

 <b>Queensland University of Technology</b>	EGH450 Group 1	Doc No: 450G1-IMP-FD-04 Issue: 4.0 Page: 1 of 36 Date: October 2019
--	----------------	--

# Final Design

## Image Processing (IMP)

Prepared by \_\_\_\_\_ Date \_\_\_\_\_  
Harry Akeroyd, Image Processing Lead

Checked by \_\_\_\_\_ Date \_\_\_\_\_  
Hope Sneddon, Project Manager

Approved by \_\_\_\_\_ Date \_\_\_\_\_  
Hope Sneddon, Project Manager

Authorised for use by \_\_\_\_\_ Date \_\_\_\_\_  
Felipe Gonzalez, Project Coordinator

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

This document is Copyright 2019 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



## Revision Record

<b>Document Issue/Revision Status</b>	<b>Description of Change</b>	<b>Date</b>	<b>Approved</b>
1.0	Initial Issue	06/04/2019	Hope Sneddon
2.0	Revising Documentation 1.0 for Found Errors	03/05/2019	Hope Sneddon
3.0	Semester 2 Revisions (changes to classifier and code)	25/07/2019	Hope Sneddon
4.0	Semester 2 Final Revision	07/10/2019	Hope Sneddon



## Table of Contents

Paragraph	Page No.
1 Introduction .....	6
1.1 Scope .....	6
1.2 Background.....	6
2 Reference Documents .....	7
2.1 QUT Avionics Documents.....	7
2.2 Non-QUT Documents.....	7
3 Subsystem Introduction .....	8
3.1 Specific System Requirements .....	8
4 Subsystem Architecture .....	9
4.1 Interfaces .....	11
4.1.1 Raspberry Pi to PiCam.....	11
4.1.2 Raspberry Pi to Modem .....	11
4.1.3 Modem to GCS .....	12
5 Design.....	13
5.1 Hardware Design .....	13
5.1.1 Pi Camera V2.....	13
5.1.2 Raspberry Pi 2 Model B.....	15
5.1.3 Laptop .....	16
5.2 Software Design .....	17
5.2.1 Raspberry Pi Model B.....	17
5.2.2 Image Processing Software .....	17
5.2.3 Cascade Classifier.....	17
5.2.4 Training the HAAR-Based Classifier. ....	18
5.2.5 Example Image for training classifier. ....	22
5.2.6 Pseudocode for UAV in operation .....	22
5.2.7 Software Flow Diagram .....	22
5.3.6.1 Pseudocode .....	<b>Error! Bookmark not defined.</b>
6 Analysis .....	24
7 Conclusion.....	31

 <b>Queensland University of Technology</b>	EGH450 Group 1	Doc No: 450G1-IMP-FD-04 Issue: 4.0 Page: 4 of 36 Date: October 2019
--	----------------	--

## List of Figures

<b>Figure</b>	<b>Page No.</b>
Figure 1: System Architecture Overview (RD/5).....	9
Figure 2: Pi Camera V2 (RD/7) .....	13
Figure 3: Raspberry Pi Microcomputer (RD/6) .....	15
Figure 4: Laptop (RD/10).....	16
Figure 5: Software Flow Diagram.....	23



## Definitions

UAV	Unmanned Aerial Vehicle
UAS	Unmanned Aircraft System
OS	Operating System
GCS	Ground Control Station
USB	Universal Serial Bus
HLO	High Level Objectives
CSI	Camera Serial Interface
FPS	Frames Per Second
CMOS	Complementary Metal-Oxide-Semiconductor
RAM	Random Access Memory
GPU	Graphics Processing Unit
CPU	Central Processing Unit
XML	eXtensible Markup Language

 <b>Queensland University of Technology</b>	EGH450 Group 1	Doc No: 450G1-IMP-FD-04 Issue: 4.0 Page: 6 of 36 Date: October 2019
--	----------------	--

## 1 Introduction

The consequence of the preliminary design documentation (Issues 1.0 and 2.0), testing and integration works has led to the final image processing subsystem documentation, presented herein. This current documentation is the final iteration off the design plan and represents the systems that will be used during the autonomous flight of the UAV. All design choices have been made ensuring that the customers' requirements are met, and the image processing can fulfil its task.

### 1.1 Scope

This document specifies the image processing subsystem (IMP) and its integration in the overall system. The detailing of the IMP hardware and software designs is described within the following documentation. The design decisions and component selections (hardware and software) will be presented in detail, with specific attention paid to; integration within the system architecture, the development of the image classification software and its use aboard the Raspberry Pi and target localisation (camera relative to the targets). The design decisions are also justified in detail with supporting evidence of these respective choices.

### 1.2 Background

The Queensland University of Technology (QUT), along with the Australian Research Centre for Aerospace Automation (ARCAA) has commissioned Group 1 to develop a UAVUSR that is able to successfully and autonomously conduct a search around a simulated urban environment after a natural disaster event. During its mission, the UAV must identify and locate two human targets and deploy the correct emergency medication. Telemetry and status information from onboard UAV systems must be available to view on the ground station in real time. Additionally, it is a requirement that the UAV hovers over identified targets for 10 seconds before deploying medication. Throughout the design and delivery of this project, the UAVUSR will be designed in accordance with industry-standard Systems Engineering practices (a rigorous and disciplined engineering methodology for complex projects) to ensure all customer requirements are met.

## 2 Reference Documents

### 2.1 QUT Avionics Documents

RD/1	UAVUSR-SUP-Customer Needs	UAVUSR Project: Customer needs for 2019
RD/2	SR19G1-UAVTAQ-2019-Urban Search and Rescue-2019-02-18-Final	UAV for Urban Search and Rescue – UAVUSR: 2019
RD/3	349G1-PM-PMP	Project Management Plan Group 1 2019.
RD/4	349G1-PM-SysReq	The System Requirements for SRUAV Group 1.
RD/5	349G1-PM-ICD	Group 1 Interface Control Document
RD/6	349G1-PWR-TR08	Image Processing Subsystem Component Testing Document
RD/7	349G1-PM-ICD	Group 1 Interface Control Document

### 2.2 Non-QUT Documents

RD/6	Raspberry Pi 2 Model B	<a href="https://www.inet.se/files/pdf/1974044_0.pdf">https://www.inet.se/files/pdf/1974044_0.pdf</a>
RD/7	Raspberry Pi Camera Module V2	<a href="http://au.element14.com/raspberry-pi/rpi-8mpcamera-board/raspberry-pi-camera-boardv2/dp/2510728">http://au.element14.com/raspberry-pi/rpi-8mpcamera-board/raspberry-pi-camera-boardv2/dp/2510728</a>
RD/9	Wall Charger and Micro USB Cable	<a href="http://mobileshop.amaysim.com.au/aos-chargerusb-cable-2a-microusb.html">http://mobileshop.amaysim.com.au/aos-chargerusb-cable-2a-microusb.html</a>
RD/10	Laptop (6 <sup>th</sup> gen Lenovo X1 Carbon)	<a href="https://www.2compute.net/image-library/LNVO/pdf/20KG0026MB.pdf">https://www.2compute.net/image-library/LNVO/pdf/20KG0026MB.pdf</a>
RD/11	Haar Feature-based Cascade Classifier for Object Detection	<a href="http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html#cascadeclassifier-cascadeclassifier">http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html#cascadeclassifier-cascadeclassifier</a>



### 3 Subsystem Introduction

The High-Level Objectives (HLOs) as stated in (RD/1) and the Systems requirements (RD/3). Completing these objectives, the aircraft will be capable of delivering specific packages to specified locations by complete autonomous action. To achieve completely autonomous delivery the UAV must be capable of recognizing markers on the ground using a downwards facing camera onboard the Unmanned Aircraft System (UAS). GPS coordinates will be derived from this subsystem, which will later be utilised to position the UAV before payload deployment is actioned. Therefore, the image processing subsystem is a known critical component of the UAS to ensure complete autonomous operation is achieved. The Raspberry Pi micro-computer with the additional Pi camera V2 attachment will be used for the project.

The Image Processing subsystem initiates with the Pi camera taking images of directly below the UAV. The images are then transferred to the Raspberry Pi utilising a lightweight Linux Operating System (OS), using a Camera Serial Interface (CSI). The images are then modified to remove the slight fisheye affect from the Pi camera, before image detection is executed. Image detection occurs on-board the UAV and using open source OpenCV packages the camera relative to the target location is calculated. These translation coordinates are associated with a specific time stamp and are published to TF2. From which navigation is able to obtain the real-world location of the targets and set a new flight plan.

#### 3.1 Specific System Requirements

Table 1 outlines the specific subsystem requirements for the IMP subsystem. The system requirements can be found in (RD/4).

[REQ-04-U]	The UAVUSR must transmit live telemetry and imagery to the ground control station for display and logging.
[REQ-04-01-U]	All sensor information must be logged and displayed graphically in an easily understandable way to the customer in real time.
[REQ-04-01-01-U]	Live single-capture imagery as observed by the UAVUSR shall be displayed to the customer on the control station displays.



## 4 Subsystem Architecture

Overview of the Image Processing subsystem, the control diagram and subsystem architecture can be seen in Figure 1.

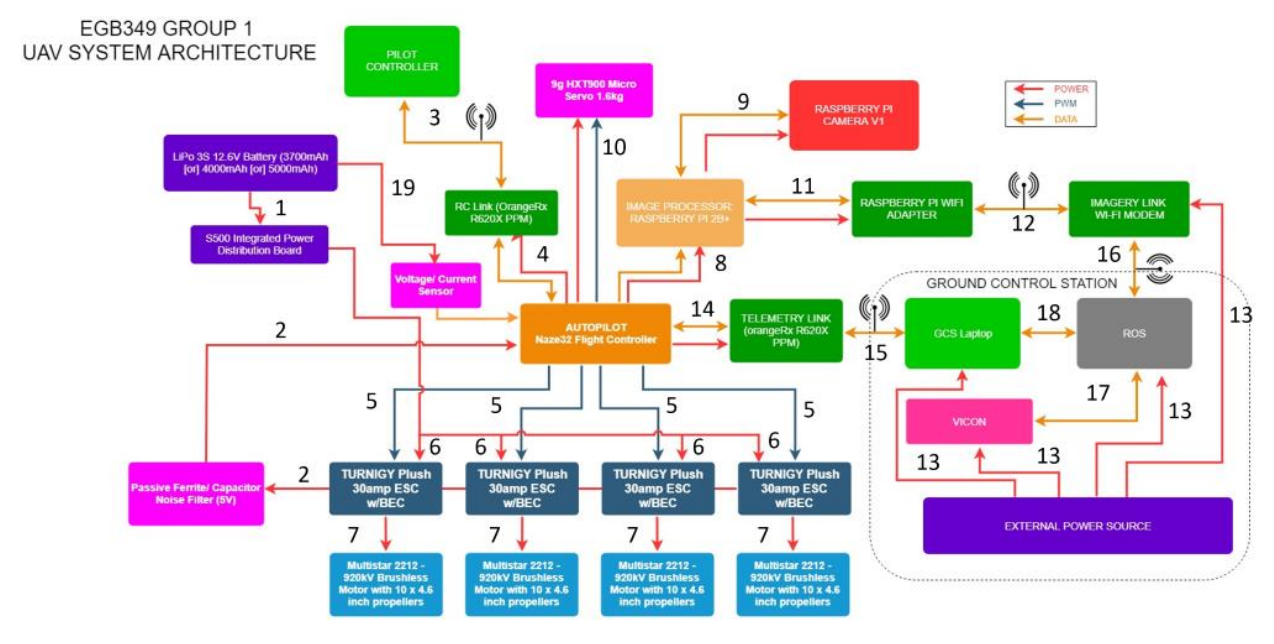


Figure 1: System Architecture Overview (RD/5)

The figure above shows how the IP subsystem operates within the complete system (see Figure 1). The subsystem is located at the GCS Laptop and is powered via a GCS Laptop charger (see Section 5.1.2). The subsystem will receive information from 3 inputs: autopilot, VICON and the Raspberry Pi. Details on these input sources is specified in Section 4.1, which outline the subsystem interfaces and is also summarised in Table 1. All data received will be outputted to the user through QUTAS Flight Stack applications RVIZ and RQT, and the Linux terminal.

ROS is used for software communication between input devices and is installed onto the GCS Laptop.



Interface Number	Power / Data being received	Interface
SSI 8	Power from the autopilot	A USB to MicroUSB connection to the Raspberry Pi, providing the power.
SSI 9	Power to camera, imagery from camera, commands to camera	CSI interface to provide a 2-way data connection and a 1-way power connection (board towards camera) between the Raspberry Pi and the Pi cam.
SSI 11	Power & data to wireless adapter	USB 2.0 interface to connect the USB WIFI Modem to the Raspberry Pi. USB WIFI Modem powered via the Raspberry Pi.
SSI 12	Data to GCS from Raspberry Pi	WIFI 802.11n connection to provide data transfer between the Raspberry Pi and the WIFI modem.
SSI 16	Data to GCS from modem	WIFI 802.11n connection to provide data transfer between the WIFI modem and the GCS.

Table 1 Summary of IMP Relevant Interfaces

## 4.1 Interfaces

This section describes the IMP subsystem interfaces and the types of data that will be transferred. For further information about the interface used to connected to other subsystems see the Group 1 Interface Control Document (RD/7).

### 4.1.1 Raspberry Pi to PiCamera

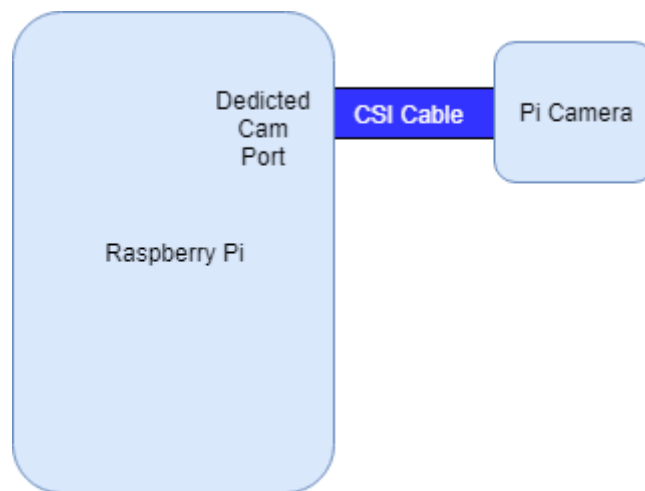


Figure 2: Interface from Raspberry Pi to Pi Camera. (RD/7)

Figure 2 shows the connection between the Raspberry Pi and the Pi Camera. The connection is a CSI cable, this allows data and power to be transmitted both ways. In this case power is being transmitted to the Pi cam and data being transmitted both ways.

### 4.1.2 Raspberry Pi to Modem

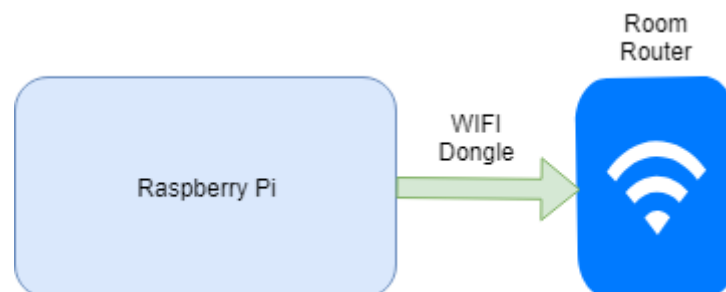


Figure 3: Interface from Raspberry Pi to Modem. (RD/7)

Figure 3 shows the connection between the Raspberry Pi and the modem. The modem is called “FlyFi-QUT-2.4” which is a 902.11n WIFI connection. The data transferred is the imagery data originally collected from the Pi Camera.

### 4.1.3 Modem to GCS



Figure 4: Interface from Modem to GCS. (RD/7)

Figure 3 shows the connection between the modem and the GCS. The connection is very similar to section 4.1.2, but with data coming from the modem to the GCS. The data transferred is the same as the previously mentioned section, imagery data originally collected from the Pi Camera.

 <b>Queensland University of Technology</b>	EGH450 Group 1	Doc No: 450G1-IMP-FD-04 Issue: 4.0 Page: 13 of 36 Date: October 2019
--	----------------	---

## 5 Design

For this assignment devices were selected from a predetermined list of products, previously the employer completed a trade study and the list of available components was given to the group. The devices selected for this project are the Raspberry Pi 2B V1.1 as the micro-controller and the Pi camera V2 add-on for the Raspberry Pi, as the camera of the system.

### 5.1 Hardware Design

The Image Processing subsystem consists of hardware and software components. The following section outlines the hardware specifications of the devices selected and the overall hardware design of the subsystem.

The Raspberry Pi is powered by an onboard Universal Serial Bus (USB), the Pi camera is powered through the Raspberry Pi. The WiFi connection is a transmitter connected to the Raspberry Pi through a USB port, meaning the dongle is powered through the board as well. The Raspberry Pi receives data from the Pi camera via CSI, which will then be processed on board the UAS, using software written for the project, before camera relative coordinates are transmitted to the GCS.

#### 5.1.1 Pi Camera V2



Figure 5: Pi Camera V2 (RD/7)

The Pi Camera v2 is light-weight, high quality camera, capable of providing video with a resolution of 1080p at 30 Frames Per Second (FPS). Additionally, the camera interfaces with the Raspberry Pi well, as the camera is built specifically to work with the Raspberry Pi, evident with the one connector for power and data.

### 5.1.1.1 Pi Camera V2: Operating Parameters

The camera was calibrated and coded to the resolution of *480x360* with a frame rate of *12 FPS*, this was selected as it provided an effective view of the operational environment on the GCS display: adhering to [REQ-04-01-01-U].



Figure 6: Camera calibration example using a 9x8 checkerboard, achieving a calibration at 480x360 resolution.

The operational parameters of the camera were also used in dictating the minimum and maximum size parameters for the classifier. These were selected as 85x85 and 500x500, respectively and were chosen as the images were identifiable within these limits. Furthermore, in doing so the lag of the camera feed was significantly decreased. The original lag was ~12 seconds, implementing these steps allowed the lag to be reduced to ~3 seconds.

### 5.1.1.2 Hardware Specifications

Connection Type:	CSI
Lens/Sensor Type:	CMOS
Optical Resolution:	8 Megapixel
Horizontal Field of View:	62.2 degrees
Vertical Field of View:	48.8 degrees

### 5.1.1.3 Hardware Diagram

The designs for CSI connections are unavailable for pinout connections as CSI is considered proprietary of Broadcom.

### 5.1.2 Raspberry Pi 2 Model B



Figure 6: Raspberry Pi Microcomputer (RD/6)

The Raspberry Pi is a small form factor, lightweight micro-computer. The Raspberry Pi is used for many things that a standard size desktop or laptop would be too large.

#### 5.1.2.1 Hardware Specifications

<i>CPU:</i>	900MHz quad-core ARM Cortex-A7
<i>GPU:</i>	Broadcom VideoCore IV
<i>Memory:</i>	1 GB
<i>USB Ports:</i>	4
<i>Video Outputs:</i>	HDMI, Composite video (PAL & NTSC)
<i>Audio Outputs:</i>	3.5mm, HDMI
<i>On-board Storage:</i>	MicroSD Card
<i>Power Ratings:</i>	800 mA (4 W)
<i>Power Source:</i>	5 volts via MicroUSB, GPIO header
<i>Size:</i>	85.60 mm x 56.5 mm
<i>Weight:</i>	45 grams

It should be noted that a heat-sink was added to the raspberry pi IC to ensure effective heat regulation when the Pi is performing onboard image processing. See below:

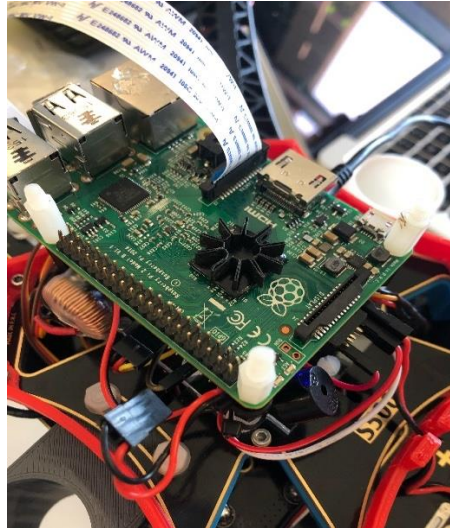


Figure 7: Raspberry Pi with star configuration heat sink on processing chip.

### 5.1.3 Laptop (Used outside of Da Vinci Facilities)



Figure 8: Laptop (RD/10)

This is a generic laptop capable of running the required GCS software. In this case the laptop is specifically a 6<sup>th</sup> generation Lenovo X1 Carbon.

#### 5.1.3.1 Hardware Specifications

<i>CPU:</i>	8th Gen Intel Core i7-8550U Processor (8M Cache, 1.8 GHz, 4.0 GHz max)
<i>GPU:</i>	Intel UHD Graphics 620
<i>Memory:</i>	16 GB



 <b>Queensland University of Technology</b>	EGH450 Group 1	Doc No: 450G1-IMP-FD-04 Issue: 4.0 Page: 17 of 36 Date: October 2019
--	----------------	---

## 5.2 Software Design

The Image Processing Subsystem requires appropriate software to provide a clear and detailed image of the ground below the UAV. Additionally, the Raspberry Pi must process the image onboard to determine if the object below the UAV is one of the intended targets.

### 5.2.1 Raspberry Pi Model B

The following subsections will be executed on the Raspberry Pi model B, as it is the operating environment for image processing aboard the UAV.

#### 5.2.1.1 Software Specifications

The Image Processing Subsystem will be based on a Linux based operating system, this allows for lightweight operating software allowing for more resources to be dedicated to image processing. UbuntuMate is the selected OS for this assignment as it is extremely lightweight and adaptable.

### 5.2.2 Image Processing Software

The video stream frames will be processed using the implementation of two HAAR-Based Cascade Classifiers. The first classifier will be specific to the first target, the orange square. The second classifier will identify the second target, the blue triangle. The consequence of these classifiers is the location of the identified targets relative to the UAV, which is sent to the GCS. The classifiers will be built using MATLAB R2019a and will interface with the Raspberry Pi through Python. Target localisation will be done using solvePnP and sent using TF functions. Python was selected as the language is relatively lightweight for what it is capable of and also allows for OpenCV functions to be utilised.

#### 5.2.3 Cascade Classifier

The frames will be searched for 'positive' targets which are trained into the classifier in accordance with the example shapes in (RD/1). Training a classifier in this way results in it being a HAAR based classifier. The need for two classifiers to be trained is a result of the HAAR type feature, it does not allow for multiple objects to be trained within the same classifier.

The classifier contains a number of stages, where each stage is recognised as an ensemble of weak learners. These weak learners (decision stumps) are classifiers and are used to eliminate negative and false positive instances. The stages in a classifier label the region of interest (ROI) defined by the current location of the sliding window and denotes it as either positive or negative. A positive

 <b>Queensland University of Technology</b>	EGH450 Group 1	Doc No: 450G1-IMP-FD-04 Issue: 4.0 Page: 18 of 36 Date: October 2019
--	----------------	---

indication tells the classifier that an object of interest was identified in the window, whereas a negative indication tells the classifier that there was no object of interest identified. Once the negative label is passed the classification of the region is concluded, sliding the window to the next location. If this label is identified as positive, the classifier passes the region to the next stage. A positive instance in the final classifier occurs when the detector is able to report an object detection in the final stage.

The training and testing process for classifier take a large amount of RAM and processing power, GPU processing is much more efficient for this stage, however the CPU can be used as well although considerably slower. This stage can take many hours, when the dataset is of an appropriate size. The training of the classifiers will be done on an external computer with more memory and processing power than the Raspberry Pi, this will produce an XML file which can be transferred to the Raspberry Pi and allow detection without the training being performed on the Pi. The above described classifier will be trained using MATLAB R2019a, and is described below (*5.2.4: Training the HAAR-Based Classifier*)

### 5.2.4 Training the HAAR-Based Classifier

The training process of each classifier is relatively simple but will require time to perform. The following is the process used to train the classifiers.

1. Gather copies of the markers to detect, images must reflect the flight area and targets to increase accuracy.
2. Gather images of the markers in different light conditions, different angles, different heights and different locations in the flight area. This would allow the classifier to detect the marker in a number of different scenarios. Approximately 400 images of each marker would be sufficient to train the classifiers.
3. Two scripts will be opened on MATLAB R2019a, however the steps are exactly the same and the example will relate to training the first classifier (target one).
4. Load positive samples from a *.mat* file stored in the working directory.
5. Selecting the bounding box from the ROI's returned from Image Labeller App. Image Labeller App will contain all the images with the ROI's identified.
6. Add the positive image folder to the current working directory.
7. Specify the folder containing the negative images.
8. Create an *imageDatastore* object which will contain the negative images (specified in the above step).
9. Train the classifiers using MATLAB's built in function, *trainCascadeObjectDetector*. Set

 <b>Queensland University of Technology</b>	EGH450 Group 1	Doc No: 450G1-IMP-FD-04 Issue: 4.0 Page: 19 of 36 Date: October 2019
--	----------------	---

the following parameters:

- a. 'FalseAlarmRate', 0.1
- b. 'NumCascadeStages', 20
- c. 'TruePositiveRate', 0.98

10. Once completed, the classifier for the first target will be stored in *classifier\_S.xml* file, the classifier for target two will be stored in *classifier\_T.xml* and can be transferred and used on Raspberry Pi.

NB: The script in Appendix 1 demonstrates the implementation of the above steps.

### 5.2.5 Target Localisation: Camera Relative to Target

Upon identification of a target by the above trained classifier the next stage in image processing is to provide the location (translational x and y coordinates) to navigation, such that the location of the target can be found in the real world. This was done using a perspective-n-point approach. More specifically, OpenCV's solvePnP function. This process of localisation will be described in its sequential order with the appropriate code extracted from Appendix 2.

#### *Step 1: Establishing the publishers*

As can be seen in figure 6, the first step of achieving target localisation was to introduce two transform publishers: functions given by the OpenCV Transform Library. As can be seen, the transforms were published under the *emulated\_uav* simulation, for all UAV-flight scenarios this was changed too 'Group1UAV'. This allowed interaction with Vicon: necessary to achieve real-world localisation. The Time assigned was indicative of when the target was found, further explanation can be found in step

```
self.broadcaster_S = tf2_ros.TransformBroadcaster()
self.pub_found_S = rospy.Publisher('/emulated_uav/Square', Time, queue_size=10)

self.broadcaster_T = tf2_ros.TransformBroadcaster()
self.pub_found_T = rospy.Publisher('/emulated_uav/Triangle', Time, queue_size=10)
```

Figure 9: Declaring the publishers for each patient under the names, Square and Triangle.

*Step 2: Declaring the first parameter of the solvePnP function: model\_objects*

The model objects shown in figure 10 represent an vector array of 3D object points in the real-world coordinate space.

```

squareLength = self.param_target_diam
self.model_object = np.array([[0.0,0.0,0.0),
                              (-squareLength/2, squareLength/2, 0.0),
                              (squareLength/2, squareLength/2, 0.0),
                              (squareLength/2, -squareLength/2, 0.0),
                              (-squareLength/2, -squareLength/2, 0.0)])

```

Figure 10: Declaration of model\_object points using the passed parameter squareLength (where squareLength is equal to 0.2m).

*Step 3: Declaring the third parameter of the solvePnP function: camera\_matrix*

The camera characteristics were given upon completion of the camera calibration and represent the Pi camera calibrated at a resolution of 480x360. The correct camera calibration and assignment was significantly important as the intrinsic parameters of the camera contributed to achieve a correct localisation: optical centre and focal length, etc were some of these parameters.

```

# Collect in the camera characteristics
def callback_info(self, msg_in):
    self.dist_coeffs = np.array([[msg_in.D[0], msg_in.D[1], msg_in.D[2], msg_in.D[3], msg_in.D[4]]], dtype="double")

    self.camera_matrix = np.array([
        (msg_in.P[0], msg_in.P[1], msg_in.P[2]),
        (msg_in.P[4], msg_in.P[5], msg_in.P[6]),
        (msg_in.P[8], msg_in.P[9], msg_in.P[10])],
        dtype="double")

    if not self.got_camera_info:
        rospy.loginfo("Got camera info")
        self.got_camera_info = True

```

Figure 11: Assignment of camera characteristics.

#### *Step 4: Implementing solvePnP function and creating the transforms*

The final step in achieving target localisation was the implementation of steps 1, 2 and 3 into the solvePnP function with the addition of the model\_image\_S. The code is initiated by the if statement, that is if a square is found. Following this is the assignment of the four ROI's coordinates (x,y,w,h) which was used to assign the model\_image\_S. This was representative of the assumed model image points as identified by the cascade classifier. The solvePnP function is then executed, the outputs of such being; a Boolean value, assigned as 'success', a rotational vector and a translational vector. The concluding if statements is ran once the solvePnP returns a 'success'. From the if statement a transform is assigned under the parent frame name 'camera' and child frame name 'square': the transform tree, figure 13, visually explains this relationship. The transform is then published along with its assigned time-stamp, which is used by navigation to find the real-world location of the target.

```

if sign_S is not None:
    for (x,y,w,h) in sign_S:
        # Calculate the pictured the model for the pose solver
        # For this example, draw a square around where the circle should be
        # There are 5 points, one in the center, and one in each corner
        self.model_image_S = np.array([
            (x+(w/2), y+(h/2)),
            (x, y),
            (x+w, y),
            (x, y+h),
            (x+w, y+h)], dtype=np.float32)

# Do the SolvePnP method
(success, rvec_S, tvec_S) = cv2.solvePnP(self.model_object, self.model_image_S, self.camera_matrix, self.dist_coeffs)

# If a result was found, send to TF2
if success:
    broadcaster_S = tf2_ros.TransformBroadcaster()
    pub_found_S = rospy.Publisher('/emulated_uav/Square', Time, queue_size=10)

    time_found_S = rospy.Time.now()
    S = TransformStamped()
    S.header.stamp = time_found_S
    S.header.frame_id = "camera"
    S.child_frame_id = "Square"
    S.transform.translation.x = tvec_S[0]
    S.transform.translation.y = tvec_S[1]
    S.transform.translation.z = tvec_S[2]

    S.transform.rotation.x = 0.0
    S.transform.rotation.y = 0.0
    S.transform.rotation.z = 0.0
    S.transform.rotation.w = 1.0

    self.broadcaster_S.sendTransform(S)
    self.pub_found_S.publish(time_found_S)

```

Figure 9: Script to achieve target localisation upon positive identification of a target. Replicated for the triangle, where 'S || Square' are replaced with 'T || Triangle'.

### 5.2.6 Subject Images for Training the Classifier



Figure 10: Target One – Test Image that will be used to Train the Classifier



Figure 1: Target Two - Test Image that will be used to Train the Classifier

### 5.2.7 Pseudocode for UAV in operation

The following pseudocode demonstrates a high-level overview of the logic performed while in operation.

While the UAV is performing its flight path:

- Receive the input from the camera to the Raspberry Pi board;
- Use the pre-trained classifier to search the image for the markers;
- If a target is found;
  - Find the relative location of the image to the camera;
  - Send the relative location to the GCS for real world localisation and display.

### 5.2.8 Software Flow Diagram

The following software flow diagram depicts the general flow of data from the Pi camera to the Raspberry Pi, through the software to process the raw image files. Finally, the images get transmitted to the GCS where the images will be displayed.

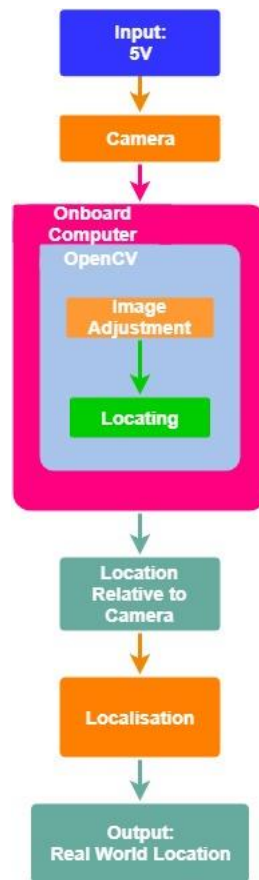


Figure 2: Software Flow Diagram

 <b>Queensland University of Technology</b>	EGH450 Group 1	Doc No: 450G1-IMP-FD-04 Issue: 4.0 Page: 24 of 36 Date: October 2019
--	----------------	---

## 6 Analysis

The following will address the performance testing and analysis of the components contributing the operation of the image processing subsystems. Hardware and software components will be addressed.

### 6.1 Hardware Performance Testing and Analysis


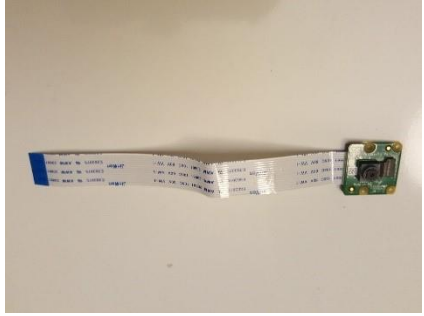


Component	Results (Pass/Fail)	Comments
GCS Laptop	Pass	
Raspberry Pi 2 Model B V1.1	Pass	
Raspberry Pi Camera V2.1	Pass	
Raspberry Pi USB to 3-pin adapter	Pass	
USB Wi-Fi Adapter	Pass	

Table 2: Hardware inspection results. (RD/6)



 <b>Queensland University of Technology</b>	EGH450 Group 1	Doc No: 450G1-IMP-FD-04 Issue: 4.0 Page: 25 of 36 Date: October 2019
--	----------------	---




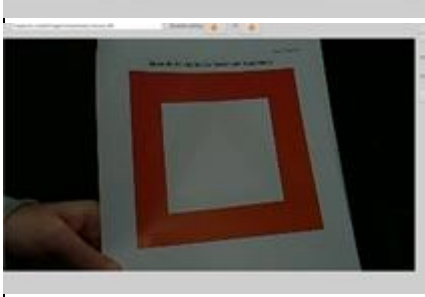
Component	Result (Pass/Fail)	Computer Display Confirmation
Raspberry Pi 2 Model B	Pass	
Pi Camera V2	Pass	
	Pass	 





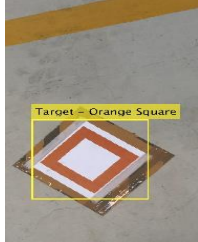
Table 3: Hardware Performance Test Results (RD/6)

*Analysis:* The design of the image processing hardware specific subsystem was found to be sufficient for the assigned task. During testing it was found that the selected components in the designed configuration would complete the system requirements for the image processing sub system. All tests were passed in the hardware inspection and the performance testing phases.

## 6.1 Software Performance Testing and Analysis

The software performance presented in the following works demonstrates the effectiveness of the cascade classifier built in Section 5.2.4 after its implementation into the SoC computer and the localisation based off of the positive identification of a target, developed and presented in Section 5.2.5.

### 6.1.1 Cascade Classifier Analysis


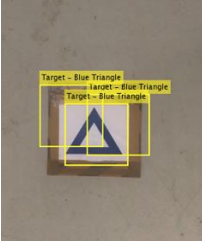




Test Number	Results (Pass/Fail)	Photo Evidence
1	Pass	
2	Fail – Identification of too many targets	
3	Pass	
4	Pass	
5	Pass	

6	Pass	
7	Pass	
8	Pass	
9	Pass	
10	Pass	

Table 4: Target Acquisition: Target One: Orange Square

*Analysis:* From Table 4 it can be seen that the first classifier had a pass rate of 90%. It can be seen that the ‘failed’ images did recognise and identify the image, however, they also identified a second ROI, this can be seen in test numbers two, five and ten. This was seen as an irregularity as during all final test flight of autonomous flight no false positive or false negatives were identified. Furthermore, this error in the identification was attributed to shadowing of the image due to the UAV being flown in ‘hand-held’ flight when the image was captured.



Test Number	Results (Pass/Fail)	Photo Evidence
1	Pass	
2	Fail – Identification of too many objects	
3	Pass	
4	Pass	
5	Pass	
6	Pass	

7	Pass	
8	Pass	
9	Pass	
10	Pass	

Table 5: Target Acquisition: Target Two: Blue Square

*Analysis:* From Table 5 it can be seen that the second classifier had a pass rate of 90%. It can be seen that the ‘failed’ images did recognise and identify the image, however, they also identified a second ROI, this can be seen in test number two. This was seen as an anomaly as during testing of autonomous flight no false positive or false negatives were identified. Furthermore, this error in the identification was attributed to shadowing of the image due to the UAV being flown in ‘hand-held’ flight when the image was captured.

### 6.1.2 Target Localisation Analysis

As can be seen in figure 12, the steps executed in Section 5.2.5 produced an accurate localisation of the target based, which was initiated by the correct identification of a target. A further application of this was that payload detection was able to determine which target had been located and which medication needed to be deployed.

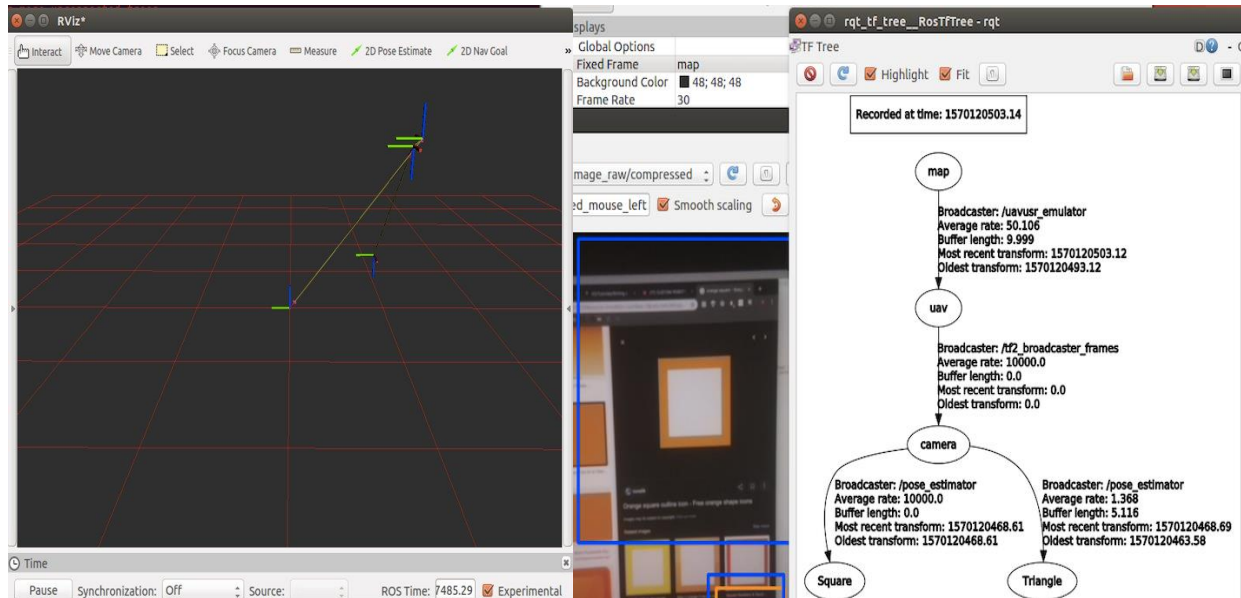


Figure 13: RVIZ display showing the target localised in map and the display of the UAV and camera components.

The transform can also be seen to include all parent-child relationships which confirms the display of RVIZ.

## 7 Risk Management

The following risks are specific to the image processing subsystem. The corresponding risk levels were selected according to figure 11.

Risk	Level	Precautions
False Positive detections	Low-Risk (4)	<ul style="list-style-type: none"> <li>The classifier will be extensively trained and tested before operations.</li> <li>Additionally, as the classifier will constantly be running as the UAV gets closer to the target, it will have opportunities to recognise the detection was not a marker.</li> </ul>
Hardware Failure	Low-Risk (4)	Caution while handling the components must be taken.

RISK MATRIX Likelihood x Consequence		CONSEQUENCE			
		Minor (1)	Moderate (2)	Major (3)	Extreme (4)
LIKELIHOOD	Likely (4) 40-100%	<b>Low-Risk (4)</b> No specific management action required	<b>Moderate-Risk (8)</b> Specific management/monitoring needed	<b>High-Risk (12)</b> Increased management actions needed	<b>High-Risk (16)</b> Increased management actions needed
	Possible (3) 10-35%	<b>Low-Risk (3)</b> No specific management action required	<b>Moderate-Risk (6)</b> Specific management/monitoring needed	<b>Moderate-Risk (9)</b> Specific management/monitoring needed	<b>High-Risk (12)</b> Increased management actions needed
	Unlikely (2) 2-10%	<b>No-Risk (2)</b> No management action required	<b>Low-Risk (4)</b> No specific management action required	<b>Moderate-Risk (6)</b> Specific management/monitoring needed	<b>Moderate-Risk (8)</b> Specific management/monitoring needed
	Remote (1) <2%	<b>No-Risk (1)</b> No management action required	<b>No-Risk (2)</b> No management action required	<b>Low-Risk (3)</b> No specific management action required	<b>Low-Risk (4)</b> No specific management action required

Figure 14: Risk matrix used to decide the severity of scenarios.

 <b>Queensland University of Technology</b>	EGH450 Group 1	Doc No: 450G1-IMP-FD-04 Issue: 4.0 Page: 32 of 36 Date: October 2019
--	----------------	---

## 8 Conclusion

The system requirements documented in RD/4 formed the guidelines for which the IMP subsystem was designed. This document represents the final iteration of the IMP subsystem design document. Once the initial system requirements were completed, the Image Processing Subsystem was designed with close regard to the pre-set budget. The Image Processing subsystem consists of three main steps, first the image data is collected from the onboard Pi camera. Next, the raw image data is processed on the Raspberry Pi using two pre-trained HAAR-Based Cascade Classifiers, and camera relative coordinates would be found. Finally, the image files and the camera relative coordinates would be transferred to the GCS where they would be either displayed or transformed into real world coordinates and sent back to the UAV. Furthermore, the injury type of the found target was able to be communicated such that the appropriate medication could be deployed by the payload lead engineer. These steps assisted in meeting the customers and system requirements:

- [REQ-01-03-F] was completed as the weight of the image processing subsystem was less than the assigned weight budget of the project.
- [REQ-03-F] required that the image detection system must be able to successfully locate and correctly identify both people using an on-board computer: this was achieved as the indicated accuracy of the
- [REQ-5-F] was able to be satisfied from the perspective of the UAV. From the identification of a target, the publisher was able to be used to alert the payload script as to which injured person had been found and done so using the child frame names.
- [REQ-03-01-F] was able to be satisfied. Referring to figure 12, the patients' locations was able to be found using the image detection software, which was operating on the on-board computer.
- [REQ-04-01-01-U] required that live imagery be displayed on the GCS, this was tested and completed by displaying live imagery of the markers that will be used in the real flight, on the GCS. The camera was moved around like that of a UAV during the testing, verifying the system works.

Future works for the image detection system may be advanced by introducing Transfer learning using TensorFlow Hub to perform the target detection, or if the software is able to be ran on a desktop computer then this would improve performance speeds and capabilities.



 <b>Queensland University of Technology</b>	EGH450 Group 1	Doc No: 450G1-IMP-FD-04 Issue: 4.0 Page: 33 of 36 Date: October 2019
--	----------------	---

## 9 Appendix

### Appendix 1: MATLAB Script – *imPro\_S.m*

```

%% IMAGE PROCESSING CODE:                                     15/08/19

%% CLEAN UP

clc, clear, close all

%% LOADING IN POSITIVE SAMPLES

load('pos_label.mat');

%% SELECTING THE BOUNDING BOXES FROM THE SUPPLIED SAMPLES

positiveInstances_S = pos_roi;

%% ADDING THE IMAGE FOLDER TO THE PATH

imDirPos_S = fullfile('/Users/h/Desktop/6--EGH450/G1_Image_Processor/training_cascade_opencv/pos_S');

addpath(imDirPos_S);

%% SPECIFYING THE NEGATIVE IMAGES FOLDER

negativeFolder_S=fullfile('/Users/h/Desktop/6--EGH450/G1_Image_Processor/training_cascade_opencv/neg_S');

%% STORING THE NEGATIVE IMAGES IN A OBJECT

negativeImages = imageDatastore(negativeFolder_S);

%% TRAINING THE CLASSIFIER USING HAAR FEATURES

trainCascadeObjectDetector('cascade_S.xml',positiveInstances_S, negativeFolder_S,...

    'FalseAlarmRate',0.1,...

    'TruePositiveRate',0.98,...

    'NumCascadeStages', 20,...

    'FeatureType', 'Haar');

```



## Appendix 2: Python Code – *image\_processor.py*

```
#!/usr/bin/env python

import sys
import rospy
import cv2
import numpy as np
import tf2_ros
from geometry_msgs.msg import TransformStamped
from std_msgs.msg import Time

from sensor_msgs.msg import Image
from sensor_msgs.msg import CompressedImage
from sensor_msgs.msg import CameraInfo
from cv_bridge import CvBridge, CvBridgeError

# Global V
#broadcaster_S = None
#pub_found_S = None

#broadcaster_T = None
#pub_found_T = None

class PoseEstimator():
    def __init__(self):
        self.time_finished_processing = rospy.Time(0)

        # Set up the CV Bridge
        self.bridge = CvBridge()

        # Square Classifier
        sign_cascade_file_S = str(rospy.get_param("~cascade_file_S"))
        self.sign_cascade_S = cv2.CascadeClassifier(sign_cascade_file_S)

        # Triangle Classifier
        sign_cascade_file_T = str(rospy.get_param("~cascade_file_T"))
        self.sign_cascade_T = cv2.CascadeClassifier(sign_cascade_file_T)

        # Load in parameters from ROS - for BLUE
        self.param_use_compressed = rospy.get_param("~use_compressed", False)
        self.param_target_diam = rospy.get_param("~target_diam", 1.0)

        # Set additional camera parameters
        self.got_camera_info = False
        self.camera_matrix = None
        self.dist_coeffs = None

        # Set up the publishers, subscribers, and tf2
        self.sub_info = rospy.Subscriber("~camera_info", CameraInfo, self.callback_info)

        if self.param_use_compressed:
            self.sub_img = rospy.Subscriber("~image_raw/compressed", CompressedImage, self.callback_img, queue_size=10)
            self.pub_overlay = rospy.Publisher("~overlay/image_raw/compressed", CompressedImage, queue_size=1)
        else:
            self.sub_img = rospy.Subscriber("~image_raw", Image, self.callback_img, queue_size=10)
            self.pub_overlay = rospy.Publisher("~overlay/image_raw", Image, queue_size=1)

        # Generate the model for the pose solver
        # For this example, draw a square around where the circle should be
        # There are 5 points, one in the center, and one in each corner
        # taken 0.0,0.0,0.0 out from the first part
        squareLength = self.param_target_diam
        self.model_object = np.array([(0.0,0.0,0.0),
                                      (-squareLength/2, squareLength/2, 0.0),
                                      (squareLength/2, squareLength/2, 0.0),
                                      (squareLength/2, -squareLength/2, 0.0),
                                      (-squareLength/2, -squareLength/2, 0.0)])

        self.broadcaster_S = tf2_ros.TransformBroadcaster()
        self.pub_found_S = rospy.Publisher('/emulated_uav/Square', Time, queue_size=10)

        self.broadcaster_T = tf2_ros.TransformBroadcaster()
        self.pub_found_T = rospy.Publisher('/emulated_uav/Triangle', Time, queue_size=10)

    def shutdown(self):
        # Unregister anything that needs it here
        self.sub_info.unregister()
        self.sub_img.unregister()
```



```
# Collect in the camera characteristics
def callback_info(self, msg_in):
    self.dist_coeffs = np.array([[msg_in.D[0], msg_in.D[1], msg_in.D[2], msg_in.D[3], msg_in.D[4]]], dtype="double")

    self.camera_matrix = np.array([
        (msg_in.P[0], msg_in.P[1], msg_in.P[2]),
        (msg_in.P[4], msg_in.P[5], msg_in.P[6]),
        (msg_in.P[8], msg_in.P[9], msg_in.P[10])],
        dtype="double")

    if not self.got_camera_info:
        rospy.loginfo("Got camera info")
        self.got_camera_info = True

def callback_img(self, msg_in):
    if msg_in.header.stamp > self.time_finished_processing:

        # Don't bother to process image if we don't have the camera calibration
        if self.got_camera_info:
            # Convert ROS image to CV image
            cv_image = None

            try:
                if self.param_use_compressed:
                    cv_image = self.bridge.compressed_imgmsg_to_cv2( msg_in, "bgr8" )
                else:
                    cv_image = self.bridge.imgmsg_to_cv2( msg_in, "bgr8" )
            except CvBridgeError as e:
                rospy.loginfo(e)
                return

            sign_S = self.sign_cascade_S.detectMultiScale(cv_image,1.01,1)
            sign_T = self.sign_cascade_T.detectMultiScale(cv_image, 1.01,1)

            # If square were detected
            if sign_S is not None:
                for (x,y,w,h) in sign_S:
                    # Calculate the pictured the model for the pose solver
                    # For this example, draw a square around where the circle should be
                    # There are 5 points, one in the center, and one in each corner
                    self.model_image_S = np.array([
                        (x+(w/2), y+(h/2)),
                        (x, y),
                        (x+w, y),
                        (x, y+h),
                        (x+w, y+h)], dtype=np.float32)

            # Do the SolvePnP method
            (success, rvec_S, tvec_S) = cv2.solvePnP(self.model_object, self.model_image_S, self.camera_matrix, self.dist_coeffs)

            # If a result was found, send to TF2
            if success:
                broadcaster_S = tf2_ros.TransformBroadcaster()
                pub_found_S = rospy.Publisher('/emulated_uav/Square', Time, queue_size=10)

                time_found_S = rospy.Time.now()
                S = TransformStamped()
                S.header.stamp = time_found_S
                S.header.frame_id = "camera"
                S.child_frame_id = "Square"
                S.transform.translation.x = tvec_S[0]
                S.transform.translation.y = tvec_S[1]
                S.transform.translation.z = tvec_S[2]

                S.transform.rotation.x = 0.0
                S.transform.rotation.y = 0.0
                S.transform.rotation.z = 0.0
                S.transform.rotation.w = 1.0

                self.broadcaster_S.sendTransform(S)
                self.pub_found_S.publish(time_found_S)

            # Draw the circle for the overlay
            cv2.rectangle(cv_image, (x,y), (x+w,y+h), (0, 140, 255), 2) # Only need this one for each one.
```



```
if sign_T is not None:
    for (x,y,w,h) in sign_T:
        self.model_image_T = np.array([(x+(w/2),y+(h/2)),(x,y),(x+w,y),(x,y+h),(x+w,y+h)], dtype=np.float32)
        (success_T, rvec_T, tvec_T) = cv2.solvePnP(self.model_object, self.model_image_T, self.camera_matrix, self.dist_coeffs)

#
pub_found_T = rospy.Publisher('/emulated_uav/Triangle', Time, queue_size=10)

time_found_T = rospy.Time.now()
T = TransformStamped()
T.header.stamp = time_found_T
T.header.frame_id = "camera"
T.child_frame_id = "Triangle"
T.transform.translation.x = tvec_T[0]
T.transform.translation.y = tvec_T[1]
T.transform.translation.z = tvec_T[2]

T.transform.rotation.x = 0.0
T.transform.rotation.y = 0.0
T.transform.rotation.z = 0.0
T.transform.rotation.w = 1.0

self.broadcaster_T.sendTransform(T)
self.pub_found_T.publish(time_found_T)

cv2.rectangle(cv_image, (x,y), (x+w,y+h), (255,0,0), 2)

#Convert CV image to ROS image and publish the mask / overlay
try:
    if self.param_use_compressed:
        self.pub_overlay.publish( self.bridge.cv2_to_compressed_imgmsg( cv_image, "png" ))
    else:
        self.pub_overlay.publish( self.bridge.cv2_to_imgmsg( cv_image, "bgr8" ))
except (CvBridgeError,TypeError) as e:
    rospy.loginfo(e)

self.time_finished_processing = rospy.Time.now()
```

NB: The above is screen shots taken from the raspberry pi and the indentions are all correctly done.