Real-time point of interest identification on a Multirotor Unmanned Aerial Vehicle

Final Year Project Report — Semester 2 2015

School of Mechanical and Chemical Engineering
Faculty of Engineering, Computing and Mathematics

THE UNIVERSITY OF
WESTERN AUSTRALIA

Supervisors:
Mr. Chris Croft
Prof. Dr. Thomas Bräunl

Author:
Jeremy Tan – 20933708
26 October 2015

Abstract

The decreasing cost and proliferation of unmanned aerial vehicles in the form of remote-controlled aeroplanes, helicopters and multirotor aircraft make for a particularly interesting platform for automation, especially in the field of aerial imaging. Combined with the technological advancements to date, there is a real possibility of complete automation in a low-cost, yet functional manner.

A common task in aerial imaging is the detection of points of interest, such as location markers, objects and buildings, for purposes including localisation, object tracking, and mapping. However, this is commonly performed off-line, and is not fully autonomous.

Working towards a fully autonomous platform, this report outlines the design of a real-time point of interest identification system controlling a hexacopter. Various image processing techniques are explored and compared for accuracy and speed. GPU processing is also explored to maximise utility of the embedded system.

Results indicate that with multi-threaded image processing, effective real-time point of interest detection is possible at resolutions up to 320x240 pixels at a processing rate of 30FPS, and up to 640x480 pixels at 10 FPS. GPU processing in combination with NEON SIMD has also realised a real-time video encoder, capable of encoding video at resolutions up to 800x600 at 30FPS with only a 3 ms or less footprint.

Furthermore, a redesigned, modern web interface to visualise and oversee the autonomous control of the hexacopter is presented. Using standard web libraries, the hexacopter is controllable both through mobile devices and standard laptop computers.

Keywords: *automation, point of interest, real-time, image processing, hexacopter, multirotor, OpenMAX, VideoCore, NEON, Bootstrap, Raspberry Pi, ArduPilot, ArduCopter*

Word count: 7930 words (12031 words including appendices)

Acknowledgements

I would like to thank both my supervisors, Chris Croft and Prof. Dr. Thomas Bräunl for their ongoing support and feedback throughout this project. I also extend my deepest gratitude to my fellow colleagues, including Richard Allen, Brett Downing and Manish Mohanty, whose combined experience, ideas and insight not only made this project successful, but moreover an enjoyable one at that. Finally, I am always thankful for the support and encouragement provided by both my friends and family over the past year.

Contents

List of Figures

List of Tables

1. Introduction

1.1 Motivation

Unmanned autonomous aerial vehicles (UAVs) have long been studied for their use in the fields of surveillance, search and rescue, data collection and payload transportation [1–3]. With great versatility, there is no fixed task that such vehicles perform. However, they have found increasing use in both the commercial sector [4–6] and across a number of research disciplines as cost-effective tools for aerial imagery. Indeed, research into the application of UAVs at the University of Western Australia for aerial imagery is not restricted to one faculty, with its use considered in the field of agricultural science to monitor the growth of pasture biomass [7].

The primary focus for this study is on the application of real-time image processing for autonomously identifying points of interest (POIs). While image processing may be performed offline, real-time image processing allows for reduced delay in identifying potential points of interest. This is critical to a number of applications, such as live object tracking. Potential applications include:

- Object detection and tracking (e.g. filming moving targets like cyclists, skiers and other sports players)
- Real-time identification of anomalies (e.g. surveillance of a known environment)
- Simultaneous localisation and mapping (SLAM) (e.g. mapping of an unknown environment)

This study extends a pre-existing Hexacopter platform. In addition to the author [8], this same platform will be used by 3 other students focusing on differing areas of automation; Richard Allen [9], Brett Downing [10] and Manish Mohanty [11]. As such, the POI identification system will be developed and integrated as part of a larger automation system.

Object identification has previously been accomplished through coloured blob (e.g. red object) tracking [12]. This simple but effective method identifies objects of a known colour in a known environment. However, it tends to work well only in environments that highly contrast in colour to the detected object, and it must also be trained to identify the specific colour of each object. Other methods of identifying objects will be explored, including glyph tracking (identifying patterns as location markers) and the feasibility of detecting more complex shapes such as humans.

With a suitably robust POI identification system, high performance object tracking may be achieved. The study of object tracking and localisation will be undertaken by Downing [10]. Similarly, POI identification is useful for mapping and 3D environment reconstruction, which will be explored by Mohanty [11]. On the other hand, path planning and collision avoidance research by Allen [9] will complement the POI identification system to good effect.

The intention is to develop a self-contained, highly cost-effective solution using readily available hardware. With this in mind, attention is paid not only to developing a POI identification system, but also to the supporting framework around it — without which, the POI system would be of little use. Inclusive to this is the development of video capture techniques for comprehensive data review, as well as the design of a web interface to control the autonomous capabilities. Considering the compute-limited nature of embedded systems, methods will also be explored to leverage the most out of the system, including *multi-threaded* computing and offloading suitable computation tasks to the GPU.

1.2 Project history

This project is a direct continuation of work undertaken since 2013. A brief outline of the work achieved and theory studied from 2013 to half-way through 2015 is provided as follows.

1.2.1 2013 – Platform establishment

The project was founded in 2013 as part of a final-year thesis by Rory O'Connor [13] and Christopher Venables [14]. The components needed for the MUAV were researched and acquired, with the group settling on a Hexacopter platform using a DJI NAZA-M V1 flight controller [13, 14]. The Raspberry Pi model B was used for autonomous processing.

By the end of 2013, the platform was able to perform basic autonomous waypoint maneuvering and object tracking, using a USB QStarz GPS, an XSens IMU and an onboard camera.

1.2.2 2014 – Building on the platform

In 2014, the project was continued by 4 students; Michael Baxter [12], Merrick Cloete [15], Alexander Mazur [16] and Omid Targargh [17]. The platform was not changed significantly from 2013. Of mention was the upgrade to the Raspberry Pi model B+ and the introduction the Piksi GPS, a purportedly centimetre-accurate RTK GPS [18]. Unfortunately the Piksi GPS proved to be too new and its firmware unstable to be of much use in the studies conducted in 2013 [12].

Work in 2014 focused largely on improving the waypoint navigation of the Hexacopter [12, 17], performance improvements in the object tracking [12], and the introduction of a web interface to control it [16]. However, the XSens inertial measurement unit more often than not remained unused, due to power restrictions on the Raspberry Pi [12]. Thus, the only sensor available to the Raspberry Pi (and hence the automation software) was the GPS. This, in addition to the automation method of emulating flight control inputs served as a significant limiting factor to the performance and ability to expand and improve on the automation system.

1.2.3 2015 – Early system improvements

Multiple improvements were made early on in the project this year, but which are not the main focus of this report; they are mentioned here for completeness. The most major change was the transition from the DJI NAZA-M V1 flight controller to the 3DR Pixhawk running the open-source ArduPilot APM software stack[1]. The power distribution layout and wiring harness used was completely remade. Furthermore, the addition of inexpensive landing feet have saved many (more expensive) arms from breaking on hard landings as well as providing greater ground clearance[2]. Finally, the switch to the Raspberry Pi 2 (available at the same price as the original Raspberry Pi) has significantly increased the compute power available by around a factor of 4, and which has proved to be highly useful for increasing image processing capabilities.

[1]More about the software changes with regards to this switch are mentioned in Section 3.1

[2]Refer to Appendix B for a comparison of the physical changes made to the system.

2. Background

2.1 Image processing

Image processing encompasses a broad scope of algorithms and methods with varying levels of complexity. While some algorithms may take many hours to complete, the focus on real-time image processing restricts scope to algorithms that can be performed 'on the fly'. It is to be noted that OpenCV [19] will be used, which is a comprehensive image processing library.

2.1.1 Blob identification

As identified in Section 1.1, blob identification is a simple but fast and effective means of identifying objects. At its core, the algorithm identifies contiguous regions in the image which have a similar properties (often colour). This process (depicted in Figure 2.1) involves:

1. Conversion to HSV colourspace, for ease of colour identification [20]
2. Thresholding the image for a predetermined colour
3. Applying identification algorithms to the thresholded image

The primary drawbacks are that it must be trained to detect objects of a specific colour [21], and that it may not work well if there is a lack of contrast between the colour of the background and the colour of the object. The limiting performance factor to this method is the HSV colourspace conversion, which is compute intensive. This is mitigated by the fact that a lookup table can be precomputed for a specified colour, reducing the processing requirements [12, 22].

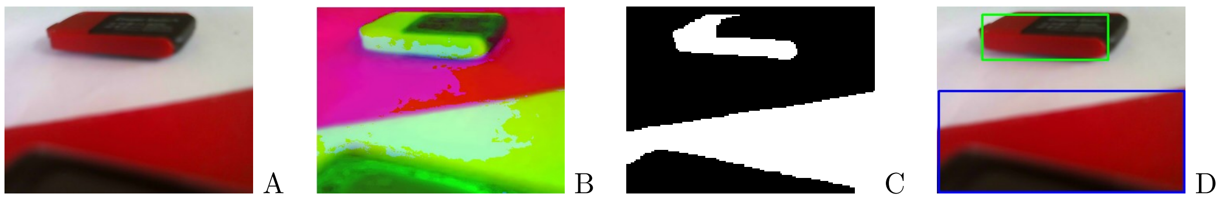


Figure 2.1: Capture (A), HSV conversion (B), thresholding (C), identification (D).

As shown in Figure 2.1, conversion to the HSV colourspace makes it easy to threshold to a specific colour, since the colour information is available on a single channel (Hue), and is mostly independent of lighting conditions [20]. Interestingly, [23] suggests that while the HSV colourspace is good for this purpose, it also suffers from *false positive* detection due to shadows and highlights, at least more so than the YC_bC_r colourspace. A potential area for improving detection accuracy may thus be to use YC_bC_r for colour thresholding.

Connected components algorithm

Once the image is thresholded, a common method to determine object position(s) within the frame is to perform connected components analysis [23]. This involves finding the 'centre of mass' of groups of thresholded pixels that are adjacent ('connected') to each other [12], such as depicted in Figure 2.1 (C) and (D), which finds two contiguous groups. Multiple libraries exist to perform this task, which can be treated as a contour finding problem - such as the OpenCV function `cv::findContours` and `cv::moments` [24].

2.1.2 Sequential image analysis

As discussed in [21] and [25], the Camshift (continuously adaptive mean-shift) algorithm also performs blob detection, but it does not use colour thresholding. Instead, it computes a *probability distribution* in the form of colour histogram [25] to create a back projection image, which is then used for tracking. Comparing the current and previous histograms with a defined search window, the object may be tracked based on how the histogram has shifted.

Using a probability distribution which is adapted with time allows this method to work even in noisy situations, and even in cases of partial occlusion [25]. Although this method is slower than simple thresholding, it has been found to be more accurate in tracking objects, especially under varied lighting conditions [21], which may make it more useful than thresholding alone.

2.1.3 Glyph/feature detection

In contrast to blob detection, it may be possible to identify objects based on the shape of the object [26]. The application of this method would be to create identifying markers of a specific shape, which are then tracked by the hexacopter.

A standard edge detection algorithm is the Canny algorithm [26, 27]. Once the edges have been detected, the Hough transform may be used to extract features of the object [26]. This is then compared to the features expected of the object (template matching) [26].

The Hough transform helps remove the effect of noise on detected edges, making it easier to identify and compare expected features [26]. However, it can be compute and memory intensive [26], so it remains to be seen whether or not this method is viable on the Raspberry Pi.

Detection of complex objects

Detecting objects without the use of identifying markers and where there is a large variability in shape, such as human figures, is a complex task. However, real-time detection of such objects is important to 'chase-cam' object tracking capabilities explored by Downing [10]. Many methods depend on using a feature extraction algorithm coupled with a classifier system to interpret the extracted features in terms of a detected object. For example, the scale-invariant feature transform (SIFT) algorithm is an established method to extract salient features from an image, while the support vector machine (SVM) is a robust classifier system [28].

A method for detecting humans is presented in [28], which uses a histogram of oriented gradients (HOG) to detect relevant features. It then couples this with an SVM classifier to detect humans. With regards to the hexacopter platform, two major limitations to implementing this capability are the lack of computational power available and the need for a training dataset for objects as viewed from an overhead perspective.

2.1.4 Processing limitations of the Raspberry Pi

The Raspberry Pi model B+ has a single core 700 MHz BCM2835 CPU with a Broadcom VideoCore IV GPU [29]. This is roughly equivalent to a 300 MHz Pentium II computer [29], which is the performance level from roughly 2 decades ago. The Raspberry Pi 2 has a quad-core 900 MHz ARM Cortex A7 CPU with the same GPU [30]. It is significantly faster than the B+, but still extremely limited as compared to the computing power of conventional systems. Limited computing power is *the* bottleneck when it comes to image processing, due to the sheer amount of data involved.

Keeping the frame rate high is key to a high performance control system for autonomous object detection and tracking. The VideoCore GPU is sufficiently powerful to capture and display full high definition (1920x1080 pixels) video at 30 frames per second [29]. This is only possible as it completely bypasses any interaction with the CPU, hence avoiding the performance constraints identified earlier. However, when image processing is required, data *must* pass through the CPU. To process images at 30 frames per second, the processing time must be under 30 milliseconds per frame. Past workarounds include reducing the image resolution (160x120 pixels in 2014 [12]) to obtain a usable frame rate.

2.1.5 GPU acceleration

Image processing

As noted earlier, the Raspberry Pi has a powerful GPU [29, 31], but at present the GPU remains unused. However, this need not necessarily be the case. Leveraging the fact that its architecture allows large data sets to be processed in parallel [32], it is possible to use items called *shaders* to apply a number of common image processing effects, including dilation, erosion, edge filtering, resizing and thresholding [33]. This unburdens the CPU, potentially increasing processing speeds.

While OpenCV has a library for GPU accelerated image processing, it is implemented using the CUDA programming language, which only works with NVIDIA brand graphics cards [34]. Instead, GPU image processing is possible through OpenGL ES 2.0 [31, 33].

Computation-latency trade-off

Care must be taken when considering GPU acceleration. Using this method, image data must inevitably pass between the GPU and CPU, and there is a speed limit to this process. Considering that a single frame at 1280x720 pixels occupies approximately 2.7MB of data, this link would need to be capable of 160 MB/s to achieve processing at 30 frames per second. Therefore, it may be found that although using the GPU decreases CPU processing requirements, the limit in data transfer speed restricts any overall gains.

Video encoding

The GPU on the Raspberry Pi provides special functions to perform hardware accelerated *encoding*, such as encoding and saving images in the JPEG format, or encoding and saving videos using the H.264 codec [31]. This makes it possible to record the video feed from the camera in real time and directly on the Raspberry Pi. This would allow completely autonomous video logging without requiring any need to connect to a base station to view and record the video stream.

Hardware accelerated encoding functions are provided through the *OpenMAX™* interface library (IL) [31]. The key limitation to video encoding is by what must be done on the CPU. In particular, the encoder only accepts video in planar YUV 4:2:0 (YV12), BGR (24bpp) and ABGR (32bpp) colourspaces [36]. OpenCV operates in BGR colourspace — except that its definition of BGR is the reverse [37] to that used in OpenMAX [38]. Thus, a colourspace conversion *must* be performed, which can only viably be done using the CPU.

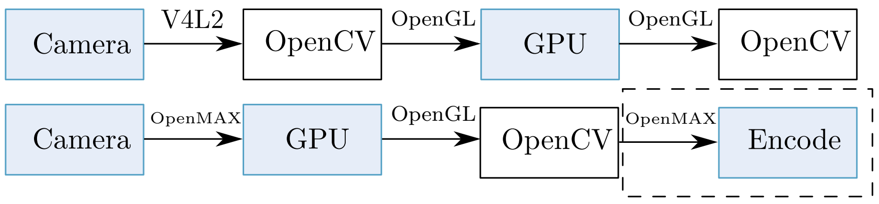


Figure 2.2: Two proposed processing pipelines. GPU actions are coloured blue. The first is easier, but requires two transfers between the CPU and GPU. The second avoids this, but is more complex. The encode section is optional, and may also be applied to the first pipeline.

2.2 Web interface design

The project inherits a web interface that was designed in 2014, which uses the Thrift protocol to communicate with the control system [16]. Overall, testing has indicated that the backend design and use of Thrift is suitable. However, the interface itself was not built with flexibility and extensibility in mind, meaning that extending the interface is difficult.

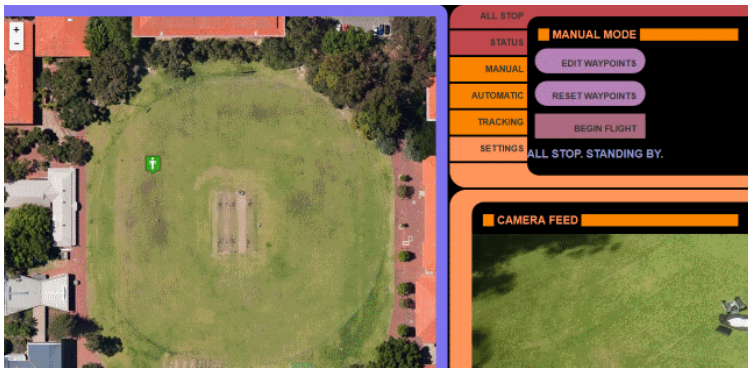


Figure 2.3: The 2014 web interface as viewed from a laptop.

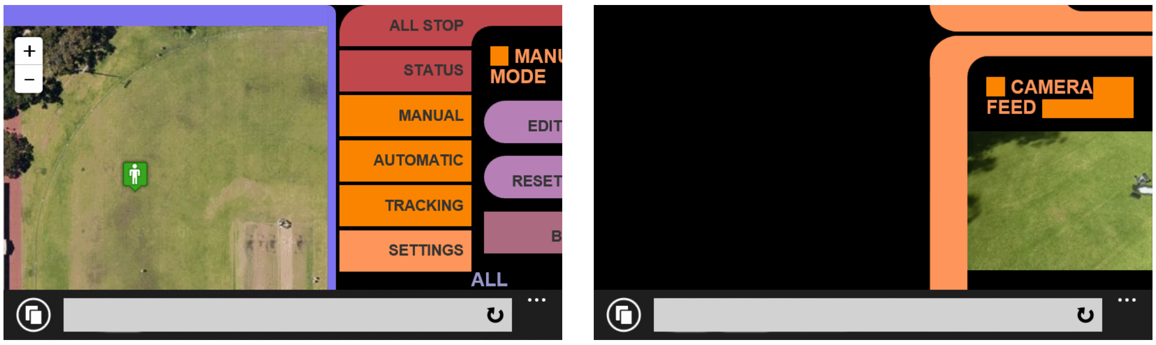


Figure 2.4: The same 2014 web interface, as viewed from a phone.

The web interface as normally displayed is shown in Figure 2.3, which requires a suitably large screen. The layout is functional, but a concerning factor is the lack of space for status information and adding extra controls to the sidebar. It also simply does not display enough information about the current status of the hexacopter to know how it is behaving. From an implementation perspective, the interface does not work well across a wide range of devices. Figure 2.4 shows how the layout is displayed incorrectly on a phone, with the right sidebar being cut off and the camera feed being awkwardly positioned.

3. Design of an autonomous POI system

3.1 ArduPilot platform

The system was switched to the ArduPilot platform, using the 3DR Pixhawk flight controller, forming a major part of the changes that were made this year. In particular, switching to this platform greatly simplified development, allowing the low-level control system to be treated as a black box. A full comparison is provided in Appendix A.

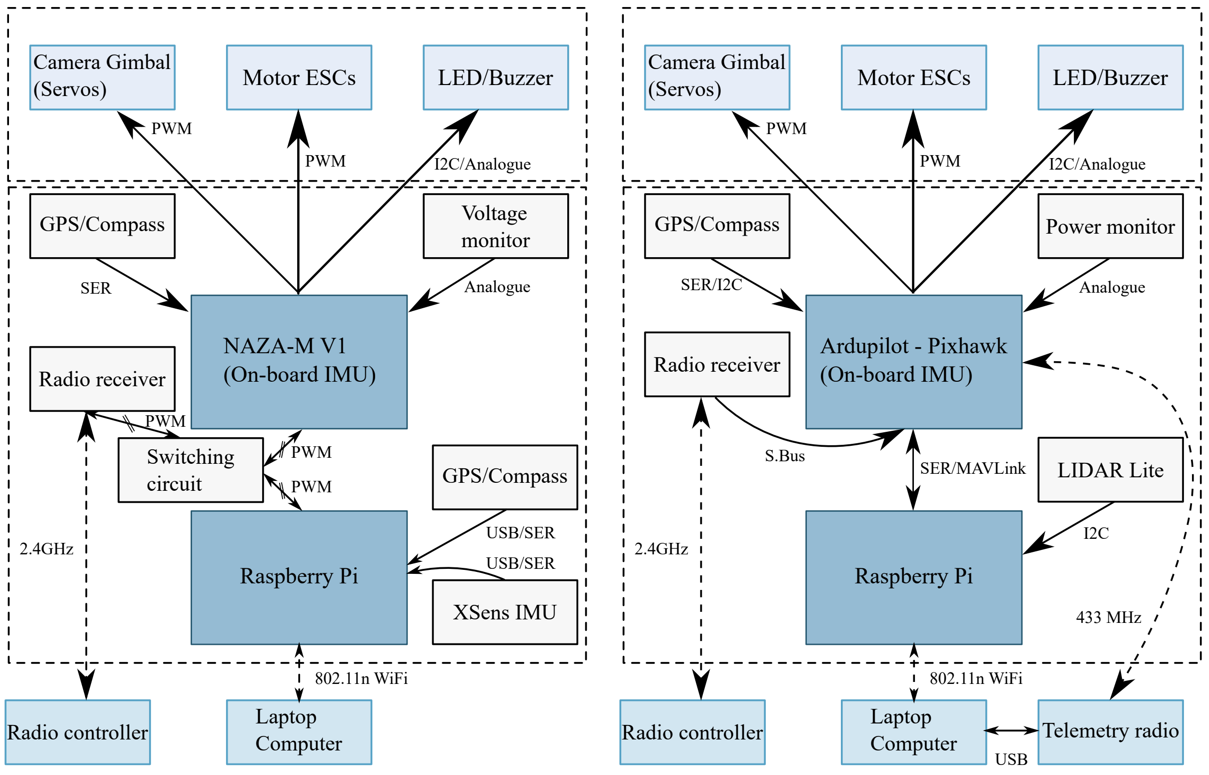


Figure 3.1: Functional block diagram comparison of the old (left) and new systems (right).

3.1.1 Communications protocol

Most flight control logic is handed off from the Raspberry Pi to the Pixhawk. These systems are interconnected via a serial link, with communications conforming to the MAVLink (Micro Air Vehicle Link) protocol [39].

This communications link allows access to almost every aspect of the flight controller, including all GPS and altitude data, accelerometer readings and power consumption levels. As shown in Figure 3.1, this eliminates the need to have a separate GPS and IMU connected to the Raspberry Pi, reducing system complexity. Furthermore, it allows the Raspberry Pi to control movement, such as instructing it to fly to particular waypoints (including at specific altitudes), or just at a particular speed and acceleration.

For safety, autonomous control is only engaged when the Pixhawk is in a special mode, called 'guided mode'. This mode can only be switched to by the operator, either via the radio controller or telemetry link. This mode can also be overridden at any time to regain manual control.

3.2 Software platform and structure

Similar to the system designed in 2014, the autonomous control software residing on the Raspberry Pi was designed as a monolithic application written in C++, conforming to the C++11 standard[1]. Threading is heavily used to ensure that sensor information can be updated in the background. In addition, and unlike in 2014, there is extensive use of both mutexes and atomic variables to ensure consistency in data that is passed between threads.

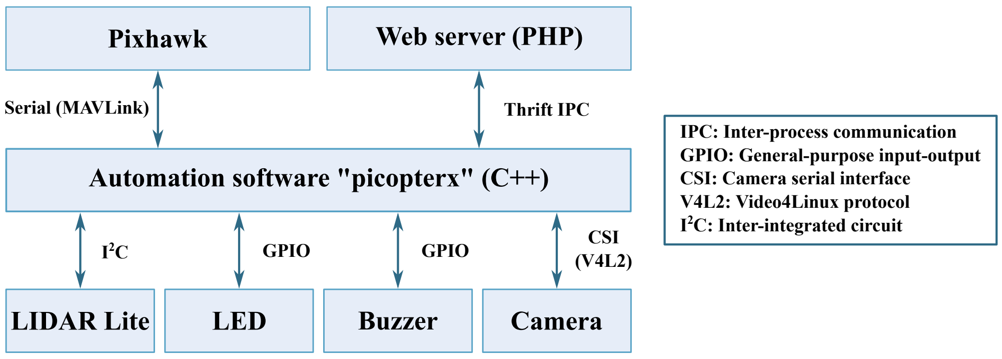

Pixhawk
Web server (PHP)
Serial (MAVLink)
Thrift IPC
Automation software "picopterx" (C++)
I²C
GPIO
GPIO
CSI (V4L2)
LIDAR Lite
LED
Buzzer
Camera
IPC: Inter-process communication
GPIO: General-purpose input-output
CSI: Camera serial interface
V4L2: Video4Linux protocol
I²C: Inter-integrated circuit

Figure 3.2: A diagram of software on the Raspberry Pi and interconnections to other components.

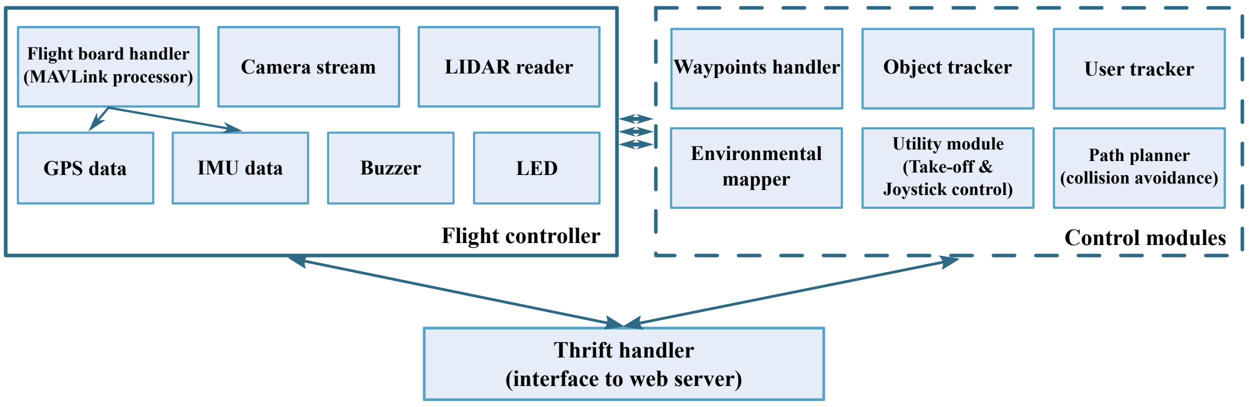

Flight board handler (MAVLink processor)
Camera stream
LIDAR reader
GPS data
IMU data
Buzzer
LED
Flight controller
Waypoints handler
Object tracker
User tracker
Environmental mapper
Utility module (Take-off & Joystick control)
Path planner (collision avoidance)
Control modules
Thrift handler (interface to web server)

Figure 3.3: A diagram of the layout of components within the automation software.

As depicted in Figure 3.3, logic is organised into a number of modules to increase maintainability and extensibility of the code. Core logic that interacts with sensors and external systems are grouped into a base ‘flight controller‘ module. The Thrift interface maintains an instance of this flight controller, which is then used by higher level control modules like the waypoints handler to perform tasks. For portability, the CMake build system is used, ensuring it can be compiled both on the Raspberry Pi and on standard Linux distributions like Ubuntu and Debian.

[1]The complete software may be found on GitHub at https://github.com/jtanx/picopterx.

3.3 Object identification system

Extending on the object identification capabilities present in 2014, a number of improvements were made both in terms of capability and performance.

3.3.1 Object identification model

A number of properties are associated with each identified object to aid in tracking:

1. The position of the object in the frame (in pixels)
2. The bounding box of the object
3. The resolution of the captured image
4. The unique identifier of the object (if any; mainly used for glyphs)
5. The global position of the object (in global coordinates; i.e. latitude/longitude/altitude)

Some parameters, such as the global position are calculated by the tracking functions provided by Downing [10] to localise an object through time.

3.3.2 Blob detection

Colour based object detection remains a significant method for identifying objects due to its computational simplicity and relative effectiveness. Improvements to blob tracking stem from the ability to track not only through HSV thresholding, but also through $Y'C_bC_r$ thresholding. Furthermore, these parameters can be configured in real-time through the web interface.

Similar to the HSV model, the $Y'C_bC_r$ model separates luminance from colour values, which are stored in the C_b and C_r chrominance channels. Assuming a range of $[0, 255]$ for values, the conversion from RGB to $Y'C_bC_r$ colourspace is [40]:

$$Y' = 0.299R + 0.587G + 0.114B \tag{3.1}$$
$$C_b = -0.168736R - 0.331264G + 0.500B + 128 \tag{3.2}$$
$$C_r = 0.500R - 0.418688G - 0.081312B + 128 \tag{3.3}$$

As with HSV thresholding, a lookup table is precomputed to minimise the computation effort needed when performing image tracking.

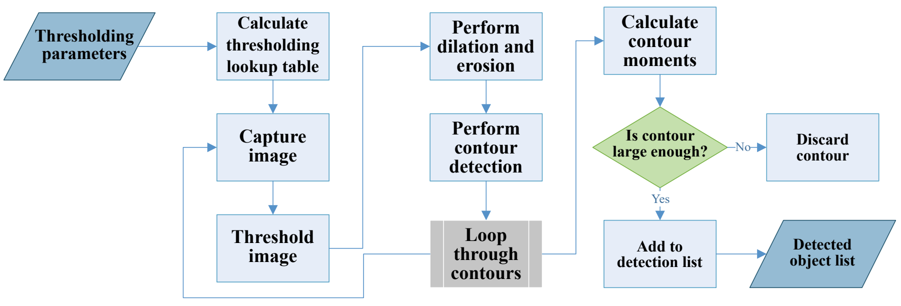


Figure 3.4: Flow chart of the colour thresholding detection algorithm.

Camshift tracking

Thresholding detection is combined with the Camshift algorithm to track objects through time. Using the thresholding algorithm, the region of interest is first calculated. From the region interest, its histogram is calculated, which is then used by the Camshift algorithm to track the object.

3.3.3 Glyph detection

A new feature added this year is the concept of glyph tracking. When used as a location marker, glyphs provide a simple yet effective method to localise the hexacopter, either relative to the marker or in absolute coordinates if the global position of the marker is also known. Purpose-made glyphs are easily identifiable, which is what makes this function attractive. It is acknowledged that marker glyphs must be installed specifically for this purpose, which may not always be possible. However, the simplicity of such a method does not detract from its effectiveness to allow for precise localisation that simply cannot be achieved through the use of a GPS alone.

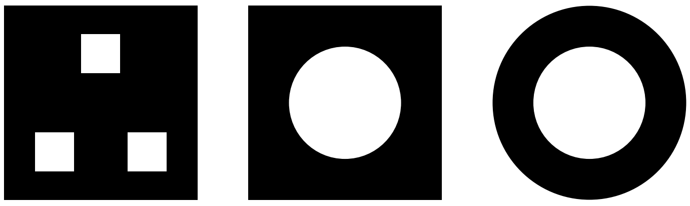
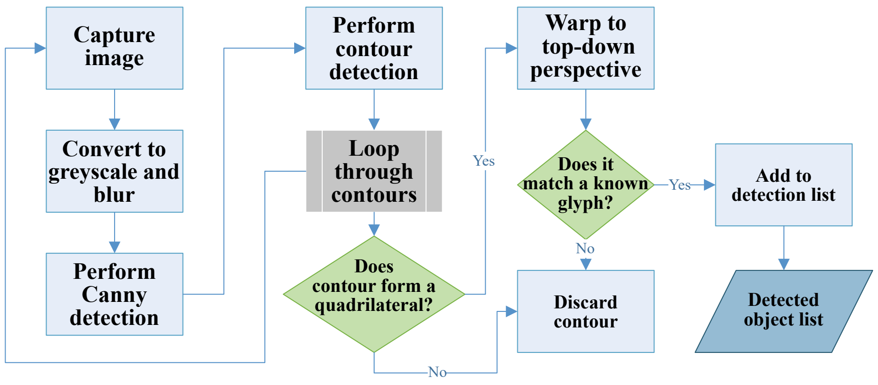


Figure 3.5: Three glyphs that could be used for identification purposes.


Capture image
Convert to greyscale and blur
Perform Canny detection
Perform contour detection
Loop through contours
Does contour form a quadrilateral?
Yes
No
Warp to top-down perspective
Does it match a known glyph?
Yes
No
Add to detection list
Discard contour
Detected object list
Figure 3.6: Flow chart of glyph detection using Canny detection for square glyphs.

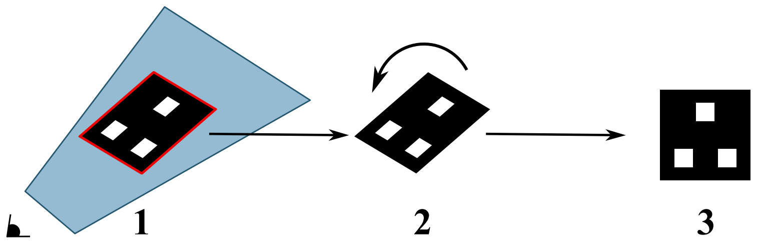


Figure 3.7: Visualisation of quadrilateral detection and transformation to top-down perspective.

The first method of glyph detection depicted in Figure 3.6 requires that the glyphs are roughly square in proportion. Through the use of the Canny edge filter, such square shapes can then be detected and extracted from the image. This is possible by:

1. Finding the contours of the image.
2. Determining the approximate number of corners for each contour.
3. Discarding all contours that do not have four corners, as well as those that are not convex.

With a detected quadrilateral, a perspective transformation matrix that warps it into a top-down perspective is then calculated. This is possible because it is known that the glyph must be square.

The perspective transformation that maps $\langle x_i, y_i \rangle$ to $\langle u_i, v_i \rangle$ is [41]:

$$\begin{bmatrix} t_i u_i \\ t_i v_i \\ t_i \end{bmatrix} = \begin{bmatrix} c_{00} & c_{01} & c_{02} \\ c_{10} & c_{11} & c_{12} \\ c_{20} & c_{21} & c_{22} \end{bmatrix} \cdot \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \qquad i = (0, 1, 2, 3), \quad c_{22} = 1 \quad (3.4)$$

Unknown coefficients are determined by solving a system of linear equations using the input transform pairs. This function is provided by OpenCV (`getPerspectiveTransform`) [41]. Once warped into a top-down perspective, the image is compared to a set of known glyphs using the process of template matching [42]. Template matching compares the reference and captured image for similarity. In this case, a normalised cross-correlation coefficient is calculated, which can be treated as a probability of how closely they match. Setting a threshold for the probability (i.e. $P_{match} > 0.8$) thus indicates if the captured image matches a known glyph.

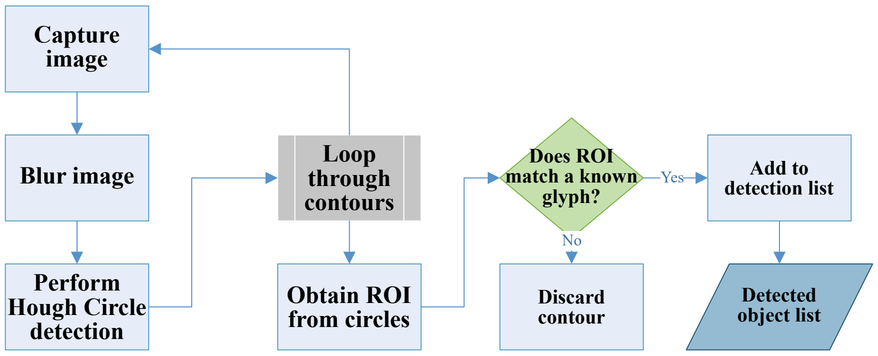


Figure 3.8: Flow chart of glyph detection using Hough circle detection for circular glyphs.

Template matching is not restricted to square glyphs. Figure 3.8 shows the use of the Hough circle detection method to detect circular glyphs. However, the Hough circle detection does not account for any warping due to the perspective of the camera relative to the glyph, making it is more difficult to both detect and compare circular glyphs.

3.3.4 Histogram of oriented gradients to detect humans

An identification system for humans is integrated through the use of the OpenCV provided class, `HOGDescriptor`, which implements a histogram of oriented gradients detector coupled to a trained SVM classifier. However, only cursory attention is paid to this method, as it is extremely computationally intensive, limiting its use in real-time object tracking.

3.3.5 Multi-threading and GPU acceleration

To increase image processing performance, two techniques are considered. The first is multi-threaded processing, which exploits the multi-core nature of the Raspberry Pi 2. As depicted in Figure 3.9, each frame is sliced into multiple chunks, and processed both independently and in parallel to one another. This method only works for algorithms that do not require adjacent pixels to perform the calculation – for example, thresholding is possible, but blurring is not possible.

The second method is that of using the GPU and OpenGL to perform the thresholding. Images are sent to the GPU as a texture (`glTexSubImage2D`), processed using vertex and fragment shaders (Figure 3.1), before being read back (`glReadPixels`).

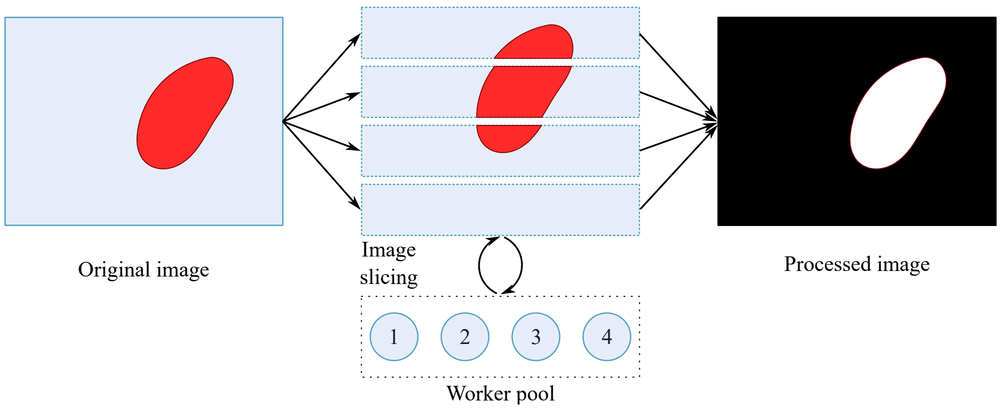


Figure 3.9: Illustration of multi-threaded image processing using thread pooling.

```
varying vec2 tcoord;
uniform sampler2D tex;
uniform vec4 threshLow, threshHigh;

void main(void)
{
    mat4 RGBtoYCrCb = mat4(0.257,  0.439, -0.148, 0.0,
              0.504, -0.368, -0.291, 0.0,
              0.098, -0.071,  0.439, 0.0,
              0.0625, 0.500,  0.500, 1.0 );

    vec4 yuv = RGBtoYCrCb * texture2D(tex,tcoord).bgra;
    bvec4 t = greaterThanEqual(yuv, threshLow) &&  lessThanEqual(yuv, threshHigh);

    if (all(t)) {
        gl_FragColor = vec4(1,1,1,1);
    } else {
        gl_FragColor = vec4(0,0,0,1);
    }
}
```

Listing 3.1: An example fragment shader to perform $Y'C_bC_r$ thresholding.

3.4 Video capture

For real-time video capture, an extremely low latency method was required. To accomplish this, hardware encoding was used, as outlined in Section 2.1.5. This software was written from scratch[2], using the OpenMAX API and the `video_encode` component [43].

To minimise encoding latency, sending and receiving data from the encoder is handled in a separate thread, with encoding conducted concurrent to any image processing. As a result, the call to encode a frame must copy it into a buffer that it holds. From Figure 3.10, this copy step is performed as part of the BGR to RGB conversion. If the encoder falls behind, it will reject any new input, effectively dropping frames.

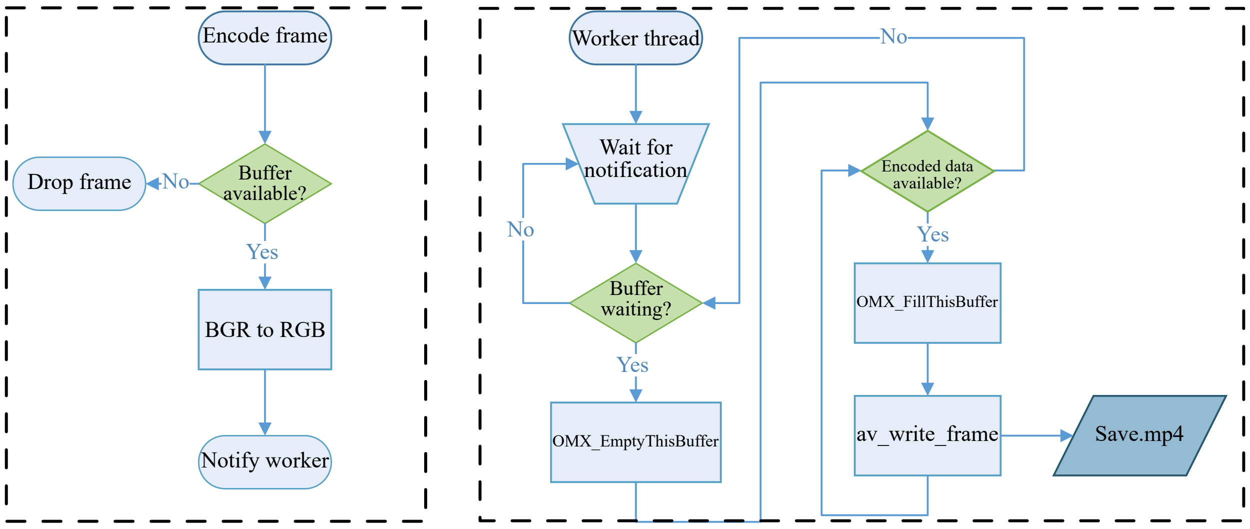


Figure 3.10: Flow chart of the encoding process.

3.4.1 BGR to RGB conversion

The BGR to RGB conversion is critical, because it forms the majority of work that must be done on the CPU before encoding by the GPU. As a common procedure, OpenCV provides a function to accomplish this (`cvtColor`). However, this step should be as fast as possible to maximise the amount of time available for doing image processing.

With the upgrade to the Raspberry Pi 2, the newer ARM Cortex A7 processor supports a single instruction, multiple data (SIMD) instruction set, called NEON. This enables data to be processed in parallel, with specific instructions being well suited to BGR to RGB conversion.

As shown in Figure 3.11 and Listing 3.2, the instruction `vld3.8` allows BGR data to be loaded and deinterlaced from memory into three NEON registers, `d0`, `d1` and `d2`, which holds 8 blue, green and red pixels respectively. The red and blue channels are then swapped with `vswp` before being interlaced and stored in memory with the corresponding `vst3.8` command.

However, Listing 3.2 is not optimal. A separate NEON processing pipeline exists, and data loading is not immediate. Performing the swap soon after the load causes a *pipeline stall* [44], where the processor must wait for data to be loaded before continuing. Listing 3.3 presents an optimised version, mitigating this issue by unrolling the loop – performing two load and two store instructions per loop. Furthermore, a special `pld` instruction hints to the processor to pre-load memory into its cache before it is used — how far ahead to preload is processor and situation dependent.

[2]The encoding software is available as a separate package at https://github.com/jtanx/omxcv.

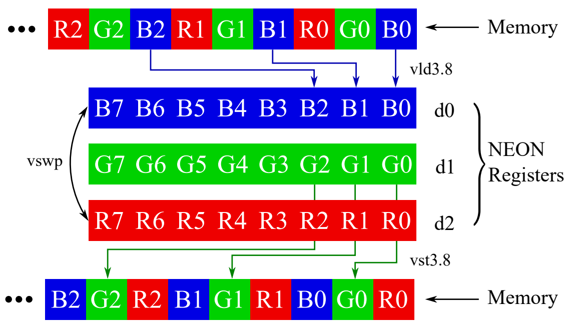


Figure 3.11: Visual representation of the BGR to RGB conversion process.

```
@void omxcv_bgr2rgb_neon(uint8_t *src, uint8_t *dst, int pixels)
omxcv_bgr2rgb_neon:
    mov r2, r2, lsr #3              @Right shift by 3 - pixels/8 (Number of runs needed)
    loop:
        vld3.8 {d0-d2}, [r0]!       @Read in 8 pixels
        vswp d0, d2                 @Swap R/B
        subs r2, r2, #1             @Decrement counter
        vst3.8 {d0-d2}, [r1]!       @Store 8 pixels
        bgt loop                    @Loop if counter is greater than zero
    bx lr                           @Return
```

Listing 3.2: The simplest version of a BGR to RGB routine using NEON instructions.

```
@void omxcv_bgr2rgb_neon(uint8_t *src, uint8_t *dst, int pixels)
omxcv_bgr2rgb_neon:
    mov r2, r2, lsr #4              @Right shift by 4 - pixels/16 (Number of runs needed)
    loop:
        pld [r0, #384]              @Preload memory for reading
        vld3.8 {d0-d2}, [r0]!       @Read in 8 pixels
        vld3.8 {d3-d5}, [r0]!       @Read in 8 more pixels
        vswp d0, d2                 @Swap R/B
        vswp d3, d5                 @Swap R/B
        subs r2, r2, #1             @Decrement counter
        vst3.8 {d0-d2}, [r1]!       @Store 8 pixels
        vst3.8 {d3-d5}, [r1]!       @Store 8 more pixels
        bgt loop                    @Loop if counter is greater than zero
    bx lr                           @Return
```

Listing 3.3: An optimised BGR to RGB conversion using memory preloading and loop unrolling.

Limitations to the NEON routines

Listing 3.2 processes 8 pixels at a time, while Listing 3.3 processes 16 pixels at a time. As such, the number of pixels that it processes must be a multiple of 8 and 16 respectively. In comparison, the version provided by OpenCV is more flexible as it does not have this restriction. As almost all video resolutions are multiples of 16 (e.g 320x240 and 640x480), restricting the width to a multiple of 16 is a reasonable trade-off for the gains in speed acquired.

3.4.2 Video storage

Raw H.264 data is returned from the encoder. For maximum compatibility, *Libavformat* [45] is used to pack (multiplex) the video into either the Matroska (MKV) or MP4 container format, which are both widely used and well supported. This also means that the video frames are properly timestamped to play back at the right speed, which is not possible with raw data alone. This is particularly important for real-time encoding, where the frame rate is not necessarily constant.

3.4.3 Image encoding

Once the framework was established to encode video, it was trivial to extend this to hardware accelerated JPEG image encoding, using the `image_encode` component. This is desirable, because saving a large number of images is required for environmental mapping and the web video feed.

3.5 Web interface

The web interface is the primary method through which the autonomous behaviour of the hexacopter is monitored and controlled. As such, the main functional requirements of the interface are:

1. Ability to instruct the hexacopter to perform autonomous functions
2. Ability to stop the hexacopter at any time, aborting any current action
3. To display enough information about the current state of the hexacopter to determine if it is performing as desired
4. Ability to review the actions of the hexacopter for later analysis

The current web interface does not meet the functional requirements, especially with respect to points 3 and 4. In particular, the only information displayed about the current state is the location and heading of the hexacopter, as well as the current state of the autonomous control software. There is also no ability to export data to review the behaviour of the hexacopter.

To address the functional requirements and concerns raised in Section 2.2, the primary design criteria for the web interface are:

1. Retain all functionality the the interface (from 2014) currently possesses
2. Increase functionality to control new routines introduced this year
3. Increase the amount of information (telemetry) displayed for diagnostics and analysis
4. Keep the interface as intuitive as possible
5. Increase compatibility of the interface across a large number of screen sizes and browsers
6. Increase maintainability and extensibility for future additions

3.5.1 Server model

Addressing the first two points of the design criteria, the same server-client model described in [16] using AJAX and Apache Thrift will be retained. Leaflet will continue to be used to display the map, with its functionality increased as needed to match the visualisations needed, such as the addition of a spiral search pattern and exclusion zones.

To increase the information displayed, the Thrift interface is extended to send additional information such as the attitude and altitude. To keep it intuitive, visual indicators will be used, which provides a quick method for the user to interpret the data.

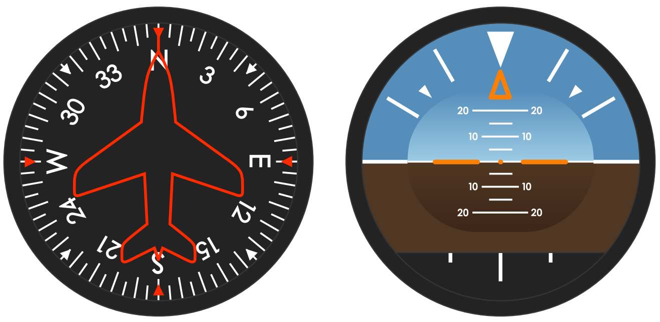


Figure 3.12: Left: A visual indicator for yaw, Right: A visual indicator for pitch and roll (artificial horizon). Copied from [46].

3.5.2 Configuration and data review — JSON and GPX

The JSON (JavaScript Object Notation) format is used for transmitting configuration data between the web interface and server. It is a widely used, human readable and standardised format [47, 48]. It is also natively supported by web browsers, making it trivial to use and interact with. On the other hand, C++ libraries like RapidJSON [49] allow for JSON parsing on the server.

```
{
    "GLOBAL" : {
        "OBSERVATION_MODE" : false
    },
    "CAMERA_STREAM": {
        "INPUT_WIDTH": 320,
        "INPUT_HEIGHT": 240,
        "PROCESS_WIDTH": 160
    }
}
```

Listing 3.4: An example of content in the JSON format.

```
<?xml version="1.0" encoding="UTF-8"?>
<gpx xmlns="http://www.topografix.com/GPX/1/0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-
    instance" xsi:schemaLocation="http://www.topografix.com/GPX/1/0 http://www.topografix.
    com/GPX/1/0/gpx.xsd" version="1.0">
  <wpt lat="-31.979708" lon="115.817756">
    <desc>P1</desc>
    <type>Lawnmower</type>
  </wpt>
  ...
  <trk>
    <name>Lawmower GPS Log</name>
    <trkseg>
      <trkpt lat="-31.97997" lon="115.817787">
        <time>2015-04-07T03:30:46Z</time>
      </trkpt>
    </trkseg>
     ...
  </trk>
</gpx>
```

Listing 3.5: An example of content in the GPX format; truncated for brevity.

Similarly, the GPS Exchange Format (GPX) is a standardised, open format for recording GPS data [50]. It is supported by tools such as Google Earth, making it an ideal format to save data for later review. For example, after an autonomous mission has been completed, the interface will provide a method to download a GPX log. The log will contain information such as the path (track) of the hexacopter and any points of interest detected along the way.

3.5.3 Interface layout

Addressing the last three design criteria, the interface will be redesigned using standard web libraries including Bootstrap [51] and jQuery [52]. In particular, Bootstrap is a modern and popular web (HTML/CSS/JavaScript) framework that allows websites to be rapidly designed and prototyped by providing a set of standardised components. Using standard layout elements provided by Bootstrap, the site can be guaranteed to display and accommodate well across a broad range of devices. Bootstrap allows for a ‘responsive’ layout, which is altered automatically to best fit both small and large devices.

4. Results

4.1 Object identification

4.1.1 Accuracy

$Y'C_bC_r$ thresholding

To compare the detection characteristics of $Y'C_bC_r$ thresholding to HSV thresholding, a test image sequence was established under a range of lighting conditions and the thresholding characteristics compared. For this comparison, the following thresholding parameters were used[1]:

$$H \geq 170 \textit{ or } H \leq 8 \qquad 116 \leq S \leq 255 \qquad 35 \leq V \leq 255 \tag{4.1}$$

$$0 \leq Y' \leq 255 \qquad 9 \leq C_b \leq 124 \qquad 152 \leq C_r \leq 255 \tag{4.2}$$

Figure 4.1: The baseline image for testing purposes.

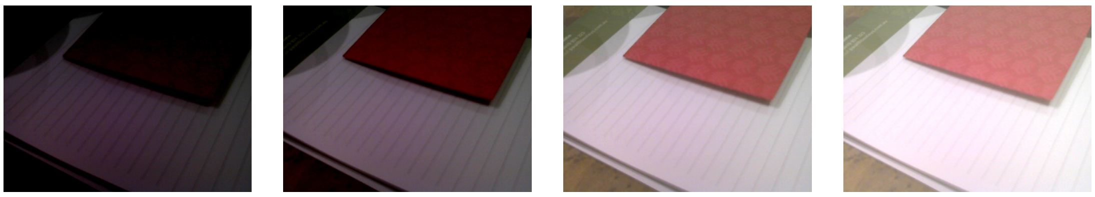


Figure 4.2: Testing images at -100%, -50%, +50% and +100% brightness.

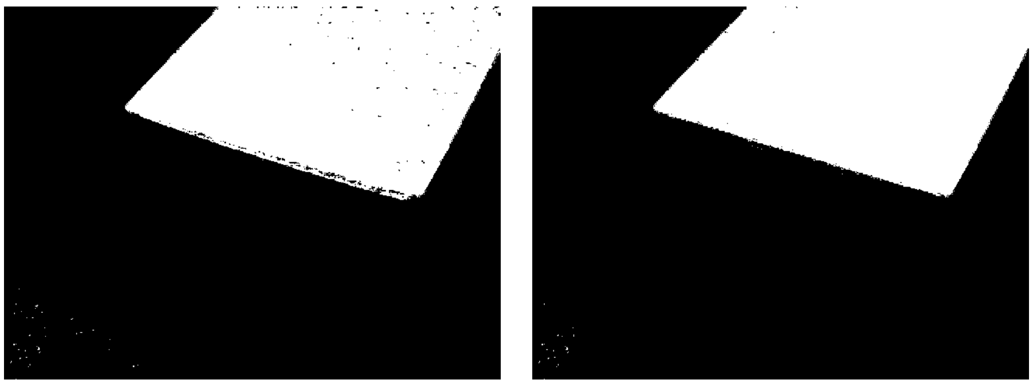


Figure 4.3: Baseline image thresholded in HSV (left) and $Y'C_bC_r$ (right) colourspaces.

[1]Following OpenCV convention, hue is in the range [0,180]. All other parameters are in the range [0,255].

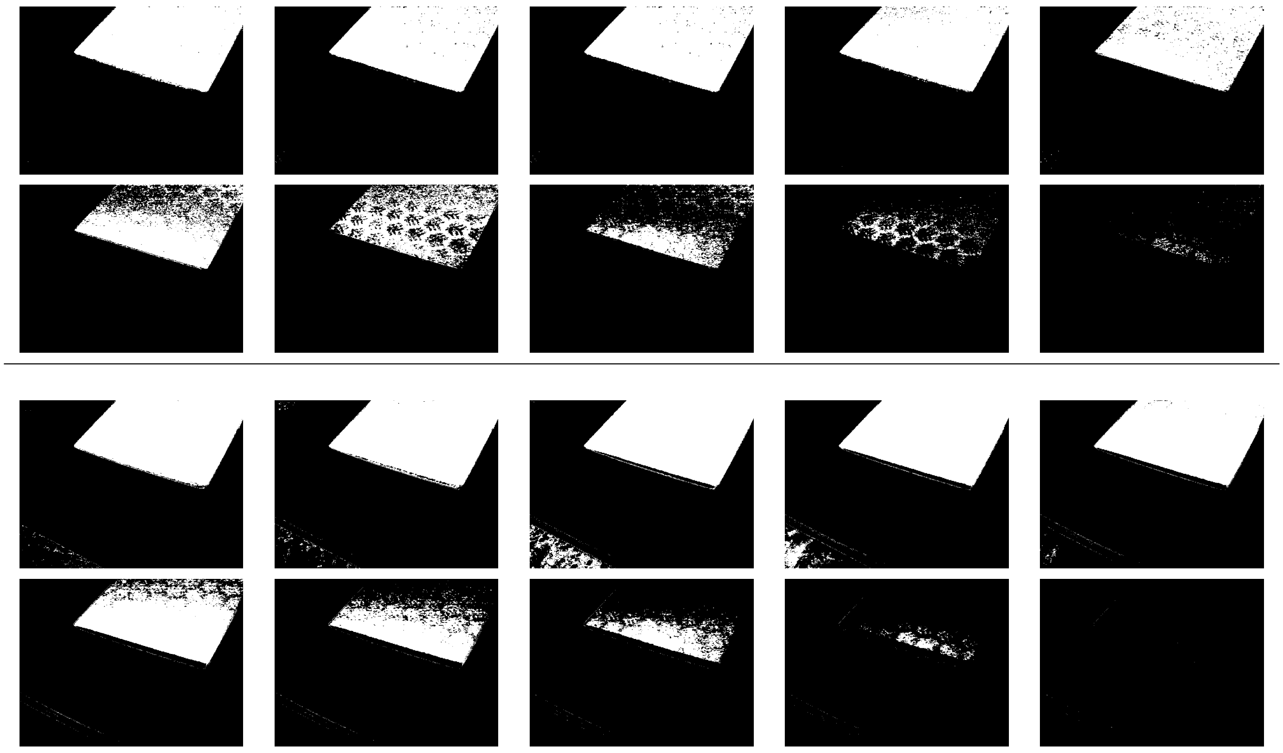


Figure 4.4: HSV thresholding of the test image from +10% up to +100% and -10% down to -100% brightness at 10% increments.

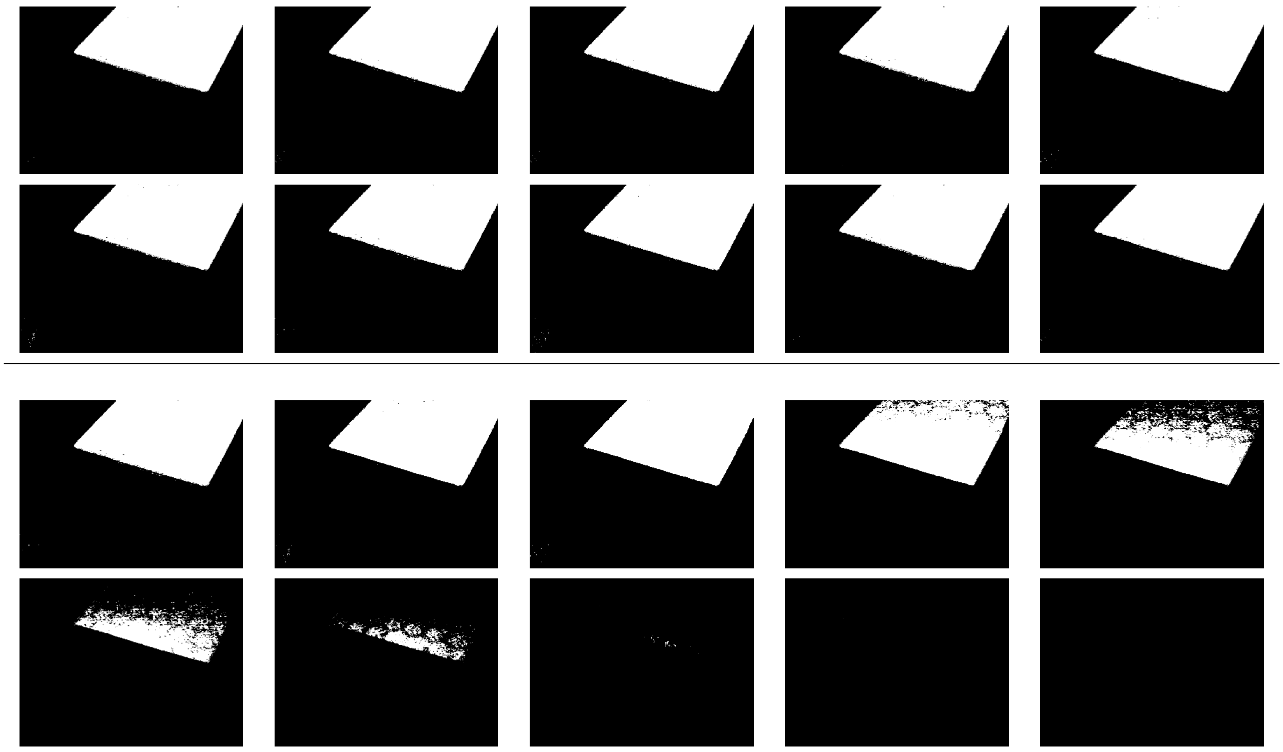


Figure 4.5: $Y'C_bC_r$ thresholding of the test image from +10% up to +100% and -10% down to -100% brightness at 10% increments.

The thresholding parameters used were based on optimising the threshold of the baseline image to capture as much of the red region as possible while rejecting all other regions in the image. These parameters were then used across the varying brightness levels. Figure 4.3 shows that with HSV thresholding, it was more difficult to capture the whole red region while rejecting the background.

Comparing Figure 4.4 to Figure 4.5, this is made clearer, with HSV thresholding performing poorer especially with increasing brightness. Past an increase of +50% in brightness, HSV thresholding gradually fails, while $Y'C_bC_r$ thresholding remains consistent. With decreasing brightness, HSV thresholding tends to work better, but not without issues. At -30% and -40% brightness, there is a distinct region of false detection in the lower left corner of the image. In comparison, little to no false positive detection is present in $Y'C_bC_r$ thresholding across the board.

Camshift detection

Camshift detection was found to behave similar to simple colour thresholding. However, it did suffer from some false positive detection, especially if the tracked object was moved off screen (Figure 4.6). Configuring minimum acceptable saturation and value parameters helped to reduce false positive detection.

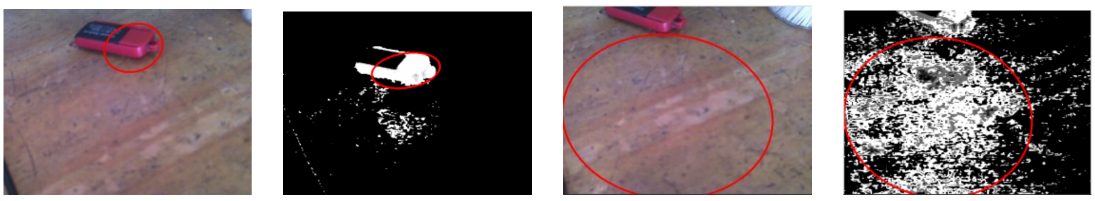


Figure 4.6: Camshift tracking, showing the back-projection image. False positive detection occurs as the tracked object is moved off-frame.

Glyph detection

Similar to testing colour thresholding, a set of test images were created to test the performance of glyph detection. The first two images in Figures 4.7 and 4.8 test detection at two differing glyph sizes. Physically, these correspond to glyphs at a size of approximately 65x65cm and 40x40cm when viewed from an altitude of 5m. For perspective, a square glyph printed on an A3 paper will be roughly 30x30cm. In these cases, the glyphs are detected correctly. The last image set testing motion blur shows one effect that can cause glyph detection to fail. In this case, motion blur in the image sufficiently deformed the glyph edge such that it could not be detected as a square object. This is apparent by observing the Canny processed image in Figure 4.8.

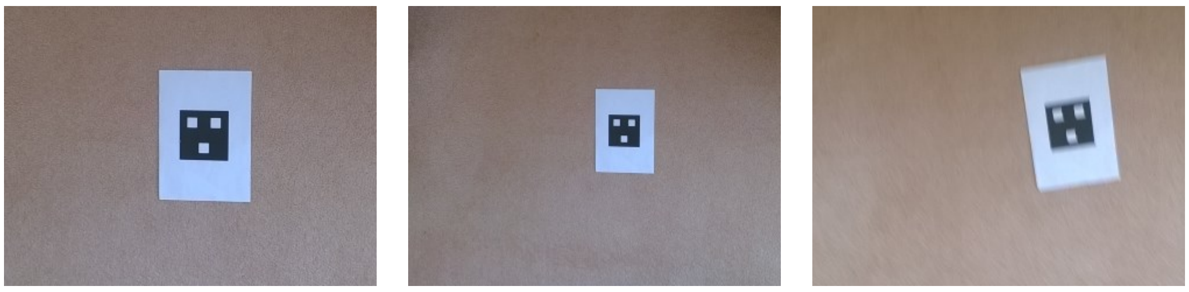


Figure 4.7: Test images at two sizes (left, center) and with motion blur (right).

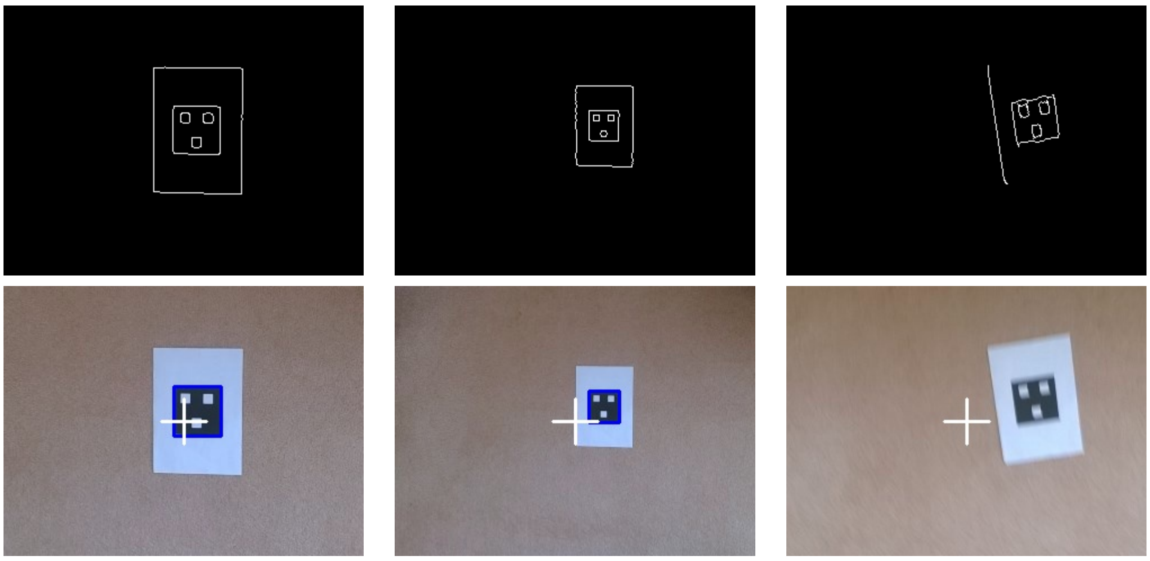


Figure 4.8: Top: Image after Canny detection. Bottom: Result of glyph detection.

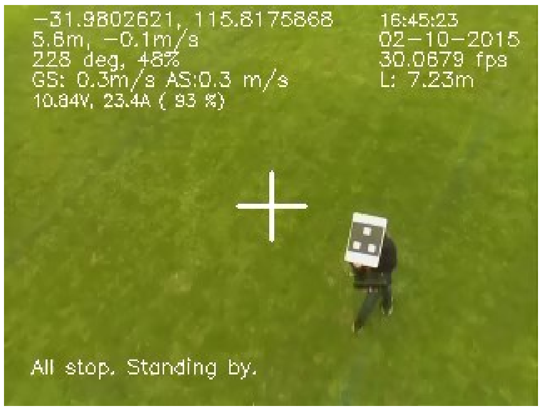


Figure 4.9: Aerial glyph testing

Further testing was conducted to determine the maximum range of detection by gradually increasing the distance from the glyph to the camera. Consistent detection fails when the glyph covers an area of 30x30 pixels or less. For a processing resolution of 320x240 pixels, this limits detection to an altitude of around 5 metres when using an A3 sized glyph.

Histogram of oriented gradients detection

The detection rate for the HOG algorithm was marginal, especially when attempting to detect humans from an aerial view. Out of a 500 frame aerial image sequence (each containing person(s)), 142 frames were correctly identified, representing a 28.4% detection rate. Better results would be attained if the detector was trained to detect humans from an aerial perspective. Nonetheless, the throughput as demonstrated in Section 4.1.2 is so low as to make this irrelevant.

4.1.2 Performance

The increased computing performance of the Raspberry Pi 2 coupled with more efficient colour thresholding has resulted in significant image processing performance improvements as compared to 2014. Note that both $Y'C_bC_r$ and HSV thresholding use the exact same lookup table method, so there are no performance differences between the two.

Table 4.1: Connected components performance characteristics (single-threaded)

Resolution	Processing Rate (FPS)	CPU (%)	MEM (%)
160x120	29.9	46.0	4.00
320x240	29.9	91.0	3.80
640x480	9.27	102.6	4.90
800x600	5.95	101.2	5.68
1280x720	3.19	101.0	7.58

Table 4.2: Connected components performance characteristics (multi-threaded)

Resolution	Processing Rate (FPS)	CPU (%)	MEM (%)
640x480	10.8	120.6	4.90
800x600	7.02	121.3	5.69
1280x720	3.78	116.4	7.45

Table 4.3: 2014 Connected components performance

Resolution	Processing Rate – No object detected (FPS)	Processing Rate – Object detected (FPS)
160x120	30	30
320x240	21	12
640x480	6.1	2.5
1280x720	2.2	0.9

Table 4.4: Camshift performance characteristics

Resolution	Processing Rate (FPS)	CPU (%)	MEM (%)
160x120	29.8	26.7	3.6
320x240	29.8	56.5	3.89
640x480	17.2	107.0	5
800x600	9.79	102.7	5.9
1280x720	5.56	103.7	7.99

Table 4.5: Canny glyph detection performance characteristics

Resolution	Processing Rate (FPS)	CPU (%)	MEM (%)
160x120	29.9	53.7	3.6
320x240	27.7	105.3	4
640x480	8.32	103.5	5.29
800x600	5.82	102.4	6.22
1280x720	3.10	101.4	8.44

Table 4.6: HOG detection performance characteristics

Resolution	Processing Rate (FPS)	CPU (%)	MEM (%)
160x120	N/A	N/A	N/A
320x240	1.81	100.6	4.2
640x480	0.248	100.3	5.62
800x600	N/A	N/A	N/A
1280x720	0.0336	100.2	10.05

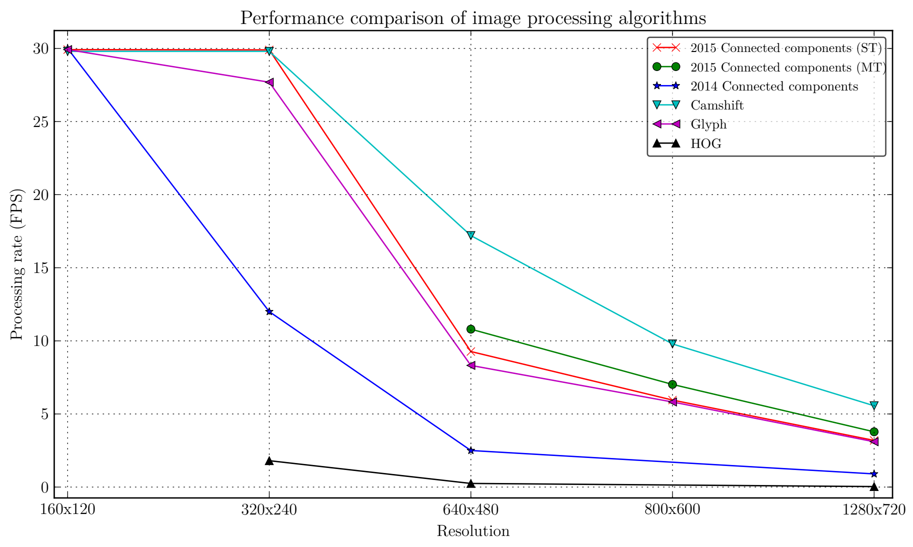


Figure 4.10: Comparison of processing rates for various image processing algorithms.

With 4 CPU cores on the Raspberry Pi 2, the CPU percentage may range up to 400%. Interestingly, these results show that Camshift detection is slightly faster than single-threaded connected components thresholding. This suggests that the added complexity of finding contours and tracking multiple blobs is slower than tracking a single blob using a probability distribution.

For comparison, the 2014 connected components algorithm [12] was run on the Raspberry Pi 2. Comparing Tables 4.1 and 4.2 to Table 4.3, the current version is consistently around 50% faster. However, its performance also degrades when objects are detected, which does not occur for the current version. In this case, performance is 150%–270% faster, depending on the resolution.

Glyph detection was slightly slower than colour thresholding. Profiling[2] suggests that this is due to the use of the Canny edge detection. Another contributing factor is that each contour detected in the image must be compared to each glyph that the system knows. For n contours and m known glyphs, this is an $O(nm)$ algorithm. Thus, as the number of known glyphs increases and as the number of contours in the image increases, processing time increases proportionally.

Finally, performance of the HOG algorithm was extremely poor. Figure 4.6 shows that performance was at 1.81FPS or less across the range of resolutions. Some results are missing because the HOG algorithm will only accept specific resolutions.

Comparison of GPU acceleration to multi-threading

Disappointingly, and as Table 4.7 and Figure 4.11 shows, there was little to no benefit gained from GPU accelerated thresholding. While it certainly performs better than single-threaded thresholding, it performs roughly equal but more often worse than multi-threaded thresholding. Based on this result, and given the relative complexity of implementation as compared to multi-threading, GPU acceleration for image processing was not considered and integrated further.

Potential scenarios where GPU acceleration may help is on single-core systems like the Raspberry Pi B+, or perhaps for other algorithms which cannot be easily multi-threaded.

[2]Refer to Appendix E.

Table 4.7: Comparison of thresholding times using various methods (CPU/GPU)

Resolution	Processing time (ms/frame)		
	Single-threaded thresholding	Multi-threaded thresholding	GPU thresholding
160x120	2.37	1.89	2.50
320x240	9.16	4.07	5.00
640x480	27.5	12.6	12.5
800x600	43.0	18.9	25.2
1280x720	82.3	31.4	41.1

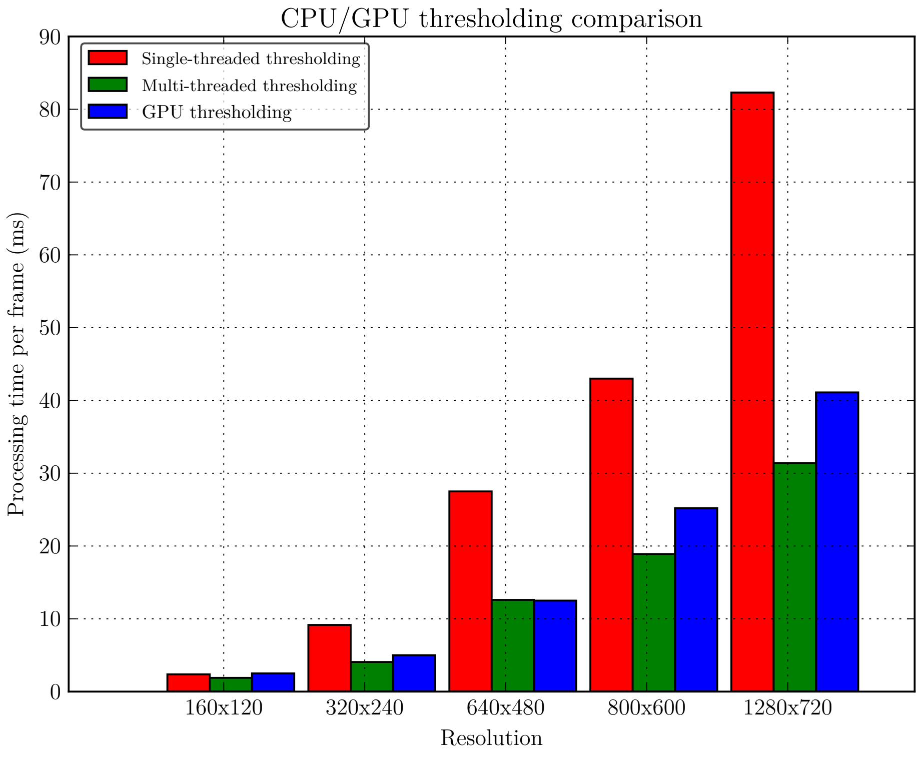


Figure 4.11: Visualisation of thresholding times using various methods (CPU/GPU)

4.2 Video encoding

Using the hardware encoder, testing has proved it capable of encoding images at resolutions up to 800x600 in real-time (30FPS). Resolutions up to 1280x720 can be processed, but only if a decreased frame rate of around 15FPS is accepted. To determine the optimal conversion process outlined in Listing 3.3, a number of variations on the BGR to RGB algorithm were tested. For testing, 500 frames at a resolution of 800x600 were processed and the time taken to process each frame observed. Three trials were run for each variation before being averaged. Variations related to the unrolling factor and preload distance used. For a baseline comparison, these results were compared to both the standard OpenCV implementation and a direct memory copy operation (`memcpy`), which is equivalent to performing no processing before encoding.

Table 4.8: Variations of the BGR to RGB routine tested

Variation	Unroll factor	Preload distance (bytes)	Average processing time/frame (μs)
`memcpy`	N/A	N/A	3254
OpenCV	N/A	N/A	7737
NEON-08-000	1	0	5477
NEON-08-192	1	192	5525
NEON-08-384	1	384	5507
NEON-16-000	2	0	3441
NEON-16-064	2	64	5865
NEON-16-192	2	192	3338
NEON-16-384	**2**	**384**	**3202**
NEON-32-000	4	0	6142
NEON-32-192	4	192	6552
NEON-32-384	4	384	6650

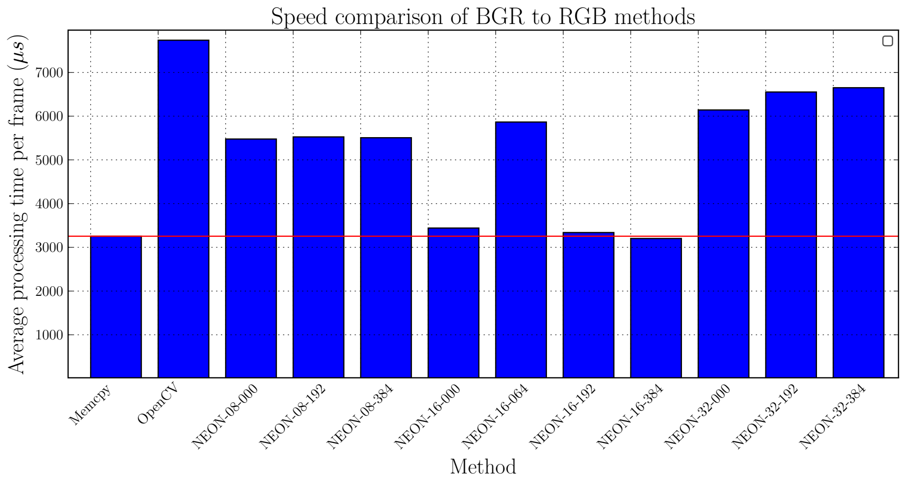


Figure 4.12: A comparison of average processing times per frame (800x600 resolution).

An unroll factor of 1 means that 8 pixels are processed per cycle, while 2 means that 16 bytes are processed, and so on. The preload distances of 0, 192 and 384 bytes were chosen based on being a multiple of the cache line size (64 bytes). The most profound finding from the results presented in Table 4.8 and Figure 4.12 is that the NEON-16-384 routine matches or slightly out-performs the performance of the baseline `memcpy` operation. Furthermore, this routine is around 240% faster than the OpenCV benchmark, which is an excellent result.

Another observation made from Figure 4.12 is that not all variations benefit from the preload instruction, which is particularly evident for the outlier NEON-16-064 trial. This may be explained by the likelihood of the incorrect memory being preloaded causing a *cache miss* [44]. For the NEON-16-064 trial, the preloaded memory is too close to the memory region that is read in that loop. The final observation is that there is clearly no benefit to unrolling to 32 pixels per loop.

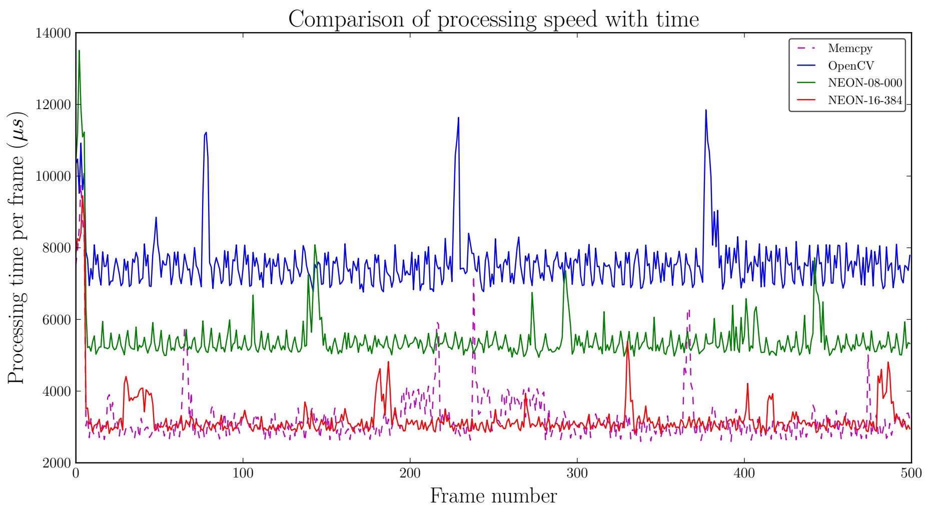


Figure 4.13: A comparison of enqueue times over the 500 frames (800x600 resolution).

Figure 4.13 shows the profile of enqueue time per frame. It indicates that there is always a startup delay, while the processing time remains fairly constant. For the NEON-16-384 algorithm, the standard deviation in processing time is around 868 µs. Periodic spikes in the processing may be explained by the fact that the system is not running a real-time operating system.

4.2.1 Speed benchmarks at various resolutions

Table 4.9: Performance characteristics for hardware H.264 video encoding.

Resolution	CPU (%)	MEM (%)	Processing rate (FPS)	Enqueue speed (µs/frame)	Encode speed (µs/frame)
160x120	6.9	3.1	29.25	183	1746
320x240	12.9	3.4	29.23	722	4643
640x480	33.6	4.3	29.21	2510	11998
800x600	51.2	5.1	29.16	3283	23072
1280x720	73.2	7.0	14.7	9107	66224
1920x1080	33.4	12.0	4.83	13092	68506

Table 4.10: Performance characteristics for hardware JPEG image encoding.

Resolution	CPU (%)	MEM (%)	Processing rate (FPS)	Enqueue speed (µs/frame)	Encode speed (µs/frame)
160x120	9.2	3.0	29.25	160	3140
320x240	14.8	3.3	27.07	662	8884
640x480	37.5	4.0	26.54	2275	29462
800x600	54.3	4.6	22.88	3481	48794
1280x720	68.5	6.0	8.31	6481	131154
1920x1080	52.5	9.9	5.13	13409	240446

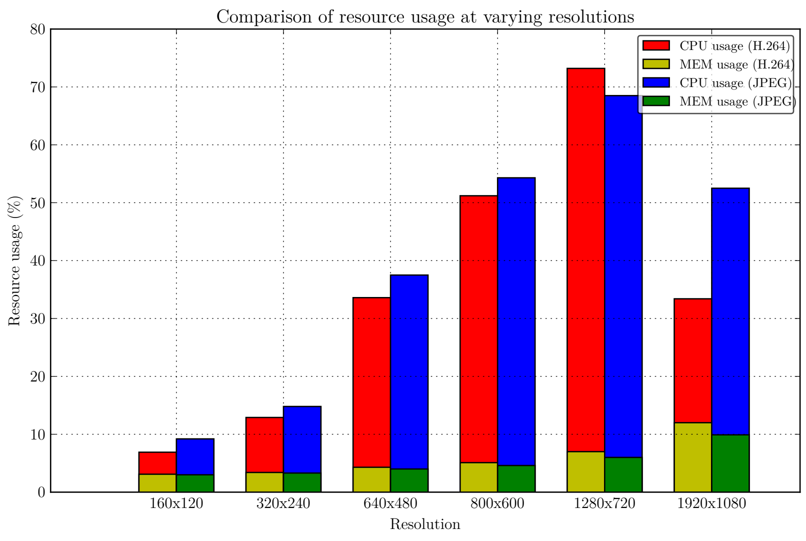


Figure 4.14: Resource use at various resolutions (H.264 columns left, JPEG columns right).

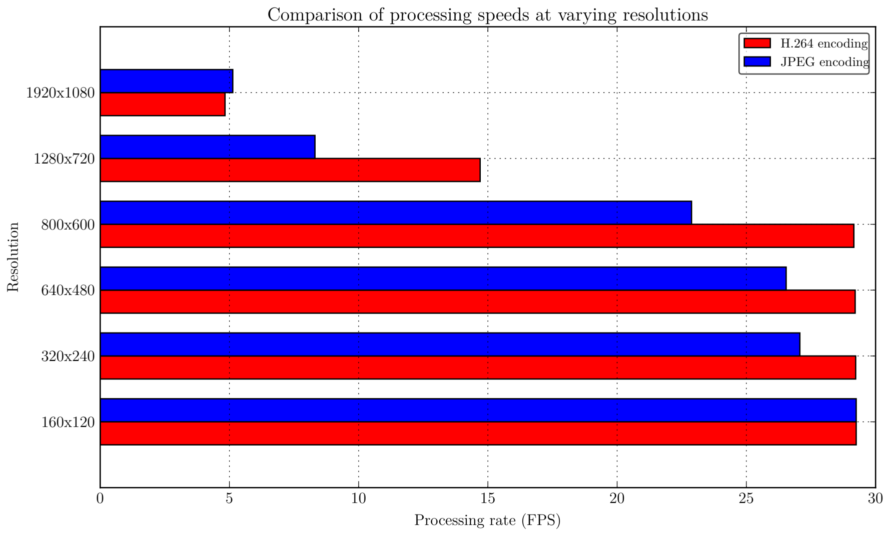


Figure 4.15: Processing speeds at various resolutions (H.264 columns bottom, JPEG columns top).

As indicated in Figure 4.15, the full processing speed of 30FPS is attained for H.264 encoding at resolutions up to 800x600. This steadily drops off as the resolution is increased further. Similarly, Figure 4.14 shows that resource usage increases with increasing resolution, with the exception of 1920x1080. As the processing speed still decreases to 4.8FPS despite the drop in CPU usage, there is likely a memory bandwidth bottleneck preventing full CPU utilisation.

JPEG encoding follows a similar trend, although it is apparent that it performs poorer than the H.264 encoder especially at larger resolutions. While this may seem surprising at first, given that it is less computationally intensive, this is likely the result of file buffering. With each image saved, the data must be flushed to disk before the next image is saved. This is because images are saved as individual files. The action of flushing to disk causes the slowdown, since for H.264 encoding, only one file is written out, meaning that it can leverage file buffering.

4.3 Web interface

Maintaining and extending functionality present in 2014, the web interface was successfully rewritten to use Bootstrap and jQuery. Extending the Thrift interface, the visual flight indicators were incorporated. For added extensibility, a Bootstrap theme provided by Bootswatch [53] was used, meaning that the visual appearance can be changed by a single line of code [3].

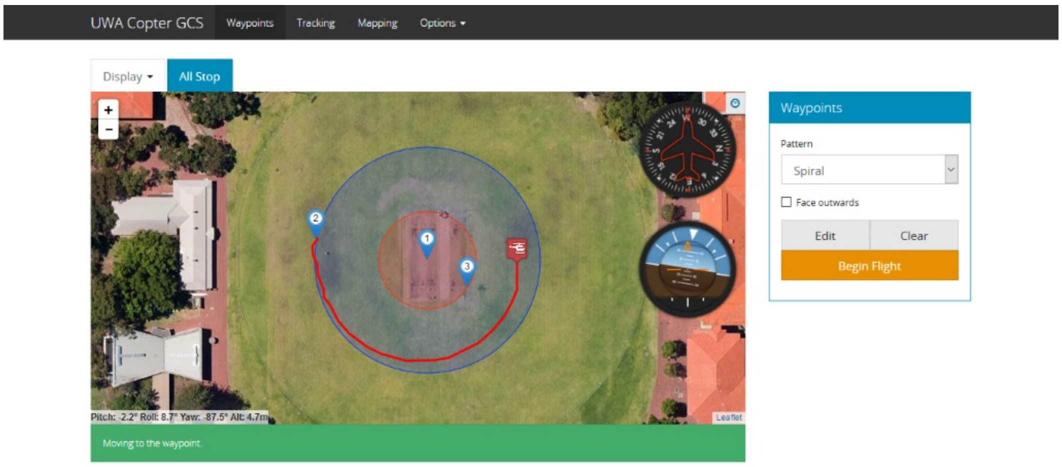


Figure 4.16: The redesigned web interface (large screen display).

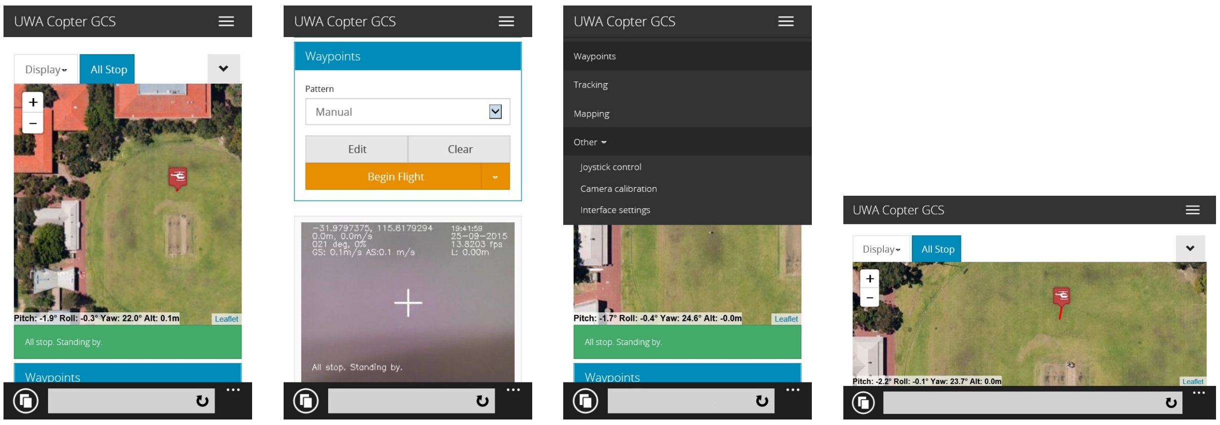
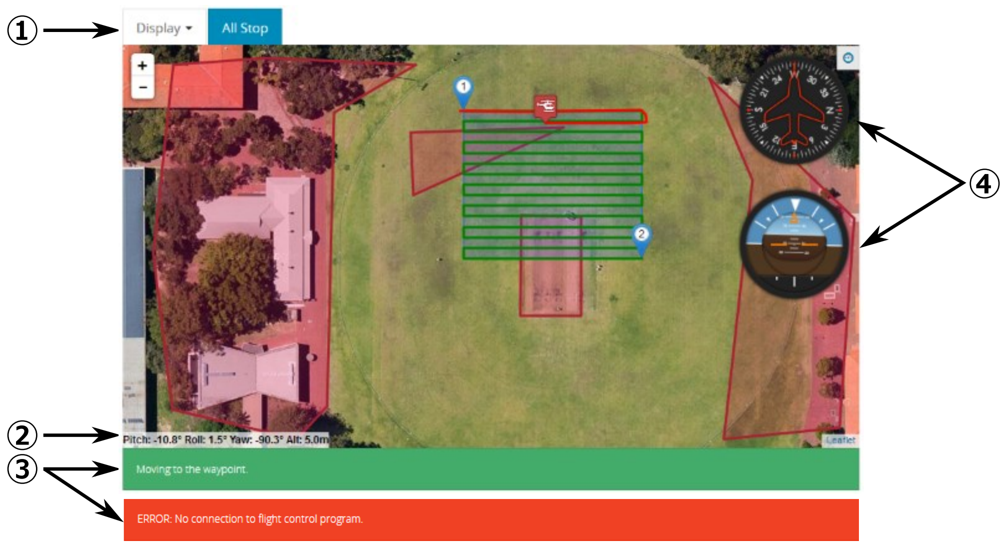


Figure 4.17: The interface as viewed from a phone.

Importantly, and as shown in Figure 4.17, the interface scales well even at small screen sizes. Both the menu and sidebar collapse to make better use of the space available. The visual flight indicators are also hidden due to the lack of space. The map display has also been updated to accommodate additional search patterns, such as the spiral pattern displayed in Figure 4.16. This capability is also extended to the configuration and display of exclusion zones (areas that should be avoided) to complement the collision avoidance system developed by Allen [9]. An example of this is provided in Figure 4.18, which depicts a creeping-line search pattern (green) overlaid with multiple exclusion zones (red).

[3]Refer to Appendix F

4.3.1 Information display



Pitch: -10.8° Roll: 1.5° Yaw: -90.3° Alt: 5.0m

Moving to the waypoint.

ERROR: No connection to flight control program.

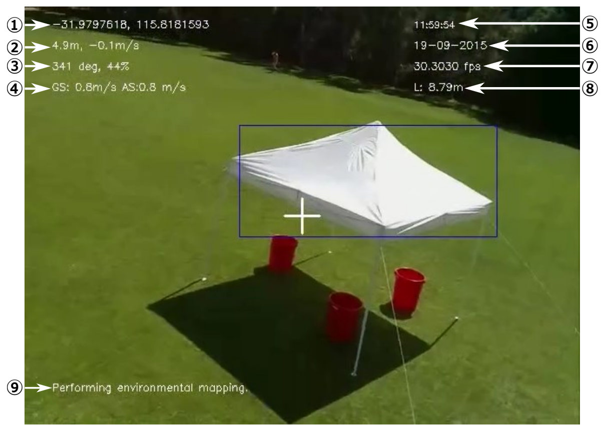
Figure 4.18: Information layout on the interface.

Extra information and visual cues provided to the user. This is depicted in Figure 4.18:

1. The map display can be switched to the camera feed as needed. The ‘All Stop’ button is always visible for quick access to stop the hexacopter at any time.
2. The attitude and altitude of the hexacopter is always displayed.
3. There is a large colour-coded status area. On error, the status bar turns red.
4. Visual indicators give a quick indication of the attitude of the hexacopter. If the screen size is too small, the visual indicators will automatically be hidden from view.

Data review

A number of methods now exist to review data captured from autonomous missions. First, the video stream captures a wealth of information through its heads-up display, in addition to displaying the video feed from the on-board camera. Second, completed missions can be reviewed in the web interface to show where detected objects were located. Finally, the web interface allows the run to be exported in GPX format, for review in other applications such as Google Earth.



−31.9797618, 115.8181593
4.9m, −0.1m/s
341 deg, 44%
GS: 0.8m/s AS:0.8 m/s
11:59:54
19−09−2015
30.3030 fps
L: 8.79m
Performing environmental mapping.

1. GPS coordinates (decimal)
2. Altitude above launch (m); climb rate (m/s)
3. Heading (degrees); throttle percentage
4. Ground/Air speed (m/s)
5. Time
6. Date (DD-MM-YYYY)
7. Image processing rate (frames per second)
8. LIDAR reading (m)

Figure 4.19: The heads-up display on the video stream.

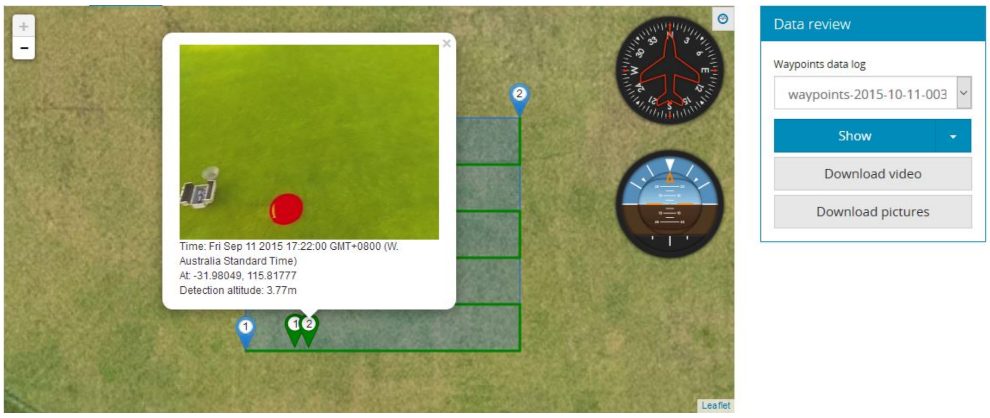


Figure 4.20: Reviewing a search in the web interface.

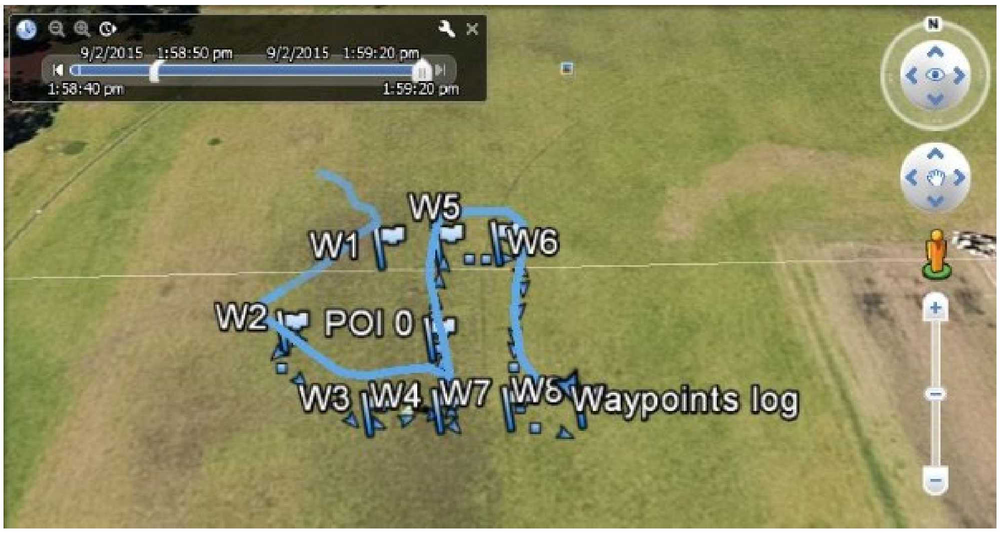


Figure 4.21: Reviewing a search in Google Earth using the exported GPX log.

5. Conclusion

This study highlights the many facets that make up a complete autonomous point of interest navigation system. It has shown how embedded computing power remains the limiting factor to image processing performance. More importantly it has explored ways to make the most out of what was available to be used. In combination with multi-threaded processing, image processing performance was significantly increased to 30FPS at 320x240 pixels or less, and 10–15FPS (algorithm dependent) for 640x480 pixels, representing a 150%–270% increase over 2014 results.

While more complex image processing routines like the histogram of gradients method still remain unfeasible due to processing constraints, multiple improvements to image processing were nonetheless made. Using $Y'C_bC_r$ thresholding proved to be a useful counterpart to HSV thresholding for object detection, being less susceptible to false-positive detection, particularly under darker lighting conditions. Glyph detection was also introduced and shown to be feasible, establishing the framework for more accurate localisation techniques.

Excellent results were also observed in integrating a low-latency, real-time video capture system capable of encoding video at up to 800x600 pixels with as little as a 3 ms or less footprint, achieved both through the use of GPU processing and NEON SIMD.

A scalable, intuitive and extensible web interface was also designed, rounding off the accomplishments made, and establishing a platform for continued research into the future.

5.1 Future work

5.1.1 Image processing

An aspect that was not considered in this report was how the glyph locations could be fused with location data to provide for more accurate positioning. This is covered in part by Downing [10], but the integration of a localisation system that incorporates this data would help to reduce the dependence on GPS positioning as the sole method for positioning. Improving glyph detection at increased distances (currently limited to around 5m at 320x240 pixels for an A3 glyph) would also help to increase the capability of the system.

Another possibility that was not covered was using OpenMAX in combination with OpenGL to capture image data and process it directly on the GPU (the second pipeline in Figure 2.2). Although GPU acceleration for image processing was not helpful in the case presented in this report, using this alternate pipeline, a better outcome may be achieved.

5.1.2 Video capture

While the heads-up display (HUD) is useful for data review, it may also be useful to perform further image processing algorithms offline. However, the HUD can interfere with this. Encoding the HUD as side data (e.g. subtitles) instead would allow it to be displayed when needed, whilst also allowing for offline processing.

5.1.3 Web interface

Including more interactive visualisation tools on the web interface is a potential area of improvement. With the extensibility and wide availability of JavaScript libraries, tools such as vis.js, three.js and D3.js can all be used to create visualisations such as 3D paths and plots, and even animations. Increasing the number of visualisations would help to make it easier to analyse the data gathered.

References

[1] Maxwell Air Force Base, “Unmanned aerial vehicles: Implications for military operations,” *Occasional Paper*, 2000.

[2] M. A. Goodrich, B. S. Morse, C. Engh, J. L. Cooper, and J. A. Adams, “Towards using unmanned aerial vehicles (uavs) in wilderness search and rescue: Lessons from field trials,” *Interaction Studies*, vol. 10, no. 3, pp. 453–478, 2009.

[3] A. Raptopoulos, D. Damm, P. Santana, M. Ling, and I. Baruchin, “Transportation using network of unmanned aerial vehicles,” Jan. 30 2014. US Patent App. 13/890,165.

[4] Parrot SA., “AR.Drone 2.0.” `http://ardrone2.parrot.com/`, 2015.

[5] senseFly, “senseFly: eBee.” `https://www.sensefly.com/drones/ebee.html`, 2015.

[6] SZ DJI Technology Co., Ltd., “Phantom 2 Vision.” `http://www.dji.com/product/phantom-2-vision`, 2015.

[7] The University of Western Australia, “The farmer takes a drone.” `http://www.news.uwa.edu.au/201504147483/research/farmer-takes-drone`, 2015.

[8] J. Tan. Undergraduate Thesis, University of Western Australia, 2015. `https://github.com/jtanx`.

[9] R. Allen. Undergraduate Thesis, University of Western Australia, 2015. `https://github.com/20739367`.

[10] B. Downing. Undergraduate Thesis, University of Western Australia, 2015. `https://github.com/brettrd`.

[11] M. Mohanty. Undergraduate Thesis, University of Western Australia, 2015. `https://github.com/manishmohanty09`.

[12] M. Baxter, “Autonomous hexacopter software design.” Undergraduate Thesis, University of Western Australia, 2014. `http://robotics.ee.uwa.edu.au/theses/2014-Hexacopter-GPS-Baxter.pdf`.

[13] R. O’Connor, “Developing a multicopter UAV platform to carry out research into autonomous behaviours, using onboard image processing techniques.” Undergraduate Thesis, University of Western Australia, 2013. `http://robotics.ee.uwa.edu.au/theses/2013-Multirotor-OConnor.pdf`.

[14] C. Venables, “Multirotor unmanned aerial vehicle autonomous operation in an industrial environment using onboard image processing.” Undergraduate Thesis, University of Western Australia, 2013. `http://robotics.ee.uwa.edu.au/theses/2013-Multirotor-Venables.pdf`.

[15] M. Cloete. Undergraduate Thesis, University of Western Australia, 2014.

[16] A. Mazur, “Autonomous operation and control of a multirotor unmanned aerial vehicle through 4G LTE using onboard GPS and image processing.” Undergraduate Thesis, University of Western Australia, 2014. `http://robotics.ee.uwa.edu.au/theses/2014-Hexacopter-Interface-Mazur.pdf`.

[17] O. Targargh, “Autonomous navigation of unmanned aerial vehicles.” Undergraduate Thesis, University of Western Australia, 2014. `http://robotics.ee.uwa.edu.au/theses/2014-Hexacopter-Navigation-Targhagh.pdf`.

[18] Swift Navigation Inc., “Piksi.” `http://www.swiftnav.com/piksi.html`, 2014.

[19] G. Bradski, “OpenCV,” *Dr. Dobb’s Journal of Software Tools*, 2000.

[20] P. Sebastian, Y. V. Voon, and R. Comley, “The effect of colour space on tracking robustness,” in *Industrial Electronics and Applications, 2008. ICIEA 2008. 3rd IEEE Conference on*, pp. 2512–2516, June 2008.

[21] S. Boubou, A. Kouno, and E. Suzuki, “Implementing camshift on a mobile robot for person tracking and pursuit,” in *Data Mining Workshops (ICDMW), 2011 IEEE 11th International Conference on*, pp. 682–688, Dec 2011.

[22] J. Tan, B. Downing, R. Allen, and M. Mohanty, “picopterx - hexacopter automation software.” https://github.com/jtanx/picopterx, 2015.

[23] P. Kumar, K. Sengupta, and A. Lee, “A comparative study of different color spaces for foreground and shadow detection for traffic monitoring system,” in *Intelligent Transportation Systems, 2002. Proceedings. The IEEE 5th International Conference on*, pp. 100–105, 2002.

[24] OpenCV development team, “Image moments.” http://docs.opencv.org/doc/tutorials/imgproc/shapedescriptors/moments/moments.html, 2014.

[25] G. R. Bradski, “Computer vision face tracking for use in a perceptual user interface,” 1998.

[26] A. Jain, Y. Zhong, and S. Lakshmanan, “Object matching using deformable templates,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 18, pp. 267–278, Mar 1996.

[27] J. Canny, “A computational approach to edge detection,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-8, pp. 679–698, Nov 1986.

[28] N. Dalal and B. Triggs, “Histograms of oriented gradients for human detection,” in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 1, pp. 886–893 vol. 1, June 2005.

[29] Raspberry Pi Foundation, “Raspberry Pi - FAQs.” https://www.raspberrypi.org/help/faqs/, 2015.

[30] Raspberry Pi Foundation, “Raspberry Pi 2 Model B.” https://www.raspberrypi.org/products/raspberry-pi-2-model-b/, 2015.

[31] “Raspberry Pi VideoCore APIs.” http://elinux.org/Raspberry_Pi_VideoCore_APIs, 2013.

[32] R. Tsuchiyama, T. Nakamura, T. Iizuka, A. Asahara, and S. Miki, *The OpenCL Programming Book: Parallel Programming for MultiCore CPU and GPU*. 2010.

[33] N. Singhal, I. K. Park, and S. Cho, “Implementation and optimization of image processing algorithms on handheld gpu,” in *Image Processing (ICIP), 2010 17th IEEE International Conference on*, pp. 4481–4484, Sept 2010.

[34] OpenCV development team, “GPU module introduction.” http://docs.opencv.org/2.4.9/modules/gpu/doc/introduction.html, 2014.

[35] J. Newmarch, “Programming and using Linux sound — Chapter 13 - OpenMAX/OpenSL.” http://jan.newmarch.name/LinuxSound/Sampled/OpenMAX/, 2013.

[36] Raspberry Pi Foundation, “OMX.broadcom.video_encode.” http://home.nouwen.name/RaspberryPi/documentation/ilcomponents/video_encode.html, 2014.

[37] OpenCV development team, “cvtColor documentation.” http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html#cvtcolor, 2014.

[38] Khronos Group, “OpenMAX™ integration layer application programming interface specification.” https://www.khronos.org/registry/omxil/specs/OpenMAX_IL_1_1_2_Specification.pdf, 2008.

[39] L. Meier, “MAVLink micro air vehicle communication protocol.” http://qgroundcontrol.org/mavlink/start, 2015.

[40] OpenCV development team, “Miscellaneous image transformations.” http://docs.opencv.org/modules/imgproc/doc/miscellaneous_transformations.html, 2014.

[41] OpenCV development team, “Geometric image transformations.” http://docs.opencv.org/modules/imgproc/doc/geometric_transformations.html, 2014.

[42] OpenCV development team, “Object detection.” http://docs.opencv.org/modules/imgproc/doc/object_detection.html, 2014.

[43] Broadcom Corporation, “OMX.broadcom.video_encode.” http://home.nouwen.name/RaspberryPi/documentation/ilcomponents/video_encode.html, 2013.

[44] ARM Limited, “Cortex-A Series Programmer’s Guide.” http://infocenter.arm.com/help/topic/com.arm.doc.den0013a/, 2014.

[45] The FFmpeg developers, “Libavformat.” https://ffmpeg.org/libavformat.html, 2015.

[46] S. Matton, “jQuery flight indicators plugin.” http://sebmatton.github.io/flightindicators/, 2014.

[47] ECMA International, “ECMA-404 – The JSON Data Interchange Format.” http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf, 2013.

[48] Internet Engineering Task Force, “RFC 7159 – The JavaScript Object Notation (JSON) Data Interchange Format (proposed standard).” https://tools.ietf.org/html/rfc7159, 2014.

[49] M. Yip and THL A29 Limited, “RapidJSON.” http://rapidjson.org/, 2015.

[50] “GPX: The graphics exchange format.” http://www.topografix.com/gpx.asp, 2004.

[51] M. Otto, J. Thornton, C. Rebert, J. Thilo, H. Fenkart, P. H. Lauke, and XhmikosR, “Bootstrap.” http://getbootstrap.com/, 2015.

[52] The jQuery Foundation, “jQuery.” https://jquery.com/, 2015.

[53] T. Park, “Bootswatch: Yeti.” http://bootswatch.com/yeti/, 2015.

[54] 3D Robotics, “3DR Pixhawk.” http://3drobotics.com/pixhawk/, 2015.

[55] DIY Drones team, “ArduPilot.” http://ardupilot.com/, 2015.

[56] C. Anderson, “CanberraUAV (ArduPlane) wins Outback Challenge!.” http://diydrones.com/profiles/blogs/canberrauav-arduplane-wins-outback-challenge, 2012.

[57] G. Mortimer, “Congratulations Canberra UAV, stunning Outback Challenge win.” http://diydrones.com/profiles/blogs/congratulations-canberra-uav-stunning-outback-challenge-win, 2014.

[58] Ardupilot community, “Simulation.” http://dev.ardupilot.com/wiki/simulation-2/, 2015.

[59] M. Oborne, “Mission planner home.” http://planner.ardupilot.com/, 2015.

[60] 3D Robotics, “Tower (DroidPlanner 3).” https://play.google.com/store/apps/details?id=org.droidplanner.android, 2015.

[61] A. Tridgell, R. Mackay, P. Hickey, *et al.*, "ArduPilot source code." https://github.com/diydrones/ardupilot/, 2015.

[62] OpenPilot Foundation, "OpenPilot source code." https://github.com/openpilot/OpenPilot, 2015.

[63] F. Ruess, G. Hattenberger, C. D. Wagter, *et al.*, "Paparazzi source code." https://github.com/paparazzi/paparazzi, 2015.

[64] MultiWii Authors, "MultiWii source code." https://github.com/multiwii/multiwii-firmware, 2015.

[65] NoLimitLab, "Setting up Futaba 14SG and R7008SB to work with the pixhawk." https://nolimitlab.wordpress.com/resources/setting-up-futaba-14sg-and-r7008sb-to-work-with-the-pixhawk/, 2015.

[66] ArduPilot community, "Communicating with Raspberry Pi via MAVLink." http://dev.ardupilot.com/wiki/companion-computers/raspberry-pi-via-mavlink/, 2015.

[67] koala ok, "Dji wheels tripod landing skids gear for f450 f550 sk480 aircraft qudcopter ok." http://www.ebay.com.au/itm/DJI-Wheels-Tripod-Landing-Skids-Gear-for-F450-F550-SK480-Aircraft-Qudcopter-OK-/251811175817, 2015.

[68] The MathWorks, Inc., "Pid controller tuning." http://au.mathworks.com/help/slcontrol/automatic-pid-tuning.html, 2015.

[69] Sufendi, B. Trilaksono, S. Nasution, and E. Purwanto, "Design and implementation of hardware-in-the-loop-simulation for uav using pid control method," in *Instrumentation, Communications, Information Technology, and Biomedical Engineering (ICICI-BME), 2013 3rd International Conference on*, pp. 124–130, Nov 2013.

[70] Open Source Robotics Foundation, "ROS - introduction." http://wiki.ros.org/ROS/Introduction, 2015.

[71] Open Source Robotics Foundation, "Gazebo." http://gazebosim.org/, 2015.

[72] J. Meyer, A. Sendobry, S. Kohlbrecher, U. Klingauf, and O. von Stryk, "Comprehensive simulation of quadrotor uavs using ros and gazebo," in *Simulation, Modeling, and Programming for Autonomous Robots* (I. Noda, N. Ando, D. Brugali, and J. Kuffner, eds.), vol. 7628 of *Lecture Notes in Computer Science*, pp. 400–411, Springer Berlin Heidelberg, 2012.

Appendix A. ArduPilot proposal [8–11]

ENGINEERING PROPOSAL

Hexacopter flight controller upgrade

24 April 2014

This document outlines the proposal for replacing and upgrading the flight controller used on the hexacopter to a 3DR Pixhawk [54] running the open-source flight control software ArduPilot [55]. This marks a significant departure from the current 'black-box' NAZA flight controller. The primary motivation for replacement is based on the strong concern that the current system is not sustainable: The current system restricts any automation improvements, provides no value to the wider community, and limits opportunities for future students.

The key benefits of switching to ArduPilot are:

- Tighter integration and feedback of the hexacopter state with the automation software;
- Avoiding sensor duplication
- Access to existing ground control and simulation software
- Well documented communication interfaces
- Wide community support
- Already being used in the research field.

ArduPilot was used in the winning entries in the 2012 and 2014 Canberra outback challenges [56,57] and is the platform of choice in the forum DIYdrones.

The Pixhawk will use the MAVLink communications protocol to communicate with the Raspberry Pi. It is acknowledged that there are risks associated in the switchover, which will be discussed and addressed in this document.

A.1 Scope

The scope of this project involves:

1. Removing the NAZA-M V1 flight controller, power monitor (PMU) and GPS/Compass module
2. Removing the autonomous/manual switching circuit
3. Installing the 3DR Pixhawk, uBlox Neo-7m GPS/Compass module, power monitor, LED and wireless telemetry kit
4. Reworking the wiring harness between the flight controller and the Raspberry Pi
5. Reconfiguring the automation software to interface with the Pixhawk

A.2 Feature comparison

Feature	**Old (NAZA) system**	**Proposed (ArduPilot) system**
Failsafe (return to base) mode	Yes.	Yes.
Low-power failsafe	Yes, voltage monitor.	Yes, voltage and current monitor.
Auto/Manual switch	Yes (via switching circuit).	Yes, built-in to Pixhawk.
Buzzer	Yes, but not controllable. Pi has separate buzzer.	Yes, controllable. Pi can also have separate buzzer as before.
Fully autonomous (takeoff and waypoint travel)	No.	Yes. Can be programmed for fully automated missions.
IMU (pitch, roll, heading, altitude) and GPS data access	Limited (undocumented, unsupported, deliberately obfuscated interface). Current break-in cable to the GPS is unreliable and may cause damage to the GPS.	Yes. Well documented communications protocol (MAVLink).
Radio telemetry	Possible (not currently installed).	Yes.
Pan/Tilt gimbal for camera	Limited (tilt control only from the Pi).	Yes.
Simulation environment	No.	Yes, both hardware and software (HITL/SITL) with physics simulation [58]
Existing software tools	Flight assistant software performs calibration only.	1. Mission planner - handles system configuration, calibration and programming for autonomous flight [59]. 2. Tower - an Android mobile app for controlling the hexacopter [60]
Modifiable	No, closed system.	Yes. It is possible to modify all aspects of the system (open-source), but not recommended for our application without thorough testing.
Autonomous control from the Raspberry Pi	Yes, via the switching circuit and PWM output.	Yes, digital control via MAVLink (a communication protocol).
Built-in support for the PIKSI GPS	No.	Yes, but it is unclear how mature this support is at this stage.

A.2.1 Further discussion of features

Failsafe

The ArduPilot configuration has a fail-safe mode equivalent to the NAZA, activated in exactly the same way. Once configured, the flight computer will return-to-land on loss of hand-held transmitter signal. The failsafe can be configured as situations demand.

Modification of the flight controller

While it is possible to modify ArduPilot itself, changes to the low-level controls is not recommended nor required. This option will remain open for future work but will require extensive testing. The ArduPilot's default flight modes are sufficient for the scope of our work.

Camera gimbal

The pan servo cannot be controlled by the Pi due to a limitation of the NAZA. This is an out-of-the-box feature on the ArduPilot.

Sensor availability

The Raspberry Pi cannot access pan or tilt data as collected by the NAZA. Any pan/tilt data received will be different to the NAZA, causing the Pi to fight the NAZA. MAVLink provides access to the GPS and IMU data directly from the flight computer, which allows both the QStarz GPS and the XSens IMU to be removed. MAVLink also reports remaining battery capacity, which will allow us to write software that issues warnings and aborts missions intelligently.

Telemetry downlink

A new redundant data channel will permit monitoring outside range of WiFi, or in the case of server failure, which has caused near misses in the past.

The ArduPilot software supports fully autonomous missions, with instruction sent over this link. However, we expect to maintain the availability of manual override at all times and will configure the fail-safe to reflect this.

Control debugging

Using an external monitor interferes with the current control interface between the NAZA and Raspberry Pi. The proposed modification will save a great deal of time and effort in working around this problem.

The PWM signal from ServoBlaster must be calibrated with the NAZA flight control software. With no hardware PWM outputs, exact calibration with the NAZA flight computer is difficult. This is not an issue using MAVLink as it is a digital protocol.

Additional sensor compatibility

The ArduPilot community has added support for a wide variety of sensors over a wide variety of interfaces. This support includes integration into the navigation Kalman filter where appropriate.

Forward compatibility

DJI has released two new flight computers since the NAZA-M V1. The NAZA-M V1 is still supported, but the latest firmware updates require a CAN bus expander valued at A$80 to make use of any new features. Very few of these new features address our current concerns.

The Pixhawk was designed to replace the APM2.6 as the firmware files became too large for the 8-bit processor it carried. It was designed to be largely future-proof, boasting a generous amount of ram, flash and CPU power. Even if the ArduPilot project out-grows the Pixhawk, it is likely to have continued support and backported features, such as the APM2.6 owners currently enjoy.

Portability to future platforms

The ArduPilot project began with fixed wing aircraft and has since been extended to various configurations of multi-rotor, Helicopters, Rovers and Boats. Any work done building functionality against ArduCopter (the multirotor firmware) is immediately relevant to almost any other autonomous platform.

A.2.2 Alternative solutions

Upgrade the current flight controller

As a commercial system, there is *no* official interface that allows for automation or access to any of its sensors. In the past, this project achieved automation by using a remotely controlled switching circuit and the Raspberry Pi to emulate the PWM signals of joystick commands (aileron, elevator and rudder controls). This approach also requires the Raspberry Pi to have its own set of sensors, where progress would be bogged down in trying to interface with the low-level components and having to re-invent, test and tune basic control functions.

Although there is some limited upgrade capability (upgrading to the PMU V2), which would provide access to the IMU data, this is only possible through an unofficial, undocumented, reverse-engineered system. Even with this extra information, this does not address the other issues outlined in terms of community support, the camera gimbal, telemetry and the existing level of software available for the ArduPilot platform.

APM 2.6

The APM2.6 is a known-good ArduPilot board. The hardware is cheaper than the Pixhawk, but it is version capped as of last year. This board is not future-proof and will not benefit from new features in ArduPilot.

MultiWii

MultiWii is another general purpose open-source flight controller. Originally developed to use the gyroscope and accelerometer system from the Nintendo Wii game controller, it has developed into a general flight controller that operates on the Arduino platform. Community support for this platform is less than that for the ArduPilot. The platform also has lesser specs than the out-moded APM2.6.

Paparazzi

Paparazzi is another open-source flight controller system, which has been developed since 2003. It is an older project that appears to be extremely versatile, but not very beginner friendly. Their focus appears to be on modularity and wide applications. The community encourages significant modification of the core flight control software, which is likely to remain out of scope for this platform. Many of the demonstrated applications are single-purpose scientific flights.

OpenPilot

The OpenPilot appears to have a strong community backing for FPV and acrobatics, but the ArduPilot appears to have a sample of simpler flight modes and a stronger autonomous focus, and a stronger Australian community. The OpenPilot hardware does support ArduCopter, but does not have the extensive feature set of the Pixhawk, nor its sensor redundancy.

A.3 Risks

A.3.1 Risk matrix

	Very unlikely	Unlikely	Possible	Likely	Very Likely
Negligible	Very low	Very low	Very low	Low	Low
Minor	Very low	Very low	Low	Low	Medium
Moderate	Low	Low	Medium	Medium	High
Significant	Medium	Medium	High	High	Very high
Severe	Medium	High	High	Very high	Very high

A.3.2 Risk register

No.	Description	Probability	Impact	Rating	Mitigation	Contingency
1.1	The UAV community moves to a different platform	Unlikely	Moderate	Low	Many people, including developers have invested in ArduPilot compatible hardware which represents a small community lock-in	Find alternative, compatible flight controller software
1.2	Unforeseen hardware incompatibilities	Possible	Moderate	Medium	Conduct prior investigation and identify any potential hardware issues	Consult community for potential fix. OR replace incompatible hardware
2.2	Late deliveries or manufacturing faults	Possible	Significant	High	Request priority delivery, AND Much of the integration testing can be performed with an existing ArduPilot owned personally by a team member.	Source parts elsewhere

A.3.3 Further risk comments

Risk 1.1 - The UAV community moves to a different platform The developer forums are extremely active, and new features are being added regularly. Of the open-source systems discussed, ArduPilot is the most actively developed [61–64]. If an open-source flight controller is to be used, it makes sense to use the most actively developed platform to avoid project stagnation. Should ArduPilot development cease, there are a number of compelling platforms with a great deal of support - OpenPilot, MultiWii and Paparazzi.

Consequences
External development of the ArduPilot software would halt, but it would still be available in its current state. Being an open source project, many of the alternative drone software projects already support the Pixhawk hardware, just as ArduPilot has been ported to hardware from other projects.

Risk 1.2 - Unforeseen hardware incompatibilities The Hexacopter has 30A Opto ESCs and a Futaba R7008SB receiver [65]. Support for these components with the ArduPilot has been demonstrated in other UAV projects, so hardware incompatibility is unlikely. The Raspberry Pi has been demonstrated to work with ArduPilot [66].

Consequences
Hardware incompatibilities would result in additional debugging work and lost time. It could also result in the replacement of other hardware, or the purchase of additional interfacing hardware.

Controls
Preliminary research into each subsystem of current hexacopter.

Risk 2.2 - Late deliveries or manufacturing faults Online purchases carry the risk of components not arriving on time. It may also make it more difficult to return components, should they be faulty.

Consequences
Late deliveries would lead to lost time, potentially putting the project on hold or incurring additional costs.

A.4 Costs

Component	Cost ($AUD)	Source	Comments
Pixhawk	$256	3DR store	
UBlox GPS + Honeywell Compass	$41	eBay	
Telemetry	$32	eBay	Not strictly necessary, but allows in-flight full recovery after a complete server crash
Indicator LED	$12	eBay	

Total cost: A$341 (24/04/2015)

A.5 Conclusion

We fully expect the components listed above to serve as a drop-in replacement for the NAZA and other ancillary components.

The removal of the NAZA flight computer and the inclusion of the Pixhawk radically simplifies the wiring harness, removing many of the components and links that the 2014 team complained about being ‘unreliable’ and the cause of multiple crashes.

The move to the Pixhawk will greatly increase the maintainability of the platform and significantly widen the plausible scope of any future work with this hardware.

Appendix B. Platform changes made in 2015

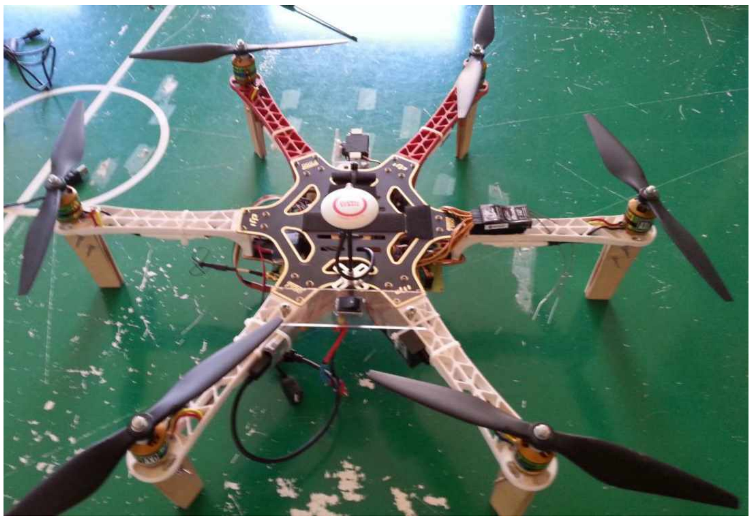


Figure B.1: The hexacopter as it was in 2014.

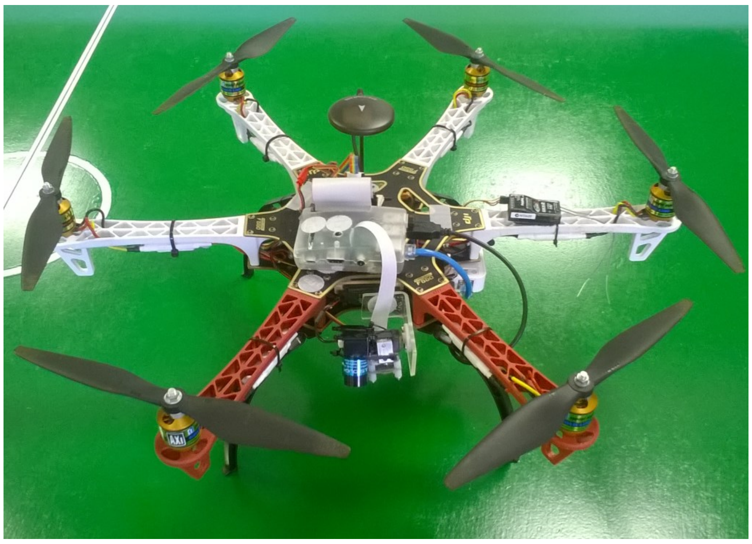


Figure B.2: The hexacopter as it is now in 2015.

The mounting arrangements on the hexacopter mean that there is no room to place the battery on the underside without raising the height of the supports. In the past, this was solved by using extenders made of unsealed MDF. As an ad-hoc solution, these were problematic. Being unsealed, they were prone to damage, especially if placed on a damp surface. Furthermore, they increased the torque action on the legs, such that in a crash, the arms were more likely to break.

This was resolved by replacing these with a commercially available landing gear that were properly designed to fit the DJI F550 frame. At a total cost of A$6.74 [67], this is a highly practical and cost effective solution. Not only was it a ‘bolt on’ solution, it is now more likely that in a crash, the landing gears will break instead of the arms. This is desired, because unlike the arms, it is a non-structural component. Furthermore, the increased height provided by the new landing gear allows for mounting extra equipment on the underside, should this be necessary.

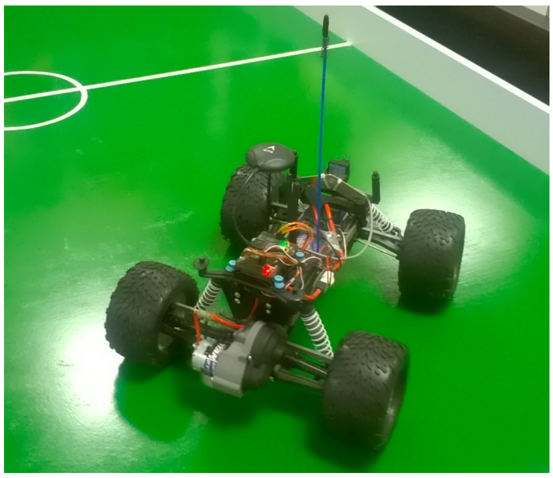


Figure B.3: The rover used for object tracking.

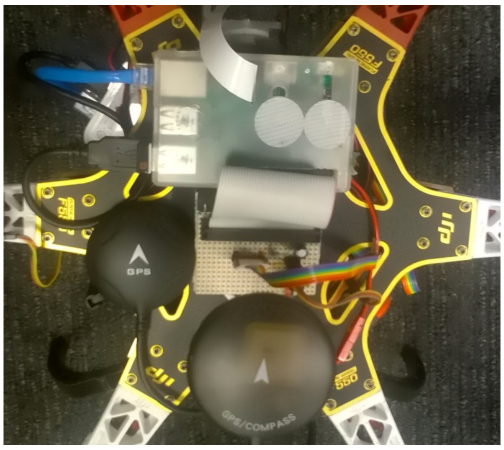


Figure B.4: The hexacopter using a dual-GPS configuration.

The remote-controlled rover shown in Figure B.3 is used for testing object tracking. It was outfitted this year with a similar automation system to the hexacopter, using the cheaper APM 2 controller instead of the Pixhawk. This makes it capable of performing autonomous waypoint navigation (entered directly into the flight controller using Mission Planner) in addition to manual remote control.

However, observing GPS glitching with the hexacopter, a late modification to the platform was a dual-GPS system as shown in Figure B.4, where the second GPS was borrowed from the rover. The Pixhawk is able to use both GPSes to reduce the effect that glitching can have on the system. With the upgrade to ArduCopter 3.3, the dual-GPS data may also be incorporated into the EKF system.

B.1 Waypoint navigation characteristics

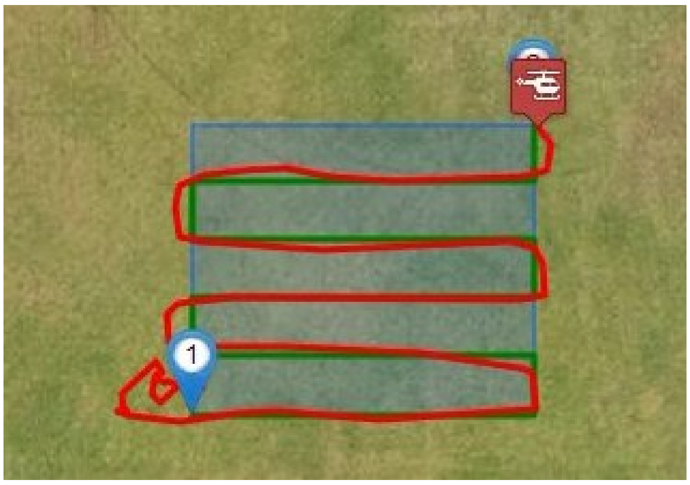


Figure B.5: Review of a creeping-line (lawnmower) search pattern (3m spacing).

The upgrade to the Pixhawk has dramatically increased the navigation capabilities of the system to follow set waypoints closely. Figure B.5 shows the result of a test run using the creeping line search pattern. The actual path closely matches the specified path, with the maximum deviation being 2m. For comparison, the system in 2014 had a maximum deviation of 5.66m [12].

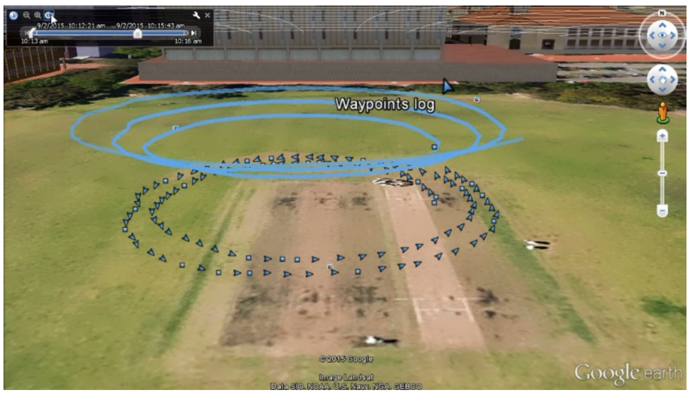


Figure B.6: Review of a 3D spiral search pattern in Google Earth.

The Pixhawk also made it possible to perform 3D search patterns, as shown in Figure B.6. Altitude control was not implemented in 2014.

Appendix C. Simulation and using the simulator

C.1 Background

A key issue identified in 2014 was the lack of simulation software and tools to be able to effectively model the current platform [17]. Without simulation, efforts to tune the control systems such as the waypoint navigation and object tracking systems were limited to physical field testing. This was extremely time consuming and counter-productive, as there was no way to rapidly tune any of the control parameters required. Although this is not directly related to POI identification, being able to rapidly simulate the system would greatly benefit the speed at which changes to the POI system can be made. Conversely, it would also make it easier to re-tune the system if changes are made to the POI system (especially with regards to tuning object tracking capabilities).

C.1.1 MATLAB simulations

MATLAB has the ability to tune PID control systems [68] with relative ease. Once a suitable model that describes the system is determined, it is then straightforward to use MATLAB to tune the system. In [69], a method is explored to provide a Hardware in the Loop simulation using the ArduPilot platform, the X-Plane simulator and MATLAB/Simulink. The Ziegler-Nichols method was used to tune the PID control system, achieving control will less than 10% overshoot and a settling time of less than 20 seconds [69]. This method would require adapting a similar model for the hexacopter platform. However, this remains an entirely feasible option for simulation.

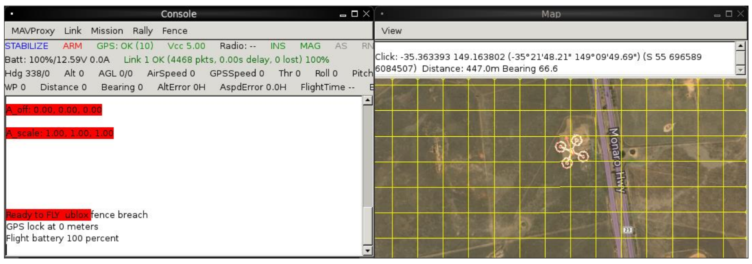
C.1.2 ROS and Gazebo

Robot Operating System (ROS) is an open-source meta-operating system that intends to make it easier to develop automation systems [70]. Gazebo is a complete, open-source software simulation suite that can be interfaced through ROS [71]. What makes this combination attractive is the availability of a comprehensive quadcopter simulation package, which models the physics of the system well in addition to providing a visual representation [72]. As before, this would require adapting to fit the hexacopter platform. Additionally, using ROS would require a complete overhaul of the code currently used, which is undesirable.

C.1.3 ArduPilot – SITL and HITL

ArduPilot comes with two simulation systems - 'Software in the Loop' (SITL) and 'Hardware in the Loop' (HITL) [58]. This is a complete simulation system that accounts for the physics of the system. While SITL is a completely software based, HITL simulation runs the simulation on the hardware, but disables the output to the motors. Note that this is a separate HITL implementation to that described in Section C.1.1. As with Gazebo, visual representation is also provided.

With the transition to the Pixhawk platform[1], it is logically sound to consider this simulation suite first, given that it has been specifically designed to work with the platform in mind.



```
STABILIZE   ARM   GPS: OK (10)   Vcc 5.00   Radio: --   INS   MAG   AS   RN
Batt: 100%/12.59V 0.0A   Link 1 OK (4468 pkts, 0.00s delay, 0 lost) 100%
Hdg 338/0   Alt 0   AGL 0/0   AirSpeed 0   GPSSpeed 0   Thr 0   Roll 0   Pitch
WP 0   Distance 0   Bearing 0   AltError 0H   AspdError 0.0H   FlightTime --   E

A_off: 0.00, 0.00, 0.00

A_scale: 1.00, 1.00, 1.00

Ready to FLY  ublox fence breach
GPS lock at 0 meters
Flight battery 100 percent
```

```
Click: -35.363393 149.163802 (-35°21'48.21" 149°09'49.69") (S 55 696589
6084507)  Distance: 447.0m Bearing 66.6
```

Figure C.1: The ArduPilot SITL simulating a quadcopter.

[1]Refer to Appendix A and Section 3

C.2 Results

Simulation was performed using the ArduPilot provided SITL simulation system. Use of the SITL simulator closely matches the physical representation, and the near-complete emulation of the system means that it can be rapidly tested to see how the system behaves to specific commands. This was invaluable for testing the new spiral search pattern, ensuring that it worked as desired before it was used on the physical platform. Apart from the MAVLink connection method, no extra code was needed to make it work in the simulator – what is run in the simulator is the same code that runs in reality.

C.2.1 Simulation requirements

An important consideration that must not be overlooked is code portability. A key limitation of the 2014 system was that the code worked *only* on the Raspberry Pi. To test new code, it had to be compiled in-situ, which is an extremely tedious and slow process.

By contrast, the current code base was made with portability in mind, and can be successfully compiled on any computer running Linux. Any component specific to the Raspberry Pi such as the GPIO library is emulated with a thin wrapper. This allows the code to be compiled on a desktop computer which can then run the simulation with ease.

C.2.2 Simulation limitations

The simulator emulates all components related to the flight controller well. However, it does not simulate components external to the flight controller, including the camera, LIDAR and gimbal. However, as OpenCV was used, any camera can be connected to the device for testing. Full emulation of these components would require the integration of a 3D emulated environment, which is beyond the scope of this study.

C.3 Simulator usage instructions

Ensure that the main server software (“`picopterx`”) is compilable on the simulation environment. It has been confirmed to compile on Ubuntu 14.04 (“Trusty”) and Debian 8 (“Jessie”). A basic understanding of working in a Linux shell environment and Git is required.

1. Clone the Git repository: `https://github.com/diydrones/ardupilot` to your home folder (e.g. `/home/pi`).

2. Edit your `~/.bashrc` file by appending the following line:

```
export PATH=$PATH:$HOME/ardupilot/Tools/autotest
alias ardusim="$HOME/ardupilot/ArduCopter/ArduCopter.elf --home
    -31.98,115.818,10,0 --model quad"
```

   Restart your terminal (or re-source `.bashrc`) for the changes to take effect.

3. Change directory into the ArduCopter folder and checkout the version that matches that running on the Pixhawk. At the time of writing, this is ArduCopter-3.3:

```
cd ~/ardupilot/ArduCopter
git checkout Copter-3.3
```

4. Still in the ArduCopter folder, compile the simulator with:

```
sim_vehicle.sh -w
```

You will require at least g++ and Make for this to work. This step may take a while. It will finish when it displays GPS Lock or when it throws an error about Numpy not being installed. Press Ctrl+C to finish.

5. The simulator can now be run. I recommend running the simulator in a special folder, as it will generate logs that accumulate in size over time. To do so, change directories to e.g. ~/picopterx/sim and run:

```
ardusim
```

   The simulator should now be running and waiting for connections.

6. Run the server software. From picopterx/code, run bin/picopter. It will connect to the simulator via the TCP port 127.0.0.1:5760. The software should now be running.

7. Once started, the simulator opens two extra ports (5762, 5763) to allow you to connect extra software via MAVLink to control it. For example, you can connect MAVProxy to port 5762 and Mission Planner to 5763.

8. At the moment, the simulator is started on James Oval, but it is still on the ground. Due to safety restrictions in our software, it cannot perform a takeoff without an external input switching the mode to GUIDED and arming the throttle. These steps may be performed in Mission Planner (graphical interface). The following steps show how it is done using MAVProxy.

9. Ensure you have Python installed. Install MAVProxy with sudo pip install MAVProxy. Connect MAVProxy to the simulator with mavproxy.py --master=tcp:127.0.0.1:5762.

10. From the web interface, prepare it to perform a guided takeoff. It should be waiting to be switched into auto (GUIDED) mode.

11. From MAVProxy enter the following commands:

```
param set arming_check 0
mode guided
arm throttle
```

   The server should now be performing the guided take-off. At this point, the simulated quadcopter is in the air and may be instructed to perform autonomous routines.

More up-to-date instructions may be found at https://github.com/jtanx/picopterx/wiki/Using-the-simulator.

Appendix D. Unit testing

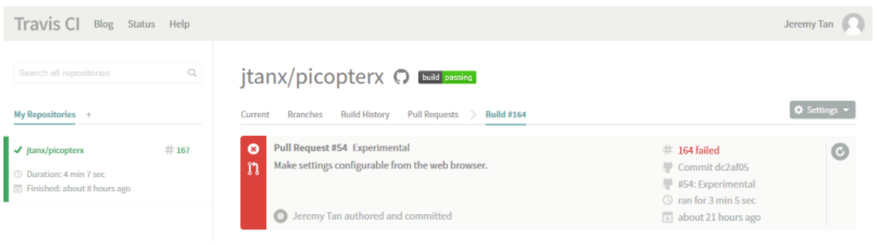
```
Running tests...
UpdateCTestConfiguration  from :/home/jeremy/fyp/picopterx/code/DartConfiguration.tcl
UpdateCTestConfiguration  from :/home/jeremy/fyp/picopterx/code/DartConfiguration.tcl
Test project /home/jeremy/fyp/picopterx/code
Constructing a list of tests
Done constructing a list of tests
Checking test dependency graph...
Checking test dependency graph end
test 1
    Start 1: picopter-testing

1: Test command: /home/jeremy/fyp/picopterx/code/bin/run-tests
1: Test timeout computed to be: 9.99988e+06
1: Running main() from gtest_main.cc
1: [==========] Running 27 tests from 3 test cases.
1: [----------] Global test environment set-up.
1: [----------] 8 tests from BuzzerTest
1: [ RUN      ] BuzzerTest.TestNegativeDuration
1: [       OK ] BuzzerTest.TestNegativeDuration (110 ms)
1: [ RUN      ] BuzzerTest.TestNegativeFrequency
1: [       OK ] BuzzerTest.TestNegativeFrequency (102 ms)
1: [ RUN      ] BuzzerTest.TestNegativeVolume
1: [       OK ] BuzzerTest.TestNegativeVolume (111 ms)
1: [ RUN      ] BuzzerTest.TestHighVolume
1: [       OK ] BuzzerTest.TestHighVolume (108 ms)
1: [ RUN      ] BuzzerTest.TestConcurrentBuzzers
1: [       OK ] BuzzerTest.TestConcurrentBuzzers (577 ms)
1: [ RUN      ] BuzzerTest.TestStop
1: [       OK ] BuzzerTest.TestStop (203 ms)
1: [ RUN      ] BuzzerTest.TestHighFrequency
1: [       OK ] BuzzerTest.TestHighFrequency (102 ms)
1: [ RUN      ] BuzzerTest.TestBlocking
1: [       OK ] BuzzerTest.TestBlocking (577 ms)
1: [----------] 8 tests from BuzzerTest (1893 ms total)
1:
1: [----------] 14 tests from OptionsTest
1: [ RUN      ] OptionsTest.TestFileLoad
1: [       OK ] OptionsTest.TestFileLoad (22 ms)
1: [ RUN      ] OptionsTest.TestFamilySetting
1: [       OK ] OptionsTest.TestFamilySetting (18 ms)
1: [ RUN      ] OptionsTest.TestInvalidFamily
1: [       OK ] OptionsTest.TestInvalidFamily (8 ms)
1: [ RUN      ] OptionsTest.TestNonExistantFile
1: [       OK ] OptionsTest.TestNonExistantFile (4 ms)
1: [ RUN      ] OptionsTest.TestModifying
1: [       OK ] OptionsTest.TestModifying (15 ms)
1: [ RUN      ] OptionsTest.TestModifyingAddingFamily
1: [       OK ] OptionsTest.TestModifyingAddingFamily (8 ms)
1: [ RUN      ] OptionsTest.TestEmpty
1: [       OK ] OptionsTest.TestEmpty (15 ms)
1: [ RUN      ] OptionsTest.TestRemove
1: [       OK ] OptionsTest.TestRemove (3 ms)
1: [ RUN      ] OptionsTest.TestRemoveExisting
1: [       OK ] OptionsTest.TestRemoveExisting (6 ms)
1: [ RUN      ] OptionsTest.TestSaveUnset
1: [       OK ] OptionsTest.TestSaveUnset (4 ms)
1: [ RUN      ] OptionsTest.TestLoadFromString
1: [       OK ] OptionsTest.TestLoadFromString (11 ms)
1: [ RUN      ] OptionsTest.TestLoadFromStringWrong
1: [       OK ] OptionsTest.TestLoadFromStringWrong (4 ms)
1: [ RUN      ] OptionsTest.TestMerge
1: [       OK ] OptionsTest.TestMerge (10 ms)
1: [ RUN      ] OptionsTest.TestSerialisation
1: [       OK ] OptionsTest.TestSerialisation (21 ms)
1: [----------] 14 tests from OptionsTest (166 ms total)
1:
1: [----------] 5 tests from NavigationTest
1: [ RUN      ] NavigationTest.TestToRadians
1: [       OK ] NavigationTest.TestToRadians (0 ms)
1: [ RUN      ] NavigationTest.TestToDegrees
1: [       OK ] NavigationTest.TestToDegrees (1 ms)
1: [ RUN      ] NavigationTest.TestInBoundsA
1: [       OK ] NavigationTest.TestInBoundsA (1 ms)
```

```
1: [ RUN      ] NavigationTest.TestCoordDistance
1: [       OK ] NavigationTest.TestCoordDistance (1 ms)
1: [ RUN      ] NavigationTest.TestCoordBearing
1: [       OK ] NavigationTest.TestCoordBearing (1 ms)
1: [----------] 5 tests from NavigationTest (10 ms total)
1:
1: [----------] Global test environment tear-down
1: [==========] 27 tests from 3 test cases ran. (2077 ms total)
1: [  PASSED  ] 27 tests.
1/1 Test #1: picopter-testing .................   Passed    2.14 sec

100% tests passed, 0 tests failed out of 1

Total Test time (real) =   2.36 sec
```

Listing D.1: Sample output from the unit test suite.

The preceding Listing (D.1) provides an example of the output that is generated by the test suite. This test suite ensures that if modifications are made to the code, that it will retain its expected behaviour ("regression testing"). For example, if a function is designed to output a red image, the test suite can be made to ensure that the function is still outputting red images, even if the code (or implementation) behind it has changed.

D.1 Continuous integration testing

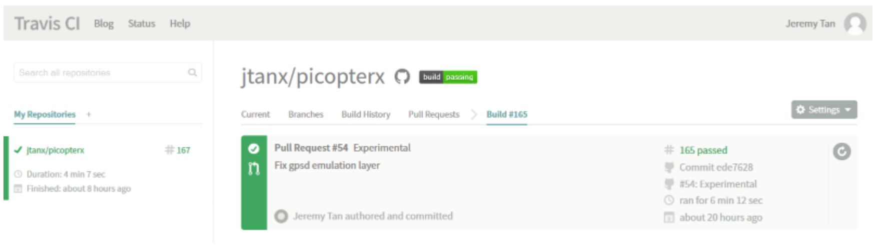


jtanx/picopterx # 167
Duration: 4 min 7 sec
Finished: about 8 hours ago

Pull Request #54 Experimental
Make settings configurable from the web browser.
Jeremy Tan authored and committed

164 failed
Commit dc2af05
#54: Experimental
ran for 3 min 5 sec
about 21 hours ago

Figure D.1: An example of Travis detecting a failed build due to the changes made.



jtanx/picopterx # 167
Duration: 4 min 7 sec
Finished: about 8 hours ago

Pull Request #54 Experimental
Fix gpsd emulation layer
Jeremy Tan authored and committed

165 passed
Commit ede7628
#54: Experimental
ran for 6 min 12 sec
about 20 hours ago

Figure D.2: An example of Travis confirming a successful build.

As shown in Figures D.1 and D.2, the use of Travis (continuous integration testing) helps ensure that the code works as expected after changes are made. After it was noted that the build was failing (Figure D.1), changes were made to fix the issue, and the next build by Travis (Figure D.2) confirms that these changes were suitable.

Appendix E. Profiling image processing functions

To determine the limiting function in image processing, the Google Performance tools CPU profiler was used. This profiler was used over the more conventional Cachegrind option because it is at least an order of magnitude faster than when running under Cachegrind. Additionally, Cachegrind and the associated Valgrind tools do not run correctly under Raspbian due to an outstanding issue (https://bugs.kde.org/show_bug.cgi?id=322935).

A typical command to the profiler is:

```
env LD_PRELOAD="/usr/lib/libprofiler.so" CPUPROFILE=picopter.prof bin/picopter
  configs/camera.json
```

After executing for a few minutes, it is terminated. It is profiled by running:

```
google-pprof --focus=ConnectedComponents --ps bin/picopter  picopter.prof
```

In this case, a call graph focusing on the 'ConnectedComponents' function is generated.

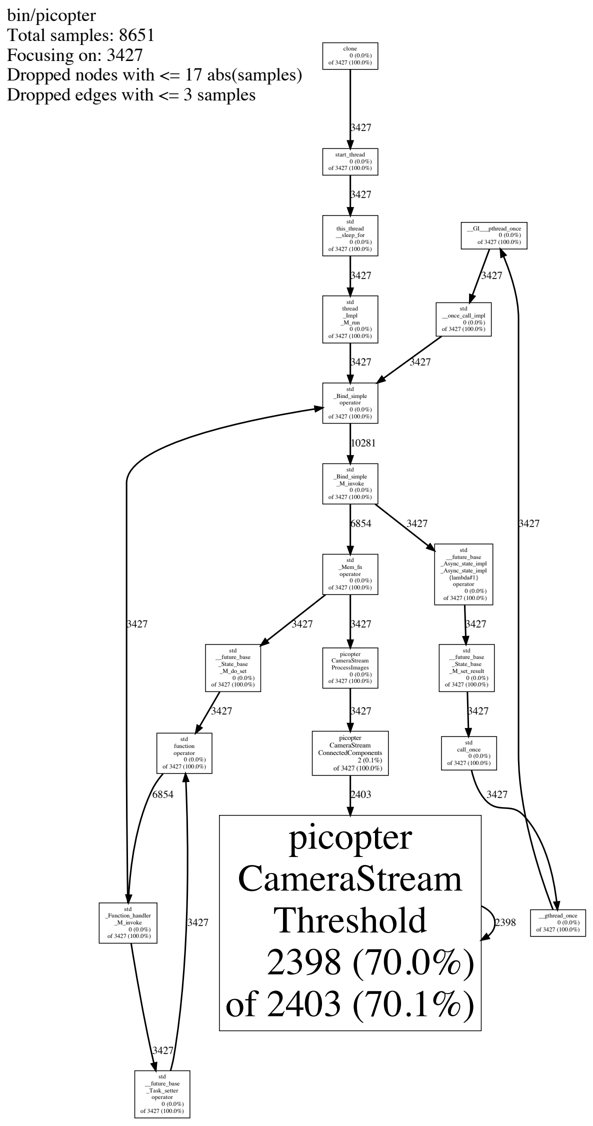

bin/picopter
Total samples: 8651
Focusing on: 3427
Dropped nodes with <= 17 abs(samples)
Dropped edges with <= 3 samples
clone
start_thread
std this_thread __sleep_for
std thread _Impl _M_run
__GI___pthread_once
std __once_call_impl
std _Bind_simple operator
std _Bind_simple _M_invoke
std _Mem_fn operator
std __future_base _Async_state_impl _Async_state_impl {lambda#1} operator
std __future_base _State_base _M_do_set
picopter CameraStream ProcessImages
std __future_base _State_base _M_set_result
std function operator
picopter CameraStream ConnectedComponents 2 (0.1%) of 3427 (100.0%)
std call_once
picopter CameraStream Threshold 2398 (70.0%) of 2403 (70.1%)
std _Function_handler _M_invoke
__pthread_once
std __future_base _Task_setter operator
3427
10281
6854
2403
2398

Figure E.1: A sample profiling graph for the 'ConnectedComponents' function.

Figure E.1 shows a sample output from the profiler (truncated for brevity). In this case, it is indicating that 70% of the call time is due to the Threshold function, so the best performance gains to be had are by optimising this function.

E.1 Debugging on the Raspberry Pi

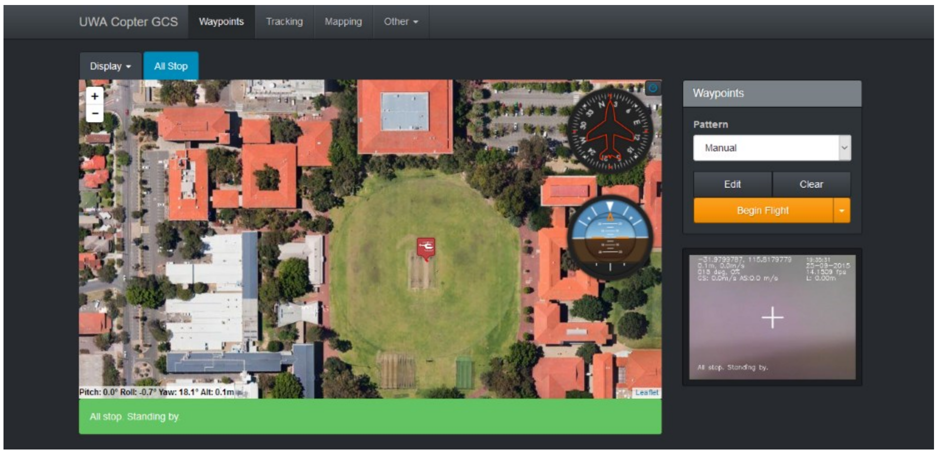
As a follow-up to profiling, debugging on the Raspberry Pi may be required. This is primarily achieved using the GNU Project Debugger (GDB). This section assumes that Raspbian Jessie is used. The versions that come with Raspbian Wheezy are too old.

1. Ensure that your software is being compiled with debugging symbols (`-g`).
2. Start your software in GDB:

   ```
   sudo gdb bin/picopter
   set args configs/camera.json
   handle SIGILL nostop
   run
   ```

   In this example, the server is being run with `configs/camera.json` being passed as arguments. `SIGILL` must be ignored because of the Thrift dependency on libcrypto. See `https://www.raspberrypi.org/forums/viewtopic.php?p=155085` for more information.

Appendix F. Further comments on the web interface



Pitch: 0.0° Roll: -0.7° Yaw: 18.1° Alt: 0.1m

All stop. Standing by

Figure F.1: The same interface using a different theme.

The theme of the web interface is configurable by replacing the Bootstrap CSS file with another one. In `index.php`, this is configured by changing:

```
<link rel="stylesheet" type="text/css" href="css/external/bootstrap.min.css">
```

Other themes may be found at `https://bootswatch.com`.

Documentation on how the interface is used may be found here: `https://github.com/jtanx/picopterx/wiki/Using-the-web-interface`.

F.1 Extending the web interface

To extend the web interface, there are roughly four areas that need to be covered.

1. The Thrift interface is specified via `picopter.thrift` (in `src/server/`). To make changes to this interface, this must be updated first.
2. Once the interface is updated, this must be implemented on both the server and the web interface. From the server, this is implemented in C++ by updating `src/server/picopter.cpp` to implement the added method.
3. On the web interface, server-side changes must be made to `www/ajax-functions.php` to interpret AJAX messages from the client and translate them to Thrift messages.
4. On the client side, new AJAX messages must be sent to the server.

A good example of extending the interface may be found here: `https://github.com/jtanx/picopterx/commit/8413f05d04168d949bfd7e1b835d0e08f95b5c4d`.

Appendix G. Using MAVLink with the hexacopter

This section lists a few pointers for working with the Pixhawk (ArduCopter) and MAVLink.

From a top-down perspective:

1. Establish a connection that MAVLink can use; commonly this is through a serial connection.
2. Establish an input message processing loop. Essentially, you should have one place where all messages are received and processed from the MAVLink stream. If more than one component needs access to that data, find some way to distribute it. On our system, this is done using threading and locks (mutexes). The basic way that this works is:

```
<Establish communications link>
<Start input thread>

Input thread:
  while !shutdown:
       if <non-blocking-read> and have message as <mavlink_msg_t msg>:
              switch(msg.msgid) {
                   case MAVLINK_MSG_ID_XXX:
                       mavlink_XXX_t xxx;
                       mavlink_msg_xxx_decode(&msg, &xxx);
                       <Do something with xxx>
              }
```

   All messages follow this same pattern. Note the MAVLink C header has more than one function to decode and encode messages. One example is shown above.

3. Send messages to the Pixhawk via the MAVLink protocol to make it do things. Luckily for us, TCP and serial communications are thread-safe (or it should be), so you don't really have to worry about only having one thread sending data at any one time. Example:

```
mavlink_msg_t msg;
mavlink_XXX_t xxx;
xxx.target_system = <System ID, get this from the heartbeat message>;
xxx.target_component = <Component ID of the Pixhawk, again get from heartbeat>;
<Fill in the rest of xxx>
mavlink_msg_XXX_encode(<System ID>, <ID for our controller; make one up>, &msg, &xxx);
<Send message>
```

But what messages do you send? What messages are there? The common message set is here: `http://mavlink.org/messages/common`, while the message set specific to the APM platform is here: `http://mavlink.org/messages/ardupilotmega`. However, beware of some pitfalls:

i. The MAVLink specification is just that — a specification. ArduCopter does not necessarily follow it!

ii. Not all messages are implemented by ArduCopter.

iii. Not everything is explained about the mechanics of the message and how it is interpreted by ArduCopter.

So — how do you know what messages will work with ArduCopter? Look at it's source code, available here: `https://github.com/diydrones/ardupilot`. I recommend reading up especially on `GCS_Mavlink.cpp` in the ArduCopter folder. This file contains most of the messages that are processed by ArduCopter. Common messages may be found in the `GCS_Mavlink` folder — look at `GCS_Mavlink.cpp` and `GCS_Common.cpp` in that folder in particular.

However, user beware! Changes are made frequently to the ArduPilot code base. What you see there now may not actually be what's currently loaded on the hexacopter. At the time of writing, it is running ArduCopter-3.3. You can checkout this version of source code from *Git* to ensure that what you see matches exactly with what is running on your hexacopter. This is the best way to ensure that what you write will actually work as intended.

Appendix H. Video encoding and OpenMAX

A number of multimedia processing components are available on the Raspberry Pi. They provide GPU accelerated image, video and sound encoding, decoding and rendering. GPU accelerated fast Fourier transforms is also possible, which may be useful for some image processing algorithms. Documentation for these components may be found here: `http://home.nouwen.name/RaspberryPi/documentation/ilcomponents/index.html`.

To perform encoding, a freely available helper "`ilclient`" library is used along with OpenMAX. The `ilclient` library is used because the OpenMAX API is inherently asynchronous, requiring the use of multiple callbacks and event handlers. Using the `ilclient` library, much of this is abstracted, allowing for sequential programming, which greatly simplifies development.

H.1 Resolution restrictions

In practice, the width/height restrictions described in Section 3.4.1 can be resolved by rounding down to the nearest multiple of 8 or 16 and processing these pixels as described. The remaining 'left-over' pixels are processed using a slower algorithm that works on a per-pixel basis.

H.2 On hardware accelerated image encoding

There are less gains to be had for hardware accelerated image encoding as compared to video encoding. First, if only one or two images need to be saved, the overhead of setting up the hardware encoder will likely exceed the total time of just encoding the image using the CPU. Thus, hardware acceleration may only be applicable when saving multiple images. Second, and as suggested in Section 3.4, the hardware encoder has specific restrictions on acceptable inputs, which are not present for software based encoding. The flexibility of software encoding to accept any input format combined with the relative computational simplicity of image encoding thus makes the use of hardware encoding restricted to specific use cases.

H.3 Structure packing

Special attention *must* be paid to how data structures (structs) are packed for sending data to the encoder. On the Raspberry Pi (Raspbian), OpenMAX expects that structs are passed with an alignment of 4 bytes, which must be set explicitly. By not setting this alignment, the compiler will choose the wrong alignment, which will result in the encoder not working properly in subtle ways. Using GCC, struct alignment is specified by:

```
extern "C" {
#pragma pack(4)
#include <bcm_host.h>
#include <ilclient.h>
#include <OMX_Broadcom.h>
#pragma pack()
}
```