
 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-TAIP-TR-01 Issue: 1.2 Page: 1 of 35 Date: 28 Oct 2022
--	--	--

Target Acquisition & Image Processing Test Report

Project: UAVPAYG19 WP Name: WP-TAIP-06 WP Number: Subsystem testing		Type of Test: Unit Test	
Test Article: Target Acquisition Accuracy Image Processing Accuracy		Part Number: N/A	Serial Number: N/A
System Requirements: <div> <div> REQ-M-03 (WEBintegration) REQ-M-04 REQ-M-06 (WEBintegration) REQ-M-12 REQ-M-14 </div> <div> REQ-M-16 REQ-M-17 REQ-M-18 REQ-M-19 (WEBintegration) </div> </div>		Test Equipment: See “equipment used” section of each test	

Queensland University of Technology
Gardens Point Campus
Brisbane, Australia, 4001.


This document is Copyright 2022 by the QUT. The content of this document, except that information which is in the public domain, is the proprietary property of the QUT and shall not be disclosed or reproduced in part or in whole other than for the purpose for which it has been prepared without the express permission of the QUT

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-TAIP-TR-01 Issue: 1.2 Page: 2 of 35 Date: 28 Oct 2022
--	--	--

Test Operators: Alex Switala Connor Harvey	Test Engineers: Alex Switala Connor Harvey
Project Manager: Marissa Bowen	Project Supervisor: Dr Felipe Gonzalez

Test Summary

This report documents the testing of the hardware and software components of the Target Acquisition and Image Processing subsystem. Each individual test documents the equipment used, procedure undertaken and the resulting outcome. The purposes of these test are to ensure that the system requirements related to the TAIP subsystem are met. All the test completed for the system were successful in verifying the system requirements.


 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-TAIP-TR-01 Issue: 1.2 Page: 3 of 35 Date: 28 Oct 2022
--	--	--

Revision Record

Document Issue/Revision Status	Description of Change	Date	Approved
1.0	Initial Issue of Subsystem test report	12/10/2022	A.S
1.1	Added Tests	20/10/2022	A.S
1.2	Minor Updates to sections	28/10/2022	C.H

Table of Contents

Paragraph	Page No.
1 Introduction	7
1.1 Scope	7
1.2 Background	7
2 Reference Documents	8
2.1 QUT avionics Documents	8
2.2 Non-QUT Documents	8
3 Test Objectives	9
4 Testing	10
4.1 Software Tests	10
4.1.1 Software Test 1: Comparison of different yoloV3 models	11
4.1.2 Software Test 2: Framerate testing	12
4.1.3 Software Test 3: Accuracy of yoloV3 model, in realistic conditions	14
4.1.4 Software Test 4: Accuracy of the ArUco marker detection	16
4.2 Hardware Tests	18
4.2.1 Hardware Test 1: Camera Mounting	18
5 Results	20
6 Analysis	21
6.1 Hardware Analysis	21
6.2 Software Analysis	21
6.2.1 Software Test 1: Comparison of different yoloV3 models:	21
6.2.2 Software Test 2: Framerate testing:	21
6.2.3 Software Test 3: Accuracy of yoloV3 model in realistic conditions:	21
6.2.4 Software Test 4: Accuracy of the ArUco marker detection:	22
7 Conclusions and Recommendations	23
8 Appendix	24
8.1 Appendix A YoloModel.py:	24
8.2 Appendix B G19-Integration.py:	26
8.3 Appendix C ArucoDetect.py:	31
8.4 Appendix D YoloModel.py:	33


 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-TAIP-TR-01 Issue: 1.2 Page: 5 of 35 Date: 28 Oct 2022
--	---	--

List of Figures

Figure	Page No.
Figure 1 Framerate testing results	11
Figure 2 Model Framerate Test Results	13
Figure 3 ArUCO markers	16
Figure 4 Camera Mounting	18


List of Tables

Table	Page No.
Table 1: Target Acquisition and Image Processing Subsystem Requirements	9
Table 2 Software Tests	10
Table 3 Detection Accuracy	14
Table 4 Results from tests	20
Table 5: Requirements Met	23

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-TAIP-TR-01 Issue: 1.2 Page: 6 of 35 Date: 28 Oct 2022
--	--	--

Definitions

Acronym	Definition
TAIP	Target Analysis and Image Processing
FPS	Frames Per Second
QUT	Queensland University of Technology
UAV	Unmanned Aerial Vehicle
AI	Artificial Intelligence
ArUCO	Augmented Reality University of Cordoba
Wi-Fi	Wireless Fidelity
WVI	Web Visualisation and Interfaces
IP	Image Processing
YOLOv3	You Only Look Once, Version 3

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-TAIP-TR-01 Issue: 1.2 Page: 7 of 35 Date: 28 Oct 2022
--	--	--

1 Introduction

The team members of Group 19 have been appointed to research, design, plan and implement an Advance Sensor Payload (ASP) for Unmanned Aerial Vehicle (UAV) target detection and air quality monitoring in GPS denied environments. The group has committed to the specified budget whilst implementing the project requirements stated by the client. The team has also committed to meeting the deadline date specified by the client with a full functioning ASP that has been tested to ensure the client requirements have been met.

This test report covers testing for the Target Acquisition and Image Processing subsystem detailing the detection of the ArUCO markers as well as the image processing component of the subsystem, along with the camera's mounting to the enclosure with how it affects the vision of the system.

1.1 Scope


The scope of the project is to research, plan, design, implement and test the ASP for UAV target detection in GPS denied environments. This document contains the objectives of the test, the equipment used, in depth descriptions of the tests, results, an analysis of these results and a conclusion with recommendations. The purpose of this test document is to see if the tests satisfies the stated System Requirements/HLO's in RD-1

1.2 Background

The Queensland University of Technology's Airborne System Lab (ASL) has commissioned the group UAVPAYG19 to design and develop a payload capable in detecting specific objects, recording air quality data to be displayed on a web interface and to pierce a ground sample. This payload is to be attached to a S500 UAV which will complete an automated flight path. The payload is mounted on the bottom of the UAV using a provided bracket. This payload must contain all components to complete its required tasks. These components are:

- Raspberry Pi 3b+
- Raspberry Pi Camera
- Pimoroni Enviro+ sensor
- DF15RSMG 360 Degree Motor

The payload is required to identify three targets, a valve (In open or closed position), a fire extinguisher and an ArUCO marker. The Pimoroni sensor is to be used to record air temperature, pressure humidity, light and potentially hazardous gas level data. This data along with a live feed of the Raspberry Pi Camera is to be visualized on a Web Interface. Lastly a soil sample must be obtained using a sampling mechanism.

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-TAIP-TR-01 Issue: 1.2 Page: 8 of 35 Date: 28 Oct 2022
--	--	--


2 Reference Documents

2.1 QUT avionics Documents

RD/1	UA–System Requirements	UAVPayloadTAQ System Requirements
RD/2	UA –Customer Needs	Advanced Sensor Payload for UAV Target Detection and Air Quality Monitoring in GPS Denied Environments
RD/32	UAVPAYG19-TAIP-FD-01	TAIP Final Design Report

2.2 Non-QUT Documents

No Non QUT Documents referenced


 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-TAIP-TR-01 Issue: 1.2 Page: 9 of 35 Date: 28 Oct 2022
--	--	--

3 Test Objectives

The objectives of the tests performed in this document are to ensure that the Target Acquisition and Image Processing subsystem (TAIP) is independently functional before integrating with the other subsystems contained within the payload. The tests achieved are used to confirm that the TAIP subsystem fulfills the system requirements listed in Table 1. If any test results are unsuccessful, a design change will be implemented to improve the subsystem until the test results meet the criteria listed within the system requirements.

Table 1: Target Acquisition and Image Processing Subsystem Requirements

Requirement Code	Description
REQ-M-04	The target identification system shall be capable of alerting the GCS of a target's type.
REQ-M-14	Developed solution shall conform to the systems engineering approach.
REQ-M-16	The UAVPayloadTAQ shall process all imagery on-board via the on-board computer
REQ-M-17	The processing must be able to analyse all data acquired from the camera and sensors while the UAV moves at a maximum speed of 2 m/s.
REQ-M-18	The processing must be able to analyse all data acquired from the camera and sensors while the UAV operates at an altitude of between 1 to 3m.

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-TAIP-TR-01 Issue: 1.2 Page: 10 of 35 Date: 28 Oct 2022
--	--	---


4 Testing

Testing will be split into two parts containing software and hardware testing. Due to software being a major component of the TAIP subsystem, this report will contain more software tests with a few hardware tests focusing on the camera and Raspberry pi itself. The tests included within this document were performed to ensure that the solution developed adhered to the system requirements, with modifications being made at points when it did not manage to meet these requirements to an acceptable degree.

4.1 Software Tests

Table 2 Software Tests

Software Test	Description of the test
Comparison of different yoloV3 models	The YoloV3 320 and YoloV3 Tiny models were trained using darknet with their performance tested against each other. The resulting outcome determining which model would be used on the pi.
Testing yoloV5 models	The YOLOv5 nano model was trained and tested on the pi to determine if it would run any better than yoloV3. The outcome determining which model could be used.
Framerate testing	The framerate of the pi with and without the yoloV3 model was tested to determine the maximum capabilities of the pi.
Code test of threading	Threading for the model was implemented and then compared with an unthreaded model on the pi so that we could determine if it improved the speed at which the image detection could be performed.
Accuracy of yoloV3 model	Three iterations of the yoloV3 model were trained and implemented with revisions being done to improve the model's accuracy each time.
Accuracy of the ArUCO marker detection	<p>Markers of differing orientations and positions will be tested for accuracy.</p> <p>The accuracy of the detection will be tested along with markers of differing orientations, positions, detecting multiple markers as well as marker id's and finally if it is checking</p>

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-TAIP-TR-01 Issue: 1.2 Page: 11 of 35 Date: 28 Oct 2022
--	--	---

4.1.1 Software Test 1: Comparison of different yoloV3 models

This software test is to compare the features of the yoloV3_320 model which is for larger images and the yoloV3 tiny which is for better performance while running, and finally yoloV5 which was a different type of model all together.

Equipment used:

The equipment used in this test consists of the following:

- Raspberry Pi
- Raspberry Pi camera
- Laptop/Desktop

Procedure:

Following is the procedure used to conduct the test:

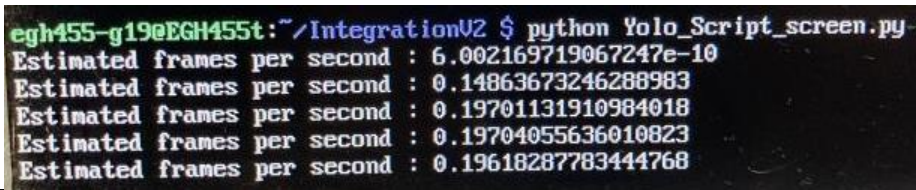
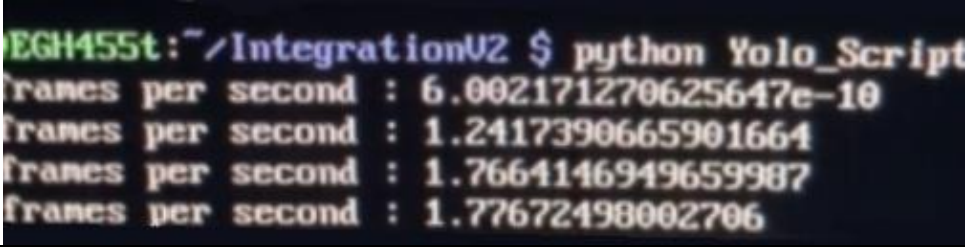
1. Turn on and log into the pi or access the pi
2. with the pi, run the yoloV3 model using the main script and write down the framerate
3. Repeat step 2 with the custom yoloV3 tiny model
4. Repeat step 2 with the custom yoloV5 nano model


Code:

The code that was run for this was the YoloModel.py script, changing the variable modelConfiguration and modelWeights, to match the respective yolo models cfg and weights files, full code included within appendix item D.

Results and Evidence:

Figure 1 Framerate testing results

Model	Frames per second
YoloV3 320	0.197
	
YoloV3 Tiny	1.77
	
YoloV5 Nano	0.22

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-TAIP-TR-01 Issue: 1.2 Page: 12 of 35 Date: 28 Oct 2022
--	--	---

```

egh455-g19@EGH455t:~/Integration02 $ python Yolo_Script_sc
Estimated frames per second : 7.502707321764913e-11
Estimated frames per second : 0.15351645047085213
Estimated frames per second : 0.21921858889071935
Estimated frames per second : 0.22154817673889507

```

During the testing it was found that YoloV3 Tiny had the best performance of the three models, with YoloV5 Nano coming in at second place with its performance of 0.22 frames per second. This however meant that the YoloV3 Tiny model was the best model for our detection as it ran the quickest on our machine which was very important for counteracting the motion blur of the drone during movement. The result of this test was the decision to use the YoloV3 Tiny model rather than the other models.

4.1.2 Software Test 2: Framerate testing

Equipment used:

The equipment used in this test consists of the following:

- Raspberry Pi
- Raspberry Pi camera
- Laptop/Desktop

Procedure:


Following is the procedure used to conduct the test:

1. Turn on and log into the pi or access the pi
2. with the pi, run the custom yoloV3 tiny model using the main script and write down the framerate
3. With the pi, run the script without the model, and write down the framerate

Code:

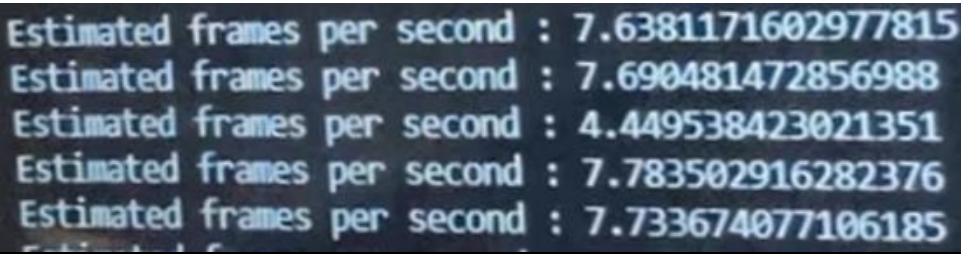
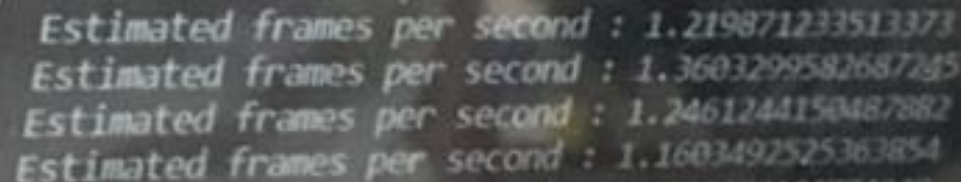
G19-Integration.py was the code used for this test, with the flag of the model being set to false for the test with no model and set to true for the test with the model.

This was a part of the main script attached to this document in the appendix item B.

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-TAIP-TR-01 Issue: 1.2 Page: 13 of 35 Date: 28 Oct 2022
--	--	---

Results and Evidence:

Figure 2 Model Framerate Test Results

No model framerate (around 7.6-7.8 fps)
 <p>Estimated frames per second : 7.6381171602977815 Estimated frames per second : 7.690481472856988 Estimated frames per second : 4.449538423021351 Estimated frames per second : 7.783502916282376 Estimated frames per second : 7.733674077106185</p>
With model framerate (around 1.1-1.4 fps)
 <p>Estimated frames per second : 1.219871233513373 Estimated frames per second : 1.3603299582687245 Estimated frames per second : 1.2461244150487882 Estimated frames per second : 1.1603492525363854</p>

The results of this test was a baseline framerate for the pi overall. This ended up being 7.6-7.8 frames per second when not detecting images. When using the model the framerate decreased to 1.1 to 1.4 frames per second and this was due to the significant processing power that the image detection model took when attempting to find and identify images.

An outcome of this test was that the ArUco marker detection and the image detection and processing power was split when a ArUco marker was detected. Therefore, it was set to only having the model running when a marker was not detected. This allows for more a better performance in frame rate when in range of a marker.

4.1.3 Software Test 3: Accuracy of yoloV3 model, in realistic conditions

Equipment used:

The equipment used in this test consists of the following:

- Desktop

Procedure:

Following is the procedure used to conduct the test:

1. Open windows powershell
2. Navigate to the installation of darknet using powershell commands e.g. >cd C:/USER/darknet
3. Run the command written in the code section of this test
4. Look at the output and look at the accuracy of it, note down the most accurate and least accurate values for items detected as well as any erroneous items detected
5. Repeat tests for 1.5m video and 2.5 m video of the actual drone, recording the values

Code:


The following code was input into windows powershell after navigating to the darknet directory:

```
./darknet.exe detector demo data/obj.data cfg/yolov3-custom-tiny.cfg backup/yolov3-custom-tiny_final.weights 2_5mTestVid.mp4 -out_filename Result_of_Test_flight_V3_2_5m.mp4
```

Results and Evidence:

Table 3 Detection Accuracy

	Highest accuracy for item Test flight 1.8m	Lowest accuracy for item Test flight 1.8m	Highest accuracy for item Test flight 2.5m	Lowest accuracy for item Test flight 2.5m
Fire Extinguisher	100%	56%	99%	67%
Open Valve	99%	32%	87%	33%
Closed Valve	95%	53%	91%	45%
Mistaken item/Error	Open Valve as closed Valve: Open 55%, Closed 31% Blurry light as Open Valve: 53% Blurry light as Closed Valve: 41%	Open Valve as closed Valve: Open 43%, Closed 31% Blurry light as Open Valve: 30% Blurry light as Closed Valve: 27%	Open Valve as closed Valve: Open 72%, Closed 92%	Open Valve as closed Valve: Open 30%, Closed 48%

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-TAIP-TR-01 Issue: 1.2 Page: 15 of 35 Date: 28 Oct 2022
--	--	---

Link to Test flight videos:

YT Link: [Result of Test flight V3 1.8m](#)

YT Link: [Result of Test flight V3 2.5m](#)

The test included speeds and heights at which the drone would be operating at. The test video of the drone flying at 1.8m and 2.5m at the maximum speed of 2m/s allowed for the model to be tested in realistic conditions. Omitted from the test however was the capped framerate of the model, although despite this, the test for the model under the conditions was a success.


Overall the results of this test was promising for fire extinguishers, with them having a very high degree of accuracy ranging from 56% to 100%. The open valve at a maximum of 99% accuracy to a low of 32% accuracy with a chance at when it drops below 55% accuracy for the model to consider it as a closed valve. The closed valves have a higher lower bound than the open valves, of 45% but a lower upper bound for accuracy of 95%.

Fire Extinguisher results:

The result of this test means that the overall accuracy of the model for fire extinguishers was excellent and a threshold value of 56%+ could be used just in case there was a mistaken item that the model picked up as a fire extinguisher that fell under that accuracy value.

Valve Results:

Overall, the model was able to pick up Closed valves with a much higher rate of accuracy (53%) when compared to the Open valves (32%) for its lower threshold. The testing also showed that open valves could be picked up as two items at once with them being able to be picked up as closed valves as well as open valves. This however could be removed using a threshold of 50% for closed valves when its being picked up as an open valve and 55% for open valves otherwise.

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-TAIP-TR-01 Issue: 1.2 Page: 16 of 35 Date: 28 Oct 2022
--	--	---

4.1.4 Software Test 4: Accuracy of the ArUco marker detection

Equipment used:

The equipment used in this test consists of the following:

- Desktop
- Webcam
- Printed ArUco marker

Procedure:

Following is the procedure used to conduct the test:

1. Start Recording with the camera
2. Open the camera interface through a script
3. Place the printed marker in front of the camera
4. Write down position values in two different measured spots moving the marker between those twice
5. Check the results and repeat if not satisfactory
6. Stop recording

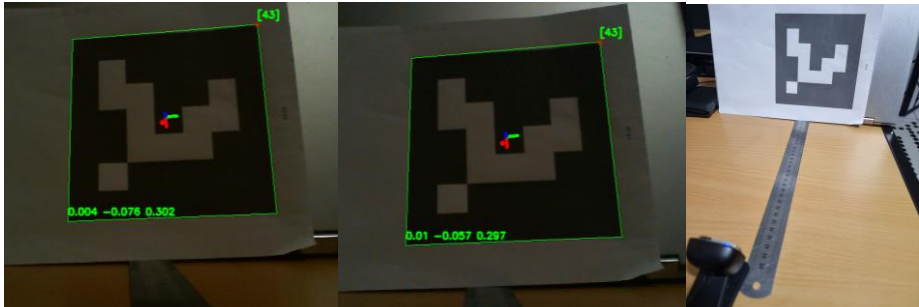
Code:


The code used to view the ArUco positions through the camera is in the ArucoDetect.py file, located in Appendix item C.

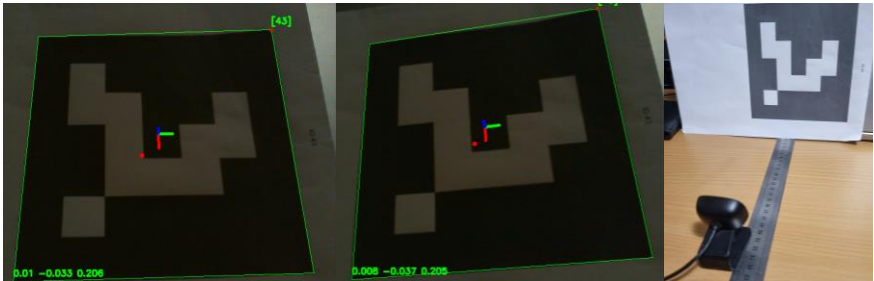
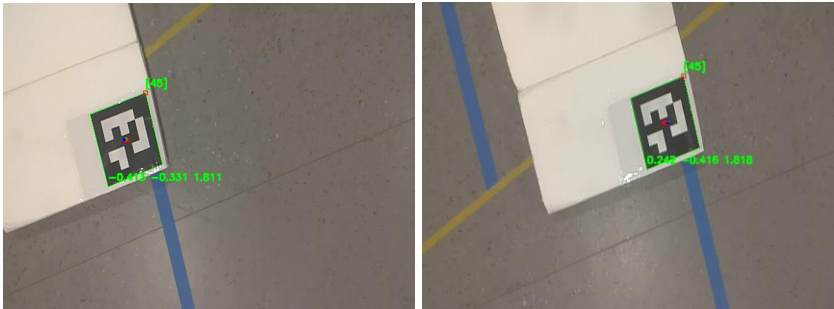
Results and Evidence:


As this is purely a software test, the camera used does not have a set restriction if it has been calibrated using the open cv checkerboard method. The code performed adequately well with detecting the distances of the ArUco markers at three separate locations. The test drone footage was also used to determine if the model could accurately detect the marker within a 1-3m range. Each location was measured twice with the maximum error range recorded to be 0.7%. This error range is within the acceptable margin resulting in a successful outcome for this test.

Figure 3 ArUco markers

Marker Distance	Image
Exact Distance: 30cm Recorded Distance: 29.7-30.2cm	

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-TAIP-TR-01 Issue: 1.2 Page: 17 of 35 Date: 28 Oct 2022
--	--	---

Exact Distance: 20cm Recorded Distance: 20.6cm	
Exact Distance: 1.8m Recorded Distance: 1.811~1.818m	

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-TAIP-TR-01 Issue: 1.2 Page: 18 of 35 Date: 28 Oct 2022
--	--	---

4.2 Hardware Tests

There is only one hardware test in this report which is the mounting of the camera.

4.2.1 Hardware Test 1: Camera Mounting

This test was to see if the camera has been mounted as well into the enclosure and what sort of effects it has on its field of view.

4.2.1.1 Equipment Used:

The equipment used in this test consists of the following:

- Raspberry Pi
- Raspberry pi camera
- Enclosure
- Laptop/Desktop

Procedure:




1. Mount the camera to the pi (ignore if already attached)
2. Mount the camera and pi to the enclosure
3. Lift the mounted enclosure off the ground up to around 2-3 meters
4. Check the field of view of the camera from the console, camera, ect.

Code:

The code to view images was the main server code and the G19-Integration.py code being run on the pi and the server respectively, code in appendix item B.

Results and Evidence:

Figure 4 Camera Mounting

Image showing that the full lens is exposed and able to view things:	Image of the position of the camera:	Image showing that the camera remains above the ground on the drone
		


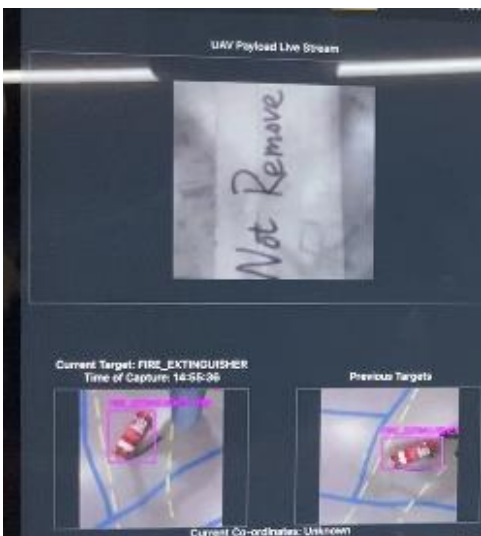
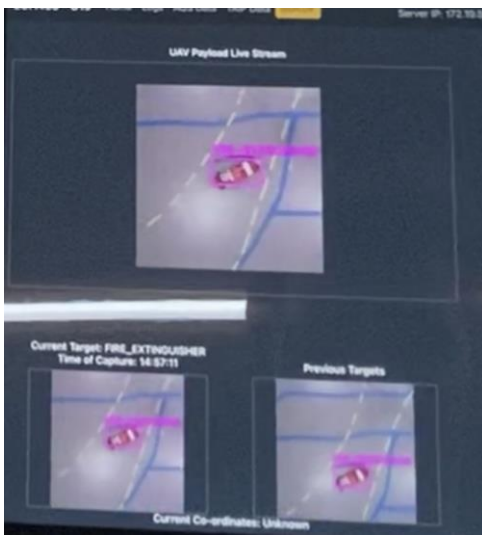

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-TAIP-TR-01 Issue: 1.2 Page: 19 of 35 Date: 28 Oct 2022
--	--	---

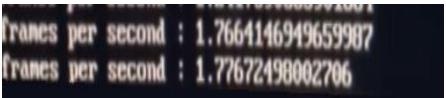
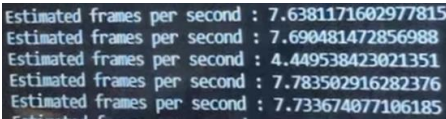
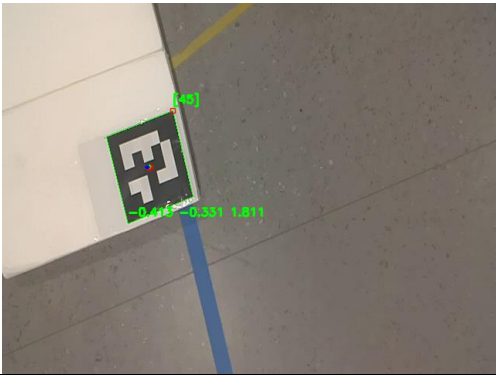

Image when drone is on the ground:	Image when drone is 2-3m up:
	


When mounted to the drone, the image came out clearly with a field of view directed towards the ground directly under the drone. When the drone was resting on the ground the camera was still able to view things and when lifted above the ground 2-3 meters able to still make out and identify images using the model.

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-TAIP-TR-01 Issue: 1.2 Page: 20 of 35 Date: 28 Oct 2022
--	--	---

5 Results

Table 4 Results from tests

Test Title	Result	IMG	Requirement Met
Comparison of different yoloV3 models	Passed		REQ-M-16, REQ-M-14, REQ-M-17
Framerate testing	Passed		REQ-M-16, REQ-M-14
Accuracy of yoloV3 model in realistic conditions	Passed		REQ-M-04, REQ-M-14, REQ-M-17, REQ-M-18
Accuracy of the ArUco marker detection	Passed		REQ-M-16 REQ-M-17, REQ-M-18
Camera Mounting	Passed		REQ-M-04

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-TAIP-TR-01 Issue: 1.2 Page: 21 of 35 Date: 28 Oct 2022
--	--	---

6 Analysis

The analysis section will discuss if the aims of the test have been achieved and were there any issues involved with this.

6.1 Hardware Analysis

During the hardware testing there were no issues with camera mounting, nor the obstruction of its vision at any point with the field of view. This allowed for the software for the target identification and detection to be unhindered by hardware. The main goal of the hardware tests to prove that the camera could fully function while attached to the enclosure. (for figures see section 4.2.1)

6.2 Software Analysis

For the software tests all aims were achieved that were set out, which were to meet the requirements listed in section 3, as well as fulfil the test objectives as listed below:

6.2.1 Software Test 1: Comparison of different yoloV3 models:

This test was designed to help the group to identify which of the 3 models within the test to use with the final model being chosen due to its performance on the raspberry pi. This was able to be achieved with no issues encountered during testing, other than the time it took to train each model which was about 10 hours for yoloV3_320, 1.5 hours for yoloV3_tiny and finally 8 hours for yoloV5_nano.

6.2.2 Software Test 2: Framerate testing:


This test was designed to check what the maximum performance of any model on the raspberry pi would be to gauge how successful our solution was. The test revealed that there was an issue with the pi that was being used and was discovered when comparing the model on different pi. The original pi had an issue of overheating which caused problems with the maximum framerate. This issue was fixed by switching the pi with another which immediately solved the issues causing the framerate to be around 10 times greater than it originally was with the fix being to switch the micro SD card.

Using what was discovered in the test, the performance was about to be increased by stopping the model when an ArUco marking was in frame. This allowed for much higher framerates and more accurate readings on these markers. This overall improved the performance of the subsystem.

6.2.3 Software Test 3: Accuracy of yoloV3 model in realistic conditions:


This test was to check if the model would work under the realistic conditions. Given videos of the actual drone flying was supplied and when using this, issues were faced during which was mainly false positives. This was fixed later by retraining the model using empty background pictures without items in it so that they would not be picked up during the actual flight.

Earlier in the development of the model, there were fewer accurate readings of the open and closed valves. These were corrected by retraining the model and gathering new separate images of the open and closed valves. These new images included the valve partially rotated. In doing so it improved the accuracy of the model for incorrect readings.

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-TAIP-TR-01 Issue: 1.2 Page: 22 of 35 Date: 28 Oct 2022
--	--	---

6.2.4 Software Test 4: Accuracy of the ArUco marker detection:


This test examined if the ArUco markers could be detected, along with the checking the accuracy of the detections. There were no issues in this test, as the ArUco detection code performed extraordinarily well resulting in a successful detection of ArUco markers and localisation of the drone.

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-TAIP-TR-01 Issue: 1.2 Page: 23 of 35 Date: 28 Oct 2022
--	--	---

7 Conclusions and Recommendations

Table 5: Requirements Met

Requirement Code	Description	How the requirement was met
REQ-M-04	The target identification system shall be capable of alerting the GCS of a target's type.	Met: - This requirement was met by having the model be accurate in its detections, proven by the Accuracy of yoloV3 model in realistic conditions as this was able to identify the target type.
REQ-M-14	Developed solution shall conform to the systems engineering approach.	Met: - This requirement was met in each test as they all conformed to the approach and methods of systems engineering.
REQ-M-16	The UAVPayloadTAQ shall process all imagery on-board via the on-board computer	Met: - This requirement was met by Comparison of different yoloV3 models, and Framerate testing which both had the model running using the on-board computer aka the raspberry pi.
REQ-M-17	The processing must be able to analyse all data acquired from the camera and sensors while the UAV moves at a maximum speed of 2 m/s.	Met: - This requirement was met by the Accuracy of yoloV3 model in realistic conditions test with the included videos in that test showing it running under realistic conditions with a max speed of 2m/s.
REQ-M-18	The processing must be able to analyse all data acquired from the camera and sensors while the UAV operates at an altitude of between 1 to 3m.	Met: - This requirement was met by the Accuracy of yoloV3 model in realistic conditions test with the included videos showing it running under realistic conditions at a height of 1.8m and 2.5m.

 Queensland University of Technology	QUT Systems Engineering UAVPAYG19	Doc No: UAVPAYG19-TAIP-TR-01 Issue: 1.2 Page: 24 of 35 Date: 28 Oct 2022
--	---	---

8 Appendix

8.1 Appendix A YoloModel.py:

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 29 09:00:00 2022

@author: Alex, Connor
Image Detection with yolov3 and opencv
"""

import cv2
import numpy as np
import time

whT = 320
#define classes
classNames = ['Closed_valve', 'Fire_extinguisher', 'Open_valve']
#confidence threshold
confThreshold = 0
#The lower this value the more aggressive it will be at removing boxes
nmsThreshold = 0.3

modelConfiguration = 'yolov3-custom-tiny.cfg'
modelWeights = 'yolov3-custom-tiny_final.weights'

class Yolo:
    def __init__(self):
        self.timeStart = time.time()
        self.net = cv2.dnn.readNetFromDarknet(modelConfiguration, modelWeights)
        self.net.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)
        self.net.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU)

    def start(self):
        return

    def update(self, img):
        #opencv needs the images as blobs, here we convert these to the blob
        #format
        blob = cv2.dnn.blobFromImage(img, 1/255, (whT, whT), [0, 0, 0], 1, crop=False)
        self.net.setInput(blob)
        layerNames = self.net.getLayerNames()
        #0 is not included within the layerNames so this is taken into account
        with
        #the -1
```



```
        outputNames = [layerNames[i-1] for i in
self.net.getUnconnectedOutLayers()]
        outputs = self.net.forward(outputNames)
        self.findObjects(outputs,img)
        self.img = img

#function to find the object on the screen
def findObjects(self, outputs,img):
    hT, wT, cT = img.shape
    #bounding box
    bbox = []
    #class identifier
    classIds = []
    #confidence intervals
    confs = []

    for output in outputs:
        for detection in output:
            scores = detection[5:]
            classId = np.argmax(scores)
            confidence = scores[classId]

            if classId == 1:
                confThreshold = 0.8
            else:
                confThreshold = 0.4

            if confidence > confThreshold:
                #print(str(classId) + ' ' + str(confidence))
                #w and h of the bounding box
                w,h = int(detection[2]*wT) , int(detection[3]*hT)
                #x and y are the centrepoint
                x,y = int((detection[0]*wT)-w/2), int((detection[1]*hT)-h/2)
                bbox.append([x,y,w,h])
                classIds.append(classId)
                confs.append(float(confidence))

    #output will be the indices of the bounding box to keep
    indices = cv2.dnn.NMSBoxes(bbox,confs,confThreshold,nmsThreshold)

    self.target = ""
    self.targetConfidence = 0

    for i in indices:
        if(int(confs[i]*100) > self.targetConfidence):
            self.targetConfidence = int(confs[i]*100)
            self.target = classNames[classIds[i]].upper()
```



```
        box = bbox[i]
        x,y,w,h = box[0],box[1],box[2],box[3]
        cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,255),2)
        cv2.putText(img,f'{classNames[classIds[i]].upper()}
{int(confs[i]*100)}%',
                    (x,y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.6,(255,0,255),2)
```

8.2 Appendix B G19-Integration.py:

```
import numpy as np
import sys
import cv2
import time
import math
import VideoStreamOverride
import AirQuality
import YoloModel
import ArucoDetect
import requests
import base64
from datetime import datetime
from time import sleep
from IP_Temp_Img import lcd_text_dis, get_ip
import servov2

import cv2 as cv
from PIL import Image
import ST7735 as ST7735

url = 'http://172.19.52.193:5001'
Imgurl = url+'/taip/taip'
AQurl = url+'/aqs/aqs'
Deployurl = url+'/st/deployButton'
Deployurl2 = url+'/st/getDeployFlag'

disp = ST7735.ST7735(
port =0 ,
cs= ST7735.BG_SPI_CS_FRONT , # BG_SPI_CSB_BACK or BG_SPI_CS_FRONT
dc=9 ,
backlight =19 , # 18 for back BG slot , 19 for front BG slot .
rotation =90 ,
spi_speed_hz = 4000000
)
WIDTH = disp.width
HEIGHT = disp.height
```



```
# Initialize display .
disp.begin()

HRes = 320
VRes = 320

UseModel = True
UseAruco = True
UseAQ = True
UseServer = True

AQSubsystem = AirQuality.AirQualitySensor()
Vision = VideoStreamOverride.VideoStream((HRes,VRes),framerate=30,UseModel=UseModel,UseAruco=UseAruco)
Model = YoloModel.Yolo()
Aruco = ArucoDetect.Aruco()
Servo = servov2.servo()

class Payload:
    def __init__(self):
        self.test = None
        self.temperature = None
        self.Deploy = False
        self.SendDeploy = False
        self.CanDeploy = False
        self.OnceDeploy = True
        self.Distances = [1000,1000,1000,1000]

    def start(self):
        print("Starting Payload")
        try:
            Vision.start()
            if(UseAQ):
                AQSubsystem.start()
            time.sleep(1)
            aa = 1
            # used to record the time when we processed last frame
            prev_frame_time = 0

            # used to record the time at which we processed current frame
            new_frame_time = 0

            delay = 0.5 # Debounce the proximity tap
            mode = 0 # The starting mode
            last_page = 0
```



```
light = 1

image_timer = time.time()
aqs_timer = time.time()

while(1):
    new_frame_time = time.time()
    frame, grabbed = Vision.read()
    if(not frame.any()):
        continue

    name = "default"
    xyz = "Unknown"

    if(UseAruco):
        Aruco.update(frame)

    if(UseModel and Aruco.ids is None):
        try:
            Model.update(frame)
            if(Model.targetConfidence > 0):
                name = f"{Model.target}"
        except:
            print("Model Failed")

    if(UseAruco):
        try:
            Aruco.draw(frame)
            if Aruco.ids is not None:
                if(time.time()-image_timer > 1):
                    name = f"Aruco: {Aruco.ids[0]}"
                    image_timer = time.time()

                xyz = Aruco.xyz
                average = sum(self.Distances)/len(self.Distances)

                if((Aruco.distance > 0.2 and average > 50) or
(average < 50)) and 45 in Aruco.ids[0]:

                    self.Distances.append(Aruco.distance)
                    self.Distances.remove(self.Distances[0])

                    average = sum(self.Distances)/len(self.Distances)
                    print(f'{average} - {Aruco.distance}')

                    if(average < 0.6 and not self.Deploy and 45 in
Aruco.ids[0]):

                        self.Deploy = True
```



```
self.SendDeploy = True

except:
    print("Couldn't Draw ArUco")

try:
    if(UseServer):
        self.sendImage(frame,name,xyz)
except:
    print("Couldn't send image")

if(UseAQ):
    AQSubsystem.update()

if((time.time() - aqs_timer > 5) and UseAQ and UseServer):
    aqs_timer = time.time()
    try:
        #print("ping aqs")
        r = requests.post(url = AQurl, json={"data":
AQSubsystem.to_json()})

    except:
        print("Couldn't ping AQS")

if(UseAQ):
    if AQSubsystem.proximity > 1500 and time.time() - last_page >
delay:

        print("Change")
        mode = (mode + 1) % 3

last_page = time.time()

if mode == 0:
    lcd_text_dis(get_ip())

if mode == 1:
    lcd_text_dis(str(round(AQSubsystem.temperature,3)) + " C")

if mode == 2:
    # This is the image display
    im_pil = Image.fromarray( frame )
    # Display the resulting frame
    # Resize the image
    im_pil = im_pil.resize(( WIDTH , HEIGHT ) )
    # display image on lcd
    disp.display( im_pil )

if(self.Deploy and self.SendDeploy and UseServer):
```



```
        try:
            r = requests.post(Deployurl1, {"deploy": True})
            if(r.status_code == 201):
                self.SendDeploy = False
                print("Enabled Button")
        except:
            print("Deploy Post Failed")
    if(self.Deploy and not self.SendDeploy and UseServer):
        try:
            getr = requests.get(Deployurl12)
            if getr.json()[0]['deployFlag']:
                self.CanDeploy = True
                self.Deploy = False
                print("Button Pressed")
        except:
            print("Get Request Failed")

    if(self.CanDeploy and self.OnceDeploy):
        print("Deploying")
        #time.sleep(5)
        Servo.start()
        print("Moving Down")
        Servo.motor_change(1)
        print("Moving Up")
        Servo.motor_change(-1)
        print("Stopping Motor")
        Servo.motor_change(0)
        Servo.cleanUp()
        self.CanDeploy = False

    #end = time.time()
    #print(f"[Iteration {aa}]: {end-start}/")
    fps = 1/(new_frame_time-prev_frame_time)
    prev_frame_time = new_frame_time
    print("Estimated frames per second : {0}".format(fps))

    aa = aa + 1

    #Display image
    #cv2.imshow('WebcamVideoa',frame)
    #cv2.waitKey(1)
    #DrawnFrame = Vision.read()
    #AQSubsystem.Display(DrawnFrame)
except:
    print("Unexpected error:", sys.exc_info())
```

```
        cv2.destroyAllWindows()
        self.stop()

#===== HELPER FUNCTIONS =====

    #stop : Closes the application gracefully
    def stop(self):
        print("stop 1")
        Vision.stop()
        cv2.destroyAllWindows()
        return

    def sendImage(self, image, name="default", xyz = "Unknown"):
        retval, buffer = cv2.imencode('.jpeg', image)
        jpg_as_text = base64.b64encode(buffer)
        base64img = {"data": jpg_as_text, "name" : name, "xyz": xyz,
"time":datetime.now().strftime("%H:%M:%S")}
        r = requests.post(Imgurl, data=base64img)
        return r

#===== MAIN PROGRAM EXECUTION =====

time.sleep(2.0)
robot = Payload()
robot.start()
print("sleep for 3")
time.sleep(3.0)
print("3 is up")
robot.stop()
```

8.3 Appendix C ArucoDetect.py:

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 29 09:00:00 2022

@author: Alex
Image Detection with yolov3 and opencv
"""

import numpy as np
import cv2
import sys
#from utils import aruco_display
#from utils import ARUCO_DICT
import argparse
import time
```

```
matrix_coefficients = np.load("calibration_matrix.npy")
distortion_coefficients = np.load("distortion_coefficients.npy")
aruco_dict_type = cv2.aruco.DICT_5X5_100
cv2.aruco_dict = cv2.aruco.Dictionary_get(aruco_dict_type)

class Aruco:
    def __init__(self):
        self.timeStart = time.time()
        self.xyz = "Unknown"
        self.distance = 10000

    def start(self):
        return

    def update(self, frame):
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        parameters = cv2.aruco.DetectorParameters_create()
        self.corners, self.ids, rejected_img_points =
cv2.aruco.detectMarkers(gray, cv2.aruco_dict, parameters=parameters,
        cameraMatrix=matrix_coefficients,
        distCoeff=distortion_coefficients)

    def draw(self, frame):
        # If markers are detected
        corners = self.corners
        ids = self.ids
        if len(corners) > 0:
            for i in range(0, len(ids)):
                # Estimate pose of each marker and return the values rvec and
tvec---(different from those of camera coefficients)
                rvec, tvec, markerPoints =
cv2.aruco.estimatePoseSingleMarkers(corners[i], 0.0225 * 2, matrix_coefficients,
                                distortion
n_coefficients)

                self.xyz =
f"X:{str(round(tvec[0][0][0],3))},Y:{str(round(tvec[0][0][1],3))},Z:{str(round(tv
ec[0][0][2],3))}"
                self.distance = round(tvec[0][0][2],3)

                cv2.aruco.drawDetectedMarkers(frame, corners)

            for (markerCorner, markerID) in zip(corners, ids):
                # extract the marker corners (which are always returned in
# top-left, top-right, bottom-right, and bottom-left order)
                corners = markerCorner.reshape((4, 2))
                (topLeft, topRight, bottomRight, bottomLeft) = corners
```



```
# convert each of the (x, y)-coordinate pairs to integers
topRight = (int(topRight[0]), int(topRight[1]))
bottomRight = (int(bottomRight[0]), int(bottomRight[1]))
bottomLeft = (int(bottomLeft[0]), int(bottomLeft[1]))
topLeft = (int(topLeft[0]), int(topLeft[1]))

# compute and draw the center (x, y)-coordinates of the ArUco
# marker
cX = int((topLeft[0] + bottomRight[0]) / 2.0)
cY = int((topLeft[1] + bottomRight[1]) / 2.0)
cv2.circle(frame, (cX, cY), 4, (0, 0, 255), -1)
# draw the ArUco marker ID on the image
cv2.putText(frame, str(markerID), (topLeft[0], topLeft[1] -
10), cv2.FONT_HERSHEY_SIMPLEX,
            0.5, (0, 255, 0), 2)

cv2.putText(frame, str(round(tvec[0][0][0], 3)) + " " +
str(round(tvec[0][0][1], 3)) +
" " + str(round(tvec[0][0][2], 3)), (bottomRight[0],
bottomRight[1] - 10), cv2.FONT_HERSHEY_SIMPLEX,
            0.5, (0, 255, 0), 2)

# Draw Axis
cv2.aruco.drawAxis(frame, matrix_coefficients,
distortion_coefficients, rvec, tvec, 0.01)

self.frame = frame
```

8.4 Appendix D YoloModel.py:

```
# -*- coding: utf-8 -*-
"""
Created on Thu Sep 29 09:00:00 2022

@author: Alex, Connor
Image Detection with yolov3 and opencv
"""

import cv2
import numpy as np
import time

whT = 320
```

```
#define classes
classNames = ['Closed_valve', 'Fire_extinguisher', 'Open_valve']
#confidence threshold
confThreshold = 0
#The lower this value the more aggressive it will be at removing boxes
nmsThreshold = 0.3

modelConfiguration = 'yolov3-custom-tiny.cfg'
modelWeights = 'yolov3-custom-tiny_final.weights'

class Yolo:
    def __init__(self):
        self.timeStart = time.time()
        self.net = cv2.dnn.readNetFromDarknet(modelConfiguration, modelWeights)
        self.net.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)
        self.net.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU)

    def start(self):
        return

    def update(self, img):
        #opencv needs the images as blobs, here we convert these to the blob
        #format
        blob = cv2.dnn.blobFromImage(img, 1/255, (whT, whT), [0, 0, 0], 1, crop=False)
        self.net.setInput(blob)
        layerNames = self.net.getLayerNames()
        #0 is not included within the layerNames so this is taken into account
        with
            #the -1
            outputNames = [layerNames[i-1] for i in
self.net.getUnconnectedOutLayers()]
            outputs = self.net.forward(outputNames)
            self.findObjects(outputs, img)
            self.img = img

    #function to find the object on the screen
    def findObjects(self, outputs, img):
        hT, wT, cT = img.shape
        #bounding box
        bbox = []
        #class identifier
        classIds = []
        #confidence intervals
        confs = []

        for output in outputs:
```



```
for detection in output:
    scores = detection[5:]
    classId = np.argmax(scores)
    confidence = scores[classId]

    if classId == 1:
        confThreshold = 0.8
    else:
        confThreshold = 0.4

    if confidence > confThreshold:
        #print(str(classId) + ' ' + str(confidence))
        #w and h of the bounding box
        w,h = int(detection[2]*wT) , int(detection[3]*hT)
        #x and y are the centrepoint
        x,y = int((detection[0]*wT)-w/2), int((detection[1]*hT)-h/2)
        bbox.append([x,y,w,h])
        classIds.append(classId)
        confs.append(float(confidence))

#output will be the indices of the bounding box to keep
indices = cv2.dnn.NMSBoxes(bbox,confs,confThreshold,nmsThreshold)

self.target = ""
self.targetConfidence = 0

for i in indices:
    if(int(confs[i]*100) > self.targetConfidence):
        self.targetConfidence = int(confs[i]*100)
        self.target = classNames[classIds[i]].upper()
    box = bbox[i]
    x,y,w,h = box[0],box[1],box[2],box[3]
    cv2.rectangle(img,(x,y),(x+w,y+h),(255,0,255),2)
    cv2.putText(img,f'{classNames[classIds[i]].upper()}
{int(confs[i]*100)}%',
                (x,y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.6,(255,0,255),2)
```