yield any new information as it cannot be changed by the engineers.

Table 26: NFR Traceability Table

Non-Functional Requirement	Test Case to Verify
PERF_001	STC_013
PERF_002	STC_002, STC_003
PERF_003	STC_004
PERF_004	STC_011, STC_013
PERF_005	STC_002, STC_003
PERF_006	STC_004
PERF_007	STC_015
PERF_008	STC_004
DES_001	None, it is a pass or fail depending on components already bought.
STD_001	STC_019
STD_002	None, it is a pass or fail depending on components already bought.
SEC_001	None, password complexity is a simple requirement and can be tested through unit tests.
SEC_002	None, pass or fail.
MTNC_001	STC_020
MTNC_002	None, may damage expensive parts. In the interest of cost, this is not tested.
MTNC_003	None, may damage expensive parts. In the interest of cost, this is not tested.
SAFE_001	STC_021
SAFE_002	STC_002
SAFE_003	STC_018
SAFE_004	STC_021
SAFE_005	None, pass or fail dependent on components bought.
USE_001	STC_004, STC_011
USE_002	STC_022
USE_003	STC_006
USE_004	STC_017
USE_005	Will be accomplished in future through a unit test.

Non-Functional Requirement	Test Case to Verify
SR_002	STC_018
SR_003	STC_005, STC_006
SR_004	None, pass or fail determined by the components bought.
SR_005	None, pass or fail determined by the components bought.
SR_006	STC_018
SR_007	STC_010
SR_008	None, redundant localization without gps
SR_009	STC_015, STC_014
SR_010	STC_018
SR_011	STC_006
SR_012	STC_007
SR_013	None, password complexity is a simple requirement and can be tested through unit tests.

6 Unit Test Description

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS] [This section should not be filled in until after the MIS (detailed design document) has been completed. —SS]

6.1 Unit Testing Scope

[What modules are outside of the scope? If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

6.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

6.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

test-id1

Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS] Initial State: Input:

Output: [The expected result for the given inputs —SS]

Test Case Derivation: [Justify the expected value given in the Output field —SS]

How test will be performed:

test-id2

Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

Initial State:

Input:

Output: [The expected result for the given inputs —SS]

Test Case Derivation: [Justify the expected value given in the Output field —SS]

How test will be performed:

...

6.2.2 Module 2

...

6.3 Tests for Nonfunctional Requirements

1. test-id1

Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

Initial State:

Input/Condition:

Output/Result:

How test will be performed:

2. test-id2

Type: Functional, Dynamic, Manual, Static etc.

Initial State:

Input:

Output:

How test will be performed:

6.3.1 Module ?

...

6.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

7 Appendix

This is where you can place additional information.

7.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.

Table 27: Symbolic Constants

ID	Symbolic Constants
Min_Hover_Params	(i_MinHoverHeight=7m, i_- -MaxHoverHeight=7m, i_DesiredHover- Height=7m)
Med_Hover_Params	(i_MinHoverHeight=20m, i_- -MaxHoverHeight=20m, i_DesiredHover- Height=20m)
Indoor_Hover_Params	(i_MinHoverHeight=1m, i_- -MaxHoverHeight=1m, i_DesiredHover- Height=1m)

7.2 Reflection

This section reviews the roles mentioned in Table 2 and evaluates each team member on the graduate attribute of Lifelong Learning.

Table 28: Required Testing

Test	Responsibility	Rationale
Dynamic Test-ing	Fady, Muham-mad, Zaid & Winnie	Dynamic testing is vital, especially for fly-ing a drone. Dynamic testing is executing test cases in variable environments to ana-lyze how the system behaves.
Integration Test-ing	Muhammad	Integration testing is important as it will test different units, modules, or components of the drone as a combined entity.
Static Testing	Fady, Muham-mad & Winnie	Static testing involves matching the require-ments from the SRS to the code written. Static testing also involves looking into code format and errors.
Visual Percep-tion and Path Planning Test-ing	Fady	Verifies that the visual perception and au-tonomous exploration algorithm is perform-ing within specifications.
Drone Finite State Machine (FSM) and Communication Testing	Ali	Verifies that all communication between the drone components and the drone to the Op-erator’s application is working correctly.
Mechanical Testing	Zaid	Verifies that all physical components and the dynamics are working within specifications.
Operator’s Ap-plication Testing	Winnie	Verifies that the Operator’s application and user manual meet the specifications outlined.

7.2.1 Knowledge & Skills

Table 29 covers different approaches to follow the plan from Table 28.

Table 29: Knowledge Acquisition

Test	Approaches	Verdict
Dynamic Test- ing	<p>Approach 1: Previous lecture notes from McMaster’s MECHTRON 3K04: Software Development.</p> <p>Approach 2: Research online and watch tutorials such as this video on YouTube.</p> <p>Approach 3: Research online and find resources for mechanical static testing such as this.</p>	Fady, Muhammad & Winnie will proceed with Approach 2 as MECHTRON 3K04 did not go in-depth with Static Testing. Zaid will proceed with Approach 3 for Mechanical Testing.
Integration Test- ing	<p>Approach 1: Previous lecture notes from McMaster’s MECHTRON 3K04: Software Development.</p> <p>Approach 2: Research online resources such as this website.</p>	Muhammad will proceed with Approach 2 to get more knowledge with Integration Testing as the MECHTRON 3K04 course did not get in-depth with the subject.
Static Testing	<p>Approach 1: Previous lecture notes from McMaster’s MECHTRON 3K04: Software Development</p> <p>Approach 2: Research online and find resources for static testing such as this website.</p>	Fady, Muhammad & Winnie will proceed with Approach 1 as Static Testing was done extensively in the MECHTRON 3K04 Pacemaker project.
Visual Percep- tion and Path Planning Test- ing	<p>Approach 1: Watch McMaster’s Google Developer’s Club video on YouTube to learn more about the OpenCV library complementing ENG 1D04: Engineering Computation.</p> <p>Approach 2: Use OpenCV tutorial documentation.</p>	Fady will proceed with Approach 1 as the tutorial was done at McMaster by McMaster students going through the fundamentals of OpenCV.
Drone Finite State Machine (FSM) and Communication Testing	<p>Approach 1: Previous lecture notes from McMaster’s MECHTRON 3K04: Software Development</p> <p>Approach 2: Use ROS.org documenta- tion on ROS’ library SMACH to learn more about the FSM implementation.</p>	Muhammad will proceed with Approach 2 to get familiar with the SMACH library which will be more relevant for drone development.

Test	Approaches	Verdict
Mechanical Testing	<p>Approach 1: Previous lecture notes from McMaster's ENG 1C03: Engineering Design & Graphics</p> <p>Approach 2: Watch SolidWorks' basic tutorial.</p>	Zaid will use Approach 2 to get familiar with SolidWorks.
Operator's Application Testing	<p>Previous lecture notes from McMaster's MECHTRON 1D04: Engineering Computation to understand the language of the application.</p> <p>Approach 2: Use Python Documentation and other online resources to understand Python syntax.</p>	Winnie will proceed with Approach 1 as ENG 1D04 had covered Python extensively.