Project Title: System Verification and Validation Plan for Mechatronics Engineering

Team # 34, ParkingLotHawk
Fady Zekry Hanna
Winnie Trandinh
Muhammad Ali
Muhammad Khan

March 9, 2023

1 Revision History

Table 1: Revision History

Date	Version	Notes
November 3, 2022	1.0	Initial Revision

Contents

List of Tables

List of Figures

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 Design Verification Plan.

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 be 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 an Autonomous 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 Autonomous Move state.
- ☐ If while in the Autonomous 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 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 ±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 Autonomous 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_MinHover-Height, 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 System Test Description. 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

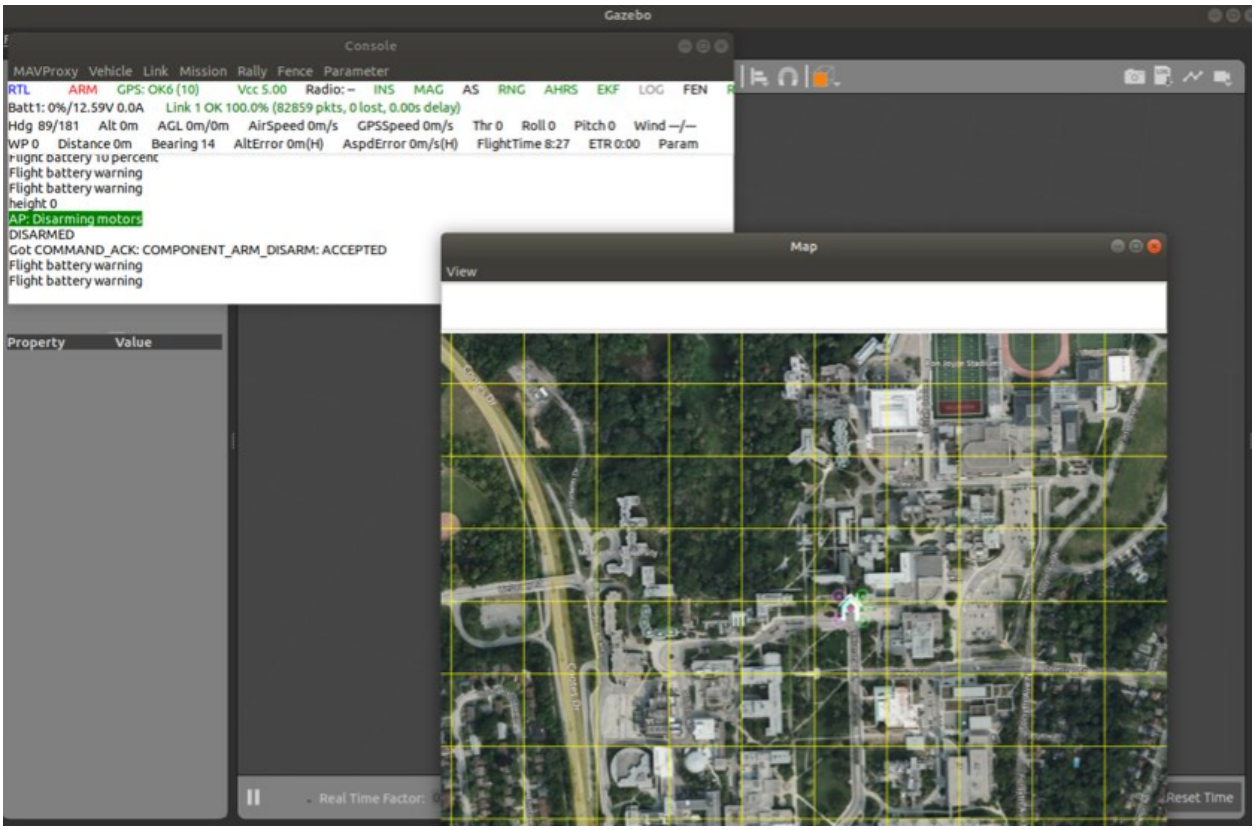
4.8 Software in The Loop Testing

In conjuncture with unit testing the modules and performing tests on the physical drone, a Software in The Loop (SITL) environment is created to replace the physical drone during testing. The software running on the drone and within SITL is exactly the same, ensuring that a test passed within SITL testing will pass on the physical drone.

The SITL environment replaces the physical drone only, thus all communication methods between the Operator's PC and the drone, and software for the Operator's Application and drone are not affected. This is accomplished through establishing a Gazebo and ROS connection, where Gazebo is a 3D physics simulation platform. An example of this Gazebo and the entire environment is shown within Figures 3 and 4.

Figure 3: Gazebo Environment

Figure 4: SITL Environment Including a Satellite Map and Console



The key difference between testing within SITL and with the physical drone is that the drone's flight performance are not involved. In many cases, this is an advantage as it increases the ease of testing and reduces the risk involved if a failed test occurs. Furthermore, custom environments can be created within Gazebo, offering increased flexibility in testing the visual perception. However, the SITL environment and the real world does not translate exactly, thus lowering the confidence in the test result. Nonetheless, the SITL environment is used for many test cases in the proceeding sections if the team is unable to test it with the physical drone.

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.
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	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. The first step of the test, configuring height parameters, is accomplished by: 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. The second step of the drone, having the drone enter and stay within the hover state, is accomplished by: 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 35 sec. During this wait measure the output variables. 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.
Test case derivation	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. While hovering the drone should stay within a 1.5 m radius laterally, without the presence of any external forces such as wind.
Purpose of test and/or relationship to other tests	• Assesses and verifies the that drone is able to fly. An indoor scenario, was selected because to minimize the influence of gusts and winds of outdoors. 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 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). Verifies configuring the height variables (GEN_003, GEN_004).

Table 4: STC_002

ID	STC_002
Control	Manual
Initial State	The product is in its Off state.
Input	Turn on the Drone, and leave it in the Idle state. Attempt to set the configuration parameters to Invalid_Hover_Params1, then attempt to set the height parameters to Invalid_Hover_Params2.
Output	Both attempts to set the Hover Parameters should fail.
How the test will be performed	The first step of the test, configuring height parameters, is accomplished by: 1. Setting the input i_Mode as Configure and turning the power switch of the drone to On. 2. Set the height parameters in the Operator's PC Application to Invalid_Hover_Params1. 3. Assert m_Launch, so the drone enters the Configure state. 4. Set the height parameters in the Operator's PC Application to Invalid_Hover_Params2. 5. Assert m_Launch, so the drone enters the Configure state. 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.
Test case derivation	As per SRS, the smallest possible height parameters are Min_Hover_Params, thus an attempt to set them smaller, such as Invalid_Hover_Params1, should fail and return an error to the user. As per SRS, the i_MaxHoverHeight must be bigger than i_MinHoverHeight, violating this contains, as in Invalid_Hover_Params1, should fail and return an error to the user.
Purpose of test and/or relationship to other tests	This verifies that the configure state can change flight parameters in accordance with the restrictions specified in the SRS (SAFE_002, GEN_004).

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	The first step of the test, configuring height parameters, is accomplished by: 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. The second step of the drone, having the drone enter and stay within the hover state, is accomplished by: 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. 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.
Test case derivation	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. 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. 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.
Purpose of test and/or relationship to other tests	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). 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 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).

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 Autonomous 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	As per SRS, the drone should enter Autonomous 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. 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. 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.
Purpose of test and/or relationship to other tests	• 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 Autonomous 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	• 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	• Assess the drone's ability to transition to the communication lost state when the wifi connection is cut. • 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	• Assess the drone's ability to transition to the communication lost state when the ParkingLotHawk Application is closed 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	Once the value of m_DesiredUserLoc is changed, the drone will enter the Compulsive Move state. The drone will continue to move forward. 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. Record the last GPS location received to determine the distance traveled before was communication lost. While flying toward the launch location the drone regains communication, it should renter the Hover state.
How test will be performed	Input steps are self-explanatory.
Test case derivation	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. 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. 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). As per SRS, if the drone regains connection in the Communication Lost state, it should transition to the hover state.
Purpose of test and/or relationship to other tests	• 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	Change m_DesiredUserLoc to a location outside the parking lot. Attempt to fly the drone to that location by using the Compulsive Move state (asserting m_CompulsieveMove) and selecting "Yes" when the prompt opens.
Output	The drone should log an error message indicating the location is not within a parking lot in c_Log and set c_UserError to Desired_Location_Out_Of_Bounds. A prompt should open up asking if the user desires the drone to move to the location. When the user selected "Yes", the drone should fly to the location.
How test will be performed	Input section contains enough detail.
Test case derivation	As per SRS, the drone should prompt the user with a message if the location they desire the drone to fly toward is invalid. It should also change c_UserError and log a message to c_Log. If the user answers "Yes" to the prompt, the drone should fly toward the location.
Purpose of test and/or relationship to other tests	• Helps verify the Compulsive Move state and transitions related to it, specifically its ability to detect and report the error when the requested location is invalid (STA_013, TRANS_005).

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	The test case can be restated in terms of the input and output variables: 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	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. 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. As per SRS, the Compulsive Move state should be entered if the m_DesiredUserLoc is changed and m_CompulsiveMove is true.
Purpose of test and/or relationship to other tests	• 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. • Helps to verify the transition into the Autonomous explore state (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	• 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) in a real physical environment. • 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) in a real physical environment. • Verifies the drone's ability to identify Parking Lot boundaries (GEN_001) in a real physical environment. • Verifies the ability to create and updating c_OccupancyMap (GEN_002). • 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 1400m$\hat{2}$ of the parking lot assuming enough time and a small enough size of the parking lot (PERF_001) in a real physical environment.

Table 16: STC_014

ID	STC_014
Control	Manual
Initial State	Disconnect the propellers from the drone. Carry the drone to an area that has grass, a roadway with lane markings, and a parking lot within walking distance.
Input	Hold the drone on a grass pasture. The walk and stand in the middle of a street with yellow and white lines. Then walk into a parking lot.
Output	The Operator's PC should display c_ParkLotDetected as false in all places except when the drone is inside a parking lot.
How test will be performed	The recreation of the initial state requires modifying the transition code such that the Idle state is never exited, to ensure safety while the tester handles the drone. The outputs of the system will be observed through the Operator's application.
Test case derivation	As per SRS, the drone should be able to detect and differentiate between parking lots and non-parking lots. The drone analyses and segments parking lot area on pre-downloaded sattaliete imagery. Thus the GPS location alone (not visual perception) is needed to help the drone determine if a given location is within the parking lot.
Purpose of test and/or relationship to other tests	• 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, propellars are disconnected, the visual perception code is active, and live camera is functional.
Input	Hold the drone over a printed picture of a parking lot. i_DesiredHoverHeight is set to 0m.
Output	The live camera view, c_CurrentView, should feature the printed image. The drone should also identify if the location it is hovering over is occupied by an object or not.
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. The outputs of the system will be observed through the Operator's application.
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 identifications, and occupancy classification.
Purpose of test and/or relationship to other tests	• Verifies the data pipeline that communicates outputs from the visual perception feature to the Operator's PC application. • Verifies the ability to identify non-occupied and occupied parking spots (GEN_005, SR_009). • Verifies display of the live camera view (c_CameraView).

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	The drone should output c_CurrentView, c_OccupencyMap, 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_OccupencyMap 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. Furthermore, the current positions of the drone should be outputted to the Operator's application: c_CurrentLoc and c_PastLoc.
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. c_CurrentView 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	• 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 c_OccupancyMap, c_CurrentLoc, and c_CurrentView (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_014, STC_016
GEN_002	STC_013, STC_016
GEN_003	STC_001, STC_003
GEN_004	STC_001, STC_002
GEN_005	STC_013, STC_016, STC_015
GEN_006	STC_013, STC_016
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
STA_012	STC_001, STC_003
STA_013	STC_001, STC_003
TRANS_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
TRANS_013	STC_001, STC_003
TRANS_014	STC_001, STC_003
TRANS_015	STC_003

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_012 without any support from developers.
Output	All volunteers read the user manual, install the necessary software and successfully conduct STC_012 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	• Build confidence that the user manual is readable and that the product is well documented. • STC_012 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	Ask each participant to highlight the sentence(s) that specify the 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	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: 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 5 people in their cars are stationed around the parking lot.
Input	Survey the 5 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	-
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_001, STC_002, STC_003
PERF_003	STC_004
PERF_004	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
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

Unit tests are important not just just for assessing the correctness of the final solution, but they also help developers narrow down issues during integration tests, such as those outlined within System Test Description. Each unit test is designed to utilize a singular module, as specified in the MIS. Sometimes the unit test pertains to a leaf module and at other times, due to the tight coupling between modules, an entire intermediate module is tested. The Unit tests are designed to verify a large, representative, and important amount of functionality in a given module, as basic functionality should be sufficient to start the extensive system testing within System Test Description. Each unit test may verify multiple functional and non-functional requirements.

6.1 Unit Testing Scope

Software modules that were not implemented by the team are not included in the Unit Testing (ROS, Ardupilot, Base Socket, and ROS-Mavlink Communication Driver).

Unit testing of the electric circuitry and the drone's hardware requires the usage of Ardupilot, but assuming Ardupilot is installed correctly the unit tests still isolate the electrical and mechanical components.

Rather than testing each leaf module independently, it sometimes makes sense to test the whole intermediate-level module. Some modules are tightly coupled with other modules, such as the Main Interface Module that was created just to execute the User Interface module, as such it makes sense to Unit Test the whole intermediate module, Interface Hiding Module. Likewise, the entire Drone Decision and Control (DDC) Hiding module is verified instead of testing each of Operation States, Operations Manager, and the Main DDC Module. Furthermore, when the Vision App, Mapper App, and Path Plan App are tested, the Algorithm Manager App and Main Algorithm Module are also used.

All modules have a high priority for being tested, except for the Mapper App, and Path Plan App, as these are stretch goals and not a part of the minimum viable product.

6.2 Tests for Functional Requirements

6.2.1 Hardware Hiding

The purpose of Unit testing the hardware is to ensure that the wiring and mechanical connections of all the various components is correct. Furthermore, the Unit tests are also designed to verify that the sensors are giving accurate readings.

Table 27: UTC_001

ID	UTC_001
Type	Functional, Manual, Dynamic
Initial State	Lay the drone flat on a surface. Remove the propellers and attach a piece of tape to each of the motors instead.
Input	Connect the battery and slowly raise the throttle on the RC Controller.
Output	Each motor should spin incrementally faster with time. Each motor should also spin in the prescribed direction.
How test will be performed	Input steps are self-explanatory.
Test case derivation	The drone can be controlled manually through an RC Controller or autonomously through Ardupilot. This test simply verifies the electrical connections.
Requirements	

Table 28: UTC_002

ID	UTC_002
Type	Functional, Manual, Dynamic
Initial State	Lay the drone flat on a surface outside.
Input	Connect the battery. Using the RC Controller, raise the throttle to fly the drone up. Then use the other joysticks on the RC Controller to make the drone fly forward, then backward, then fly left and finally fly to the right.
Output	The drone should fly up, then move forward longitudinally, then backward longitudinally, then left, and finally fly right.
How test will be performed	Input steps are self-explanatory.
Test case derivation	The drone can be controlled manually through an RC Controller or autonomously through Ardupilot. This test verifies the drone's mobility, which is mostly dependent on the electronics being connected correctly and the drone having an aerodynamic mass distribution.
Requirements	

Table 29: UTC_003

ID	UTC_003
Type	Functional, Manual, Static
Initial State	Lay the drone flat on a surface roughly 1m above the ground.
Input	Connect the battery. SSH into the drone and utilize MavROS to read Compass, Barometer and global GPS values.
Output	Compare the drone to the correct sensor values, they should be relatively close (within the tolerance specified by the Hardware manufacturer).
How test will be performed	The correct compass and GPS readings can be estimated using a smartphone. Using a measuring tape, an estimate of the correct Barometer reading can be made.
Test case derivation	This test case verifies that the Compass, Barometer, and GPS sensors are functional.
Requirements	

Table 30: UTC_004

ID	UTC_004
Type	Functional, Manual, Dynamic
Initial State	Lay the drone flat on a surface.
Input	Power on the drone using a USB C connector. SSH into the drone and utilize MavROS to read IMU Acceleration values. Pickup the drone slighly off the ground, move the drone forward, then place it on the ground. Pickup the drone slighly off the ground, move the drone rightward, then place it on the ground. Pickup the drone straight up. This concludes the test.
Output	When the drone is moved forward, the acceleration in the X should be positive. When the drone is moved rightward, the acceleration in the Y should be positive. When the drone is moved upward, the acceleration in the Z should be positive.
How test will be performed	Input steps are self-explanatory.
Test case derivation	The purpose of this test is to verify the orientation of the drone and the IMU sensors. The positive and negative conventions are made in accordance with the hardware manufacturer. The battery was not connected in this case to ensure that the drone's propellers do not spin and a human can safely handle the drone.
Requirements	

6.2.2 Operator Camera

The purpose of the Operator Camera module is to receive images from the Video Streamer, which streams live camera images on a specific port address. Thus it is sufficient to verify that live drone images can be collected.

Table 31: UTC_005

ID	UTC_005
Type	Functional, Manual, Static
Initial State	Power on the drone using a USB C connector.
Input	Pick up the drone so that one's face is within the camera view. Make various facial and hand gestures.
Output	The various gestures should be visible on the live camera video feed window created on the Operator's PC. Furthermore, the video feed should have a speed of at least 0.5 FPS.
How test will be performed	Write a stub within the Operator Camera module to collect and display the live images in a small window (e.g. using OpenCV). In this way the User Interface is not needed to conduct this test.
Test case derivation	The battery was not connected to ensure that the propellers do not turn on, and thus the drone can safely be handled by a human.
Requirements	GEN_002, PERF_004

6.2.3 Drone Camera

The purpose of the Operator Camera module is to receive images from the Drone Camera module, thus it is sufficient to verify that live drone images can be collected.

Table 32: UTC_006

ID	UTC_006
Type	Functional, Manual, Static
Initial State	Power on the drone using a USB C connector.
Input	Pick up the drone so that one's face is within the camera view. Make various facial and hand gestures.
Output	The various gestures should be visible as JPEGs in a folder on the Raspberry PI. Furthermore, the JPEGs should have a speed of at least 0.5 FPS.
How test will be performed	Write a stub within the Drone Camera module store live images in as JPEGs within any folder on the Raspberry Pi.
Test case derivation	The battery was not connected to ensure that the propellers do not turn on, and thus the drone can safely be handled by a human.
Requirements	GEN_002, PERF_004

6.2.4 Message Socket

The Message Socket module is used for bidirectional string communication between the drone and the Operator's PC. The Unit tests therefore simply verify that strings can be sent between the two platforms.

Table 33: UTC_007

ID	UTC_007
Type	Functional, Automatic, Static
Initial State	Power on the drone.
Input	Using the Message Socket on the drone, send an empty heartbeat message to the Message Socket on the Operator's PC. Using the Message Socket on the Operator's PC, send the string "{'Type': 'Command', 'Action':'None'}" to the Message Socket on the Drone.
Output	Empty heartbeat messages should be printed on the console of the drone, and "{'Type': 'Command', 'Action':'None'}" should be printed on the console of the Operator's PC. At least 1 heartbeat message must be printed every 2 seconds, and at least 1 "{'Type': 'Command', 'Action':'None'}" message should be printed every 2 seconds.
How test will be performed	A new python application on the Operator's PC will need to be written and run to complete this unit test. In the script, a Message Socket object should be created and continuously send the message "{'Type': 'Command', 'Action':'None'}" while printing all messages it has received to the console. Likewise a new python application on the drone will need to be written and run to complete this unit test. In the script, a Message Socket object should be created and continuously send the message empty heartbeat message, while printing all messages it has received to the console.
Test case derivation	The purpose of this test case is to verify string communication, as well as verify that the strings can be communicated at a sufficiently high speed (to meet the 0.5 frames per second requirement)
Requirements	PERF_004

6.2.5 Interface Hiding

The Interface Hiding module is used to interact with the user, to get commands from the Operator as well as visualize drone's results. It is an intermediate module, made up of the User Interface and Main User Interface. Several unit tests are created to verify the ability of the drone to display output, the ability to capture input, and verify the security requirements required by the User Interface.

Table 34: UTC_008

ID	UTC_008
Type	Functional, Automatic, Static
Initial State	-
Input	Run the testing script to hardcode a blank red image in the Operator's Camera, as well as hardcode the drone's heartbeat message shared by the Message Socket (the heartbeat message should specify an altitude of 5m, state of "Test", battery of 42%, "Unhealthy" health status, user error as "None", and a GPS coordinate of McMaster's Student Center.
Output	The live camera window should feature a blank red image. Furthermore, the User Interface should visually match the heartbeat message that was hard coded. In particular the GPS map should show the Drone as being located at McMaster's Student Center, health status should be "Unhealthy", the battery capacity should be 42%, altitude should be displayed as 5m, user error should be "None", and the drone state should be "Test".
How test will be performed	A stub will need to be written to hardcode the images from the Operator's Camera and the heartbeat message received from the Message Socket.
Test case derivation	The purpose of this test is to verify the ability of the User Interface to display output to the user.
Requirements	USE_005, SR_002

Table 35: UTC_009

ID	UTC_009
Type	Functional, Manual, Static
Initial State	The drone is off.
Input	The user enters an incorrect username and/or password. Then the user enters the correct username and password.
Output	When the Operator enters an incorrect username/password, the User Interface should prevent the Operator from using the user interface.
How test will be performed	Input steps are self-explanatory.
Test case derivation	The purpose of this test is to verify the security features of the User Interface.
Requirements	SR_013, SEC_001

Table 36: UTC_010

ID	UTC_010
Type	Functional, Manual, Static
Initial State	A volunteer is available.
Input	The volunteer goes through the list of inputs and outputs in the SRS, as well as the specifications specified in the Design Document (MIS).
Output	The volunteer makes sure that the user interface contains a means for capturing each input, a means for displaying each output, and implements the specification in the MIS.
How test will be performed	The input section is self-explanatory.
Test case derivation	The purpose of this test is to verify that the User Interface is complete.
Requirements	

6.2.6 Vision App

The Vision App contains two key functionalities. Firstly, the Vision App must accurately segment the parking lot from a given satellite image. The segmentation feature is done on the Operator's PC, prior to flight. Secondly, the Vision App must classify the central pixel of a given image as parking lot or non parking lot. This occupancy classification feature is ran live on the Raspberry PI.

Table 37: UTC_011

ID	UTC_011
Type	Functional, Automatic, Static
Initial State	-
Input	Atleast 50 images from any of the public parking lot camera datasets are fed into the Occupancy Classifier feature, running on the Raspberry PI. The Occupancy Classifier assigns the central pixel as either "Parking Lot", "Nature", or "Occupied". At least 20% of the dataset should be during sunny conditions, and 20% of the dataset should be from overcast decisions. The camera should be within at a height within 7m and 20m. Atleast 5% of the dataset should be from low light conditions (evening or night) images. The camera can be tilted at any angle so long as the sky is not visible in the image.
Output	Measure the average time taken processing a single image (FPS), and use it to verify the 0.5 frames per second requirement. Also, measure the accuracy of the classification.
How test will be performed	The test will be through a special unit testing script. The script should hardcode the image from one of the datasets, run the algorithm, and record the result on some kind of table.
Test case derivation	Atleast 50 images were selected because they are an ample testing size.
Requirements	GEN_005

Table 38: UTC_012

ID	UTC_012
Type	Functional, Automatic, Static
Initial State	-
Input	12 satellite images of parking lots are fed into the Vision App's parking lot segmentation feature.
Output	Measure the accuracy of the segmentation, pixels correctly classified divided by the total number of pixels.
How test will be performed	The test will be through a special unit testing script. The script should hardcode the image to be a satellite image of a parking lot, run the algorithm, and record the result in some kind of table.
Test case derivation	12 images were selected because they are an ample testing size.
Requirements	GEN_001

6.2.7 Mapper App

The purpose of the Mapper App is to verify the ability of the drone to create an occupancy map. Unit testing the Mapper App requires a verified and correct Vision App.

Table 39: UTC_013

ID	UTC_013
Type	Functional, Automatic, Static
Initial State	Have the drone powered on.
Input	Feed a prerecorded video of the drone flying over a parking lot into the Vision App.
Output	Compare the final occupancy map to the actual occupancy map of the parking lot in the video.
How test will be performed	The test will require writing a stub in the Vision App to use the prerecorded videos instead of live camera images.
Test case derivation	The purpose of this test is to verify the occupancy map functionality. The Mapper App depends on the results of the Vision App. A sample video of the drone flying over a parking lot is fed into the vision app, by keeping the vision app outputs constant the test is repeatable.
Requirements	

6.2.8 Path Plan App

There are no unit tests for the Path Plan App because it requires closed-loop testing. The Path Plan App suggests a subsequent path for exploration, assessing its performance requires the drone to actually follows the suggested path at each time step (i.e. requires closed-loop testing in SITL or live with the drone). The Path Plan App is also a stretch goal, so it should be developed once all the other modules have been verified and tested. Thus STC_016 can be used as a test case for developing the Path Plan App.

6.2.9 Drone Decision and Control Hiding

The purpose of this module is to verify the Drone Decision and Control Module. It is an intermediate module that is designed to verify the highly-coupled leaf modules Operation States, Operations Manager, and Main DDC Module. The unit tests are representative because they cover all possible Operation states (but not all transitions). Many of the system tests can be turned into Unit tests by hard-coding the outputs of other modules (such as User Interface, Vision App, etc.). The unit tests are also conducted in SITL, as testing outdoors with the real drone would require the Hardware Hiding module (and thus be a system/integration test).

Table 40: UTC_014

ID	UTC_014
Type	Functional, Automatic, Static
Initial State	The drone is in the Idle State.
Input	Send a user command to configure the height parameters to Min_Hover_Params.
Output	The height parameters stored in the 'Params.txt' file should be Min_Hover_Params. The height parameters of the drone should also be Min_Hover_Params.
How test will be performed	A stub will need to be written in the Message Socket to mimic as if the user wants the height parameters to be set (e.g. MessageSocket should return "{'Type':'Configure', 'Min':7m, 'Des':7m, 'Max':7m}". A stub will need to be written to print the height parameters of the drone and the current state to the console.
Test case derivation	A height configure command should make the drone enter the Configure state, which changes the height parameters of the drone. 'Params.txt' is a file used to permanently store the height parameters, so that they can be used the next time the drone boots up.
Requirements	STA_004, GEN_003, TRANS_003, TRANS_015

Table 41: UTC_015

ID	UTC_015
Type	Functional, Automatic, Static
Initial State	The drone is in the Idle State.
Input	Send a user command to arm the drone, takeoff, move 5m to the left, and finally land at the original launch location.
Output	The drone should arm, takeoff to the maximum hover height, move 5m left, and then land at the original location.
How test will be performed	A stub will need to be written in the Message Socket to mimic as if the user wants the drone to arm, takeoff, move left 5m, and then land (e.g. MessageSocket should chronologically return "{'Type':'Arm'}", "{'Type':'Takeoff'}", "{'Type':'Compulsive Move', 'X':-5, 'Y':0 }", and "{'Type':'Land'}"). Although the drone can be monitored by the SITL window, a stub can also be written to print the current drone location, the current state, and the current altitude to the console.
Test case derivation	The test case is designed to make the drone enter the Arm, Takeoff, Hover, Compulsive Move, and Land states. It, therefore, verifies the behavior of each of the states.
Requirements	STA_000, STA_001, STA_004, STA_011, STA_012, STA_013, TRANS_003, TRANS_009, TRANS_012, TRANS_013, TRANS_014

Table 42: UTC_016

ID	UTC_016
Type	Functional, Automatic, Static
Initial State	The drone is in the Idle State.
Input	Send a user command to arm the drone and takeoff.
Output	The drone should arm and take off. After a sufficient amount of time, the battery level will decrease below 20% capacity, at which point the drone enters the malfunction state and lands.
How test will be performed	A stub will need to be written in the Message Socket to mimic as if the user wants the drone to arm and takeoff (e.g. MessageSocket should chronologically return "{'Type':'Arm'}" and "{'Type':'Takeoff'}"). Although the drone can be monitored by the SITL window, a stub can also be written to print the current drone location, the current state, and the current altitude to the console.
Test case derivation	The test case is designed to verify the Malfunction state.
Requirements	STA_009, SR_007, SR_011

Table 43: UTC_017

ID	UTC_017
Type	Functional, Automatic, Static
Initial State	The drone is in the Idle State.
Input	Hardcode the Message Socket to be disconnected once the drone takeoffs.
Output	The drone should arm and take off. After a sufficient amount of time, when the Message Socket is hardcoded to be disconnected, the drone should enter the Communication Lost state and land.
How test will be performed	A stub will need to be written in the Message Socket to arm the drone, takeoff the drone, and then disconnect. Although the drone can be monitored by the SITL window, a stub can also be written to print the current drone location, the current state, and the current altitude to the console.
Test case derivation	The test case is designed to verify the Malfunction state.
Requirements	STA_010, TRANS_010

Table 44: UTC_018

ID	UTC_018
Type	Functional, Automatic, Static
Initial State	The drone is in the Idle State.
Input	Arm and takeoff the drone. Then send a user command to enter the autonomous explore state while no parking lot is detected.
Output	After the drone takes off, it should enter the No Parking Lot Detected Error State.
How test will be performed	A stub will need to be written in the Message Socket to arm the drone, takeoff, and then request the autonomous explore feature (e.g. MessageSocket should chronologically return "{'Type':'Arm'}", "{'Type':'Takeoff'}", and "{'Type':'Autonomous Explore'}"). The Vision App will need a stub to constantly publish "No Parking Lot Detected" on the Algorithm Topic Interface. Although the drone can be monitored by the SITL window, a stub can also be written to print the current drone location, the current state, and the current altitude to the console.
Test case derivation	The test case is designed to verify the No Parking Lot Detected Error state. This state is entered when the user requests an autonomous explore operation but no parking lot is detected.
Requirements	STA_008, TRANS_008

Table 45: UTC_019

ID	UTC_019
Type	Functional, Automatic, Static
Initial State	The drone is in the Idle State.
Input	Arm and takeoff the drone. The Vision App should constantly be returning parking lot detected while the path planning algorithm should suggest a path of 5m to the left.
Output	After the drone takes off, it should enter the Autonomous Explore state and begin to move left 5m.
How test will be performed	A stub will need to be written in the Message Socket to arm the drone and then takeoff. The Vision App will need a stub to constantly publish "Parking Lot Detected" on the Algorithm Topic Interface. The Mapper App will need a stub to constantly publish a suggested exploration path of 5m left. Although the drone can be monitored by the SITL window, a stub can also be written to print the current drone location, the current state, and the current altitude to the console.
Test case derivation	The test case is designed to verify the Autonomous Explore state. This state is entered when automatically when the drone is hovering and detects a parking lot.
Requirements	STA_003, TRANS_004

6.2.10 DDC Topic Interface

The Topic interface contains numerous functions to receive and publish data, as such only a single-topic subscription and single-topic publishing are tested.

Table 46: UTC_020

ID	UTC_020
Type	Functional, Automatic, Static
Initial State	The drone is powered on.
Input	Use the state publisher (currStatePub) to publish "Unit test 101". Turn the drone counterclockwise.
Output	"Unit test 101" should be published on the currStatePub topic. Using the compass in a smartphone, the DDC Topic Interface's reading of the compass values should roughly match the readings of the smartphone.
How test will be performed	Write a stub to print out the orientation recorded by the DDC Topic Interface (part of the local pose struct). The string being published on currStatePub can be monitored in a console, by using ROS's command line interface for subscribing to and printing data being published to a topic.
Test case derivation	The purpose of this test is to verify the DDC Topic Interface
Requirements	

6.2.11 Algorithm Topic Interface

The Topic interface contains numerous functions to receive and publish data, as such only a single-topic subscription and single-topic publishing are tested.

Table 47: UTC_021

ID	UTC_021
Type	Functional, Automatic, Static
Initial State	The drone is powered on.
Input	Use the vision app health publisher (visionAppHealth) to publish True. Publish "Unit test 101" on the "current state" topic, and print the drone state being read by the Topic Interface to the console.
Output	"Unit test 101" should be printed to the console. The visionAppHealth topic should be publishing True.
How test will be performed	The string "Unit Test 101" can be published on the current state topic using ROS's command line interface for publishing data to a topic. Write a stub to print out the current state recorded by the Algorithm Topic Interface to the console.
Test case derivation	The purpose of this test is to verify the Algorithm Topic Interface
Requirements	

6.2.12 DDC Service Interface

The Service interface contains routines to call five services, all four are tested in the following unit test. The test is performed in SITL.

Table 48: UTC_022

ID	UTC_022
Type	Functional, Automatic, Static
Initial State	The drone is in the Idle state.
Input	Call each of the service routines to set the RtlAlt to 10m, call the mode service to set the mode to "Guided", call the arm service to arm the drone, call the takeoff service to takeoff the drone, and then call the landing service (using the callService_TypeCommand routine).
Output	The drone should takeoff the ground to a height of 10m, and then lands.
How test will be performed	A unit testing script must be written to create a service interface object and then call each of the routines in the Input section in the chronological order it specified.
Test case derivation	The purpose of this test is to verify all of the service interface routine.
Requirements	

6.3 Traceability Between Test Cases and Modules

This section summarizes the traceability between Test Cases and Modules, see Table 49.

Table 49: Traceability Between Test Cases and Modules

Hardware Hiding	UTC_001, UTC_002, UTC_003, UTC_004
Operator Camera	UTC_005
Drone Camera	UTC_006
Message Socket	UTC_007
Interface Hiding	UTC_008, UTC_009, UTC_010
Vision App	UTC_011, UTC_012
Mapper App	UTC_013
Drone Decision and Control Hiding	UTC_014, UTC_015, UTC_016, UTC_017, UTC_-018, UTC_019
DDC Topic Interface	UTC_020
Algorithm Topic Interface	UTC_021
DDC Service Interface	UTC_022

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 50: Symbolic Constants

ID	Symbolic Constants
Invalid_Hover_Params1	(i_MinHoverHeight=1m, i_-MaxHoverHeight=1m, i_DesiredHover-Height=1m)
Invalid_Hover_Params2	(i_MinHoverHeight=8m, i_-MaxHoverHeight=7m, i_DesiredHover-Height=6m)
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 4.3 and evaluates each team member on the graduate attribute of Lifelong Learning.

Table 51: Required Testing

Test	Responsibility	Rationale
Dynamic Testing	Fady, Muhammad, Zaid & Winnie	Dynamic testing is vital, especially for flying a drone. Dynamic testing is executing test cases in variable environments to analyze how the system behaves.
Integration Testing	Muhammad	Integration testing is important as it will test different units, modules, or components of the drone as a combined entity.
Static Testing	Fady, Muhammad & Winnie	Static testing involves matching the requirements from the SRS to the code written. Static testing also involves looking into code format and errors.
Visual Perception and Path Planning Testing	Fady	Verifies that the visual perception and autonomous exploration algorithm is performing within specifications.
Drone Finite State Machine (FSM) and Communication Testing	Ali	Verifies that all communication between the drone components and the drone to the Operator's application is working correctly.
Mechanical Testing	Zaid	Verifies that all physical components and the dynamics are working within specifications.
Operator's Application Testing	Winnie	Verifies that the Operator's application and user manual meet the specifications outlined.

7.2.1 Knowledge & Skills

Table 52 covers different approaches to follow the plan from Table 51.

Table 52: Knowledge Acquisition

Test	Approaches	Verdict
Dynamic Testing	Approach 1: Previous lecture notes from McMaster's MECHTRON 3K04: Software Development. Approach 2: Research online and watch tutorials such as this video on YouTube. Approach 3: Research online and find resources for mechanical static testing such as this.	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 Testing	Approach 1: Previous lecture notes from McMaster's MECHTRON 3K04: Software Development. Approach 2: Research online resources such as this website.	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	Approach 1: Previous lecture notes from McMaster's MECHTRON 3K04: Software Development Approach 2: Research online and find resources for static testing such as this website.	Fady, Muhammad & Winnie will proceed with Approach 1 as Static Testing was done extensively in the MECHTRON 3K04 Pacemaker project.
Visual Perception and Path Planning Testing	Approach 1: Watch McMaster's Google Developer's Club video on YouTube to learn more about the OpenCV library complementing ENG 1D04: Engineering Computation. Approach 2: Use OpenCV tutorial documentation.	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	Approach 1: Previous lecture notes from McMaster's MECHTRON 3K04: Software Development Approach 2: Use ROS.org documentation on ROS' library SMACH to learn more about the FSM implementation.	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	Approach 1: Previous lecture notes from McMaster's ENG 1C03: Engineering Design & Graphics Approach 2: Watch SolidWorks' basic tutorial.	Zaid will use Approach 2 to get familiar with SolidWorks.
Operator's Application Testing	Previous lecture notes from McMaster's MECHTRON 1D04: Engineering Computation to understand the language of the application. Approach 2: Use Python Documentation and other online resources to understand Python syntax.	Winnie will proceed with Approach 1 as ENG 1D04 had covered Python extensively.