**Group Member NAMES: Ryan Sauer, Demetrius Johnson,**

**Jonathan Schall & Olivia Pellegrini**

**7-17-23**

***Test Specifications***

# **Table of Contents**

[**Table of Contents 1**](#_dkop2mmhn1fn)

[**1.0 Introduction 2**](#_jrsfza6balm9)

[1.1 Goals and objectives 2](#_r7ouxmlv78ex)

[1.2 Statement of scope 2](#_6qyh9kl1obcb)

[1.3 Major constraints 2](#_wtgnxtor7swj)

[**2.0 Test Plan 3**](#_xclrb4dt9nj8)

[2.1 Software Configuration Items (SCIs) to be tested 3](#_70v5w86y06s7)

[2.2 Testing strategy 5](#_ny7d648ruldn)

[2.2.1 Unit testing 5](#_ijuc3tisiap)

[2.2.2 Integration testing 5](#_k3m686vqhhm5)

[2.2.3 Validation testing 5](#_jbzw1lb4zl98)

[2.2.4 High-order testing 6](#_hkafce48gy8z)

[2.3 Testing resources and staffing 6](#_ez1rl1p2q29d)

[2.4 Test work products 6](#_wfqhb28kj9bx)

[2.5 Test record keeping 7](#_7663bdekwf9j)

[2.6 Test metrics 7](#_u9pvpzfwvsim)

[2.7 Testing tools and environment 7](#_8utu5cndhgb2)

[2.8 Test schedule 7](#_wxmb5bho45nd)

[**3.0 Test Procedure 8**](#_6b9vd4rqaufm)

[3.1 Software (SCIs) to be tested 8](#_msulz0lzt3a2)

[3.2 Testing procedure 9](#_mcot18nzqkjk)

[3.2.1 Unit test cases 9](#_ui5faph1tum6)

[3.2.2 Integration Testing 31](#_oiy85671mvjn)

[3.2.3 Validation Testing 34](#_nmjkyijx7o2y)

[3.2.4 High-order testing (a.k.a. System Testing) 34](#_drnljnabd0uq)

[3.3 Testing resources and staffing 35](#_dzwj3hopchi5)

[3.4 Test work products 36](#_ngeq6998fff6)

[3.5 Test record keeping and test log 36](#_g23ddsad6uog)

[**Traceability Matrix 37**](#_zpsoil3zj4g)

# 

# 1.0 Introduction

This section gives a general overview of the Sky Socket project testing specifications.

## 1.1 Goals and objectives

Sky Socket is a tool used to aid in the research and development of drone-car collaboration. All users are expected to be relatively expert users within the research area, however, this does not mean that they will be familiar with the testbed and requirements for the system to work. This means the program should be simple and easy to use, but flexible and reliable. In order to reach these goals, it is important that our product is as bug free as possible.

The testing process for Sky Socket has several goals. The software will be vigorously tested for logic errors and bugs as well as responsiveness and ease of use. The final product should feel approachable and understandable for any user with the appropriate qualifications to research such an area of engineering. Beyond that, we would like our product to feel easier to use than alternatives while providing the dependability and flexibility of said competitors.

## 1.2 Statement of scope

There will be several layers of testing. First will be unit testing. This will be done via coded white box testing methods via Python’s built-in unit test framework. Next will be integration testing using black box tests via unit testing once again. Validation testing will also be applied along with high order system testing with several controlled, system-wide tests being run.

Many design specifications will be validated during testing including a simple and easy-to-use interface, accurate and robust file interactions, and a reliable “click and play” UI experience. The GUI should properly interact with the desired devices, locate files accurately, and maintain a reasonable refresh rate and response time.

The manual must be complete enough for even the average user to install and run the entire system. There should also be examples in the form of multimedia (screenshots, etc.) to complement the written instructions. Installation instructions should encompass all aspects necessary in order to use our open-source product. Uninstallation should also completely remove all files installed onto target machines.

## 1.3 Major constraints

* Sky Socket has a deadline of August 14th, 2023. This will limit the testing time, especially for such a robust research framework that we envisioned.
* The team has 4 developers but only 1 set of hardware for the testing system. All tests will be run on the target hardware, but with limited access time.
  + This will be mitigated by thorough unit and integration testing on nontarget hardware before high level testing.
* Target hardware is expected to be no more than the middle line Raspberry Pi running RaspbianOS.
* Sky Socket GUI is expected to run only on ***Windows 10*** machines.
* During integration testing or system testing when the drone is actually flying, there is a big risk that the flight mechanism can be damaged due to some known or unknown error or due to faulty hardware that is out of our control; in fact, the drone could be destroyed altogether. We do not have any spare drones, and drone repair time (if even possible) would take up a lot of time, and we would have to do possibly a lot of reinstallations and thus run testing from the very beginning.
  + To mitigate this, our backup plan would be to use a node that is not really a drone but that can still run most of our components and act as a drone.

# 2.0 Test Plan

This section describes the overall testing strategy and the project management issues that are required to properly execute effective tests on Sky Socket.

## 2.1 Software Configuration Items (SCIs) to be tested

The software items to be tested are identified by name. Exclusions are noted explicitly:

*Graphical User Interface (GUI)*

* Simulation Tab
  + Video Player
  + Terminal
  + Connection window
  + Start Button
* Testing Tab
  + Terminal
  + Test window
  + SSH window
* Settings Tab
  + Text inputs
  + IP inputs
  + Link inputs
  + Save/Load dialogs
  + Buttons
* Connections
  + SSH connections
  + Python script running
  + iPerf3 test running
* Files
  + Folder validation
  + File creation
  + File writing
  + Remote file validation
  + Remote folder validation
* Installation and Uninstallation

*Vision Systems*

* Adjusting video frame size and color
  + Adjust size of input width and height
  + OpenCV Color conversion
* Detect Red
  + Apply HSV mask to video frame
  + Get contours
  + Draw bounding box
* Drone Detects Shape
  + Count number of sides
  + Display object name
* Training Custom Model
  + Label image files
  + Upload files to darknet environment
  + Prediction confidence displayed for custom dataset
* Deploy Model on Edge Device
  + Setup darknet on device
  + Identify objects in single images
  + Identify objects on live video feed
  + Display confidence threshold

*Communication and Control Systems*

* Wireless Local Area Network (WLAN) Connectivity
  + Drone WLAN broadcast and DHCP Server Functionality (via DNSmasq and WPA supplicant ran on the Drone)
  + Raspberry Pi Carx Node WLAN connectivity
  + Edge Server (Windows 10 Computer) Node WLAN connectivity
  + Windows 10 Computer with GUI Node WLAN connectivity
* ROS topic publications of the drone via web server interface provided by the drone
  + Image topics (and thus test the camera is operational)
  + Battery life topic
  + Flight control status topic (i.e. manual or autonomous mode, armed or disarmed, flying or static, etc.)
* Drone flight tests
  + Manual flight test
    - Including battery life testing
  + Hover program
    - Including battery life testing
  + Oscillation program
    - Including battery life testing
  + Hardware and software status check program (selfcheck.py)
  + Calibration tests (i.e. sensor calibrations)
  + Visual inspection of the mechanical and electrical connection components
* Car driving tests
  + Calibration tests (i.e. forward and reverse frame of reference for driving controls)
  + Driving tests - hardware (i.e. motor) and software (respond to software commands from picarx library) checks
    - forward/backwards
    - Left and right turn
    - Stop and go
  + Visual inspection of the mechanical and electrical connection components
  + Battery life testing
* Car command server program
  + Test the command server connectivity on the picarx car
    - Drone, edge server, and GUI connectivity
  + Test the command keys (values for changing speed, direction, stop, go, etc.)
  + Test that command keys can be received from the drone client program
  + Test that command keys can be received from the edge server client program
  + Test that command keys can be received from the GUI client program
* Drone client program
  + Test the client program connectivity
    - Car command server, edge server, and GUI connectivity
  + Test that command keys can be sent to the car command server program
  + Test that drone can send frames to edge server
* Edge-server node client program
  + Test the client program connectivity
    - Car command server, drone (i.e. receiving frames), and GUI connectivity
  + Test that command keys can be sent to the car command server program
  + Test that edge server can receive frames from the drone for processing

## 2.2 Testing strategy

The following will describe the testing strategy Sky Socket will use.

### **2.2.1 Unit testing**

Unit tests will be written as code is written. The programmer who programs a method or class will write unit tests for as high of coverage as reasonable. They will perform the testing using expected inputs and outputs. The programmer will be on the lookout for any erroneous inputs and outputs. This will ensure that each part works prior to Integration testing.

Testing the vision system is unique. Since the input is taken in the form of video frames, (rather than a single variable or parameter), we can input different images into our program.

Each module will aim to have near full coverage via unit testing. The programmer who programs said module will write the tests and run them, confirming that all tests pass. The unit testing phase for a module will be considered complete when another member of the team has run the same test on separate hardware and also achieved a 100% pass rate.

### 2.2.2 Integration testing

Sky Socket will overall be implementing bottom up integration testing where components will be created and tested then slowly put together to form the full product. However, within each component, there are some differences.

Sky Socket GUI will also follow bottom up integration testing. As components are made, they are incorporated into the UI and tested for completeness and compatibility. However, the computer vision system will undergo top down testing to function. The entire project's integration testing will be in a black box style. Tests will simply have expected outputs per input and pass/fail will be determined via the outputs.

Finally, the integration testing will consist of two major degrees of integration that includes all components: the first degree is to test that all components of the system have been properly integrated during nonflight. The second degree of integration is to integrate flight while also verifying (and reintegrating if necessary) all components that were already integrated during nonflight.

Once these two degrees of integration are complete, then we can begin actually testing our system (system testing) because we know that all components have been successfully integrated even while the drone is actually flying, which is critical, since flying the drone is a major aspect of the project that can also introduce a lot of uncertainty into many of the components of our system that we expect to work from nonflight integration (i.e. is the camera still working, are all nodes still communicating, are topics being published, is the drone’s processing power enough or is it overworked once the drone is flying?, etc.).

### 2.2.3 Validation testing

Validation testing will be completed alongside our client. We will thoroughly examine the requirement specifications and ensure all requested components and features are present. Should we find any discrepancies or confusion, items will be added to the defect list for clarification or repair. Items that were specified in the requirements that were found to be missing or unsatisfactory will be given high priority for completion. Should something unspecified be brought up, the item will also be added to the defect list, but priority will be low.

A set of controlled and predefined tests will be run on all parts of the system once it has been fully integrated with our client to ensure quality and compliance.

### 2.2.4 High-order testing

Once the software coding is complete, there will be a suite of tests run to ensure the high level function.

Stress Testing: Rapid and unexpected inputs will be applied and the system is expected to be robust enough to resolve confusing or erroneous inputs without causing harm to the hardware, crashing the program, or creating unexpected results.

Performance Testing: The system should be able to run on the hardware defined in the design specifications. The system should be able to perform well with no bottlenecks given the expected hardware. The GUI should also run with relatively low hardware stress and a low amount of network impact.

Alpha/Beta Testing: The system will be tested through simulations to ensure that the whole system is accurate. Alpha will be done by the team and beta testing will be done with some independent testers and the client.

## 2.3 Testing resources and staffing

We will use many resources to carry out the tests on our software and hardware. Since time is one of the largest constraints for our testing, we will attempt to utilize our human resources as efficiently as possible.

* Sky Socket Staff
  + Test Team Lead: Demetrius Johnson
  + Unit Test Lead: Ryan Sauer
  + Integration Test Lead: Jonathan Schall
  + High Level Test Lead: Olivia Pellegrini
* Sky Socket Staff computers
* Google Colab
* Darknet, YOLOv4, OpenCV
* 1 Clover Drone
* 1 PiCarX
* 1 UMD computing device
* 3rd Party Testing Assistance:
  + Validation Testing Assistance: Professor Zheng Song
  + Validation and High Level Test Assistance: Khairul Mottakin

## 2.4 Test work products

A custom widget was created in Kivy that allows us to monitor system usage details which will allow for performance metrics to be calculated and tested.

## 2.5 Test record keeping

Test records will be created, monitored, and held in a shared google drive using Google Docs and Sheets. Feedback from testing assistants and the Sky Socket team will be stored in an adjacent folder. For more information related to test time keeping, see section 3.5.

## 2.6 Test metrics

* Bytes: The metric used to measure memory usage via Sky Socket GUI
* Percent Usage: the metric used to measure the volume of CPU and GPU usage taken by Sky Socket GUI
* mAP: mean average precision. This is used during training as a recall value to show how precise our object detection model is.
* Confidence level: a percentage shown by our bounding box to show the user the confidence of a prediction.
* Gigabits and Megabits per second (Gb/s Mb/s): The metric used for bandwidth and network usage.
* Battery percentage (referring to the voltage of the battery compared to the maximum voltage capacity of the battery).

## 2.7 Testing tools and environment

The testing tools used for testing include Python’s unittest framework, the custom kivy widget, and the system hardware. Google Colab was used while training our custom dataset for object detection. It used darknet, YOLOv4-tiny and OpenCV for training. The Environment used by the hardware is the RaspianOS. Windows 10 will be used for Sky Socket GUI testing.

## 2.8 Test schedule

* Unit Testing:
  + Active throughout programming phase
  + March 13th - July 31st
* Integration Testing:
  + Active during major intersections
  + April 3rd - April 7th
  + July 17th - August 4th
* Validation Testing:
  + Active once project programming is complete
  + August 5th - 7th
* High Order Testing:
  + Active once project programming is complete
  + August 5th - August 11th

# 3.0 Test Procedure

This section describes a detailed test procedure including test tactics and test cases for the software.

## 3.1 Software (SCIs) to be tested

The software to be tested is identified by name. Exclusions are noted explicitly.

a) GUI

* Status Bar
* Simulation Tab
  + Video Player
  + Terminal
  + Connection window
  + Start Button
* Testing Tab
  + Terminal
  + Test window
  + SSH window
* Settings Tab
  + Text inputs
  + IP inputs
  + Link inputs
  + Save/Load dialogs
  + Buttons
* Connections
  + SSH connections
  + Python script running
  + iPerf3 test running
* Files
  + Folder validation
  + File creation
  + File writing
  + Remote file validation
  + Remote folder validation

b) Vision System

* Adjusting video frame size and color
* Bounding box overlay
* Confidence level

c) Communication and Control Systems

* Wireless Local Area Network (WLAN) Connectivity
  + Drone WLAN broadcast and DHCP Server Functionality (via DNSmasq and WPA supplicant ran on the Drone)
  + Raspberry Pi Carx Node WLAN connectivity
  + Edge Server (Windows 10 Computer) Node WLAN connectivity
  + Windows 10 Computer with GUI Node WLAN connectivity
* ROS topic publications of the drone via web server interface provided by the drone
  + Image topics (and thus test the camera is operational)
  + Battery life topic
  + Flight control status topic (i.e. manual or autonomous mode, armed or disarmed, flying or static, etc.)
* Drone flight tests
  + Manual flight test
    - Including battery life testing
  + Hover program
    - Including battery life testing
  + Oscillation program
    - Including battery life testing
  + Hardware and software status check program (selfcheck.py)
  + Calibration tests (i.e. sensor calibrations)
  + Visual inspection of the mechanical and electrical connection components
* Car driving tests
  + Calibration tests (i.e. forward and reverse frame of reference for driving controls)
  + Driving tests - hardware (i.e. motor) and software (respond to software commands from picarx library) checks
    - forward/backwards
    - Left and right turn
    - Stop and go
  + Visual inspection of the mechanical and electrical connection components
  + Battery life testing
* Car command server program
  + Test the command server connectivity on the picarx car
    - Drone, edge server, and GUI connectivity
  + Test the command keys (values for changing speed, direction, stop, go, etc.)
  + Test that command keys can be received from the drone client program
  + Test that command keys can be received from the edge server client program
  + Test that command keys can be received from the GUI client program
* Drone client program
  + Test the client program connectivity
    - Car command server, edge server, and GUI connectivity
  + Test that command keys can be sent to the car command server program
  + Test that drone can send frames to edge server
* Edge-server node client program
  + Test the client program connectivity
    - Car command server, drone (i.e. receiving frames), and GUI connectivity
  + Test that command keys can be sent to the car command server program
  + Test that edge server can receive frames from the drone for processing

## 3.2 Testing procedure

In this section we will describe the test procedures for Sky Socket in detail.

### 3.2.1 Unit test cases

The procedure for unit testing is described for each software component (that will be unit tested) is presented.

#### 3.2.1.1.a Testing Procedure for Component: GUI Status Bar

The status bar will be tested in a white box fashion. There are 4 statuses that can be displayed and 3 different colors for the background. All states must be tested to ensure user feedback.

#### 3.2.1.1.b Stubs and/or drivers for component: GUI Status Bar

#### 

Kivy’s event handler

#### 3.2.1.1.c Test cases component: GUI Status Bar

* Connection set to offline
* Connection set to online
* Connection set to connected
* Connection set to active
* All connections are active
* All connections are online
* All connections are connected
* All connections are offline
* One connection is online, the rest go through all stages
* One connection is connected, the rest go through all stages

#### 

#### 3.2.1.1.d Purpose of tests for component: GUI Status Bar

The purpose of these tests is to ensure the correct user feedback for the connection status of the hardware as well as accurate awareness of connection status.

#### 3.2.1.1.e Expected results for component: GUI Status Bar

The status bar to display the selected status at the appropriate time and for the color to display the lowest common activity.

#### 3.2.1.2.a Testing Procedure for Component: GUI Connection Window

The connection window will be tested in a white box fashion. It will receive user input via buttons and perform commands based upon the given input. SSH connections are established and monitored via these presses, so connections should be created and monitored asynchronously and accurately.

#### 3.2.1.2.b Stubs and/or drivers for component: GUI Connection Window

Kivy’s event handler

SSH

Fabric (Python SSH module)

#### 

#### 3.2.1.2.c Test cases component: GUI Connection Window

* Hardware Online
  + Display accurately
  + Ping button works
  + Ping button fails
  + Connect button works
  + Connect button failed
  + Connected displays accurately
  + Restart Button works
* Hardware Offline
  + Display accurately
  + Ping button works
  + Ping button fails
  + Connect button doesn't work
  + Restart button doesnt work
  + Connected displays false

#### 

#### 3.2.1.2.d Purpose of tests for component: GUI Connection Window

This interface is the main interaction for the user to run simulations and experiments, thus these buttons are the most used area of the program and must be consistent in all situations and edge cases.

#### 3.2.1.2.e Expected results for component: GUI Connection Window

The component is expected to perform within the design specifications by displaying accurate information, give responsive feedback, and prevent error states or race conditions.

#### 3.2.1.3.a Testing Procedure for Component: GUI Start Button

The start button will be tested in a white box fashion. It will receive user input then run a set method to run multiple programs across multiple pieces of hardware.

#### 3.2.1.3.b Stubs and/or drivers for component: GUI Start Button

Kivy’s event handler

Unittest

#### 

#### 3.2.1.3.c Test cases component: GUI Start Button

* Pressed when not ready (unpressable)
* Pressed to start
* Pressed during startup sequence (unpressable)
* Pressed during connection (stop)
* Pressed during shutdown sequence (unpressable)
* System stops being active without button press

#### 

#### 3.2.1.3.d Purpose of tests for component: GUI Start Button

The start button is the button to initiate tests and simulations, this is a critical button that can never cause errors or issues with the program so all edge cases must be taken into account.

#### 3.2.1.3.e Expected results for component: GUI Start Button

The start button is expected to be unpressable during startup and shutdown sequences, unpressable when not available, and should change to a stop button while a test is running.

#### 3.2.1.4.a Testing Procedure for Component: GUI Terminal Window

The terminal will be tested in a white box fashion. Output should be sent to the terminal and displayed with the source identified. The text should be scrollable and default to show the most recent output.

#### 3.2.1.4.b Stubs and/or drivers for component: GUI Terminal Window

Kivy Event Handler

Unittest

#### 3.2.1.4.c Test cases component: GUI Terminal Window

* Standard output from each source to terminal window
* Error output from each source to terminal window
* Enough text to scroll to terminal window

#### 

#### 3.2.1.4.d Purpose of tests for component: GUI Terminal Window

The terminal window will be the primary source of user feedback for debugging and understanding how the system is currently handling the test. Without a satisfactory or functional terminal, there will be limited user feedback.

#### 3.2.1.4.e Expected results for component: GUI Terminal Window

The terminal is expected to behave like a normal terminal application and as specified in the System Requirement Specification document.

#### 3.2.1.5.a Testing Procedure for Component: GUI Testing Window

The testing window will be tested in a white box fashion. Checkboxes should provide user feedback, text input should be validated, and buttons should initiate methods.

#### 3.2.1.5.b Stubs and/or drivers for component: GUI Testing Window

#### 

Kivy Event Handler

Unittest

#### 3.2.1.5.c Test cases component: GUI Testing Window

* Each checkbox clicked for expected feedback
* Valid checkbox input
* Invalid checkbox input
* Button press without test being run
* Button press with test currently active (disabled)

#### 

#### 3.2.1.5.d Purpose of tests for component: GUI Testing Window

The testing window is an important tool for gathering connection information regarding the hardware, without this module, there will be a long and laborious process to manually test these features.

#### 3.2.1.5.e Expected results for component: GUI Testing Window

The inputs should handle user feedback appropriately and the button should initiate a test when pressed and ignore presses while a test is in progress.

#### 3.2.1.6.a Testing Procedure for Component: GUI SSH Connect Window

The SSH connect window will be tested in a white box fashion. The spinbox should attach the corresponding session, the text input should be validated, the connect button should call the correct method, and the send button should run the command.

#### 3.2.1.6.b Stubs and/or drivers for component: GUI SSH Connect Window

Kivy Event handler

Unittest

#### 

#### 3.2.1.6.c Test cases component: GUI SSH Connect Window

* Each spinwheel option selected
* The connect button clicked when a connection exists
* The connect button clicked when a connection doesn't exist
* An empty command given
* A valid command given
* The send button hit when a command isn’t being run
* The send button hit when a command is being run (disabled)

#### 

#### 3.2.1.6.d Purpose of tests for component: GUI SSH Connect Window

Interfacing with multiple pieces of hardware is difficult and laborious, thus having this window be functional allows for a single interface to communicate between each individual computer.

#### 3.2.1.6.e Expected results for component: GUI SSH Connect Window

Commands should be run to the selected destination and have appropriate feedback.

#### 3.2.1.7.a Testing Procedure for Component: GUI Settings Tab

The settings tab will be tested in a white box fashion. Each input should be validated and update the corresponding setting in both the file and memory.

#### 3.2.1.7.b Stubs and/or drivers for component: GUI Settings Tab

#### 

Kivy Event Handler

Unittest

#### 3.2.1.7.c Test cases component: GUI Settings Tab

* Valid test input for each input
* Invalid test input for each input
* Apply button with no input (dissabled)
* Apply button with input

#### 

#### 3.2.1.7.d Purpose of tests for component: GUI Settings Tab

The settings tab is the main way the user can customize the experiment. In order to reach our goal of having a flexible system, the settings tab must be functional and easy to use.

**3.2.1.7.e Expected results for component: GUI Settings Tab**

All input fields must be functional and validate the input.

#### 3.2.1.8.a Testing Procedure for Component: GUI Video Player

The video player will be tested in a white box fashion. It should display video from the video link provided by the user and update with that link being changed.

#### 

#### 3.2.1.8.b Stubs and/or drivers for component: GUI Video Player

#### 

Kivy Event Handler

Unittest

#### 

#### 3.2.1.8.c Test cases component: GUI Video Player

#### 

* Invalid link given
* Stream and valid link given
* No stream but valid link given
* Link changed
* Play pressed
* Pause pressed

#### 3.2.1.8.d Purpose of tests for component: GUI Video Player

Having video feed allows the user to understand the systems current status and gives a first hand view of the experiment.

**3.2.1.8.e Expected results for component: GUI Video Player**

Video player is expected to function as a standard video player with the given link as a source.

#### 3.2.1.9.a Testing Procedure for Component: GUI File Writing

The file writing system will be tested in a white box fashion. It should validate paths, files and successfully write to said files.

#### 3.2.1.9.b Stubs and/or drivers for component: GUI File Writing

#### 

Kivy Event Handler

Unittest

#### 

#### 3.2.1.9.c Test cases component: GUI File Writing

#### 

* Invalid file path given
* Valid file path given
* Empty file
* Appended file

#### 3.2.1.9.d Purpose of tests for component: GUI File Writing

Logs and system feedback are a core part of the functionality of this product. Without this functionality, there is no testing using this system.

**3.2.1.9.e Expected results for component: GUI File Writing**

Files should be found and appended to. Appropriate error feedback should be given.

#### 3.2.1.10.a Testing Procedure for Component: Vision System-Adjusting video frame size and color

Our program for object detection takes a live video stream and analyzes each video frame. While testing, we ran the program on different cameras to ensure that our program can handle different input. We also tested with different sized jpeg images to make sure our program can handle them. We then did color conversion to make our video frames detectable by OpenCV.

#### 3.2.1.10.b Stubs and/or drivers for component: Vision system-Adjusting video frame size and color

Before we wrote our program that took input as a livestream, we tested with different sized images to see how our program adjusted image size and color.

#### 

#### 3.2.1.10.c Test cases component: Vision system-Adjusting video frame size and color

#### 

* Adjust size of input width and height
* OpenCV Color conversion

#### 

#### 3.2.1.10.d Purpose of tests for component: Vision system-Adjusting video frame size and color

#### 

Our program requires all input to be uniform to give consistent results. It also allows our program to properly utilize darknet and OpenCV library.

**3.2.1.10.e Expected results for component: Vision system-Adjusting video frame size and color**

We expect all input to be converted to a uniform size that is compatible with darknet and to be in RBG color format so color is detectable using OpenCV.

#### 3.2.1.11.a Testing Procedure for Component: Vision System-Create Bounding Box and Class names

#### 3.2.1.11.b Stubs and/or drivers for component: Vision System-Create Bounding Box and Class names

#### 

This component relies on a trained data set with a confidence level over 50. This was a requirement to get our bounding boxes outputted with our correct class name. We use the OpenCV library to create the bounding box.

#### 3.2.1.11.c Test cases component: Vision System-Create Bounding Box and Class names

* Get corners of bounding box
* Label classes

#### 

#### 3.2.1.11.d Purpose of tests for component: Vision System-Create Bounding Box and Class names

It is crucial for our bounding box to be properly displayed because it shows the user what item our program successfully detected. It also shows the proper class anime to decipher which of the various objects are detected.

**3.2.1.11.e Expected results for component: Vision System-Create Bounding Box and Class names**

The result of our testing should show a bounding box, the class and the confidence level around the item we trained to be detected.

#### 3.2.1.12.a Testing Procedure for Component: Vision System-Confidence Level

Confidence level shows how accurate the program can recognize an object based on the result of the custom trained data set. This can be used to test how well trained our data set actually is and apply it to live video/images.

#### 3.2.1.12.b Stubs and/or drivers for component: Vision System-Confidence Level

This component relies on a trained dataset to be present and the use of darknet.

#### 3.2.1.12.c Test cases component: Vision System-Confidence Level

* High mAP from custom dataset
* Confidence threshold

#### 

#### 3.2.1.12.d Purpose of tests for component: Vision System-Confidence Level

Confidence level is a vital component that needs to be tested. Displaying the confidence level itself does not require much testing. But getting a high mAP is what is important. To test this, we can run our live video stream and hold up different objects to test our detection. We set the threshold to 50% confidence level. If the confidence level seems low, then we know we must retrain to get our desired results.

**3.2.1.12.e Expected results for component: Vision System-Confidence Level**

When the object we want to be detected is in the video frame, we expect a bounding box, class name and confidence level to be displayed if the confidence level is over 50%.

#### 3.2.1.13.a Testing Procedure for Component: Wireless Local Area Network (WLAN) Connectivity

1. Drone WLAN broadcast and DHCP Server Functionality:
   * Testing Procedure:
     + Set up the drone with WLAN broadcast and DHCP server functionality.
     + Connect nodes to the drone's AP network and verify IP address assignment.
     + Monitor DHCP server logs and confirm successful IP address assignments.
   * Notes: Ensure that the drone is properly configured with DNSmasq and WPA supplicant for DHCP server functionality.
2. Raspberry Pi Carx Node WLAN connectivity:
   * Testing Procedure:
     + Configure the Raspberry Pi Carx node to connect to the drone's WLAN.
     + Initiate WLAN connection from the Raspberry Pi Carx node.
     + Verify successful WLAN connectivity by checking the obtained IP address.
   * Notes: Confirm that the Raspberry Pi Carx node is set up to connect to the specific WLAN network of the drone.
3. Edge Server (Windows 10 Computer) Node WLAN connectivity:
   * Testing Procedure:
     + Configure the Edge Server (Windows 10 Computer) node to connect to the drone's WLAN.
     + Establish WLAN connection from the Edge Server node.
     + Verify successful WLAN connectivity by checking the obtained IP address.
   * Notes: Ensure that the Edge Server node is configured to connect to the designated WLAN network of the drone.
4. Windows 10 Computer with GUI Node WLAN connectivity:
   * Testing Procedure:
     + Configure the Windows 10 Computer with GUI node to connect to the drone's WLAN.
     + Establish WLAN connection from the Windows 10 Computer with GUI node.
     + Verify successful WLAN connectivity by checking the obtained IP address.
   * Notes: Confirm that the Windows 10 Computer with GUI node is configured to connect to the appropriate WLAN network of the drone.

#### 3.2.1.13.b Stubs and/or drivers for Component: Wireless Local Area Network (WLAN) Connectivity

1. Drone WLAN broadcast and DHCP Server Functionality:
   * Stubs and/or Drivers: None required.
2. Raspberry Pi Carx Node WLAN connectivity:
   * Stubs and/or Drivers: None required.
3. Edge Server (Windows 10 Computer) Node WLAN connectivity:
   * Stubs and/or Drivers: None required.
4. Windows 10 Computer with GUI Node WLAN connectivity:
   * Stubs and/or Drivers: None required.

#### 3.2.1.13.c Test Cases for Component: Wireless Local Area Network (WLAN) Connectivity

1. Drone WLAN broadcast and DHCP Server Functionality:
   * Test Case 1: Verify IP address assignment for a single connected node.
   * Test Case 2: Test simultaneous connection of multiple nodes and validate IP address assignments.
2. Raspberry Pi Carx Node WLAN connectivity:
   * Test Case 1: Validate successful WLAN connectivity of the Raspberry Pi Carx node.
3. Edge Server (Windows 10 Computer) Node WLAN connectivity:
   * Test Case 1: Validate successful WLAN connectivity of the Edge Server node.
4. Windows 10 Computer with GUI Node WLAN connectivity:
   * Test Case 1: Validate successful WLAN connectivity of the Windows 10 Computer with GUI node.

#### 3.2.1.13.d Purpose of Tests for Component: Wireless Local Area Network (WLAN) Connectivity

1. Drone WLAN broadcast and DHCP Server Functionality:
   * Purpose: To ensure that the drone effectively broadcasts its WLAN and functions as a DHCP server, assigning IP addresses to connected nodes.
2. Raspberry Pi Carx Node WLAN connectivity:
   * Purpose: To verify the successful WLAN connectivity of the Raspberry Pi Carx node to the drone's WLAN network.
3. Edge Server (Windows 10 Computer) Node WLAN connectivity:
   * Purpose: To validate the successful WLAN connectivity of the Edge Server node to the drone's WLAN network.
4. Windows 10 Computer with GUI Node WLAN connectivity:
   * Purpose: To confirm the successful WLAN connectivity of the Windows 10 Computer with GUI node to the drone's WLAN network.

#### 3.2.1.13.e Expected Results for Component: Wireless Local Area Network (WLAN) Connectivity

1. Drone WLAN broadcast and DHCP Server Functionality:
   * Expected Results: The drone effectively broadcasts its WLAN, and connected nodes receive IP addresses from the DHCP server without conflicts.
2. Raspberry Pi Carx Node WLAN connectivity:
   * Expected Results: The Raspberry Pi Carx node successfully connects to the drone's WLAN and obtains a valid IP address.
3. Edge Server (Windows 10 Computer) Node WLAN connectivity:
   * Expected Results: The Edge Server node successfully connects to the drone's WLAN and obtains a valid IP address.
4. Windows 10 Computer with GUI Node WLAN connectivity:
   * Expected Results: The Windows 10 Computer with GUI node successfully connects to the drone's WLAN and obtains a valid IP address.

#### 3.2.1.14.a Testing Procedure for Component: ROS topic publications of the drone via web server interface

1. Image topics:
   * Testing Procedure:
     + Access the drone's web server interface.
     + Subscribe to image topics and receive image data.
     + Verify the presence of image data and check for proper image transmission.
   * Notes: Ensure that the drone's web server interface is accessible and configured to publish image topics.
2. Battery life topic:
   * Testing Procedure:
     + Access the drone's web server interface.
     + Subscribe to the battery life topic and receive battery life data.
     + Verify the accuracy and reliability of the battery life data.
   * Notes: Ensure that the drone's battery life topic is properly published and reflects the actual battery level.
3. Flight control status topic:
   * Testing Procedure:
     + Access the drone's web server interface.
     + Subscribe to the flight control status topic and receive status data.
     + Verify the accuracy and consistency of the flight control status information.
   * Notes: Confirm that the flight control status topic accurately represents the drone's operational mode and status.

#### 3.2.1.14.b Stubs and/or drivers for Component: ROS topic publications of the drone via web server interface

1. Image topics:
   * Stubs and/or Drivers: None required.
2. Battery life topic:
   * Stubs and/or Drivers: None required.
3. Flight control status topic:
   * Stubs and/or Drivers: None required.

#### 3.2.1.14.c Test Cases for Component: ROS topic publications of the drone via web server interface

1. Image topics:
   * Test Case 1: Verify the presence and integrity of image data received from the drone's image topics.
   * Test Case 2: Test the performance of image topic transmission under varying network conditions.
2. Battery life topic:
   * Test Case 1: Validate the accuracy and consistency of the battery life data received from the drone's battery life topic.
   * Test Case 2: Test the behavior of the battery life topic during extended drone operations.
3. Flight control status topic:
   * Test Case 1: Confirm the accuracy and consistency of the flight control status information received from the drone's topic.
   * Test Case 2: Validate the synchronization between the flight control status topic and actual drone operations.

#### 3.2.1.14.d Purpose of Tests for Component: ROS topic publications of the drone via web server interface

1. Image topics:
   * Purpose: To ensure the proper transmission and availability of image data through the drone's web server interface.
2. Battery life topic:
   * Purpose: To verify the accuracy and reliability of the battery life data published through the drone's web server interface.
3. Flight control status topic:
   * Purpose: To validate the accuracy and consistency of the flight control status information published through the drone's web server interface.

#### 3.2.1.14.e Expected Results for Component: ROS topic publications of the drone via web server interface

1. Image topics:
   * Expected Results: Image data is successfully transmitted and received from the drone's image topics without loss or corruption.
2. Battery life topic:
   * Expected Results: The battery life data received from the drone's topic accurately reflects the actual battery level and remains consistent.
3. Flight control status topic:
   * Expected Results: The flight control status information received from the drone's topic accurately represents the drone's operational mode and status consistently.

#### 3.2.1.15.a Testing Procedure for Component: Drone flight tests

1. Manual flight test:
   * Testing Procedure:
     + Prepare the drone for a manual flight test.
     + Execute flight control commands (e.g., throttle, pitch, roll, yaw) manually.
     + Observe and record the drone's response to the commands.
     + Monitor the battery life during the flight test.
   * Notes: Ensure a suitable flight area and adhere to safety regulations while conducting the manual flight test.
2. Hover program:
   * Testing Procedure:
     + Activate the hover program on the drone.
     + Monitor and record the drone's stability and ability to maintain a hover position.
     + Observe the drone's response to any manual control inputs during the hover.
     + Monitor the battery life during the hover program.
   * Notes: Ensure that the hover program is properly configured and activated on the drone.
3. Oscillation program:
   * Testing Procedure:
     + Activate the oscillation program on the drone.
     + Observe and record the drone's oscillatory movement pattern.
     + Verify the drone's stability and responsiveness during the oscillation program.
     + Monitor the battery life during the oscillation program.
   * Notes: Ensure that the oscillation program is properly configured and activated on the drone.
4. Hardware and software status check program (selfcheck.py):
   * Testing Procedure:
     + Execute the selfcheck.py program on the drone.
     + Monitor and verify the reported status of hardware and software components.
     + Ensure that the selfcheck.py program identifies any issues or discrepancies accurately.
   * Notes: Confirm that the selfcheck.py program is properly implemented and covers all relevant hardware and software components.
5. Calibration tests (i.e., sensor calibrations):
   * Testing Procedure:
     + Initiate sensor calibrations on the drone (e.g., IMU, compass, gyro).
     + Verify the calibration process and ensure accurate sensor calibration.
     + Test the drone's behavior and responsiveness after the calibration procedure.
   * Notes: Follow the specific calibration procedures recommended for the drone's sensors.
6. Visual inspection of the mechanical and electrical connection components:
   * Testing Procedure:
     + Physically inspect the drone's mechanical components (e.g., propellers, frame) and electrical connections.
     + Ensure that all components are securely attached and properly connected.
     + Report any visible damages, loose connections, or anomalies.
   * Notes: Use visual inspection to identify any potential issues or risks related to the mechanical and electrical components.

#### 3.2.1.15.b Stubs and/or drivers for Component: Drone flight tests

1. Manual flight test:
   * Stubs and/or Drivers: None required.
2. Hover program:
   * Stubs and/or Drivers: None required.
3. Oscillation program:
   * Stubs and/or Drivers: None required.
4. Hardware and software status check program (selfcheck.py):
   * Stubs and/or Drivers: None required.
5. Calibration tests (i.e., sensor calibrations):
   * Stubs and/or Drivers: None required.
6. Visual inspection of the mechanical and electrical connection components:
   * Stubs and/or Drivers: None required.

#### 3.2.1.15.c Test Cases for Component: Drone flight tests

1. Manual flight test:
   * Test Case 1: Verify the drone's response to manual flight control commands (throttle, pitch, roll, yaw).
   * Test Case 2: Monitor and record the battery life during the manual flight test.
2. Hover program:
   * Test Case 1: Validate the drone's ability to maintain a stable hover position.
   * Test Case 2: Monitor and record the battery life during the hover program.
3. Oscillation program:
   * Test Case 1: Verify the drone's oscillatory movement pattern during the oscillation program.
   * Test Case 2: Monitor and record the battery life during the oscillation program.
4. Hardware and software status check program (selfcheck.py):
   * Test Case 1: Confirm that the selfcheck.py program accurately reports the status of hardware and software components.
   * Test Case 2: Verify the program's ability to identify and report any hardware or software issues.
5. Calibration tests (i.e., sensor calibrations):
   * Test Case 1: Validate the accuracy and effectiveness of sensor calibrations on the drone.
   * Test Case 2: Verify the drone's behavior and responsiveness after sensor calibrations.
6. Visual inspection of the mechanical and electrical connection components:
   * Test Case 1: Physically inspect and confirm the integrity of the drone's mechanical components.
   * Test Case 2: Ensure the secure attachment and proper connection of electrical components.

#### 3.2.1.15.d Purpose of Tests for Component: Drone flight tests

1. Manual flight test:
   * Purpose: To evaluate the drone's response to manual flight control commands and monitor battery life during normal flight operations.
2. Hover program:
   * Purpose: To assess the drone's ability to maintain a stable hover position and monitor battery life during hover operations.
3. Oscillation program:
   * Purpose:To verify the drone's ability to execute an oscillatory movement pattern and monitor battery life during the oscillation program.
4. Hardware and software status check program (selfcheck.py):
   * Purpose: To ensure the accurate reporting of hardware and software status through the selfcheck.py program.
5. Calibration tests (i.e., sensor calibrations):
   * Purpose: To validate the accuracy and effectiveness of sensor calibrations for improved flight performance.
6. Visual inspection of the mechanical and electrical connection components:
   * Purpose: To visually confirm the integrity of the drone's mechanical components and ensure proper electrical connections for safe and reliable operation.

#### 3.2.1.15.e Expected Results for Component: Drone flight tests

1. Manual flight test:
   * Expected Results: The drone responds accurately to manual flight control commands, and the battery life remains within the expected range during normal flight operations.
2. Hover program:
   * Expected Results: The drone maintains a stable hover position during the hover program, and the battery life remains within the expected range.
3. Oscillation program:
   * Expected Results: The drone exhibits the expected oscillatory movement pattern during the oscillation program, and the battery life remains within the expected range.
4. Hardware and software status check program (selfcheck.py):
   * Expected Results: The selfcheck.py program accurately reports the status of hardware and software components, identifying any issues or discrepancies.
5. Calibration tests (i.e., sensor calibrations):
   * Expected Results: Sensor calibrations improve the drone's flight performance, resulting in more accurate and responsive movements.
6. Visual inspection of the mechanical and electrical connection components:
   * Expected Results: The visual inspection confirms the integrity of the drone's mechanical components and ensures secure and proper electrical connections.

#### 3.2.1.16.a Testing Procedure for Component: Car driving tests

1. Calibration tests (i.e., forward and reverse frame of reference for driving controls):
   * Testing Procedure:
     + Calibrate the car's driving controls to establish the forward and reverse frame of reference.
     + Test the car's response to forward and reverse commands to ensure correct movement in the respective directions.
     + Monitor and record any calibration discrepancies or issues.
   * Notes: Follow the specific calibration procedures recommended for the car's driving controls.
2. Driving tests - hardware and software checks (forward/backwards, left and right turn, stop and go):
   * Testing Procedure:
     + Execute hardware and software checks by issuing commands for forward/backward movements, left and right turns, and stop/go actions.
     + Observe and record the car's response to the commands, ensuring that it aligns with the expected behavior.
     + Monitor and record any hardware or software issues encountered during the driving tests.
   * Notes: Ensure that the car's hardware components (e.g., motors) and software commands from the picarx library are functioning properly.
3. Visual inspection of the mechanical and electrical connection components:
   * Testing Procedure:
     + Physically inspect the car's mechanical components (e.g., wheels, chassis) and electrical connections.
     + Ensure that all components are securely attached and properly connected.
     + Report any visible damages, loose connections, or anomalies.
   * Notes: Use visual inspection to identify any potential issues or risks related to the mechanical and electrical components of the car.
4. Battery life testing:
   * Testing Procedure:
     + Monitor the battery usage of the car during the driving tests.
     + Record the duration of operation and battery consumption.
     + Compare the actual battery usage with the expected battery life.
   * Notes: Ensure that the car's battery is fully charged before conducting the tests and measure the battery life accurately.

#### 3.2.1.16.b Stubs and/or drivers for Component: Car driving tests

1. Calibration tests (i.e., forward and reverse frame of reference for driving controls):
   * Stubs and/or Drivers: None required.
2. Driving tests - hardware and software checks (forward/backwards, left and right turn, stop and go):
   * Stubs and/or Drivers: None required.
3. Visual inspection of the mechanical and electrical connection components:
   * Stubs and/or Drivers: None required.
4. Battery life testing:
   * Stubs and/or Drivers: None required.

#### 3.2.1.16.c Test Cases for Component: Car driving tests

1. Calibration tests (i.e., forward and reverse frame of reference for driving controls):
   * Test Case 1: Verify the car's response to forward command inputs and ensure correct movement in the forward direction.
   * Test Case 2: Verify the car's response to reverse command inputs and ensure correct movement in the reverse direction.
2. Driving tests - hardware and software checks (forward/backwards, left and right turn, stop and go):
   * Test Case 1: Test the car's response to forward and backward command inputs and ensure accurate movements in the respective directions.
   * Test Case 2: Test the car's response to left and right turn command inputs and ensure accurate turning movements.
   * Test Case 3: Test the car's response to stop and go command inputs and ensure proper stopping and starting behaviors.
3. Visual inspection of the mechanical and electrical connection components:
   * Test Case 1: Physically inspect and confirm the integrity of the car's mechanical components.
   * Test Case 2: Ensure the secure attachment and proper connection of electrical components.
4. Battery life testing:
   * Test Case 1: Measure the car's battery usage during the driving tests and compare it to the expected battery life.
   * Test Case 2: Monitor and record the battery life during extended car operations to assess its performance.

#### 3.2.1.16.d Purpose of Tests for Component: Car driving tests

1. Calibration tests (i.e., forward and reverse frame of reference for driving controls):
   * Purpose: To establish the forward and reverse frame of reference for the car's driving controls, ensuring correct movements in the respective directions.
2. Driving tests - hardware and software checks (forward/backwards, left and right turn, stop and go):
   * Purpose: To verify the proper functioning of the car's hardware components (e.g., motors) and software commands from the picarx library for accurate and reliable driving movements.
3. Visual inspection of the mechanical and electrical connection components:
   * Purpose: To visually confirm the integrity of the car's mechanical components and ensure proper electrical connections for safe and reliable operation.
4. Battery life testing:
   * Purpose: To assess the performance of the car's battery and verify its usage against the expected battery life during normal driving operations.

#### 3.2.1.16.e Expected Results for Component: Car driving tests

1. Calibration tests (i.e., forward and reverse frame of reference for driving controls):
   * Expected Results: The car responds correctly to forward and reverse command inputs, exhibiting accurate movements in the respective directions.
2. Driving tests - hardware and software checks (forward/backwards, left and right turn, stop and go):
   * Expected Results: The car accurately responds to forward/backwardand left/right turn commands, exhibiting the expected movements. It stops and goes as commanded.
3. Visual inspection of the mechanical and electrical connection components:
   * Expected Results: The visual inspection confirms the integrity of the car's mechanical components, ensuring that they are securely attached. The electrical connections are properly connected without any loose connections or anomalies.
4. Battery life testing:
   * Expected Results: The car's battery usage during the driving tests aligns with the expected battery life. It operates within the expected range during normal driving operations.

#### 3.2.1.17.a Testing Procedure for Component: Car command server program

1. Test the command server connectivity on the picarx car:
   * Testing Procedure:
     + Start the car command server program on the picarx car.
     + Verify the connectivity of the car command server with the drone, edge server, and GUI.
     + Send test commands to the car command server and validate the response.
   * Notes: Ensure that the car command server program is properly configured and able to establish connections with the specified components.
2. Test the command keys:
   * Testing Procedure:
     + Send command keys (values for changing speed, direction, stop, go, etc.) to the car command server program.
     + Verify that the command keys are correctly interpreted and result in the desired actions.
   * Notes: Confirm that the car command server program accurately processes the command keys and performs the corresponding actions.
3. Test receiving command keys from the drone client program:
   * Testing Procedure:
     + Send command keys from the drone client program to the car command server program.
     + Validate that the car command server program successfully receives and processes the command keys from the drone client program.
   * Notes: Ensure that the communication between the drone client program and the car command server program is established and functional.
4. Test receiving command keys from the edge server client program:
   * Testing Procedure:
     + Send command keys from the edge server client program to the car command server program.
     + Validate that the car command server program successfully receives and processes the command keys from the edge server client program.
   * Notes: Ensure that the communication between the edge server client program and the car command server program is established and functional.
5. Test receiving command keys from the GUI client program:
   * Testing Procedure:
     + Send command keys from the GUI client program to the car command server program.
     + Validate that the car command server program successfully receives and processes the command keys from the GUI client program.
   * Notes: Ensure that the communication between the GUI client program and the car command server program is established and functional.

#### 3.2.1.17.b Stubs and/or drivers for Component: Car command server program

1. Test the command server connectivity on the picarx car:
   * Stubs and/or Drivers: None required.
2. Test the command keys:
   * Stubs and/or Drivers: None required.
3. Test receiving command keys from the drone client program:
   * Stubs and/or Drivers: None required.
4. Test receiving command keys from the edge server client program:
   * Stubs and/or Drivers: None required.
5. Test receiving command keys from the GUI client program:
   * Stubs and/or Drivers: None required.

#### 3.2.1.17.c Test Cases for Component: Car command server program

1. Test the command server connectivity on the picarx car:
   * Test Case 1: Verify the connectivity of the car command server with the drone.
   * Test Case 2: Verify the connectivity of the car command server with the edge server.
   * Test Case 3: Verify the connectivity of the car command server with the GUI.
2. Test the command keys:
   * Test Case 1: Verify that the car command server correctly interprets and processes command keys for changing speed.
   * Test Case 2: Verify that the car command server correctly interprets and processes command keys for changing direction.
   * Test Case 3: Verify that the car command server correctly interprets and processes command keys for stop and go actions.
3. Test receiving command keys from the drone client program:
   * Test Case 1: Validate that the car command server successfully receives and processes command keys from the drone client program.
4. Test receiving command keys from the edge server client program:
   * Test Case 1: Validate that the car command server successfully receives and processes command keys from the edge server client program.
5. Test receiving command keys from the GUI client program:
   * Test Case 1: Validate that the car command server successfully receives and processes command keys from the GUI client program.

#### 3.2.1.17.d Purpose of Tests for Component: Car command server program

1. Test the command server connectivity on the picarx car:
   * Purpose: To ensure the proper connectivity of the car command server with the drone, edge server, and GUI.
2. Test the command keys:
   * Purpose: To verify that the car command server accurately interprets and processes command keys for speed, direction, stop, go, etc.
3. Test receiving command keys from the drone client program:
   * Purpose: To validate the successful reception and processing of command keys from the drone client program.
4. Test receiving command keys from the edge server client program:
   * Purpose: To validate the successful reception and processing of command keys from the edge server client program.
5. Test receiving command keys from the GUI client program:
   * Purpose: To validate the successful reception and processing of command keys from the GUI client program.

#### 3.2.1.17.e Expected Results for Component: Car command server program

1. Test the command server connectivity on the picarx car:
   * Expected Results: The car command server successfully establishes connectivity with the drone, edge server, and GUI.
2. Test the command keys:
   * Expected Results: The car command server accurately interprets and processes command keys for changing speed, direction, stop, go, etc.
3. Test receiving command keys from the drone client program:
   * Expected Results: The car command server successfully receives and processes command keys from the drone client program.
4. Test receiving command keys from the edge server client program:
   * Expected Results: The car command server successfully receives and processes command keys from the edge server client program.
5. Test receiving command keys from the GUI client program:
   * Expected Results: The car command server successfully receives and processes command keys from the GUI client program.

#### 3.2.1.18.a Testing Procedure for Component: Drone client program

1. Test the client program connectivity:
   * Testing Procedure:
     + Start the drone client program.
     + Verify the connectivity of the drone client program with the car command server, edge server, and GUI.
     + Send test commands from the drone client program and validate the response.
   * Notes: Ensure that the drone client program is properly configured and able to establish connections with the specified components.
2. Test sending command keys to the car command server program:
   * Testing Procedure:
     + Send command keys (values for changing speed, direction, stop, go, etc.) from the drone client program to the car command server program.
     + Validate that the car command server program successfully receives and processes the command keys from the drone client program.
   * Notes: Ensure that the communication between the drone client program and the car command server program is established and functional.
3. Test sending frames to the edge server:
   * Testing Procedure:
     + Send frames from the drone client program to the edge server.
     + Verify that the edge server successfully receives and processes the frames from the drone client program.
   * Notes: Ensure that the communication between the drone client program and the edge server is established and functional.

#### 3.2.1.18.b Stubs and/or drivers for Component: Drone client program

1. Test the client program connectivity:
   * Stubs and/or Drivers: None required.
2. Test sending command keys to the car command server program:
   * Stubs and/or Drivers: None required.
3. Test sending frames to the edge server:
   * Stubs and/or Drivers: None required.

#### 3.2.1.18.c Test Cases for Component: Drone client program

1. Test the client program connectivity:
   * Test Case 1: Verify the connectivity of the drone client program with the car command server.
   * Test Case 2: Verify the connectivity of the drone client program with the edge server.
   * Test Case 3: Verify the connectivity of the drone client program with the GUI.
2. Test sending command keys to the car command server program:
   * Test Case 1: Validate that the car command server program successfully receives and processes command keys sent from the drone client program.
3. Test sending frames to the edge server:
   * Test Case 1: Validate that the edge server successfully receives and processes frames sent from the drone client program.

#### 3.2.1.18.d Purpose of Tests for Component: Drone client program

1. Test the client program connectivity:
   * Purpose: To ensure the proper connectivity of the drone client program with the car command server, edge server, and GUI.
2. Test sending command keys to the car command server program:
   * Purpose: To validate the successful transmission and processing of command keys from the drone client program to the car command server program.
3. Test sending frames to the edge server:
   * Purpose: To validate the successful transmission and processing of frames from the drone client program to the edge server.

#### 3.2.1.18.e Expected Results for Component: Drone client program

1. Test the client program connectivity:
   * Expected Results: The drone client program successfully establishes connectivity with the car command server, edge server, and GUI.
2. Test sending command keys to the car command server program:
   * Expected Results: The car command server program successfully receives and processes command keys sent from the drone client program.
3. Test sending frames to the edge server:
   * Expected Results: The edge server successfully receives and processes frames sent from the drone client program.

#### 3.2.1.19.a Testing Procedure for Component: Edge-server node client program

1. Test the client program connectivity:
   * Testing Procedure:
     + Start the edge-server node client program.
     + Verify the connectivity of the edge-server node client program with the car command server, drone (receiving frames), and GUI.
     + Send test commands from the edge-server node client program and validate the response.
   * Notes: Ensure that the edge-server node client program is properly configured and able to establish connections with the specified components.
2. Test sending command keys to the car command server program:
   * Testing Procedure:
     + Send command keys (values for changing speed, direction, stop, go, etc.) from the edge-server node client program to the car command server program.
     + Validate that the car command server program successfully receives and processes the command keys from the edge-server node client program.
   * Notes: Ensure that the communication between the edge-server node client program and the car command server program is established and functional.
3. Test receiving frames from the drone:
   * Testing Procedure:
     + Receive frames from the drone in the edge-server node client program.
     + Validate the successful reception and processing of frames from the drone.
   * Notes: Ensure that the communication between the edge-server node client program and the drone for frame reception is established and functional.

#### 3.2.1.19.b Stubs and/or drivers for Component: Edge-server node client program

1. Test the client program connectivity:
   * Stubs and/or Drivers: None required.
2. Test sending command keys to the car command server program:
   * Stubs and/or Drivers: None required.
3. Test receiving frames from the drone:
   * Stubs and/or Drivers: None required.

#### 3.2.1.19.c Test Cases for Component: Edge-server node client program

1. Test the client program connectivity:
   * Test Case 1: Verify the connectivity of the edge-server node client program with the car command server.
   * Test Case 2: Verify the connectivity of the edge-server node client program with the drone for receiving frames.
   * Test Case 3: Verify the connectivity of the edge-server node client program with the GUI.
2. Test sending command keys to the car command server program:
   * Test Case 1: Validate that the car command server program successfully receives and processes command keys sent from the edge-server node client program.
3. Test receiving frames from the drone:
   * Test Case 1: Validate the successful reception and processing of frames from the drone in the edge-server node client program.

#### 3.2.1.19.d Purpose of Tests for Component: Edge-server node client program

1. Test the client program connectivity:
   * Purpose: To ensure the proper connectivity of the edge-server node client program with the car command server, drone (for frame reception), and GUI.
2. Test sending command keys to the car command server program:
   * Purpose: To validate the successful transmission and processing of command keys from the edge-server node client program to the car command server program.
3. Test receiving frames from the drone:
   * Purpose: To validate the successful reception and processing of frames from the drone in the edge-server node client program.

#### 3.2.1.19.e Expected Results for Component: Edge-server node client program

1. Test the client program connectivity:
   * Expected Results: The edge-server node client program successfully establishes connectivity with the car command server, drone (for frame reception), and GUI.
2. Test sending command keys to the car command server program:
   * Expected Results: The car command server program successfully receives and processes command keys sent from the edge-server node client program.
3. Test receiving frames from the drone:
   * Expected Results: The edge-server node client program successfully receives and processes frames from the drone.

#### 3.2.1.20.a Testing Procedure for Component:

#### 3.2.1.20.b Stubs and/or drivers for component:

#### 

#### 3.2.1.20.c Test cases component:

#### 

#### 3.2.1.20.d Purpose of tests for component:

**3.2.1.20.e Expected results for component:**

### 3.2.2 Integration Testing

The integration testing procedure is specified.

#### 

#### 3.2.2.1 Testing Procedure for Integration:

* Conduct bottom-up integration testing, gradually integrating components into the Sky Socket GUI. Start by integrating individual components and gradually combine them to form the complete system.
* ***First Degree of Integration (non flight integration):***
  + Test the integration of the Sky Socket GUI with each component individually, including the car command server program, drone client program, and edge-server node client program.
  + Verify the proper communication, command processing, and data exchange between the GUI and each integrated component.
  + **Input**: Commands and data exchanged between the GUI and the connected components.
  + **Output**: Successful integration and functioning of the individual components within the GUI.
* ***Second Degree of Integration (flight integration):***
  + Test the integration of all components during non flight and flight scenarios.
  + Integrate the previously integrated components, such as the car command server program, drone client program, and edge-server node client program, to ensure they work together effectively.
  + Validate the communication and data exchange between the GUI and all integrated components during both nonflight and flight scenarios.
  + **Input**: Commands, data, and image frames exchanged between the GUI and the integrated components during non flight and flight scenarios.
  + **Output**: Successful integration and functioning of all components within the GUI in both nonflight and flight scenarios.
* Perform end-to-end testing to ensure the entire system works as expected, incorporating all integrated components.
  + **Input**: Various input scenarios and user interactions with the Sky Socket GUI and connected components.
  + **Output**: Expected system behavior, accurate responses to user commands, and successful integration of all components.
* Track and download log files generated on the drone to monitor battery life, bandwidth between the drone and all connections, frame rate of the camera, and frame rate of processing frames.
  + **Input**: Log files generated on the drone during operation.
  + **Output**: Downloaded log files containing information about battery life, bandwidth, camera frame rate, and processing frame rate.
  + Purpose: To track and analyze performance metrics for the drone and system during integration testing.

#### 3.2.2.2 Stubs and Drivers Required:

* Stubs and drivers specific to each component as mentioned in the previous responses.
* Stub for simulating the drone's image capture functionality and processing algorithms during integration testing without physical hardware.
* Simulator or test environment to replicate the edge server node's capabilities for second-level image analysis.

#### 3.2.2.3 Test Cases and Their Purpose:

1. Test Case: Verify the integration of the Sky Socket GUI with each component individually during the first degree of integration.
   * Inputs: Commands and data from the GUI to each individual component.
   * Expected Output: Proper communication, command processing, and data exchange between the GUI and each integrated component.
   * Purpose: To ensure the successful integration and functioning of individual components within the GUI.
2. Test Case: Validate the integration of all components during nonflight scenarios in the second degree of integration.
   * Inputs: Commands, data, and image frames exchanged between the GUI and the integrated components during nonflight scenarios.
   * Expected Output: Successful integration and functioning of all components within the GUI during nonflight scenarios.
   * Purpose: To verify the communication, data exchange, and proper integration of all components in non flight scenarios.
3. Test Case: Validate the integration of all components during flight scenarios in the second degree of integration.
   * Inputs: Commands, data, and image frames exchanged between the GUI and the integrated components during flight scenarios.
   * Expected Output: Successful integration and functioning of all components within the GUI during flight scenarios.
   * Purpose: To verify the communication, data exchange, and proper integration of all components in flight scenarios.
4. Test Case: Perform end-to-end testing of the entire system, incorporating all integrated components.
   * Inputs: Various input scenarios and user interactions with the Sky Socket GUI and connected components.
   * Expected Output: Expected system behavior, accurate responses to user commands, and successful integration of all components.
   * Purpose: To ensure the entire system functions correctly, meeting the goals and objectives of the Sky Socket project.
5. Test Case: Validate the command line prompt functionality of the Sky Socket GUI.
   * Inputs: Direct commands entered through the command line prompt.
   * Expected Output: Proper interpretation and processing of commands by the GUI, resulting in the expected actions or responses from the connected car.
   * Purpose: To ensure the command line prompt feature functions correctly, allowing direct commands to be sent to the car command server program.
6. Test Case: Test the integration between the drone client program and the car command server program.
   * Inputs: Commands and data exchanged between the drone client program and the car command server program.
   * Expected Output: Proper communication and command processing between the drone client program and the car command server program, resulting in accurate control of the car's movements.
   * Purpose: To verify the integration and communication between the drone and the car for coordinated actions based on the drone's analysis.
7. Test Case: Validate the integration between the drone client program and the edge-server node client program.
   * Inputs: Image frames and data exchanged between the drone client program and the edge-server node client program.
   * Expected Output: Successful transmission of image frames from the drone to the edge server for second-level analysis, with accurate processing and detection of critical information.
   * Purpose: To verify the integration and communication between the drone and the edge server for advanced image analysis and detection.
8. Test Case: Verify the integration between the edge-server node client program and the car command server program.
   * Inputs: Commands and data exchanged between the edge-server node client program and the car command server program.
   * Expected Output: Proper communication and command processing between the edge server and the car command server program, resulting in accurate control of the car's movements.
   * Purpose: To validate the integration and communication between the edge server and the car for coordinated actions based on the edge server's analysis.
9. Test Case: Monitor and download log files from the drone to track battery life, bandwidth, camera frame rate, and processing frame rate.
   * Inputs: Log files generated on the drone during operation.
   * Expected Output: Downloaded log files containing information about battery life, bandwidth, camera frame rate, and processing frame rate.
   * Purpose: To track and analyze performance metrics of the drone and system during integration testing.

#### 3.2.2.4 Expected Results:

1. Expected Result: The Sky Socket GUI successfully integrates and communicates with each component individually during the first degree of integration, demonstrating proper command processing and data exchange.
2. Expected Result: The Sky Socket GUI effectively integrates and functions with all components during non flight scenarios in the second degree of integration, demonstrating successful integration and communication in nonflight scenarios.
3. Expected Result: The Sky Socket GUI successfully integrates and functions with all components during flight scenarios in the second degree of integration, demonstrating successful integration and communication in flight scenarios.
4. Expected Result: The entire system, including the Sky Socket GUI and all integrated components, exhibits expected behavior, accurate responses to user commands, and successful integration, meeting the goals and objectives of the Sky Socket project.
5. Expected Result: The command line prompt of the Sky Socket GUI correctly interprets and processes direct commands, resulting in the expected actions or responses from the connected car.
6. Expected Result: The drone client program integrates and communicates effectively with the car command server program, enabling accurate control of the car's movements based on the drone's analysis.
7. Expected Result: The drone client program successfully transmits image frames to the edge server for second-level analysis, with accurate processing and detection of critical information.
8. Expected Result: The edge-server node client program integrates and communicates effectively with the car command server program, enabling accurate control of the car's movements based on the edge server's analysis.
9. Expected Result: Log files are generated on the drone, containing information about battery life, bandwidth between the drone and all connections, camera frame rate, and processing frame rate.
10. Expected Result: Log files can be downloaded from the drone to the Sky Socket GUI for analysis and tracking of performance metrics.

### 3.2.3 Validation Testing

#### 3.2.3.1 Testing procedure for validation

The features and functionality of Sky Socket will be cross-referenced with the Software Requirements Specification document to verify that it conforms with the requirements and is up to standard. We will also do validation testing with our client to ensure that it is to their standard.

#### 3.2.3.3 Expected results

The software will perform at the specified requirements from the Software Requirements Specification document.

#### 3.2.3.4 Pass/fail criterion for all validation tests

The features in Sky Socket are allowed to not be within the specified requirements verbatim, but the spirit of all correlated requirements must be fulfilled to ensure compliance.

### 3.2.4 High-order testing (a.k.a. System Testing)

#### 3.2.4.1 Recovery testing

Recovery testing will not be performed. While system failure is not desired, there is no requirement for recovery in the event of a crash.

#### 3.2.4.2 Security testing

Security testing will not be performed. There are no concerns with security for our product. Any security researchers will be required to add onto our system.

#### 3.2.4.3 Stress testing

A series of targeted button presses will be done in sequence to try and cause Sky Socket GUI to crash. The program should have systems in place to avoid this from causing system failure.

#### 3.2.4.4 Performance testing

The machine learning algorithm will be loaded onto expected hardware as well as Raspberry Pi’s. This will test how high of a frame count as well as the accuracy of our model for systems with limited resources.

#### 3.2.4.5 Alpha/beta testing

Alpha testing will occur in-house. Members of the testing team will attempt to run complete and predetermined simulations with predetermined outcomes to assess the accuracy and completeness of the product.

Beta testing will also be in-house with members of the testing team as well as some independent testers and the client assisting in the simulation process. They will not be given instructions for one run of the testing and then given instructions for the second. The new additions will be asked for their feedback on using the product as well as the documentation’s completeness and user friendliness.

#### 3.2.4.6 Pass/fail criterion for all validation tests

The features in Sky Socket are allowed to not be within the specified requirements verbatim, but the spirit of all correlated requirements must be fulfilled to ensure compliance.

## 3.3 Testing resources and staffing

**Staffing:**

* Sky Socket Staff

We have a dedicated team of developers who will be taking the lead on the testing process.

* 3rd Party Testing Assistance:

There will be assistance in validation and beta testing from the client Zheng Song and some graduate students who are doing work in a similar field.

**Resources:**

* Sky Socket Staff computers

The Sky Socket staff will be utilizing their personal computers for most of the testing phases. This includes a range of Windows, Linux, and virtual machines.

* Google Colab

Google Colab is used for early testing and creation of the machine learning system and will be utilized for unit testing on said systems.

* 1 Clover Drone

The team has a clover drone which is one of the pieces of hardware used for the simulation. It will be used for the second phase of integration and high level testing.

* 1 PiCarX

The team has a PiCarX which is used as one of the pieces of hardware used for the simulation. It will be used for the second phase of integration and high level testing.

* 1 UMD computing device

The team will have access to a UMD computing devices to cover testing on unknown hardware for the GUI.

## 3.4 Test work products

**Custom Kivy Widget:**

To monitor the hardware usage on the system the GUI is being run on, we created a custom Kivy widget that will display the impact the software has on the system. This will be valuable for system performance testing.

## 3.5 Test record keeping and test log

The following is an example of the test log format that will be used:

| Date | Test Type | Component | Module | Coordinator | Result |
| --- | --- | --- | --- | --- | --- |
| 4-6 | Unit | Communication | Drone Client Program | Demetrius | Drone connected to server successfully |
| 4-6 | Unit | Communication | Car Server program | Demetrius | Car established connections successfully |
| 5-12 | Unit | GUI | Connection Window | Ryan | All buttons function as specified. To establish connections |
| 6-12 | Unit | GUI | Status Bar | Ryan | Colors and status messages are displaying as expected with given inputs |
| 6-12 | Integration | GUI | Tabs | Ryan | The GUI contains all tabs and the status bar and they respond/interact as expected |
| 6-12 | Unit | Vision | Bounding Box | Olivia | The algorithm correctly detects objects and places a bounding box around the expected area |
| 6-12 | Unit | Vision | Confidence Level | Jonathan | The confidence level is shown and meets the requirements |

# Traceability Matrix

