Optimization Of Command Execution Testing Procedures

Akhila Yellapragada, Vedika Ghildyal, Sydney Whilden, Deepak Mishra

akhila.yellapragada@uga.edu, Vedika.ghildyal@uga.edu, sydney.whilden25@uga.edu, dmishra@uga.edu,
Small Satellite Research Laboratory, University of Georgia

Research Background

The Multiview Onboard Computational Imager (MOCI) is a 6U Cube Satellite under development at The University of Georgia Small Satellite Research Laboratory. MOCI will capture images of the Earth's surface from low-earth orbit (LEO) and utilize custom in-house Structure-from-Motion (SfM) algorithms to produce Digital Elevation Models (DEMs) of ground targets. MOCI will also be measuring the effectiveness of its in-house machine learning algorithms to perform object classification of pre-defined man-made ground targets. The testing and evaluation of each command that will be sent to the MOCI satellite is a crucial procedure that must be completed in order to ensure that the satellite and its subsystems successfully execute against commands received from the SSRL ground station over the course of the mission. MOCI is projected to launch early next year.

Abstract + Motivation

- Current Command Execution Testing (CET) procedures are conducted manually and can be arduous and impractical for frequent testing
- Inefficiency of an integral testing procedure can lead to delays within the other teams involved in the development of the cube satellite

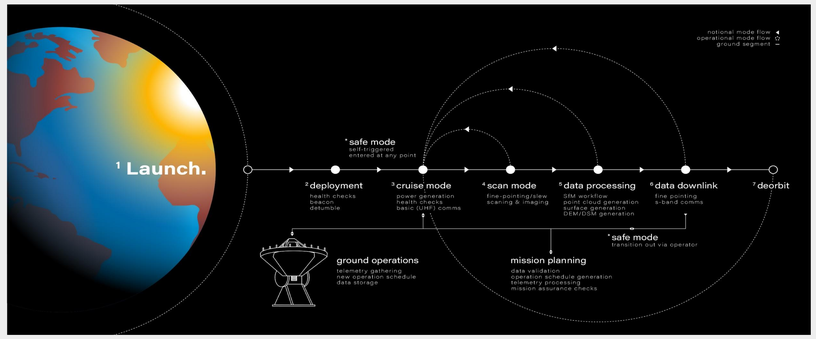

Figure 1 - Modes of MOCI

Methods

Method 1: Automation of the program

- Integration of scripting techniques enables full automation of the test.
- Minimal human intervention is required with this approach.
- Automation can help address the challenges of manual command execution.
- Optimal results in the efficiency of the program can streamline overall testing development amongst the flight software and MOPS teams.

Additional Methods

Method 2: Match commands with related anomalies

- Identify anomalies using the MOCI RVMS and other subsystems working
- Match those anomalies with the command and telemetry list
- Ensure the anomalies exist and are being tested by flight software in TMTCLab
- Specify which commands go into CET

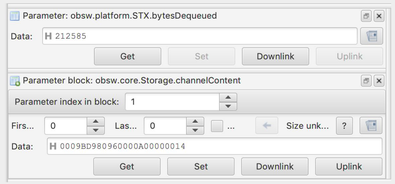

Figure 2 (R): TMTCLab
Figure 3 (L): Automation Script

```
class TestCommandExecution(unittest.TestCase):

    def setUp(self):
        pass

    # HK_BEACON Actions
    def test_hkbeacon_reset(self):
        result = tmtc.invoke('chd.tmtc.HKBeacon.reset')
        print(type(result))

    def test_hkbeacon_send(self):
        result = tmtc.invoke('chd.tmtc.HKBeacon.send')
        print(type(result))
```

Method 3: Perform Command Execution Testing for each of the respective tests

- Classify sections of commands according to their associated associated testing procedures
- Incorporate Command-Execution-Testing Procedures into the remaining four tests
- Evaluate commands that are not attributed to a specific testing procedure within the original Command Execution Test

Results & Evaluation

Method 1:

- Simultaneous execution of commands is not possible with the On Board Computer (OBC) due to its limited capability.
- The OBC can only process one command at a time and lacks multithreading support.
- Commands can be run concurrently based on a priority queue approach.
- The fragmented distribution of commands across different code sections makes comprehensive automation in a single step unfeasible.

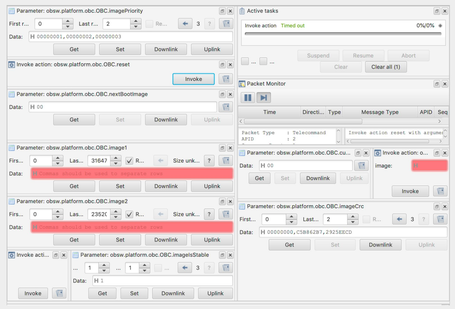

Figure 4: TMTCLab Parameters

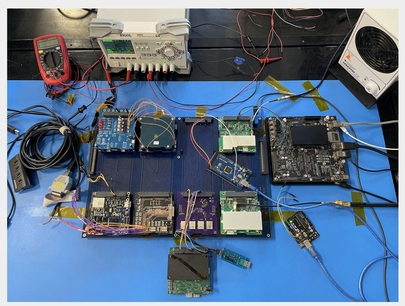

Figure 5: FlatSat

Results & Evaluation (contd.)

- Methods 2 and 3 involve rehashing and procedural changes to the existing CET procedures
- Efficiency of each method depends on the ability for either method to eliminate potential bottlenecks
- Bottlenecks may include dependencies on external teams, insufficient testing documentation, and inconsistencies in the testing process.

The table below demonstrates how method 3 can be accomplished through classification of commands according to testing procedure

batteryVoltage	Charge Cycle Test
batteryTemperature	
batteryHeaterStatus	
ADCSState	Attitude Determination and Control System
ADCSMeasurements	
ADCSPowerControl	
DownLinkSA	Simulated Communications
ActiveDownlinkSA	
recoverToCruise	Day in The Life
recoverToSafe	
setModeWithScheduleID	

Future Research:

The proper evaluation of methods 2 & 3 to determine the most optimal one is yet to be determined through performance evaluation of Command Execution Testing procedures at a future date.

Acknowledgements:

Cameron Bonesteel, Josh Messitte, Current & Former lab members