BRIGHAM YOUNG UNIVERSITY

AUVSI CAPSTONE TEAM (TEAM 45)

Imaging Test Procedures

ID	Rev.	Date	Description	Author	Checked By
IM-006	1.0	12-11-2018	Initial release	Tyler Miller	Jake Johnson

1 Introduction

In order to verify that the imaging subsystem independently meets its key success measures a series of tests and procedures were developed. Requirements for the subsystem can be seen in IM-007. However, some of the subsystem's requirements cannot be fully measured until it is integrated with the full system and testing in flight.

2 Continuous Integration

A prerequisite to meeting our key success measures, is a reliable, bug-free codebase. Imaging undertook a large task this year with our server-client model requiring us to rewrite our entire codebase. This rewrite presented an opportunity to build a proper test framework in parallel with our server-client code. A series of unit and integration tests were developed for each component of the server and client. These tests are extremely beneficial because they ensure proper functionality as changes are made to the code. Tests are run automatically by our github repository on every commit and test results are reported back to the team. This allows us to pinpoint breaking changes and respond to them quickly.

The CI tests start with a series of unit tests to examine server-database functionality. A unit test was created for every Database Access Object (DAO) method used by the server to interact with the database. These methods perform basic database operations, such as insert, select, update, and delete. The unit tests perform these operations through the DAO methods, and then confirm that the database reflects the expected results of such operation. These methods also inherently test constraints placed within the database itself (ie: primary key and unique constraints).

Next, integration test are performed. The purpose of these tests are to verify the client rest library (described in IM-001) interacts with the server as expected. The client rest library presents top-level methods useful to the client gui to interact with the imaging server. Thus integration tests confirm that the client rest library and the api endpoint handlers on the server function properly.

Regression tests, which are added as bugs are found and patched, naturally fall within one of the existing unit or integration tests and are placed there. Between these unit and integration tests, we can confidently say the server performs as expected. Having a verified codebase allows us to reliably perform tests on the system in the future as we integrate with other subsystems.

3 Bandwidth Test Procedure

One of our most important success measures, as described in the imaging requirements matrix (IM-007) is streaming images at a high enough rate, so we're able to perform classifications in an actionable (near real time) manner. A high stream rate will also guarantee that we dont miss any targets as the plane performs its search pattern since higher stream rates will allow more image overlap.

The bandwidth test itself is fairly simple to perform and can done on the ground.

1. With the plane's onboard computer running, connect it to the network using the ubiquiti bullet and litebeam, as it would be connected during an actual flight.

2. On a computer connected to the router over ethernet, confirm you are on the proper ros network.

3. Start the ros handler code with: rosrun imaging_ros_handler ros_handler.py

4. On the plane, confirm the camera is connected to the onboard computer and start the camera driver with: rosrun a6000_ros_node a6000_ros

5. On the server machine (connected to the router), monitor the image stream rate using: rostopic hz, this will print out the average number of messages/second every few seconds.

6. Walk the plane away from the base station, tracking distance and stream rate every 10-20 feet.

With this method we saw stream rates between 1.5 - 2.5Hz. It was also observed that the main rate limiter was the camera itself. Since the camera captures and compresses the image before sending it to the plane's onboard computer, it seems to consistently take 1s for capture and 1 additional second for it to be successfully sent to the computer (presumably this time is used to compresses the image before it is transmitted). Given the camera's high resolution, wide field of view, and the plane's slow speed; this rate should be adequate to capture the entire competition field.