 Queensland University of Technology	QUT Systems Engineering UAVPayload^{TAQ}-G2	Doc No: FD-UAVPayload ^{TAQ} -G2-IMG-01 Issue: 1 Page: 1 of 15 Date: 23 October 2020
--	--	---

UAVPayload^{TAQ}

Final Design – Image Processing

Prepared by

Date 23/10/2020

James Arnold & Oliver Campbell

Checked by

Date 23/10/2020

Alexander Inftene, Project Manager

Approved by

Date 23/10/2020


James Arnold & Oliver Campbell

Authorised for use by

Date 23/10/2020

Dr. Felipe Gonzalez, Project Coordinator

This document is Copyright 2020 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 UAVPayload^{TAQ}-G2	Doc No: FD-UAVPayload ^{TAQ} -G2-IMG-01 Issue: 1 Page: 2 of 15 Date: 23 October 2020
--	--	---

Revision Record

Document Issue/Revision Status	Description of Change	Date	Approved
1.0		23 October 2020	James Arnold & Oliver Campbell


 Queensland University of Technology	QUT Systems Engineering UAVPayload^{TAQ}-G2	Doc No: FD-UAVPayload ^{TAQ} -G2-IMG-01 Issue: 1 Page: 3 of 15 Date: 23 October 2020
--	--	---

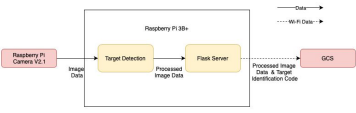


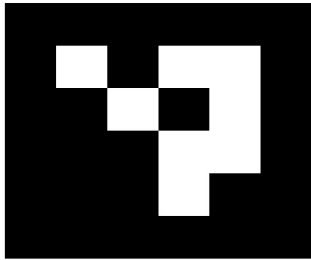
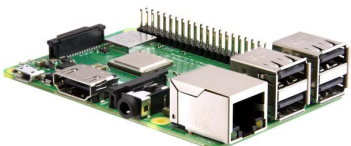
Table of Contents

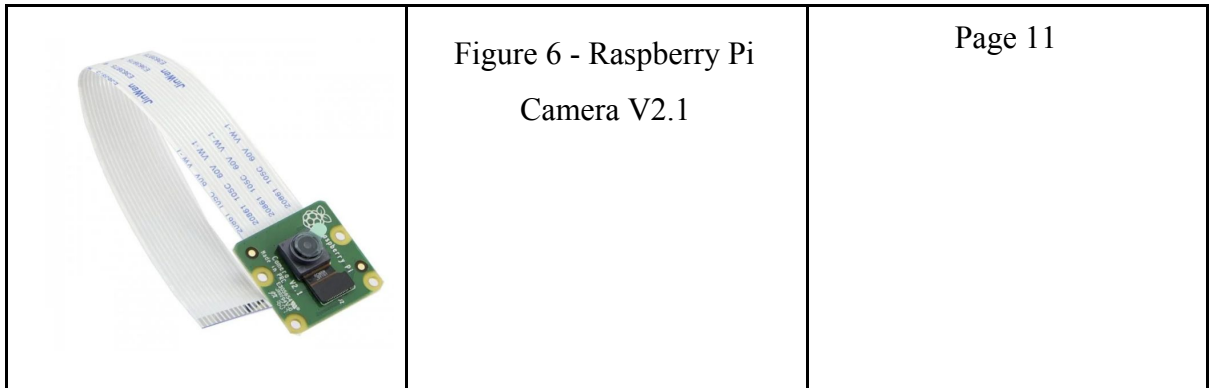
Paragraph	Page No.
Introduction	6
Scope	6
Background	6
Reference Documents	6
QUT Avionics Documents	7
Non-QUT Documents	7
Subsystem Introduction	8
Subsystem Architecture	8
Interfaces	9
Final Design Description	9
Hardware Design	10
Software	11
Framework	12
Machine Learning/Training of Data	12
ARUCO Marker Detection	13
Conclusion	14
Appendices	15
Aruco Detection Code	15

List of Figures

Figure


Page No.

	<p>Figure 1 - IMG Subsystem Architecture</p>	<p>Page 9</p>
	<p>Figure 2 - Hazardous marker for target identification</p>	<p>Page 10</p>
	<p>Figure 3 - Hazardous Marker for target identification</p>	<p>Page 10</p>
	<p>Figure 4 - Acruo marker for target identification</p>	<p>Page 10</p>
	<p>Figure 5 - Raspberry Pi 3 Model B+</p>	<p>Page 11</p>



Definitions

UAV	Unmanned Aerial Vehicle
UAVPayload	Unmanned Aerial Vehicle Payload
GCS	Ground Control Station
CO2	Carbone Dioxide
WEB	Web Visualization Subsystem
FTP	File Transfer Protocol
CNN	Convolution Neural Network
GUI	Graphical User Interface
HLO	High Level Objective

	<p align="center">QUT Systems Engineering</p> <p align="center">UAVPayload^{TAQ}-G2</p>	<p>Doc No: FD-UAVPayload^{TAQ}-G2-IMG-01</p> <p>Issue: 1</p> <p>Page: 6 of 15</p> <p>Date: 23 October 2020</p>
---	---	--

1 Introduction

The following document outlines the design and integration of the image processing subsystem. This subsystem commenced independently of the other subsystems, and commenced immediately after the initial trade study and background research. The web development partially depends on the completion of the image processing subsystem before it was completely implemented, thus the image processing subsystem was completed as a priority. There was a strong emphasis on integration with other subsystems and our computationally limited hardware. This was a constant consideration during the design process of the subsystem.


1.1 Scope

The design section below demonstrates where the image processing subsystem fits in with the project as a whole. The image processing subsystem works mostly independently to the signal processing subsystem, however was integrated closely with the web design subsystem in order to interface in real time with the end user.

The image processing subsystem was required to identify a target using a 640x480 camera interfacing with our Raspberry Pi 3. More details regarding hardware specifications are outlined below. This target identification was done on-board the RPi in a computationally limited environment, using a machine learning model trained prior to deployment. Once a target was identified, we interface with the WEB subsystem and transmit the image to the flask web server via wifi.

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 UAVPayload^{TAQ} 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 UAVPayload^{TAQ} is designed and developed using Systems Engineering to ensure QUT Airborne Sensing Systems requirements are met. Taken from RD/3.

 Queensland University of Technology	QUT Systems Engineering UAVPayload^{TAQ}-G2	Doc No: FD-UAVPayload ^{TAQ} -G2-IMG-01 Issue: 1 Page: 7 of 15 Date: 23 October 2020
--	--	---


2 Reference Documents

2.1 QUT Avionics Documents

RD/1	CN-UAVPayload ^{TAQ} -01	UAV Payload Customer Needs
RD/2	SR-UAVPayload ^{TAQ} -01	UAV Payload System Requirements
RD/3	PMP-UAVPayload ^{TAQ} -01	UAV Payload Project Management Plan
RD/4	PD-UAVPayload ^{TAQ} -G2-WEB-01	Preliminary Web Visualisation Subsystem Design
RD/5	PD-UAVPayload ^{TAQ} -G2-DAT-01	Preliminary Sensor Data Subsystem Design
RD/6	PD-UAVPayload ^{TAQ} -G2-IMG-01	Preliminary Image Processing Subsystem Design
RD/7	PD-UAVPayload ^{TAQ} -G2-ENC-01	Preliminary Payload Enclosure Subsystem Design

2.2 Non-QUT Documents

RD/8	https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/	Raspberry Pi 3 Model B+
RD/9	https://www.raspberrypi.org/documentation/hardware/camera/	Raspberry Pi Camera V2.1
RD/10	https://www.raspberrypi.org/documentation/hardware/raspberrypi/schematics/rpi_SCH_3bplus_1p0_reduced.pdf	Raspberry Pi Hardware Diagram

	<p align="center">QUT Systems Engineering</p> <p align="center">UAVPayload^{TAQ}-G2</p>	<p>Doc No: FD-UAVPayload^{TAQ}-G2-IMG-01</p> <p>Issue: 1</p> <p>Page: 8 of 15</p> <p>Date: 23 October 2020</p>
---	---	--

3 Subsystem Introduction

The Drone must satisfy all system requirements outlined in RD/2. The majority of these system requirements are relevant to the IMG subsystem, as the subsystem was integrated heavily with other subsystems. A considerable amount of on-board processing was required to capture and process images at a high enough framerate to reliably acquire targets. This may have bottlenecked our device, limiting the performance of our signal processing subsystem or may cause thermal throttling. This made hardware design decisions (e.g. air flow in physical enclosure, heat sink on chip, etc.) very relevant to this subsystem. This is reflected in the design section below.

Software development for efficient and reliable image processing is the most significant aspect of this subsystem. During deployment of the drone, a number of targets were laid flat on the floor on A4 sheets of paper. These targets are outlined in figures 2, 3 and 4 above. As outlined in REQ M-13 in RD/2, the drone was travelling at a speed of 1m/s, 2m above the target. As such, the machine learning model was trained under similar conditions, using the same Raspberry Pi camera to capture the images.

The Raspberry Pi was operating on the Linux OS, using Ubuntu. The images will be captured using a 640x480 camera, attached via a 15cm flexible ribbon cable. The images were captured at 2 fps whilst moving at 1m/s. Once an image was detected, the image was flagged and uploaded to the web server via File Transfer Protocol (FTP) using a LAN wifi connection. This was then transferred to the end-user and displayed on a front-end GUI.

The cascade classifiers were trained on a total of 1000 images. 850 of these were positive images (containing our target) and 150 of these were negative images (not containing our target). These images were captured at a distance of 0.5m, 1m and 2m at a variety of different angles. The cascade classifier was developed in python using the OpenCV library.

4 Subsystem Architecture

An overview of the subsystem's interfaces are outlined below.

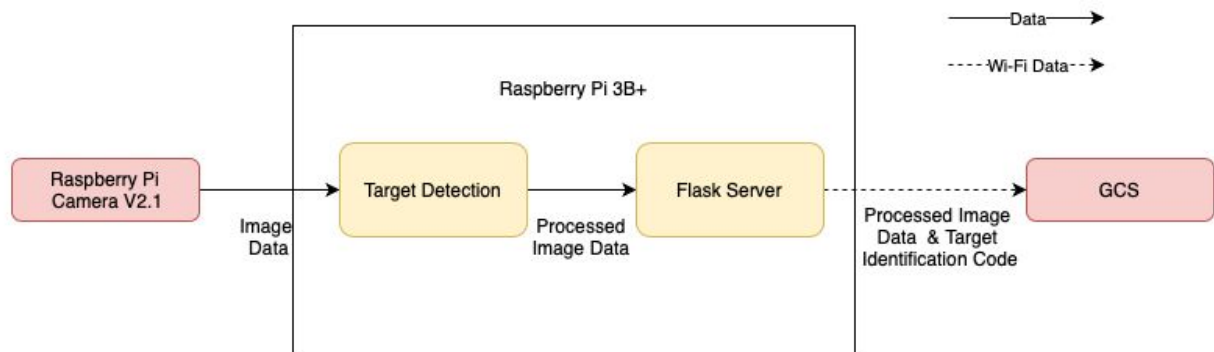


Figure 4 - IMG Subsystem Architecture


4.1 Interfaces

Interface 1: 2A/5V Power Supply to Micro-USB (type B)

Interface 2: Proprietary flat ribbon cable. Power and Data supplied via Push Pins.

Interface 3: USB 2.0 dongle SD card reader to read SD card running Linux and relevant programs

Interface 4: On-board wifi chip to send data over LAN network.

 Queensland University of Technology	QUT Systems Engineering UAVPayload^{TAQ}-G2	Doc No: FD-UAVPayload ^{TAQ} -G2-IMG-01 Issue: 1 Page: 10 of 15 Date: 23 October 2020
--	--	--

5 Final Design Description

5.1 Hardware Design

The following subsection will focus on the Hardware design aspect of the Image Processing Subsystem. Although the image processing subsystem primarily focuses on an effective and efficient implementation of an image processing algorithm, there are some hardware design considerations as well.

The Raspberry Pi 3 Model B+ was the micro-computer of choice, due its light weight, low cost and plenty of IO for all necessary peripherals. Raspberry Pi is by far the most supported micro-computer, allowing for plenty of choice regarding supported OS and relevant frameworks. The Pi 3 Model B+ was chosen as it was the latest release as of the commencement of the project and previous versions with worse performance would have been prohibitive for our use-case (would have had to lower frame rate, etc.). RPi 3 Model B+ can be found in figure 5. The hardware specifications are outlined below.

The Raspberry Pi Camera V2.1 was chosen primarily due to its low cost and light weight. It uses a proprietary interface to connect to the board. The connector is a flat ribbon with push pins on both ends. Other cameras such as the logitech C525 HD with a USB 2.0 interface may have been a sturdier option, as the ribbon connector on the Raspberry Pi camera is quite flimsy. The weight and price of alternative cameras make them prohibitive for our customer needs, especially considering the project only requires one successful demonstration. The RPi Camera V2.1 can be found in figure 6. The hardware specifications are outlined below.

Raspberry Pi 3 Model B Hardware Specs:

SOC: 64-bit Cortex-A5, 1.4GHz

RAM: 1GB DDR2

USB Ports: 4x USB 2.0 ports

Video/Audio Ports: 1 full size HDMI, MIPI DSI display port, MIPI CSI camera port

Ethernet Port: Gigabit Ethernet over USB2.0

Micro-SD slot: 1x Micro SD slot

Size: 85x56mm

Weight: 50g

Raspberry Pi Camera V2.1 Hardware Specs:

Still Resolutions: 8 Megapixels

Video: 1080p/30fps, 720p/60fps, 480p/90fps

Lens size: 35mm

Overall size: 25x24mm

Weight: 3g

5.2 Software


The software design of our subsystem revolves around an infinitely iterating loop which attempts to detect our target on the latest image capture using our Aruco detection classifier and our two cascade classifiers. Each target is being searched for independently from each other. Once a target is identified, we return an ID depending on which target was identified. This ID is then sent to the web server to indicate which target is being captured. If a target is identified, the script will not attempt to capture any other targets. Our main loop can be found in the figure below.

```

74 while (1):
75     print('beginning iteration')
76     target_id = 0 #This gets set to 1, 2 or 3 depending on detection. Remains 0 if no detection.
77     #Delay 0.5s between images
78     sleep(0.5)
79     #Capture & save image
80     camera.capture('/home/pi/photos/input/image.jpg')
81     #Record img path
82     imgpath = '/home/pi/photos/input/image.jpg'
83     # read in image
84     image = cv2.imread(imgpath)
85     #grayscale
86     grayscale = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
87
88     #Call Aruco, return 3 if detected
89     if (target_id == 0):
90         target_id = detectMarkers(image, dictionary, param, grayscale)
91
92     #Call mdg9 classifier, return 2 if detected
93     if (target_id == 0):
94         target_id = mdg9_classifier(image, grayscale)
95
96     #Call toxic classifier, return 1 if detected
97     if (target_id == 0):
98         target_id = toxic_classifier(image, grayscale)
99
100    #If target ID is still 0, save image anyway with no boxes and exit loop.
101    if(target_id == 0):
102        writeName = '/home/pi/photos/output/outputimage.jpg'
103        cv2.imwrite(writeName, image)
104        print('no classifiers detected')
105

```

Figure 5 - Iterating loop to identify target

 Queensland University of Technology	QUT Systems Engineering UAVPayload^{TAQ}-G2	Doc No: FD-UAVPayload ^{TAQ} -G2-IMG-01 Issue: 1 Page: 12 of 15 Date: 23 October 2020
--	--	--


By structuring our code like this, we are unable to identify multiple targets in the same image capture. As this was not a design requirement, this was an intentional design decision in order to save on performance. As soon as we have identified a target, we can immediately start processing the next image instead of attempting to detect additional targets which generally are not present.

5.3 Framework

OpenCV was chosen for its vast image handling properties. An open-source, BSD licensed product, OpenCV is often used with success in a variety of commercial applications, and has a wide range of online support available, making it an ideal software for use for this project. The ARUCO marker library will also be used for the detection of the ARUCO markers targets. Python also has a variety of open source machine learning libraries available to use, including TensorFlow, Scikit, and Pytorch. All of the libraries provide tools for simple machine learning tasks, such as regression and clustering, with TensorFlow and Pytorch providing the capability of constructing machine learning models and neural networks. Tensorflow was originally chosen as the machine learning library for use in the UAVPayload^{TAQ} tasks, due to the existence of Keras. Keras is a high-level, user-friendly API that is written in Python. The Image Processing team has limited experience with machine learning, and so the added functionality and modular approach of Keras make it well suited to the project. Due to unforeseen difficulties, the development of a neural network became implausible within the time span, and it was decided to instead use a Cascade Classifier. Cascade Classifiers have a documented use in image recognition tasks, and are also easy to train, with more detailed classifiers possible and only dependent on the computational power available.

5.4 Machine Learning/Training of Data

To identify the targets, a Cascade classifier was developed using OpenCV. The usage of a Cascade Classifier for image recognition (particularly face recognition) is an established principle, commonly used in mobile devices and cameras. They are commonly used in low-power environments as they are highly computationally efficient compared to alternative machine learning algorithms. To develop the model, 1000 100x100 grayscale images were used from a large number of image samples. The desired targets were overlaid on 900 of these targets, and used as our positive image dataset. The background images were used on 150 of these images and


 Queensland University of Technology	QUT Systems Engineering UAVPayload^{TAQ}-G2	Doc No: FD-UAVPayload ^{TAQ} -G2-IMG-01 Issue: 1 Page: 13 of 15 Date: 23 October 2020
--	--	--

these were designated as our negative image dataset. The command `opencv_traincascade` was then used to train the classifiers. One classifier was developed for each of the non-ARUCO markers.

When training the classifiers, there were a number of parameters that were adjusted to improve its effectiveness. These included the number of positive and negative images, the number of stages in the classifier, the minimal hit rate, and the maximum false alarm rate. The number of positive and negative images was very important in the training process, as with an increased number of stages, the amount of data needed increases. With more stages, the cascade classifier has more chances to eliminate negative images, which reduces computation power as well as improving accuracy. Minimal hit rate and the maximum false alarm rate are only two of a variety of parameters that can be changed in `opencv`. To reduce complexity, these two were empirically chosen to be the parameters of interest. Minimal hit rate refers to the minimally desired rate of success for each stage of the classifier. The maximum false alarm rate refers to the maximum number of false positives allowed per stage. Changing these two parameters can improve the accuracy of the classifier by making it better at detecting a wider range of features, but by increasing them too much can lead to a large number of false positives. A testing procedure was performed on the classifier after it was trained, and adjustments to the parameters were made, until classifiers with satisfactory performance were achieved.

5.5 ARUCO Marker Detection

The ARUCO markers are detected using functions from the ARUCO library that are located in the `opencv python` library. The input image is turned into greyscale, and the appropriate ARUCO dictionary is selected. For the purposes of this project, the library was the `5x5_250` dictionary. The function `aruco.detectMarkers` is then applied to the greyscale image, which provides a list of positions, and ids. If there are any markers detected in the image, their numerical id is contained within the list 'ids', otherwise the list is empty. If the list is not empty, an id code signifying the presence of an ARUCO marker is sent to the ground control station. As identification of the marker's numerical id was not a requirement, it is not performed. An image of the code is visible in appendix 7.1.

 Queensland University of Technology	QUT Systems Engineering UAVPayload^{TAQ}-G2	Doc No: FD-UAVPayload ^{TAQ} -G2-IMG-01 Issue: 1 Page: 14 of 15 Date: 23 October 2020
--	--	--

6 Conclusion

This subsystem design document outlines all the requirements pertaining to the IMG subsystem outlined in RD/2 have defined the overall design of the subsystem. This subsystem design serves as an overview of the overall design for the subsystem.

7 Appendices

7.1 Aruco Detection Code

```
def detectMarkers(im_name):
    imported_img = cv2.imread(im_name)
    img = cv2.cvtColor(imported_img, cv2.COLOR_RGB2BGR)

    # make sure to convert form RGB to BGR (stupid opencv)

    plt.figure()
    plt.imshow(img)
    plt.show()

    """

    grayscale = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    dictionary = aruco.Dictionary_get(aruco.DICT_5X5_250)
    param = aruco.DetectorParameters_create()
    corners, ids, rejectedImgPoints = aruco.detectMarkers(grayscale, dictionary, parameters=param)
    img_markers = aruco.drawDetectedMarkers(img.copy(), corners, ids)
    """

    plt.figure()
    plt.imshow(img_markers)
    if ids is None:
        print("There are no aruco markers in this image")
    else:
        for i in range(len(ids)):
            c = corners[i][0]
            plt.plot([c[:, 0].mean()], [c[:, 1].mean()], "o", label = "id={0}".format(ids[i]))
    plt.legend()
    plt.savefig("markers_with_legend.pdf")
    plt.show()

    return ids, corners
```