UAVPayloadTAQ

Subsystem Unit Testing – Image Processing and Target Detection

Project: UAVPayloadTAQ-G2 **WP Name**: Subsystem Integration Testing **WP Number**: WP-IMG-5	**Type of Test**: Unit Test	
Test Article: **Image Processing and Target Detection**	**Part Number**: N/A	**Serial Number**: N/A
System Requirements: REQ-M-3, REQ-M-4, REQ-M-6, REQ-M-12, REQ-M-13, REQ-M-14	**Test Equipment**: Raspberry Pi V2.1 Camera, Raspberry Pi 3B+	
Test Operators: James Arnold, Oliver Campbell	**Test Engineers**: James Arnold, Oliver Campbell	
Project Manager: Alexander Iftene	**Project Supervisor**: Dr Felipe Gonzalez	

Revision Record

Document Issue/Revision Status	Description of Change	Date	Approved
1.0	Initial Document	9 October 2020	Brian Sivertsen
1.1	Adding Post Integration Tests	19 October 2020	Brian Sivertsen

Table of Contents

List of Figures

Definitions

UAV	Unmanned Aerial Vehicle
UAVPayload	Unmanned Aerial Vehicle Payload
GCS	Ground Control Station
WEB	Web Visualization Subsystem
FTP	File Transfer Protocol
HLO	High Level Objective
SR	System Requirements
MDG9	Miscellaneous Dangerous Goods 9

1 Introduction

In order to ensure proper intended functionality for each subsystem it is important to conduct adequate testing to ensure they each operate correctly prior to integration.

Before a subsystem can be integrated with the other subsystems to form a complete product, testing must be performed on the subsystem. This serves to ensure the subsystem is operating as desired, and that it fulfills all the requirements set out for it.

1.1 Scope

This document outlines the Unit testing for the IMG subsystem. Testing is conducted using a Raspberry Pi serving dummy data to a Windows computer which is outputting the web server that is to be accessed locally by other devices on the network.

This document outlines the Unit testing performed on the IMG subsystem. Testing will aim to evaluate the performance of the target detection software, through repeated tests. The target detection will be tested with realistic sized targets at set intervals

1.2 Background

The Queensland University of Technology (QUT) Airborne Sensing Lab have appointed Group 2 of the EGH455 (Advanced Systems Design) class to design a UAV Payload for indoor air quality to be installed on a S500 UAV designed for navigating in GPS denied environments. The UAVPayloadTAQ is required to conduct constant air quality sampling in a simulated underground mine. During monitoring, it must find and identify multiple markers placed by miners around the mine. Additionally, QUT Airborne Sensing Systems requires that the UAVPayloadTAQ is designed and developed using Systems Engineering to ensure QUT Airborne Sensing Systems requirements are met. Taken from RD/2.

2 Reference Documents

2.1 QUT Avionics Documents

RD/1	CN-UAVPayloadTAQ-01	UAV Payload Customer Needs
RD/2	SR-UAVPayloadTAQ-01	UAV Payload System Requirements

2.2 Non-QUT Documents

3 Test Objectives

Unit testing has been conducted on the IMG subsystem to ensure that the specified system requirements were met. From RD/2 the relevant system requirements are:

- **REQ-M-3:** The UAVPayload^{TAQ} shall communicate with a ground station computer to transmit video, target detection and air quality data.
- **REQ-M-4:** The target identification system shall be capable of alerting the GCS of a targets type.
- **REQ-M-6:** The Web Interface is required to display the images of the targets that are taken directly from the UAVPayloadTAQ and updated every time a new picture is taken.
- **REQ-M-12:** The UAVPayload^{TAQ} shall process all imagery on-board via the on-board computer.
- **REQ-M-13:** The processing must be able to analyse all data from the flight path while the UAV moves at a maximum speed of 1m/s.
- **REQ-M-14:** The processing must be able to analyse all data from the flight path while the UAV operates at an altitude of 2m.

4 Test Setup and Equipment

4.1 Pre-Integration Testing Setup and Equipment

Testing was performed on the IMG subsystem prior to integration with the other subsystems. It was impossible to create the exact same conditions as our demonstration environment during testing, however we attempted to replicate the condition as closely as possible to ensure a successful demonstration.

The script was modified to capture 20 images and save it locally to the SD card. The script would otherwise behave in the exact same manner as it would post-integration. Each target was printed out on an A4 sheet of paper and several images were taken of the target at a range of distances. This will be further elaborated upon in section 5 below. Careful consideration was taken to the background of the testing environment to ensure results were fair and consistent.

No specialised equipment was necessary during testing.

4.2 Post-Integration Testing Setup and Equipment

After integration, the same testing environment was replicated, however the images would output to our web server in a loop indefinitely. We also ran our signal processing subsystem concurrently in order to ensure our flask server and hardware could handle the load. Our resolution and image capture rate was slowly adjusted until we reached the highest quality possible without encountering any bottlenecks for at least a 20 minute runtime. An app called ‘Conky’ was installed on the pi to monitor our CPU temp and other resources during post-integration testing.

5 Test Procedures

5.1 Pre-Integration Test Procedures

During pre-integration testing, the RPi and camera was mounted to the end of a wooden stick and maneuvered by the tester in such a way to simulate the movement of a drone flying at 1m/s. The script was modified to capture 20 images over a period of 10 seconds. The targets were printed on A4 sheets of paper and layed on the ground, to simulate the same testing conditions as the demonstration. The number of positive, negative and false positive images were recorded for each test, in order to determine which classifier was the most accurate.

The cascade classifiers were trained at specific parameter values for each test. When testing a new classifier, we would be sure to only change one parameter from one test to the next. Be minimising our number of variables being changed simultaneously, it becomes clear which variable was resulting in a change in our results.

During testing, we would capture 20 images at a distance of 0.5m, 1m and 2m altitude. These images would be saved locally to the Pi and transferred to a computer using file transfer software. The camera would be angled in order to capture the image at as many orientations as possible. A spreadsheet containing our parameter values and the test results was maintained in order to keep track of our most effective cascade classifier parameters.

5.2 Post-Integration Test Procedures

Post integration test was very similar, however images were outputted to the web server rather than being saved locally. Additionally, the signal processing subsystem was running concurrently, in order to assess the load on the web server and hardware. To assess hardware and particularly CPU temperature, ‘Conky’, a resource monitoring app was used. In addition to the above parameters, CPU temperature was recorded whilst the Pi was running all subsystems concurrently.

6 Test Results

These test results apply to the tests outlined above in section 5

6.1 Pre-Integration Test Results

All the tests were performed after loading the classifier onto the raspberry pi, and using it to apply the image detection software to the images. This satisfies **REQ-M-12**.

Parameters used in the opencv_traincascade presented as: number of positive images, number of negative images, number of stages, minimal hit rate, max false alarm rate. As tests were performed at a range of 2 metres, this shows that the software can detect the target at a distance of at least 2 metres, satisfying **REQ-M-14.**

MDG9

Parameters	0.5m Test	1m Test	2m Test	Notes
200, 100, 3, 1, 0.3	50%	45%	40%	Low detection overall
200, 100, 3, 0.5, 0.3	55%	50%	50%	
200, 100, 3, 0.2, 0.3	60%	55%	50%	
200, 100, 3, 0.1, 0.3	60%	60%	55%	
200, 100, 4, 0.1, 0.3	60%	45%	45%	
200, 100, 4, 0.1, 0.1	60%	60%	50%	
200, 100, 4, 0.3, 0.1	70%	70%	60%	
450, 100, 5, 0.3, 0.1	75%	70%	70%	
450, 100, 5, 0.1, 0.1	70%	60%	70%	
450, 100, 5, 0.1, 0.05	70%	60%	65%	Low detections
450, 100, 6, 0.1, 0.1	75%	70%	70%	
450, 100, 6, 0.1, 0.3	95%	95%	90%	Few false positives
450, 100, 6, 0.3, 0.3	90%	90%	90%	Increased false positive rate

The highlighted classifier was deemed satisfactory. However, it was realised that this classifier had only been trained to recognise the targets in an upright orientation, and upon further testing found that it struggled when the targets were in orientations with a rotation of more than approximately 20 degrees in either positive or negative direction. The testing process was repeated by changing the training dataset to have the targets in a complete range of orientations.

MDG9 - Repeat

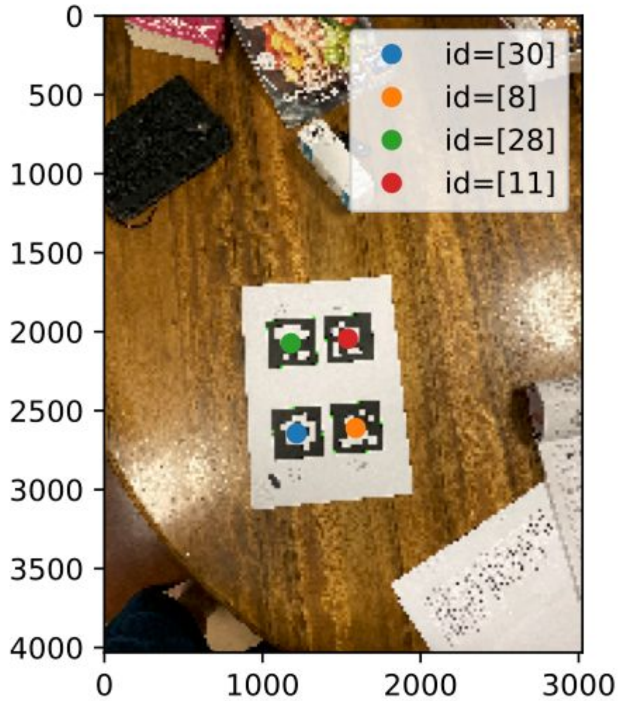
Parameters	0.5m Test	1m Test	2m Test	Mass Test
850, 150, 5, 1, 1	75%	70%	70%	Very high # of false pos
850, 150, 5, 0.5, 1	65%	70%	60%	Very high # of false pos
850, 150, 5, 0.5, 0.5	60%	60%	60%	High # of false pos
850, 150, 5, 0.5, 0.1	65%	70%	65%	Some false pos
850, 150, 5, 0.1, 0.1	75%	75%	80%	
850, 150, 6, 0.1, 0.1	80%	80%	75%	
850, 150, 6, 0.3, 0.1	95%	90%	95%	Extremely low # false pos

Training the cascade classifier for the second target, the Toxic Goods sign, started using similar parameters to the successful MDG9 classifier.

Toxic

Parameters	0.5m Test	1m Test	2m Test	Mass Test
850, 150, 5, 0.5, 0.5	70%	70%	70%	Very high # of false pos
850, 150, 5, 0.5, 0.1	65%	70%	65%	High # of false pos
850, 150, 5, 0.1, 0.1	65%	65%	70%	
850, 150, 6, 0.1, 0.1	60%	75%	60%	Lower detection
850, 150, 6, 0.3, 0.1	75%	75%	80%	
850, 150, 6, 0.3, 0.15	80%	80%	80%	
850, 150, 6, 0.2, 0.15	95%	95%	90%	Extremely low # false pos

A photo taken from the ARUCO training is shown below, from an extension of the task aiming to test detections full capabilities, trying to identify 4 targets at once at a size smaller than required. All ARUCO markers were identified correctly.



6.2 Post-Integration Test Results

MDG9

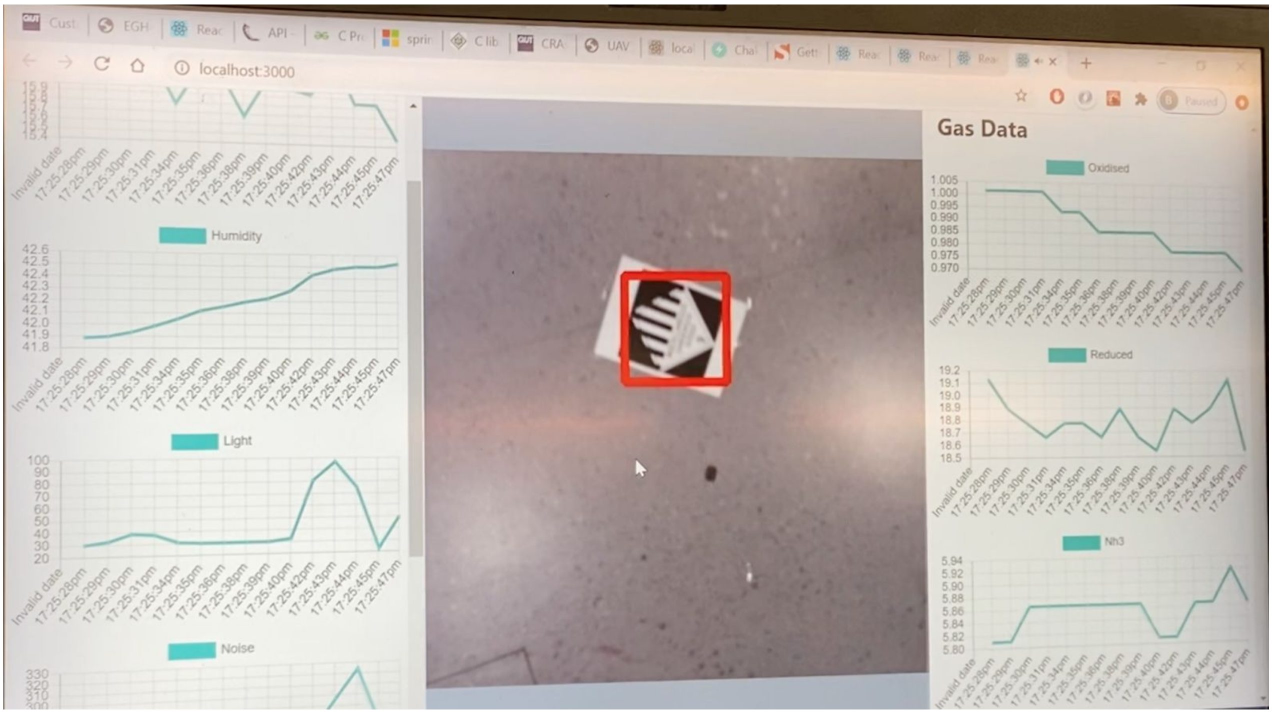
Parameters	0.5m Test	1m Test	2m Test	Notes	CPU Temp
850, 150, 6, 0.3, 0.1	90%	85%	90%		63°C
850, 150, 6, 0.3, 0.1	95%	95%	90%		65°C
850, 150, 6, 0.3, 0.1	90%	90%	95%		64°C

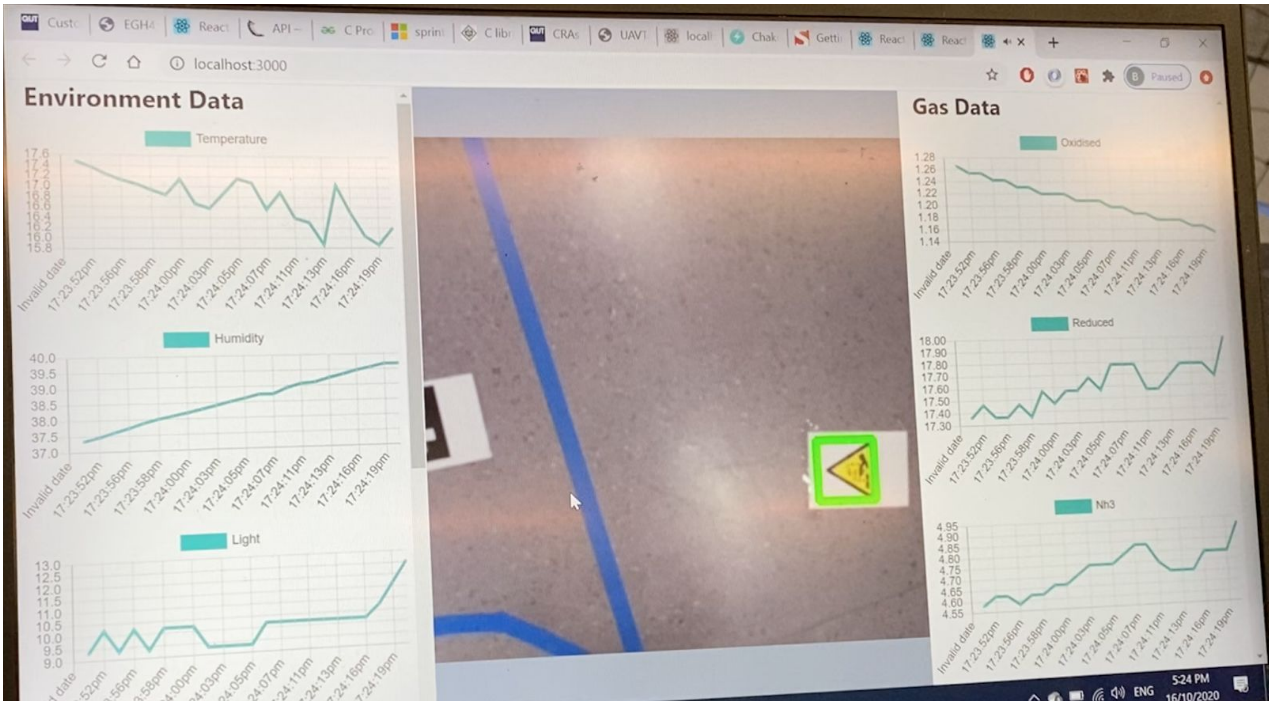
Toxic

Parameters	0.5m Test	1m Test	2m Test	Notes	CPU Temp
850, 150, 6, 0.2, 0.15	90%	90%	90%		65°C
850, 150, 6, 0.2, 0.15	95%	90%	85%		66°C
850, 150, 6, 0.2, 0.15	90%	90%	90%		65°C

The ARUCO marker detection was tested in a similar manner, but meaningful results could not be visually displayed, only by audio feedback. The ARUCO marker detection software worked as pre-integration, that is to say successfully.

During the demonstration, the assessment team verified an appropriate audio signal was played upon target detection. Additionally, the below images show that image displayed on the web server, with bounding boxes outlining the targets. This shows satisfaction of requirements **REQ-M-3**, **REQ-M-4**, & **REQ-M-6**.





localhost:3000
Environment Data
Temperature
Humidity
Light
Gas Data
Oxidised
Reduced
Nh3

7 Conclusion

Our testing methodology was fairly robust, however we didn't target any specific value that we would classify as being ready for use. We simply continued to test until we hit severe diminishing returns in performance. We chose our cascade classifier parameters in a pseudo-random fashion. If a particular parameter value was clearly not working, we would change it significantly instead of adjusting it gradually, which allowed us to save some time but is not the most objective testing process.

8 Appendices

8.1 Miscellaneous Dangerous Goods Sign

8.2 Toxic Target Sign