An autonomous drone system, with intelligent flight-planning, to effectively undertake indoor photogrammetry

Submitted April 24th 2018, in partial fulfilment of

the conditions for the award of the degree, **Computer Science (G404).**

Matthew Clarke

With supervision from Steven Bagley

[redacted]

School of Computer Science

University of Nottingham

I hereby declare that this dissertation is all my own work, except as indicated in the text:

Signature: [redacted]

Date: 24/04/2018

Abstract

Modern online mapping tools provide users a 3-dimensional perspective view of cities and landmarks, to allow for better visualisation of the map itself. To provide these 3D views, a reconstruction technique, *photogrammetry*, is utilised over an input dataset of aerial imagery. However, photogrammetry is not yet widely utilised for interior spaces in these online mapping tools, which are of equal importance such as for tourism purposes.

This dissertation aims to provide 3D reconstruction for interior spaces, with an automated system to collate a dataset of images and to automatically undertake photogrammetry upon it. A multi-copter is utilised as the hardware platform for dataset generation, connected to extensive ground control software through which to control its flight. Furthermore, a solution to the problem of indoor GPS reliability is implemented via Visual Simultaneous Localisation and Mapping, to provide appropriate positioning estimates to the multi-copter.

Word count: 15,451

Contents

List of Figures

List of Tables

Fonts

`Example text:` denotes a system component or utilised library
`Example text:` denotes a class name, inline code or shell command

1 Introduction

1.1 General Background

Many modern mapping solutions typically integrate 3-dimensional models of buildings to allow users to better visualise the environment being represented; one example is that of Google Maps[1]. These models are typically generated via a technique known as photogrammetry[2]. As Schenk notes, *"there is no universally accepted definition of photogrammetry"* (Schenk, 2005)[2]. Therefore, this dissertation treats the term *photogrammetry* to be defined as:

> *"[...] the science of obtaining reliable information about the properties of surfaces and objects without physical contact with the objects, and of measuring and interpreting this information."*
>
> **Source:** Schenk (2005)[2]

More specifically, this dissertation considers Structure-from-Motion photogrammetry, which provides the generation of 3D objects from a set of images taken at differing angles and translations.

Aerial photography is typically utilised to generate the required imagery to reconstruct building exteriors, from platforms such as multi-copters, aeroplanes and satellites. Interior modelling, however, is not typically represented in these aforementioned mapping solutions. Whilst research does exist focusing on utilising photogrammetry for such an application, such as that from Al Khalil et al (2001)[3], and indeed even industry focus from Swiss company Mosini Caviezel SA[4] – whom focus on generating 2D floor plans from resulting 3D models – it can be argued that this is a niche that can be filled for high-fidelity mapping solutions.

1.2 Project Motivation and Overview

A solution to the lack of interior 3D modelling can be achieved through allowing the user to photograph an indoor space, and subsequently feed these images into a program implementing photogrammetry. For a dissertation however, this is not ambitious enough, and is not novel. Furthermore, it doesn't provide a significant enough external aspect – it is likely there will be little users, as copious existing photogrammetry applications provide this functionality.

To resolve this lack of novelty, the author presents the idea of producing an end-to-end interior 3D modelling solution that builds upon existing photogrammetry libraries, to produce output with ideally a single button press in a GUI. Production of images will be handled by a moving platform with an attached camera, with an appropriate platform being a multi-copter due to their manoeuvrability and existing usage for outdoor photogrammetry. Utilising a multi-copter would also be beneficial to the requirement of an external aspect to the dissertation; the end solution can be targeted towards a hobbyist community, chiefly that of multi-copter enthusiasts.

Further to the idea of utilising a multi-copter, affording it autonomous capabilities lends well to a "single click" solution. By intelligently choosing which positions within the multi-copter's known surroundings to move to, an interior model can be generated with little to no prior knowledge regarding photogrammetry on the part of the user.

Thus, this dissertation focusses upon building.

1.3 Aims and Objectives

Thus, the core aim of this dissertation is to implement an approach to indoor photogrammetry that utilises an autonomous aerial platform, in this case a multi-copter, with intelligent and dynamic flight planning, such that it is cost-effective enough to be utilised by hobbyists.

To achieve this aim, the following objectives will need to be met:

1. To implement duplex communication between a multi-copter and ground control station program to send positioning commands and receive live video data.
2. To provide effective indoor localisation for the multi-copter.
3. To integrate with an existing photogrammetry library to produce both an interim 3D model, and the final end indoor model.
4. To intelligently reason about the interim 3D model to produce a flightpath for the multi-copter to follow.
5. To assess differing flightpaths in their ability to generate useful sets of images for photogrammetry to be undertaken.

1.4 Multi-disciplinary Nature of the Project

The multi-copter utilised has been built from scratch to meet the cost-effective clause of the core aim, with all components sourced and assembled by the author. Resulting from this, this project can be argued to be multi-disciplinary, combining Computer Science with that of Electronics, and to some degree Aerospace Engineering.

Therefore, a percentage of this dissertation covers the electronics and engineering aspects of building this hardware; a significant degree of learning, research, and effort has been put into the construction of the multi-copter.

1.5 Externality

In terms of externality and as stated earlier, this project aims to appeal to the wider multi-copter hobbyist community. It is due to this that this project should be cost-effective – low-cost solutions are far easier for a community of users to corral behind due to a lower financial entrance barrier.

To this end, the author has contacted a specific hobbyist user, in this case Dr Michael Pound of the University of Nottingham, to ask for his input into this dissertation's software project ("the Project"). As a stakeholder, he has been able to provide an external aspect to the Project that would act as an extension from the hobbyist community as a whole. This input has been wholly invaluable, such as through providing support on construction of the multi-copter itself.

2 Research

2.1 Existing Works

To commence the Project, existing works both in literature and industry were examined. Hao et al (2011)[5] present a prototype mobile 3D mapping system to *"build dense and complete models for his/her personal environments, running on a laptop in real-time and interacting with the user on-the-fly"*. This system utilises an RGB-D (colour and depth) camera such as the Microsoft Kinect[6], alongside manual user guidance to map an environment. Whilst not automated as per this dissertation's proposal, this system provides robustness through *"[computing] 3D alignments of depth frames on-the-fly, so that the system can detect failures [...] and prompt the user to "rewind" and resume scanning"* (Hao et al, 2011)[5]. The end result of their system provides 3D models of large indoor spaces such as an office, to a centimetre level of accuracy. For the Project, an RGB-D camera may therefore be a key hardware component.

Focus on particular points of interest within an interior model is presented by Díaz-Vilariño et al (2015)[7]. Their approach is that of *"a data-driven method for geometric reconstruction of structural elements using the point cloud, and a model-driven method for the recognition of closed doors in image data based on the generalized Hough transform"* (Díaz-Vilariño et al, 2015). By detecting doors in image data, their work allows for improved reconstruction of building interiors, allowing for modelling the location of doors in relation to other interior maps. Existing approaches such as that of Budroni and Boehm (2010)[8] rely upon low density regions of resultant point clouds, from for example LiDAR[9] data, to find connections between rooms. The issue here is that closed doors cannot be detected, as they are presented in such data as regions of higher point density – highly similar to walls and other obstructions. Therefore, the usage of both LiDAR and RGB camera data by Díaz-Vilariño et al allows for using colour information to detect closed openings at a much better rate of success. The rationale provided by Díaz-Vilariño et al gives a further insight into why RGB data was used in conjunction with LiDAR: *"a laser beam usually penetrates window glasses, so that no laser points are reflected, causing areas with low raw laser information"* (Díaz-Vilariño et al, 2015)[7].

Lehtola et al (2017)[10] provide an in-depth narrative comparing state-of-the-art scanning and point cloud generation methods. The compared approaches comprise of commercial and experimental interior mapping technologies: the Matterport 3D camera, the NavVis 3D Mapping Trolley, the Zebedee hand-held 3D mapping system, the hand-held Kaarta Stencil, the Leica Pegasus Backpack, the Würzburg Backpack, the Aalto VILMA, and the FGI Slammer. Details on each platform are covered in Lehtola et al's paper; this is recommended further reading. Through qualitative evaluation, they found that the experimental FGI Slammer and commercial

NavVis platforms provided the most precise point clouds. Both utilise Simultaneous Localisation and Mapping (SLAM) approaches[11] for localising themselves within the current environment, alongside multiple laser scanners to generate these point clouds. That said, these platforms are both wheeled, and so are restricted to flat environments. For the purposes of the Project, multiple scanners may not be feasible; the cost of a single laser scanner is too prohibitive for hobbyist users. Though, the methodology utilised to compare generated point clouds may be appropriate to determine the precision of results generated by the Project.

In industry, Skydio has recently produced a fully autonomous multi-copter: the Skydio R1[12], marketed as a *"self-flying camera"*. Its capabilities are impressive: highly accurate SLAM, subject tracking through Skydio's in-house artificial intelligence software, along with on-the-fly route planning for the extreme near future. Powering this is NVIDIA's Jetson TX2[13]– a 256-core GPU alongside a quad-core 64-bit ARM CPU – along with 13 cameras for input. At a price point of $2,499.00 this hardware is far beyond the low-cost core aim of the Project, though can serve as a useful reference.

2.2 Required Hardware

Through the process of researching existing works, the author has remained settled on the prospect of utilising a multi-copter for the Project, due to the manoeuvrability it allows when generating a 3D point cloud. Care must be taken to ensure any camera installed on the multi-copter is vibrationally isolated as mentioned by Skydio's documentation[12]. This is since any vibration will affect any SLAM algorithm in use, by inducing erroneous translations in the observed physical world.

As mentioned in Section 1.3, a clause of the core aim is to be financially cost-effective for a hobbyist. Therefore, to control the exact cost the author opted to build the multi-copter himself. Generally, any multi-copter requires the following basic components: a frame upon which to mount hardware, a flight controller, a battery, motors, Electronic Speed Controllers (ESCs), and propellers[14].

To progress with research, the flight computer and camera for the multi-copter needed to be chosen. Thus, the flight computer was chosen to be the Raspberry Pi Zero W[15], on account of its small size and exceedingly low cost – £11. Removing the cost-effective restriction would lead the author to instead invest in the NVIDIA Jetson TX2[13] as utilised in the Skydio R1, due to its compute capability of 1.5 teraflops. To connect the Raspberry Pi to the motors of the built multi-copter, an addon board was required. Two such boards exist: the Navio2[16] and the PXFmini[17], both depicted overleaf in Figure 1.

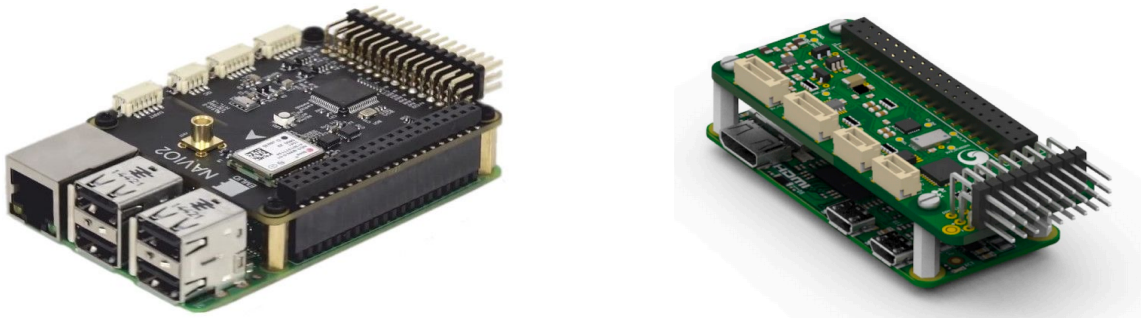


Figure 1: The Navio2 and the PXFmini
Note: The Navio2 (left) is connected to a Raspberry Pi 2, and the PXFmini (right) to a Raspberry Pi Zero.
Sources: Emlid Ltd (Updated 2018)[16] and Erle Robotics (Updated 2017)[17].

The author decided upon the PXFmini as the addon board due to two factors: it is more optimised for the form factor of the Raspberry Pi Zero, and costs £58 less than the Navio2 yet also provides the minimum diversity of sensors required for the Project.

In terms of camera input, existing works have demonstrated that the utilisation of both RGB-D cameras and LiDAR in conjunction produce precise point clouds. On the lower financial spectrum for LiDAR is that of Scanse's Sweep[18], providing a range of 40 metres at a cost of $349.00. Unfortunately, this alone is above the maximum budget for the Project, totalling £250.00. Therefore, the author has directed efforts into utilising a low cost RGB or RGB-D camera such as the Intel RealSense D435[19]. This particular RGB-D camera provides a wide-angle lens over both stereo cameras, allowing for better tracking of rapidly moving subjects, or in the Project's case a rapidly changing environment due to the multi-copter's translations and rotations. However, this hardware was not available at the time of multi-copter construction, instead being released during the first quarter of 2018. Also available through Intel is the SR300 RGB-D camera[20], though is far too large to fit on a multi-copter nimble enough for indoor flight.

Due to these limitations in acquiring a suitable RGB-D camera, the author instead opted for the Raspberry Pi v2.1 camera module[21], resulting in only a monocular RGB video stream. As Pagnutti et al (2017)[22] have found, this camera has the capability to provide scientific and engineering-grade output; *"raw imagery is shown to be linear with exposure and gain (ISO), which is essential for scientific and engineering applications"* (Pagnutti et al, 2017)[22]. Due to the usage of monocular input, care must now be taken when choosing a localisation approach such that monocular input is well-supported.

2.3 Flight Software and Communications

Simply assembling hardware components is not sufficient to build a functional multi-copter; a flight stack must be running upon the flight computer to correctly output commands to each motor. Due the usage of the Raspberry Pi Zero W, the flight stack must be capable of running atop GNU/Linux. Two such major projects exist: `ArduPilot` (APM)[23], and `PixHawk`

(PX4)[24]. Note that decisions regarding the flight stack need not be undertaken until building the multi-copter's OS image.

Before continuing, a definition must be provided for the term *flight stack*. Typically, such a stack comprises of three layers: a real-time operating system, middleware providing device drivers and peripheral support, and flight control such as telemetry and motor control[25]. In the case of this project, the real-time operating system will be Linux with a real-time patch applied to its kernel[26], and so will be identical between APM and PX4.

Comparing the two flight stacks end-to-end, it can be seen that APM's base OS support is far more varied, allowing running on: Linux, Arduino, and ChibiOS. PX4 instead requires that its base OS has real-time capabilities, along with POSIX-compliance. At the middleware level, differences are less apparent. Both flight stacks support the use of the MAVLink protocol to communicate with companion computers; *"a very lightweight, header-only message marshalling library for micro air vehicles"* (Meier, 2009)[27]. Differentiation occurs over which additional communication is supported. APM provides support for both the Robot Operating System (ROS)[28] alongside the CAN bus[29], whereas PX4 supports the Real Time Streaming Protocol (RTSP)[30] along with the micro Object Request Broker (uORB)[31] protocol specific to PX4.

Both flight stacks were worked upon simultaneously under the Dronecode project from late 2014, aiming to foster *"[a] neutral place where industry and community developers can contribute technology in order to reduce costs and time to market"* (Dronecode, updated 2018)[32]. However, the team behind `ArduPilot` exited the organisation due to leadership concerns in 2016. As a result, PX4 retains a greater array of commercial partners including Microsoft and Intel.

2.4 Indoor Positioning

Due to the aim of this project to map interior spaces, localisation of the multi-copter is a key concern; typical GPS-based solutions are not reliable in an indoor setting[33]. To combat this, alternative approaches can be undertaken: Differential GPS (DGPS), Visual Odometry (VO) or Simultaneous Localisation and Mapping (SLAM).

As presented by Mertikas (1983)[34], the concept of DGPS is derived from the original implementation of GPS by the United States Department of Defence. The concept of Selective Availability was applied to limit the usefulness of GPS to unauthorised entities such as the general public, with only military users able to utilise GPS' maximum accuracy of to within 10 metres. To circumvent this issue, Mertikas proposed the technique of Differential GPS: a ground station is placed at a known position, receives GPS signals and applies a differential correction to improve accuracies for users connected to the ground station. Whilst no longer necessary post-2000, this technique can still be applied to indoor positioning, with a ground station placed at a known location outdoors relaying corrected positions to indoor GPS users.

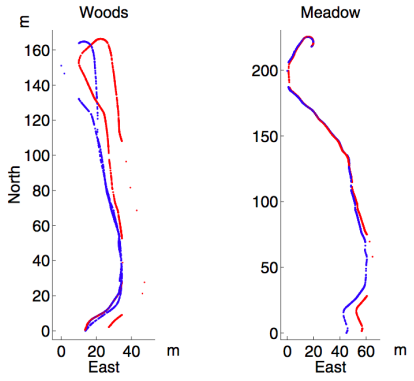


Figure 2: Accuracy comparison of DGPS (dark blue) against VO (red)
Source: Nister, Naroditsky and Bergen (2004)[35]

Alternatively, VO could potentially be utilised. Due to the advent of inexpensive pinhole cameras, multitudes of visual algorithms have been developed to extract meaningful positioning data from a stream of incoming video frames. As presented by Nister, Naroditsky and Bergen (2004)[35], VO matches features between pairs of video frames, and computes a trajectory the platform has travelled along. The end positioning results can thus be combined with additional sources of positioning such as inertial sensing and GPS input. Their results comparing VO to DGPS can be seen to the left in Figure 2. Note that their results have been generated through the usage of stereo cameras, which may prove challenging for this project due to the previous decision to utilise monocular camera input. VO has proved highly successful, to the degree that two rovers placed on the surface of Mars make use of it to maintain a robust position estimate (Cheng and Maimone, 2005)[36].

Building atop VO is that of Visual SLAM (V-SLAM). As Durrant-Whyte and Bailey state:

> *"The simultaneous localization and mapping (SLAM) problem asks if it is possible for a mobile robot to be placed at an unknown location in an unknown environment and for the robot to incrementally build a consistent map of this environment while simultaneously determining its location within this map."*
>
> **Source:** Durrant-Whyte and Bailey (2006)[1]

Extending from this definition, V-SLAM is the approach of utilising camera input to solve the SLAM problem, instead of inertial or other input data. Recent developments in CPU and GPU technologies have allowed for V-SLAM to be undertaken in real-time, due to the significant computational cost it typically requires[37]. In terms of implementation, typically five modules are present in any given V-SLAM algorithm: initialisation, tracking, mapping, re-localisation, and global map optimisation[37]. A recent example of monocular V-SLAM is that of `ORB-SLAM` (Mur-Artal et al, 2015)[38], with its system architecture presented to the right in Figure 3.

This algorithm makes use of ORB to undertake feature matching, combining the FAST feature detector to find object corners, alongside the BRIEF feature descriptor (Rublee et al, 2011)[39]. Initialisation is undertaken through tracking features over multiple frames of input, resulting in triangulation of points from which to generate a map, though in an arbitrary reference frame. Since these points have high uncertainty of depth,

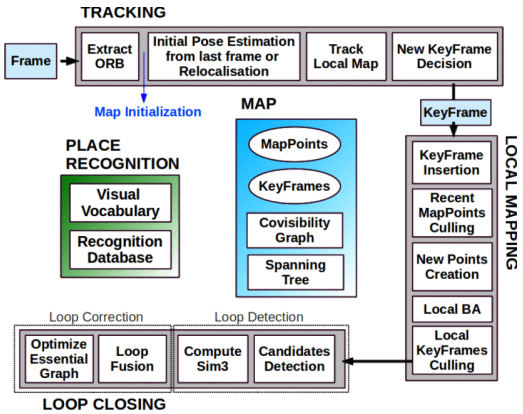


Figure 3: System architecture of `ORB-SLAM`
Source: Mur-Artal et al (2015)[38]

initialisation does not complete until this uncertainty has been reduced over successive triangulation iterations.

An alternative monocular V-SLAM algorithm is that of `LSD-SLAM` (Engel and Cremers, 2014)[40]. Rather than detecting features through corner detection, edges are also utilised allowing for greater accuracy and robustness. Furthermore, the algorithm incorporates the effect of noisy depth values into tracking to achieve this robustness. That said, the algorithm faces difficulties if tracking is lost – re-localisation occurs by re-initialising the algorithm, effectively shifting the origin position.

2.5 Video Streaming

To provide input into any visual-based localisation approach, data must be sourced. Due to the flight computer being that of a Raspberry Pi, it is highly likely that the end multi-copter will be running Linux. However, this hardware is limited on compute power, and therefore will likely need to off-board any localisation computations to a ground station computer. In this case, video streaming will be required to provide video data to this computer.

The most popular video streaming utility available to Linux is `gstreamer`, *"a library for constructing graphs of media-handling components"* (GStreamer Team, updated 2018)[41]. A pipeline for consuming and processing video or audio input can be built through `gstreamer`, typically starting with an input `source` element and ending in a `sink` element; the latter may render video to a display, for example. Furthermore, a `caps` element can be provided to provide detailed metadata about expected inputs and outputs throughout the pipeline.

In terms of streaming video, `gstreamer` allows for providing a `source` element to read directly from a connected camera, and `sink` element outputting over a UDP link to a specified destination. Figure 4 presents an example pipeline to stream video:

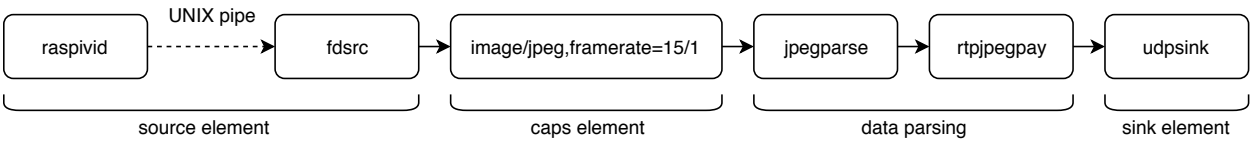


Figure 4: Example `gstreamer` pipeline

An appropriate pipeline can be constructed on the ground station computer to receive input from the resultant UDP stream.

In terms of media codecs, two were considered for video streaming: H.264 and Motion JPEG (MJPEG). Both provide lossy compression for video, with the former also providing motion compensation at the cost of greater computational complexity[42]. H.264 encodes a full static frame, referred to as an I-frame, alongside Predicted frames (P-frames) and Bi-directionally predicted frames (B-frames). Each P-frame encodes a motion vector for changes in the I-frame; a given frame is subdivided into 16x16 blocks, where each block may not change between frames.

To reduce storage requirements, only those blocks with a motion vector need to be updated. B-frames follow the same approach, except also encode future motion vectors along with any current vectors. Therefore, an H.264 decoder uses the last I-frame and subsequent P-frames alongside B-frames to reconstruct any given frame of video.

Conversely, MJPEG simply encodes each frame of video as a JPEG image[43]. This is far less computationally complex than H.264, and so can easily run on any commodity hardware. However, outputted video is compressed at a lesser rate compared to H.264, giving larger output sizes. Therefore, if using MJPEG care must be taken to configure the framerate, bits-per-second and resolution of input video to ensure bandwidth limitations are taken into account.

2.6 Photogrammetry

As stated in Section 1.1, this dissertation only considers Structure-from-Motion (SfM) photogrammetry, providing the generation of 3D objects from monocular camera inputs. The general approach as outlined by Micheletti et al (2015)[44] for SfM is to capture multiple views of an object from a range of positions, identify common features across the images, and subsequently transform these features into 3D co-ordinates to be presented in a sparse point cloud. This point cloud can then be pipelined into Multi-View Stereo (MVS)[45] techniques to densify the point cloud, resulting in a textured 3D model.

In terms of specific implementations of SfM photogrammetry, multitudes exist. The public notes of Falkingham (2017)[46] have been instrumental in researching existing implementations. His findings showed that the combination of `COLMAP`[47] and `openMVS`[48] to generate a point cloud and densify it to a 3D model respectively produced the most accurate results.

The author has generated an example output using this combination of software to test accuracy. Figure 5 below shows the model generated from a self-made dataset of 150 images at a resolution of 640x480 pixels, taken from an iPhone X with automatic ISO and white balance:

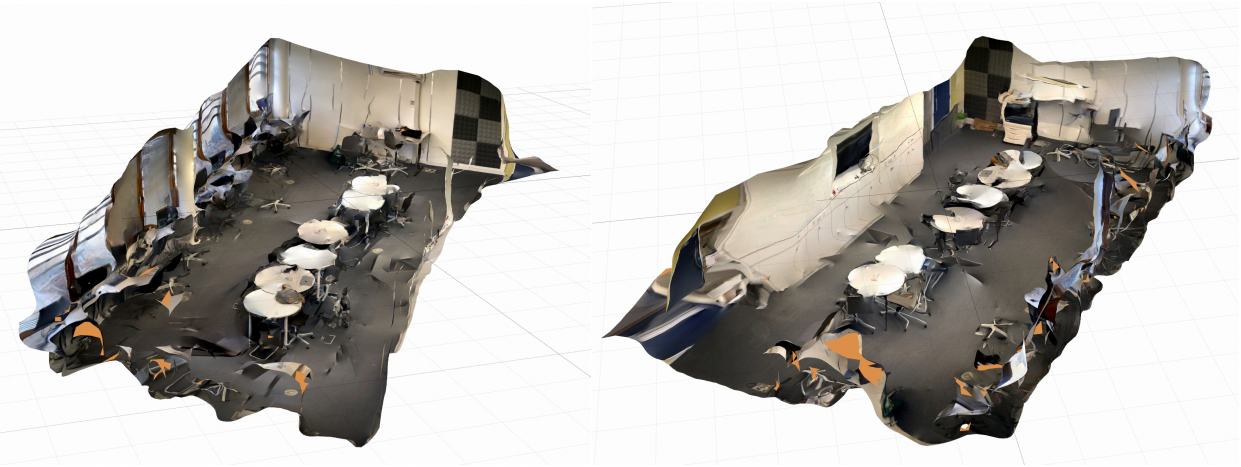


Figure 5: Example output from `COLMAP` and `openMVS` on a self-made dataset

Whilst not completely true to the physical space, these models prove that this combination of software is viable. Thus, work will need to be done to ensure input images are sufficient enough to generate high-quality output models.

3 System Architecture

The system produced to meet the core aim of the Project provides the necessary inter-communication between the disparate research areas covered by Section 2.

3.1 Development Methodology

The general development methodology has been an individual-orientated SCRUM[49]-based approach; there is no need for morning stand-up meetings, and supervisor meetings are typically treated as end-of-sprint meetings. However, this was not strictly enforced in the first half of the Project, as the time required for multi-copter construction proved extremely variable. Furthermore, ZenHub[50], a web application integrating with GitHub that allows for moving issues on a project through a pipeline, has been utilised to organise tasks. A screenshot is provided below in Figure 6:

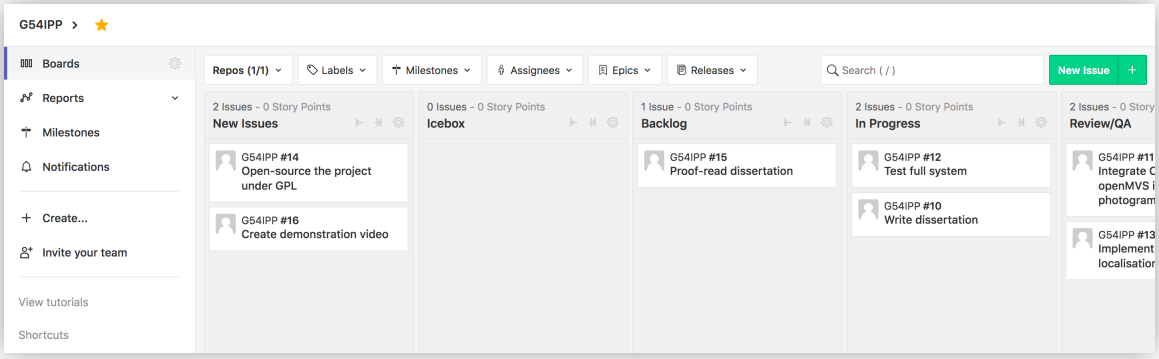


Figure 6: ZenHub board utilised by the author to organise tasks

Since Dr. Michael Pound is a stakeholder as mentioned in Section 1.5, it also has been necessary to ensure he remains updated on the Project's progress. The majority of communication with him has been over email and some face-to-face interactions. Overall, this communication has related typically to the construction of the multi-copter, due to his expertise in the matter.

3.2 Design

The system built for this project is multi-threaded and also comprises of multiple processes. Due to the complexity of integrating components such as localisation and photogrammetry, each major subsection of the Project was designed as a module of the whole. Figure 7 overleaf shows this modular design in full.

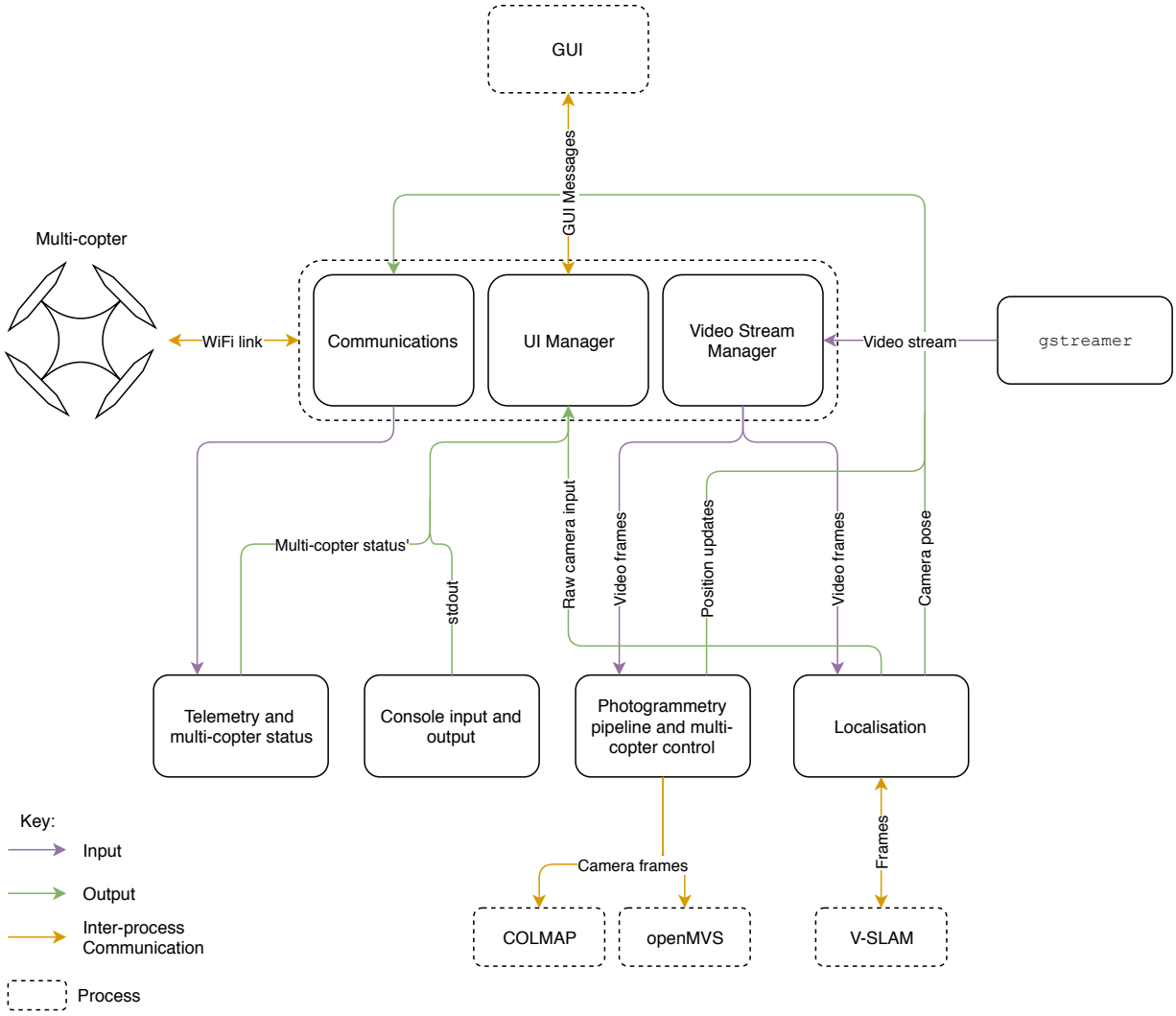


Figure 7: System architecture design for the Project

Of the components of the diagram, the following are third-party: *COLMAP*, *openMVS*, *V-SLAM* and *gstreamer*.

Four modules have been designed; *telemetry*, *console I/O*, *photogrammetry* and *localisation*. The first handles incoming packets from the multi-copter, and processes them to provide, for example, battery statistics for the GUI. Second, the console module affords the user an interactive command-line style interface to communicate directly to the multi-copter, and to view messages that are written to the standard output (`stdout`) of the system. The third module, photogrammetry, provides the necessary implementation to interact with both `COLMAP` and `openMVS`. Alongside this, this module also generates waypoints for the multi-copter to navigate to. To enable waypoint navigation, the localisation module is required to generate positioning data based upon incoming video frames. This communicates with the to-be-defined approach to localisation running in a separate process.

The main process of the system co-ordinates each of these modules, by providing three "management" objects: *Communications* for sending and receiving messages from the multi-copter, *UI* to send and receive messages from the GUI process, and *Video* to arbitrate incoming video frames from the `gstreamer` thread.

3.3 Multi-Copter OS Considerations

As discussed in Section 2.2, the Raspberry Pi Zero W (RPi) has been utilised as the flight computer for the multi-copter. In terms of software, it was decided to base the operating system (OS) of the multi-copter on Linux, due to the author's pre-existing familiarity with it. Furthermore, this ensures that either `ArduPilot` or `PixHawk` can be utilised as the flight stack with no change to the underlying OS. In addition, an example OS image is provided by the manufacturer of the PXFmini addon board utilised – Erle Robotics[51] – which is also Linux-based.

Unfortunately, this example OS could not be utilised as a basis; any derivatives are non-redistributable, requiring pre-purchase of hardware from Erle Robotics to access the image. Thus, the author opted to design an OS image from scratch. Due to the low compute power available to the RPi, the system components in use must be as minimal as possible. Curiously, the example OS ships with additional components that are simply not necessary, including a copy of the popular sandbox game Minecraft[52].

In terms of the flight stack, `ArduPilot` was chosen. The primary reasoning for this was due to its greater maturity over `PixHawk`, with the assumption that the flight control layer has received a greater degree of testing and stability adjustments. Furthermore, its inclusion in the example OS provides a useful reference as to how it should be configured.

The general design of the OS image can be found below in Figure 8:

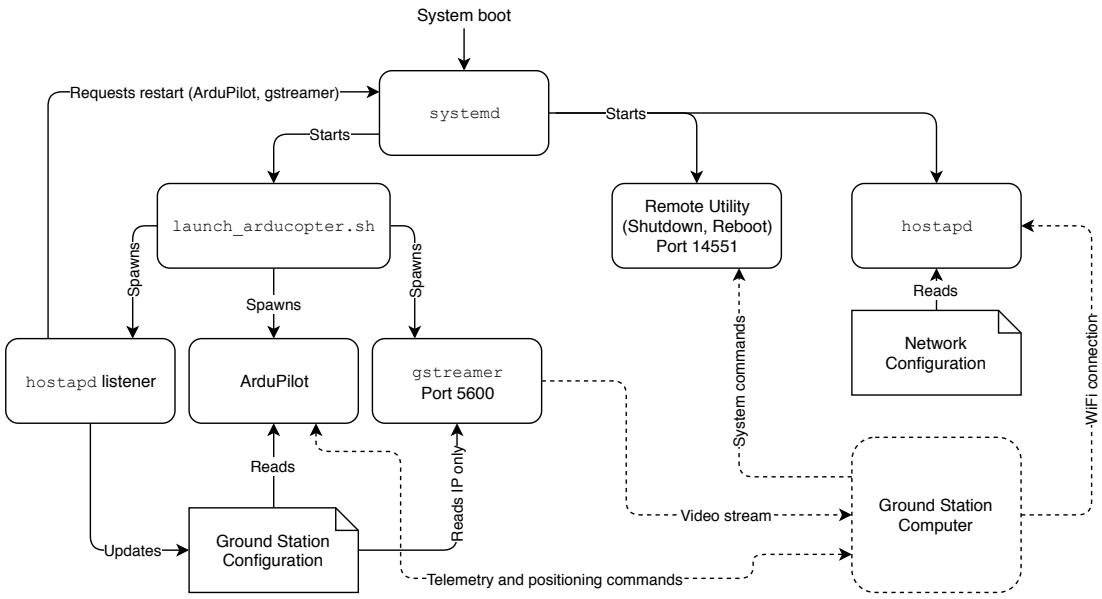


Figure 8: General OS image design

As can be seen in Figure 8, all components are started by `systemd`, providing *"a system and service manager that runs as PID 1 and starts the rest of the system"* (freedesktop.org, updated 2017)[53]. The shell script `launch_arducopter.sh` is started by `systemd`, which spawns three sub-processes: a listener for changes regarding devices connected to the WiFi hotspot generated by `hostapd`, `ArduPilot` itself, and the `gstreamer` pipeline to stream video to the ground station computer. Note that `hostapd`, `gstreamer`, `ArduPilot` and `systemd` are not developed by the author.

A WiFi hotspot is generated by the multi-copter to allow the user to connect without needing to configure an external WiFi hotspot. Since IP addresses assigned to connected devices may change, and since `ArduPilot` treats the ground station computer as a server, an up-to-date record of which IP address represents the ground station computer is required. When a change is detected regarding connected devices, a configuration file referencing the ground station computer is updated, recording its IP address alongside a user-configurable port to connect over. `ArduPilot` and `gstreamer` are subsequently restarted to utilise these configuration changes.

Furthermore, a Remote Utility is present to process system-related commands from the ground station computer, such as to shutdown or reboot the OS. This runs separately from `ArduPilot`, ensuring that some degree of control can still be maintained in the event the flight stack crashes.

3.4 `MAVProxy`

To ensure cross-compatibility between both `ArduPilot` and `PixHawk` if the user decides to change the flight stack, MAVLink was chosen to be the communication protocol to the multi-copter. However, there are multitudes of messages supported by the protocol; for the full listing, see the further reading in Section 8.1. Therefore, it falls outside the Project's scope to implement the protocol.

Existing projects are available that implement the majority of the protocol, with `MAVProxy`[54] being a key open-source and extensible example. Developed by CanberraUAV initially for competing in the 2012 UAV Outback Challenge[54], `MAVProxy` supports running atop any POSIX-compliant operating system. Furthermore, the capability is provided to extend `MAVProxy` through building additional modules written in Python. These modules have full duplex communication with the multi-copter, such as the ability to request a take-off, and to monitor system status.

Thus, `MAVProxy` was chosen as the basis of this project, with all architecture as presented in Figure 6 implemented as module to it. This also dictates the programming language of the Project: Python. Ideally, no modifications to `MAVProxy` itself need to be made, allowing for updates produced by CanberraUAV to be incorporated without breaking core functionality of the Project. Another benefit of utilising `MAVProxy` is that the author's own module has the capability to communicate with other modules bundled with `MAVProxy`, greatly reducing development efforts for mundane lower-level details, such as calibrating the multi-copter.

An interesting challenge at this point of the Project was uncovered: the author has little prior experience with Python. Therefore, all learning of the language must be undertaken alongside developing the architecture of presented in Figure 6. Whilst this has the potential to lead to inferior code quality, the author has treated this requirement as an opportunity to broaden his skillset. Furthermore, Python is cross-platform, and so ensures that nobody in the hobbyist community will be discriminated against in their choice of desktop operating system.

3.5 Video Streaming Configuration

As prior shown in Figure 8, `gstreamer` is utilised to stream video from the multi-copter to the ground station computer. The pipeline utilised to undertake this is highly similar to that presented in Figure 4, with the exact shell command utilised shown below in Figure 9:

```
raspivid -t 0 -cd MJPEG -w 640 -h 480 -fps 15 -b 1500000 -o - | gst-launch-1.0 fdsrc ! "image/jpeg,framerate=15/1" ! jpegparse ! rtpjpegpay ! udpsink host=XXX.XXX.XXX.XXX port=5600
```

Figure 9: `gstreamer` pipeline to stream video to the ground station computer

Initially, the `raspivid` utility, built-in to the OS image, is utilised to read directly from the camera hardware. Here, the output resolution, codec, framerate and bitrate are hardcoded at:

Resolution: 640 pixels wide by 480 pixels high
Framerate: 15 frames per second
Bitrate: 1500000 bits per second
Codec: Motion JPEG (MJPEG)

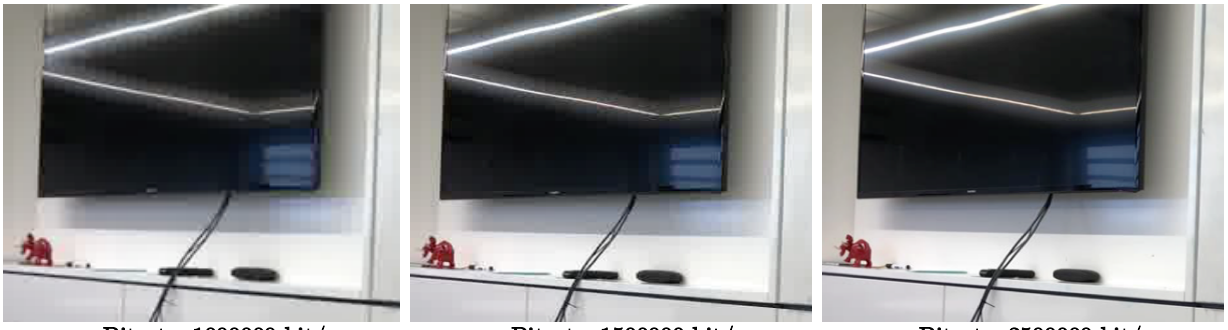
Initial testing of video streaming showed that `ArduPilot` typically utilises on average 75% CPU, leaving scarce resources for video encoding and streaming. Therefore, the greater computational cost of H.264 could not be afforded after accounting for hardware acceleration. This is due to the maximum amount of RAM available on the RPi being allocated for the CPU to access, and subsequently `ArduPilot` to utilise, leaving only 16MB for the GPU. Therefore, MJPEG was utilised as the video codec. However, the outputted stream is less compressed compared to using H.264, and so the framerate, resolution and bitrate needed to be reduced to conserve bandwidth. Table 1 below shows a range of parameter values, and output traffic for a 5 second example video:

Resolution (w×h px)	Framerate (frame/s)	Bitrate (bit/s)	Output Traffic (KB/s)
1920×1080 (control)	60	340000000	4080
1920×1080	30	170000000	2040
1920×1080	15	85000000	1020
1280×720	60	240000000	3560
1280×720	30	120000000	1780
1280×720	15	60000000	890
640×480	60	110000000	1640
640×480	30	55000000	820

640×480	15	27500000	410
640×480	15	10000000	410
640×480	15	5000000	410
640×480	15	2500000	410
640×480	15	1500000	300
640×480	15	1000000	280

Table 1: Output bandwidth usage for differing video parameters

For each table row, the example video has been converted from the control video file via `FFmpeg`[55] with the presented parameters. The output traffic is calculated by dividing the file size of the converted video by 5, due to its length of 5 seconds.



Bitrate: 1000000 bit/s **Bitrate: 1500000 bit/s** **Bitrate: 2500000 bit/s**

Figure 10: Visual comparison of lower bitrates

Due to the greatly reduced bandwidth usage compared to higher resolutions, a resolution of 640×480 and framerate of 15 was chosen. Figure 10 shows the difference in video quality at lower bitrates with this resolution and framerate. At 1000000 bits/sec, there are significant visual artefacts, yet little reduction in bandwidth compared to 1500000 bits/sec. Thus, the bitrate of 1500000 bits/sec was chosen due to its further reduction in bandwidth usage with little visual artefacts.

The streaming pipeline terminates in a `udpsink` element directed at port 5600 on the ground station computer. This choice of port was guided by the documentation of existing ground control software, qGroundControl[56], which expects a UDP stream of video input on port 5600. Therefore, any multi-copter configured for this software should also function as expected with the software produced for this dissertation.

3.6 Simultaneous Localisation and Mapping

Multiple approaches to indoor localisation were presented in Section 2.4: Differential GPS (DGPS), Visual Odometry (VO) and Visual Simultaneous Localisation and Mapping (V-SLAM). Due to the need for users to utilise an external station to undertake DGPS, it was dismissed as a potential option. Since V-SLAM is built atop VO, it was decided to make use of an existing V-SLAM algorithm to provide position estimates, taking advantage of the most recent research in the field.

As Section 2.2 stated, only a singular RGB camera is installed on the multi-copter. Therefore, the V-SLAM algorithms that can be potentially utilised are restricted to those supporting RGB monocular input. Of these, two were shortlisted: `ORB-SLAM` and `LSD-SLAM`, both discussed in some detail in Section 2.4. Deciding which to utilise was simplified through comparing their robustness at recovering from loss of localisation. This is a key issue, since if localisation is lost for an extended period during flight, the multi-copter is likely to behave in an undefined manner. Since `LSD-SLAM` cannot recover without shifting the origin position, `ORB-SLAM` was chosen as the algorithm to be utilised.

To integrate the position estimates generated by `ORB-SLAM`, some understanding of how GPS data is handled within `ArduPilot` is required. To estimate the multi-copter's current position in 3D space, `ArduPilot` fuses accelerometer, gyroscope, magnetometer and barometer (collectively, IMU) data in an Extended Kalman Filter[57] (EKF), alongside optional GPS positioning data. This produces extensive outputs, including a quaternion defining the current attitude of the multi-copter along with current air-speed velocity, and as mentioned a 3D position.

At the time of architecture design, `ArduPilot` did not support input of visual position estimates into the EKF. As such, it was required to generate faux GPS co-ordinates from the estimates provided by `ORB-SLAM`, treating the position **`[0,0,0]`** as the origin in both the EKF's and ORB-SLAM's reference frames.

Due to the significant processing requirements `ORB-SLAM` demands, the flight computer cannot run this algorithm on-board. Therefore, due to the likely availability of a reasonably powerful CPU on the ground station computer, it was chosen to off-load `ORB-SLAM` to the ground station. Faux GPS co-ordinates can then be transmitted over MAVLink to `ArduPilot`'s EKF through the `GPS_INJECT_DATA` MAVLink message[58].

Off-loading positioning in this manner does lead to the multi-copter being tethered to the range of the WiFi link to the ground station computer. If flown beyond WiFi range, the multi-copter will not receive further positioning data, and will be forced to rely upon only IMU data to estimate its current position. However, this is not a concern for this project, since indoor flight restricts the distance the multi-copter can travel sufficiently to remain within coverage of the ground station computer.

Further to this, latency is incurred during localisation; the transmission time of faux GPS co-ordinates along with the computation time for localisation can potentially reach 500ms. Thus, the multi-copter will be forced to move at a low velocity to ensure it doesn't fly too far before its current position is correctly updated. However, this is not an issue, as a low velocity is required for indoor flight to allow for greater manoeuvrability in such a constrained environment.

3.7 Photogrammetry Pipeline

As stated in Section 2.6, the public notes of Falkingham (2017)[46] showed that the combination of `COLMAP` and `openMVS`, to generate a point cloud and densify it to a 3D model respectively, produced the most accurate end results. Therefore, `COLMAP` and `openMVS` were chosen to be integrated as an end-to-end pipeline to generate a 3D model as per the Project's aims.

To generate images for this pipeline, it was chosen to request the multi-copter to travel to specific waypoints in 3D space. These waypoints are simple 3D co-ordinates relative to the origin of the multi-copter's reference frame. `ArduPilot` provides the capability to undertake such waypoint-based navigation via a single MAVLink message containing a 3D co-ordinate, provided that input (faux) GPS data is both present and providing a reasonable degree of accuracy. At each waypoint, the multi-copter is commanded to rotate 360 degrees, stopping at 45-degree increments to record an image from the camera. A subsequent waypoint is then specified, repeating the process until sufficient image data is generated.

Once sufficient image data has been generated, the multi-copter is requested to land ideally at the origin of its reference frame. The photogrammetry pipeline is then undertaken to process the generated dataset, resulting in a 3D model presented to the user. This model is to be represented in the standardised Wavefront OBJ format[59], allowing for simplified exporting on the part of the user.

Throughout the flight, it was decided to visually represent the progress of generating the dataset on the GUI, in the form of rendering an up-to-date point cloud generated by `COLMAP`. However, this may prove difficult due to the time taken to generate a point cloud being exponential in accordance with the count of images in the dataset. More specifically, the time taken is $O(N^2)$, due to the need to match each image's features with all other images' features to triangulate each feature in this point cloud.

3.8 Flight Planning

To achieve the aim of autonomy, the waypoints utilised as discussed in Section 3.7 are to be generated automatically. The end goal is from the central position of a given indoor point cloud, all furthest edges should be interconnected high-density regions, assuming that they represent walls, the floor and ceiling of the interior space. Therefore, if a low-density region is adjacent to a high-density region, it can be assumed it has not yet been mapped fully.

Waypoints are generated in a best-effort approach to cover these less dense regions. For each new waypoint, the closest "area of disparity", a high-density region adjacent to a low-density region, to the multi-copter's current position is found. The new waypoint is then generated by taking the point of the high-density region closest to the low-density region at a pre-specified altitude, then translating it towards the most central point of the entire point cloud by 2 metres. An example of this process is found overleaf in Figure 11.

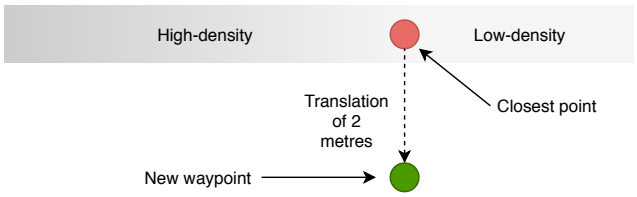


Figure 11: Example waypoint generation process

To gain access to a point cloud upon which to generate these waypoints, two approaches can be utilised. First is through the photogrammetry pipeline; `COLMAP` generates a point cloud through Structure-from-Motion, which can be updated with new images without the requirement of re-processing all previous images. The advantage here is that the point cloud generated is written to disk after processing new data and so can be readily read when required. However, the drawback is that time must be spent waiting for this cloud to be re-generated after accumulating new imagery, leading to the multi-copter idling whilst airborne and thus draining its on-board battery.

Alternatively, `ORB-SLAM`'s internal point cloud can be utilised. After features are extracted from input video frames, each is represented as a point in a sparse point cloud. Therefore, by extracting this point cloud, it will be possible to gain some understanding of the physical world upon which to generate waypoints. Due to localisation requiring an up-to-date point cloud, the multi-copter needn't wait for a new cloud to be generated; it is updated in real-time. However, this approach does have a major drawback owing to the usage of the FAST feature detector in `ORB-SLAM`. Since FAST detects only the corners of objects, the resultant point cloud will not provide representation of feature-less objects such as walls. Therefore, either the approach described previously will need to be adjusted to handle this issue of sparseness, or an alternative Visual SLAM (V-SLAM) algorithm that generates an appropriately detailed point cloud, such as `LSD-SLAM`, may be utilised.

It was chosen to utilise the point cloud generated through `COLMAP` to generate waypoints, due to the sparseness present in `ORB-SLAM`'s point cloud. That said, real-world testing will show whether utilising the point cloud of an alternate V-SLAM algorithm is more appropriate.

4 Implementation – Multi-Copter

4.1 Construction

Since the author began the Project with no prior knowledge base on how to build the multi-copter, and with little skill in electronics, there was significant degree of knowledge to obtain. This knowledge has been generated through a number of online resources, typically through practical-orientated articles on the topic, along with asking questions to an acquaintance of the author over the Twitter social media platform.

To begin construction, the author perused Liang's (Updated 2017)[14] article on the typical components required for a multi-copter. This gave a listing of components to source: a frame, Electronic Speed Controllers (ESCs), motors, and a battery. Choosing of these components was greatly guided by an article published by Jacobs (2017)[60], providing a tremendously useful flowchart reproduced in Figure 12 to the right.

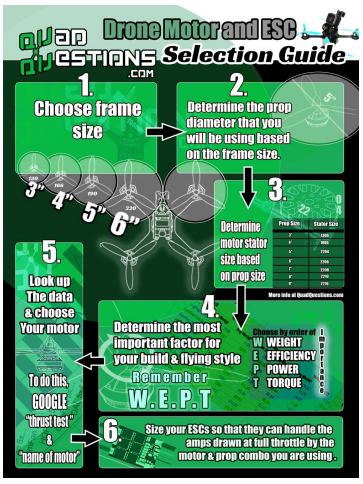


Figure 12: Flowchart for sourcing components
Source: Jacobs, A. (2017)[60]

A full listing of the components sourced can be found in the Bill of Materials presented in Appendix A. The cost of the multi-copter totalled **£235.48**, within the budget of £250.00 as defined in Section 2.2. For reference, the Parrot Bebop 2[61], which also communicates over MAVLink with similar hardware, retails at £449.99.

Once all components had been ordered and subsequently delivered, construction commenced. The steps followed are outlined in Appendix B. During the latter half of the Project, the author designed and 3D printed an enclosure for the Raspberry Pi and PXFmini assembly from ABS plastic. Photography of the components and finished multi-copter can be found below in Figure 13:

Figure 13: Photography of components, and constructed multi-copter

4.2 OS Configuration

4.2.1 Kernel

To build the operating system (OS) image for the multi-copter, the author first reverse engineered the configurations made by Erle Robotics in their example OS image. It was found a real-time pre-emption patch[26] was to be made to the Linux kernel, required by `ArduPilot` to improve the response time of the Linux kernel to physical events. To achieve this, the author cloned the Linux kernel via `git`, version 4.4.50 to be precise, inside a Linux virtual machine and applied the patch before cross-compiling the kernel for the Raspberry Pi (RPi). Whilst this is an older version of the Linux kernel at the time of writing, changes made to the handling of I^2C[62] devices in the 4.9 kernel tree cause issues with `ArduPilot` and so cannot yet be utilised.

The resultant kernel binaries were packaged inside a Debian archive[63] for easier installation on the RPi, and also allowing for easier distribution of these binaries to end users. Once the kernel

was installed, the author adjusted hardware settings on the RPi; `I²C`, `SPI`, serial over the GPIO pins and `uart` were all enabled. These are pre-requisites for allowing `ArduPilot` to communicate with the PXFmini addon board from the RPi.

4.2.2 Debugging Improvements

For improved debugging, the author opted to enable Ethernet over a USB connection to the RPi. A scarcely advertised feature of the RPi Zero specifically is that of USB On-The-Go mode[64]. This is a specification introduced in 2001 allowing a USB port to be dynamically configured as a USB master or as a USB slave. In terms of definitions, the former allows other USB devices (also: slaves) to connect to itself to then undertake I/O on them, with the latter being that of a USB device. In this case, the RPi was configured such that its micro-USB port acts as a USB slave, advertising a virtual Ethernet connection.

After configuring the micro-USB port of the RPi, a micro-USB to USB-A cable could be utilised to construct an Ethernet connection between the author's development machine and the RPi. This then allowed for an `SSH` connection to be undertaken over static IP addresses configured on both ends, without the need for a functional wireless connection. The benefits of this were found to be the "loosening of a tether" from pre-configured WiFi networks for `SSH` access into the RPi, along the ability to easily recover from incorrect WiFi configurations made in Section 4.2.3.

4.2.3 Wireless Communications

As mentioned in Section 3.3, the multi-copter generates its own access point for others to connect to. This was achieved through the combined usage of `hostapd` and `udhcpd`, providing the access point itself and IP address allocation via DHCP respectively. The notes of Chaharbakhshi (2017)[65] proved fruitful in configuring this access point, alongside simultaneously allowing the RPi to connect as a client to other WiFi networks. This capability is made possible through the chipset installed on the RPi for WiFi connectivity, and can be enabled through adding a new virtual wireless device for `hostapd` to broadcast on. Whilst additional configuration is required, simultaneously advertising an access point along with connecting to other networks allows for providing simpler Internet access to the RPi. Without it, there is a need to periodically switch between configurations for running as an access point or as a client when requiring Internet connectivity; this is not practical for development.

Alongside this configuration is the `hostapd` listener described in Section 3.3. This has been implemented as a script ran whenever device connection states change, and is provided to `hostapd` as a command-line option. The main state change observed is `AP-STA-CONNECTED`[66], communicating that a new device has connected to the access point. The script's logic is presented overleaf in Figure 14.

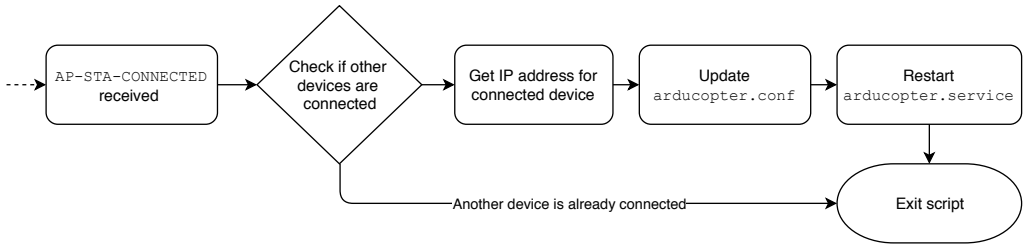


Figure 14: Logic of the `hostapd` listener script

The check if other devices are connected is a simple approach to ensuring new connections cannot hijack the multi-copter whilst in flight. For example, if a flight was in session when a new device connected, the resultant restart of `ArduPilot` to update configuration details would cause the multi-copter to behave in an undefined manner.

4.2.4 `ArduPilot`, `gstreamer` and the Remote Utility

Configuring `ArduPilot` in the operating system (OS) image was found to be simple. To start the flight stack, its binary is executed as `root`, with optional command line parameters. These parameters define the IP address of the ground station computer and where to write generated log files on the filesystem. A script was written to automate starting `ArduPilot` on system boot through a `systemd` service: `launch_arducopter.sh`, as referenced in Figure 8. The logic of this script is presented below in Figure 15:

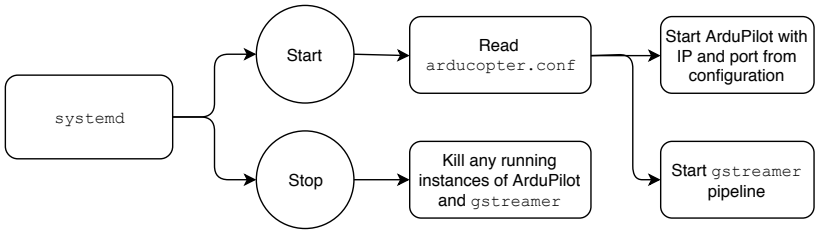


Figure 15: Logic of `launch_arducopter.sh`

After writing new configuration changes to `arducopter.conf`, the `hostapd` listener requests `systemd` to restart `ArduPilot` and `gstreamer`. This occurs by `systemd` executing the script with the *Stop* flow, and subsequently the *Start* flow as shown above.

To process ground station commands such as a request to reboot or shutdown the multi-copter, a Remote Utility was implemented in Python. This is a small WebSockets[67] server that listens for incoming messages on port 14551. Each message is a serialised dictionary object, containing a mandatory message identifier to differentiate incoming commands, along with arbitrary data relevant to each command. Communication is duplex; results of commands are sent back to the client over the same WebSockets connection.

4.3 Testing and Calibration

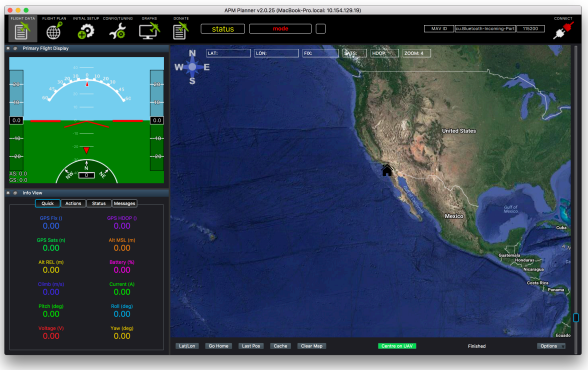


Figure 16: Screenshot of APM Planner 2.0

Once built, the multi-copter needed to be calibrated and then tested. To undertake this, the author utilised an existing MAVLink-aware program; APM Planner 2.0, with a screenshot available to the right in Figure 16. This software provides graphical output of telemetry data, allows for starting (also: arming) the motors, specifying waypoints, and also for controlling the multi-copter via a gamepad over a UDP link. It also provides the capability to calibrate the sensors on-board the multi-copter, which occurs by moving the multi-copter in all 6 degrees of freedom.

Outdoor testing of the multi-copter was undertaken; indoor testing was not yet an option, as the author was unsure of how responsive the multi-copter would be to input. This was a wise decision, as the first of these tests saw the multi-copter accelerating upwards rapidly and veering wildly to the left. Regarding test results, it was found that the motors began spinning once 50% throttle was reached, which was far more than necessary to conduct a hovering behaviour. Also, the gamepad used for manual control had tremendous latencies when manually controlling the multi-copter, with severe adverse results occurring on the second outdoor test. Suffice to say, the author recommends a risk assessment to be undertaken for future work, due to personal injuries acquired.

Calibration of the multi-copter's camera was also undertaken to find its distortion parameters. Modern pinhole cameras have a major drawback: constant image distortion, due to large-scale manufacturing processes[68]. This distortion is typically a systematic error, and so can be corrected through remapping after calculating a given camera's distortion parameters. This is of high importance due to the usage of `OpenCV` in `ORB-SLAM`, which expects input images to be undistorted before undertaking any processing. Calibration was achieved through the recommended approach for `OpenCV`: capture multiple images from differing angles of a checkerboard pattern with known size then execute the provided calibration script from `OpenCV`. The RPi camera module's technical specifications along with calibration results are found below in Table 2 and 3 respectively.

Sensor	Sony IMX219
Sensor Resolution	3280 × 2464 px
Pixel Size	1.12µm × 1.12µm
Focal Length	3.04 mm
Horizontal field of view	62.2 degrees
Vertical field of view	48.8 degrees

Table 2: Raspberry Pi Camera Module v2.1 specifications
Source: Raspberry Pi Foundation (2016)[69]

Focal length X	499.523
Focal length Y	498.172
Optical centre X	320.742
Optical centre Y	242.846

Table 3: Computed distortion parameters

5 Implementation – Ground Control Software

As stated in Section 3.4, the ground control software (GCS) has been implemented in Python as a module to `MAVProxy`, with the localisation subprocess utilising C++.

5.1 User Interface

For simplicity on the part of the user, all functionality provided in the GCS is presented in a GUI. This is implemented through the `wxPython` library, *"[a] cross-platform GUI toolkit for the Python language"* (wxPython Team, updated 2018)[70], and can be viewed below in Figure 17.

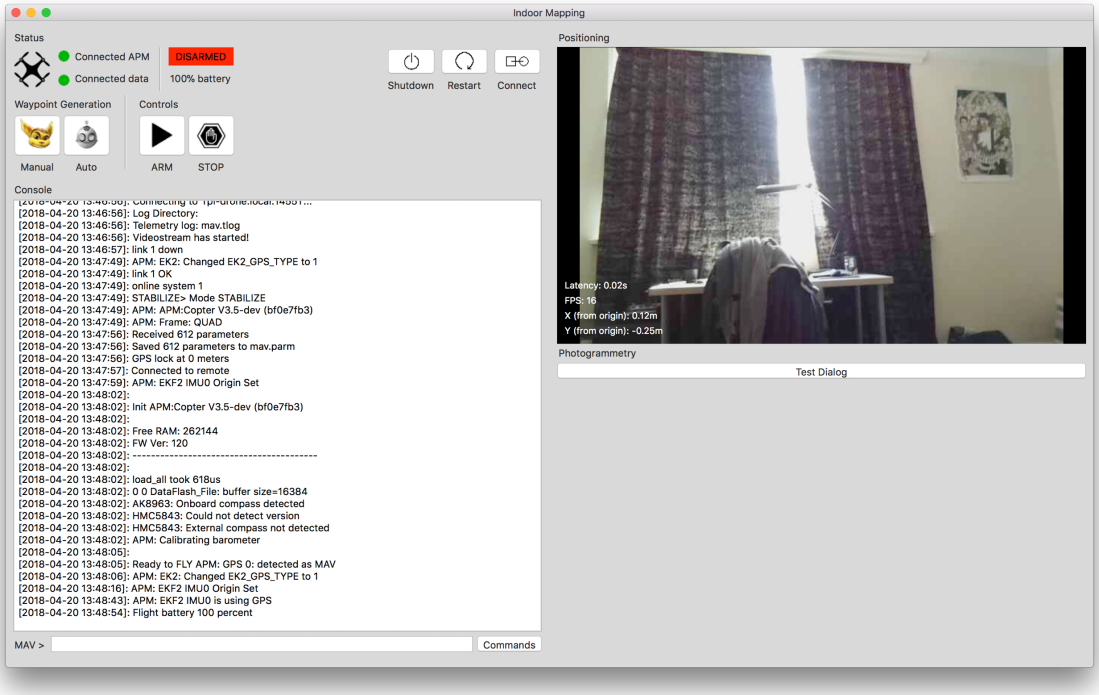


Figure 17: Screenshot of the developed GUI

The GUI itself is ran inside a subprocess to the main process of the GCS developed, with duplex inter-process communication achieved via UNIX pipes. Messages sent over this duplex link are serialised dictionary objects, each containing a hardcoded identifier string referencing a tuple of a sub-section (also: panel) of the GUI and a sub-module of the main process.

This messaging approach is not dissimilar to the Blackboard architecture for Artificial Intelligence, wherein *"a global database [...] mediates all communication within a system"* (Craig, 1995)[71]. The general idea is to place all data in a globally accessible manner, with a number of Knowledge Specialists acting on the data in turn to contribute towards an end solution. To compare to this Project, each message sent to the GUI is accessible by all panels, with a scheduler mediating access based upon the identifier string of the message.

The `DroneMainWindow` class handles this message scheduling via forwarding incoming messages to the appropriate panel based upon the identifier string, and ensures each discrete panel is drawn to the user's display correctly. Each panel is a separate subclass of `BasePanel`, ensuring a polymorphic approach to providing panel classes with expected base functionality. This base class provides an abstraction over sending messages from the GUI back to the GCS' main process, along with a default implementation of UI elements.

Note that the licensing requirements of the utilised iconography have been upheld through appropriate attribution in the source code.

5.2 System Status

Displaying the current status, such as battery conditions, of the multi-copter is of use both during development and to the end user. Furthermore, `MAVProxy` presents a command-line interface to the user by default, which becomes in-accessible if a GUI is visible. Therefore, two panels were created, both visible in the left-hand side of Figure 17: display of system status and important controls, and an interactive console. Two sub-modules within the GCS' main process were also created, handling the backend of each panel respectively. Each sub-module is sub-classed from `SubmoduleBase`, again providing convenience functionality to child classes.

The system status panel integrates both data received from regular MAVLink-based updates from the multi-copter, along with user-facing controls to the Remote Utility implemented in Section 4.2.4. These updates comprise of data such as the current battery level and current attitude of the multi-copter. Once received in the system status' sub-module, this data is sent as a message through to the corresponding panel. The user-facing controls of the Remote Utility are presented as three buttons: *Shutdown*, *Restart* and *Connect*. Once pressed, these buttons generate an appropriate message sent back to the system status' sub-module, to be subsequently parsed and forwarded to the multi-copter's Remote Utility server on port 14551.

Regarding the interactive console, the user is afforded two text regions: display of `stdout` messages, and a command input box. To feed the former with data, the sub-module handling the console overrides Python's standard implementation of `stdout`, and instead redirects each write to it into a message sent to the console panel. The textual content of this message is subsequently written to the display region. For the command input box, the user's current input is messaged to the backing sub-module on each press of the Return key, whereupon it is forwarded to `MAVProxy` to process.

5.3 Integration of `gstreamer`

Integration of `gstreamer` was achieved through implementing a "management" class: `VideoStreamManager`. Along with forwarding incoming video frames throughout the GCS, it also handles the construction of the client-side `gstreamer` pipeline, defined overleaf in Figure 18.

```
udpsrc port=5600 !
application/x-rtp,format=(string)RGB,media=(string)video,clock-
rate=(int)90000,encoding-name=(string)JPEG,a-framerate=(string)15.000000,a-
framesize=(string)640-480,payload=(int)26 !
rtpjpegdepay ! jpegparse ! jpegdec !
videoconvert ! video/x-raw,height=480,width=640,framerate=15/1,format=RGB !
dronemappingvideosink
```

Figure 18: `gstreamer` client-side pipeline

This pipeline comprises of multiple stages. First, the `source` element receives a UDP stream of data on port 5600. Next, this input is afforded metadata by a `caps` element, whereupon each frame of video is decoded into an in-memory JPEG through `rtpjpegdepay`, `jpegparse` then `jpegdec`. Finally, this in-memory representation is converted into an RGB bitmap, and passed through into a custom `sink` element: `dronemappingvideosink`. This final element has been implemented by the author using the Python-to-C bindings of `gstreamer`, encapsulating each incoming RGB bitmap into an object of type `GdkPixbuf` – available in the `GDK` Python library[72]. These encapsulated objects are then forwarded into the singleton instance of `VideoStreamManager`.

Ensuring that incoming frames are distributed throughout the GCS is achieved through "object registration". Each object expecting video input is maintained in an internal array of the `VideoStreamManager` singleton instance. Once a new video frame is available, this array is iterated through, calling `handle_videostream()` on each object in turn with the new frame as a parameter.

The conversion of the in-memory JPEG to RGB data was included as a result of a severe issue. Beforehand, as frames arrived they were encapsulated into a `GdkPixbuf` object correctly, though when displayed in the GUI each colour channel was rendered separately in vertical stripes. The cause of this was each frame of the MJPEG stream represented each colour channel as a separate contiguous block of pixel values. However, it was expected that the channels were instead combined as 24 bits per pixel; 8 bits per channel. The inclusion of the `videoconvert` pipeline element resolved this issue with no additional work on the part of the author.

5.4 ORB-SLAM

As stated in Section 3.6, integration of `ORB-SLAM` has been off-loaded to the ground station computer. This has been implemented with two major components, the first being an additional sub-module to the GCS' main process, registered to the singleton instance of `VideoStreamManager`. Alongside this is a subprocess: a simple wrapper around the C++ library of `ORB-SLAM`, communicating to the sub-module via UNIX pipes.

The format used for data interchange between the subprocess and sub-module is JSON. This allows for each video frame to be encoded as base64 and written to the `stdin` of the subprocess, with output from the subprocess undertaken via writing JSON encoded strings to its `stdout` via `printf()`. The output of the subprocess is either new 3D positions as computed by `ORB-`

SLAM, or information about the state of ORB-SLAM. This state information includes the current state of initialisation, and the current status of feature tracking.

As mentioned in Section 2.4, the output positions from ORB-SLAM are in an arbitrary reference frame due to the usage of monocular input. To remap this reference frame into metric units, the author implemented a simple, yet effective, solution. First, the user is requested to translate the multi-copter in 3D space to achieve initialisation of ORB-SLAM. Then, they are requested to place the multi-copter on a flat surface, to move the multi-copter rightwards by exactly 50cm, and finally forwards by the same distance. At the flat surface step, an average of the current position is taken for the X and Y position components in the arbitrary reference frame. At the end of the rightwards movement, a new average is taken for the X component, with the original component subtracted, an absolute value taken of the result, with it finally being multiplied by 2. The same process occurs for the Y component in the forwards direction. This produces two multiplicative factors that can be used to map incoming positions from ORB-SLAM into metric units. A recommendation for future work has been given by Dr Steven Bagley (2018)[73] to utilise an exact distance of 297mm. This is the length of an A4 sheet of paper, which provides an improved measurement guide due to ease of access for end users.

Real-world testing showed that the author's development machine can only run ORB-SLAM in real-time if the majority of incoming video frames are ignored. It is assumed this will also be the case for end users. Thus, only 3 video frames per second are forwarded to the localisation subprocess, whereupon they are additionally downscaled to a resolution of 360×240px before being processed by ORB-SLAM.

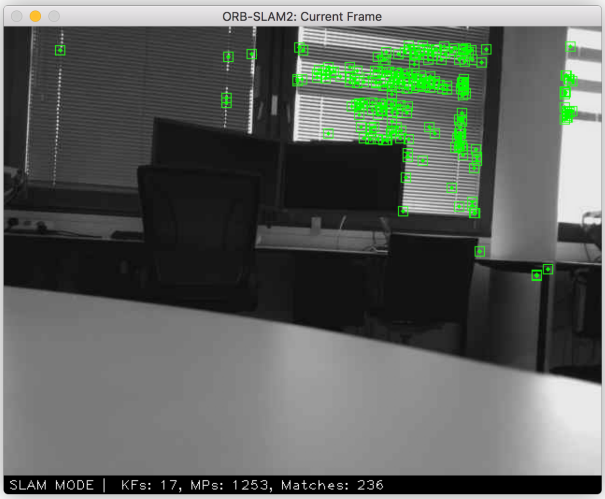


Figure 19: Visualisation of detected points in ORB-SLAM

Further issues were also found during real-world testing. Initially, the points extracted by ORB-SLAM were to be visualised in the GUI, with each frame of visualisation to be encoded as base64 and written to the subprocess' stdout. However, this was not possible – the volume of incoming data to the localisation sub-module caused corruption on Python's memory heap, leading to unexpected crashes. To resolve this, the author utilised OpenCV's support for visualisation to render these points in a separate window. An example of this visualisation can be observed in Figure 19 to the left.

5.5 Flight Control and Photogrammetry

One additional sub-module for the GCS was also implemented, providing both waypoint-based navigation and integration with the photogrammetry pipeline. This sub-module initially waits

upon a MAVLink message to indicate the motors have been started (also: armed), and subsequently requests a take-off via a `MAVProxy` API call. The system status panel provides a user-facing control to initiate this motor arming. Once the altitude of the multi-copter reaches ±10cm of 1 metre above the start position, the main loop for flight control is begun.

Flight control can be summarised as follows: rotate 360 degrees stopping at each 45-degree increment, request the next waypoint from a subclass of `BaseWaypointGenerator`, send the resulting waypoint over a MAVLink message to the multi-copter, wait until it is navigated to by `ArduPilot`, and then repeat the loop.

To rotate the multi-copter, data about the current attitude of the multi-copter is utilised. Before a MAVLink message is sent to the multi-copter with a new relative yaw angle, the current yaw angle of the multi-copter is recorded. A short busy loop is begun after the message is sent, with an end condition such that the current yaw angle subtracted from the recorded angle is ±1 degree. This ensures that no further commands are sent to the multi-copter until there is high confidence the rotation has fully completed. The time taken to undertake this yaw is specified in the MAVLink message to be 4 seconds, to ensure `ORB-SLAM` does not lose localisation tracking whilst undertaking the rotation. It was found during real-world testing that rapid rotation would lead to `ORB-SLAM` becoming disorientated, likely due to too few triangulated features present in the newly observed scene. This potentially may lead to undefined multi-copter behaviour.

Waypoint generation is provided by a subclass of `BaseWaypointGenerator`, in this case `ManualWaypointGenerator`, contained within the sub-module's instance. As can be inferred, implementation of automated waypoint generation was not completed at the time of writing. Instead, a small window is currently presented to the user to manually specify each new waypoint relative to the multi-copter's current spatial location.

The integration of the photogrammetry pipeline – `COLMAP` and `openMVS` – was completed to some degree. Whilst the process of dense reconstruction is implemented within the photogrammetry pipeline, no code is present to commence this after a flight has ended. Though, code is present to record images to the filesystem during a flight, allowing for dense reconstruction to still be undertaken manually. For this purpose, a separate shell script is present in the source code. Incidentally, this shell script was utilised to produce the example output displayed Figure 5.

Through utilising this shell script, it was found the generation of both sparse point clouds and the final dense reconstruction have a major issue: generation is CPU-bound on all operating systems but Windows, with the time taken for point generation being far too excessive as a result. Therefore, utilising this sparse cloud for autonomy purposes is be infeasible. Plus, the pre-existing implementation of `ORB-SLAM` prevents the photogrammetry pipeline from being undertaken for many users, due to its own CPU resource usage. Therefore, the multi-copter would be required to idle whilst airborne for multiple minutes towards the end of a flight, due to the size of the image dataset at that time, when requesting a waypoint.

Thus, future work will require transitioning to an alternative Visual SLAM algorithm, which must provide an improved point cloud to enable autonomous waypoint generation. Even after this adjustment, the photogrammetry pipeline may only be undertaken at the end of a flight; for any Visual SLAM algorithm, its subprocess must be first be exited to increase the availability of compute resources, which cannot occur during flight.

6 Evaluation

6.1 Test Flights for Implemented Ground Control Software

Once the majority of the ground control software (GCS) had been implemented, indoor test flights were undertaken to ascertain its quality.

The first flight saw a null result, due to `ArduPilot` refusing to arm the motors without a GPS fix. This was due to `ArduPilot`'s Extended Kalman Filter (EKF)[57] refusing `GPS_INJECT_DATA` messages containing faux GPS data from the GCS. This was resolved by tuning the `GPS_TYPE` parameter within `ArduPilot`, specifying a value of 14 to signify the EKF to accept these messages as GPS input. Furthermore, the messages themselves were required to be modified to provide a low, yet non-zero, value defining the accuracy of the faux GPS data. This was hardcoded to a value of 1, resulting in the EKF accepting faux GPS data without issue.

The second flight saw the motors again refusing to arm. `ArduPilot` provides a failsafe system to ensure a multi-copter returns to land if its battery level drops below a specified value. It was discovered the author has misconfigured this failsafe, leading to it triggering whilst already on land. Therefore, the motors refused to arm, due to `ArduPilot` believing the failsafe had successfully caused the multi-copter to land. The resolution was to adjust the voltage multiplier parameter applied to the detected battery voltage on-board the multi-copter, producing corrected battery level outputs above the failsafe's activation value.

The third test flight saw the motors correctly arming, but due to the high speed the propellers were rotating the author activated a kill-switch, implemented for safety reasons. Unfortunately, this lead to a rapid unexpected disassembly of the propellers, with all four making contact with the ceiling of the room used as a test site. Upon inspection, it was found the motors were installed on the multi-copter incorrectly, allowing the propellers to loosen their fastenings.

Due to the need to activate the kill-switch, the author has inferred the multi-copter constructed is overly powerful for indoor flight, to the degree of risk to personal safety. As a result, the code implementing flight control has not been tested to its fullest extent.

6.2 Localisation Accuracy

Of note is the accuracy of localisation afforded to the multi-copter. Provided the calibration procedure described in Section 5.4 produced correct multiplicative factors, empirical testing

showed accuracy to typically be within ±10cm whilst airborne. However, this accuracy is reduced to within ±2m when the multi-copter is placed such that a significant proportion of its view comprises of a flat, featureless surface. In effect, a greater accuracy was observed whilst the multi-copter was held aloft and can be assumed to be similar during flight.

In cases whereupon the multiplicative factors were not correct, real-world translations of 50cm producing remapped translations of up to 5m have been observed. This incorrectness typically results from placing the multi-copter centrally on a flat surface and translating it from this point for calibration. This introduces the inaccuracy of ±2m, causing the assumption that the magnitude of raw position data remains stable whilst calibrating to be false. No possibility to correct this error is present at the time of writing, requiring the GCS to be restarted to allow a new calibration procedure to be undertaken. This manual correction can be implemented at a future date, with a single button to request the localisation sub-module to restart calibration.

It is recommended that further work should utilise an RGB-D camera if available. This usage will avoid the need to remap calculated positions into the metric reference frame, thus reducing the potential for error in localisation to occur. That said, the calibration procedure provided for monocular input does ensure a suitably accurate remapping to metric units whilst airborne.

6.3 Safety and Privacy

As the initial outdoor testing outlined in 4.3 found, safety is a key concern when working with multi-copters. This typically relates to the exposed propellers for the production of thrust, which may spin at velocities great enough to cause injury; this is the case with the multi-copter built. Therefore, multiple safety measures are in place.

Primarily, a kill-switch is provided to the user in the event of an emergency. This is implemented as the sending of a MAVLink message to land at the current location, intended to override any other command that may be in progress. In the GUI, this is presented as a button appropriately entitled *Stop*. Furthermore, a hardware-based kill-switch has been configured for the manual flight transmitter used during testing. This transmitter has been provided by the external sponsor, Dr Michael Pound, due to the issues found with the gamepad initially used for outdoor testing as in Section 4.3. This will still function if the multi-copter leaves WiFi coverage to the ground station computer and was required to be utilised in the testing described in Section 6.1. The multi-copter also has been configured to require such a connected transmitter, to ensure that manual control can always be provided if necessary.

Furthermore, `ArduPilot` itself has a set of in-built safety features. These features can be generalised to be failsafes, with one such failsafe encountered during indoor testing. These failsafes may be triggered via: a low battery level, a lack of connection to the ground station computer, or bad heading estimates[74]. This list is not exhaustive; other failsafes not useful for this project are also available. Upon triggering of any, the multi-copter will either immediately attempt a landing, or loiter in place. For this project, this behaviour has been configured to only

attempt a landing, due to the loiter behaviour requiring GPS data. This data will not be present if outside WiFi coverage.

Since data is produced by the installed camera, privacy of others must be respected. The data produced by the camera is only recorded for the purposes of undertaking photogrammetry on the resultant images. That said, it is indeed being recorded. Therefore, to protect privacy when undertaking testing during development, image capture for the photogrammetry pipeline was only enabled whilst no other person was present at the test site. Due to the photogrammetry pipeline being the last major component developed, image capture was not a concern throughout the majority of development. With regards to `ORB-SLAM` and its usage of camera data, its internal point cloud is not saved to the filesystem. Therefore, no abuse can occur from the resultant output of its image processing.

When utilising the implemented GCS as an end-to-end system to map an environment, the author highly recommends ensuring the indoor space to be emptied of others prior to a flight. If this is not possible, it is strongly urged to ensure others are aware a flight is to be undertaken, for both privacy and safety reasons.

6.4 Legal Considerations

Multi-copters and other aerial hobbyist vehicles are an emerging concern in legislation. The UK Civil Aviation Authority (CAA) has published a set of guidelines[75] for hobbyist flight, summarised as: fly at least 50 metres away from other people and property, fly below 120 metres, and to remain outside any restricted airspace including airports. The author was required to find a suitable location when undertaking outdoor testing. Such a location was found on the Jubilee Campus of the University of Nottingham. This location is marked on the map displayed in Figure 20. Note that the guidelines for indoor flight are equal to those for outdoors, though with the added benefit that it is easier to control access to the venue a flight is occurring within.

Figure 20: Location of outdoor flights
Map data © Google 2018: *maps.google.com*

Over the summer of 2017, the UK government defined new incoming regulations for aerial hobbyist vehicle owners[76]. Once in effect, these regulations will require owners to register any aerial vehicle they own weighing over 250 grams, and to sit a safety awareness exam. This legislation was not introduced during the Project, and so did not pose any restriction on development. However, the author will be required to undertake both of these additional ownership steps at a later date, since the constructed multi-copter weighs roughly 600 grams.

7 Summary and Reflections

7.1 Coverage of Objectives

A key measure of this dissertation's success can be derived through comparing the implemented software project against the expected objectives laid out in Section 1.3.

The first objective was to implement duplex communications between the multi-copter and the ground control software (GCS). This has been implemented through the tight integration with `MAVProxy`, deriving from the choice to utilise `ArduPilot` as the flight stack and MAVLink as the communication protocol. Furthermore, the implementation of the Remote Utility as detailed in Section 4.2.4 extends this integration, allowing for communicating directly to the underlying OS of the multi-copter through which to enact system commands.

Second was that of providing effective indoor localisation. This is arguably the most significant component of the implemented software project. This is due to the requirement of configuring and accessing an incoming video stream through `gstreamer`, along with subsequently piping each individual video frame into the integration with `ORB-SLAM`. Effectively, the implementation of localisation provides the required intricate inter-communication between myriad existing works, alongside a solution to the metric scale problem associated with monocular input to `ORB-SLAM`.

The third objective was to integrate with an existing photogrammetry library for 3D model production for the end system output, along with an interim model for autonomous waypoint generation. This has been completed to an extent; `COLMAP` and `openMVS` have been integrated to the degree such that dense reconstruction can be undertaken manually. However, neither an interim 3D model nor a point cloud is outputted such that intelligent reasoning for flightpaths can be undertaken. This is due to a late decision to focus upon instead providing manual waypoint generation, to afford some useful utility to the final GCS.

Fourth was to autonomously produce a flightpath for the multi-copter to follow, such that a useful image dataset for photogrammetry can be produced. Additionally, the fifth objective was to assess differing algorithms producing flightpaths on their output dataset quality. These objectives were instead adapted to focus upon manual waypoint generation, due to the rapidly approaching dissertation deadline; focus was required upon the authoring of the dissertation document. As a result, the fourth objective has been implemented with modifications as described in Section 5.5, whereupon the user is requested to manually provide a new waypoint relative to the multi-copter. However, there is no possibility to assess the resultant flightpath as per objective five. This requires assessment of the user's judgement regarding the next waypoint, falling within the field of Psychology.

7.2 Project Management

7.2.1 Time Management

As can be inferred, the progress of the Project did not coincide with some milestones set out in the dissertation proposal document. This discrepancy has been driven by two factors: an underestimation of the time commitments required in the fourth year of university, along with an underestimation of the intricacy inherent in constructing a multi-copter. The latter caused the largest disruption; the initial milestone for the completion of construction was due to be reached by the end of October 2017. In actuality, this was mid-December 2017. This is due to this milestone not accounting for the need for outdoor testing and subsequent re-tuning of the multi-copter, with the latter being unexpected. As a result, all other milestones were forced backwards by two months.

Much effort was invested into recovering this loss of time. This was a factor in the decision of utilising `MAVProxy` as a basis to the ground control software (GCS), leading to the time required for implementing duplex communication being greatly reduced. As a result, other objectives of the dissertation could still be met in the remaining time, due to their reliance on a functional duplex link to the multi-copter. Outside of this decision, the usage of ZenHub as described in Section 3.1 for task organisation has been instrumental in ensuring that a degree of time management has been upkept in the face of this time discrepancy. In the course of his previous dissertation, the author learnt to *"generate a list of tasks for the coming weeks, with a granularity of to the day if needs be, taking into account variance that can occur from unexpected issues"* (Clarke, 2017)[77] through which to balance time. Each milestone was thus decomposed into separate tasks and added to the task pipeline offered by ZenHub. A useful feature provided by this software was found to be that of "Epics": a collection of tasks, with each milestone comprising an Epic with its constituent tasks grouped together.

Though, due to underestimation of other time commitments associated with the fourth year of university, only a modicum of time was reclaimed. These commitments were typically that of coursework, which themselves unexpectedly spanned multiple weeks. Thus, this lead to less time availability for the dissertation than had been planned. Furthermore, much of the developmental work on the GCS, such as for localisation, required a live connection to the multi-copter. Therefore, this necessitated access to the multi-copter which was not always possible; it was typically stored at the author's residence. As a result, even if a window of time became available for development such as between teaching lectures, the journey time required for hardware access did not allow for development to reasonably occur.

On reflection, the allocation of time provided in both the dissertation proposal and the interim report was appropriate given the knowledge available at the time. It was not known ahead-of-time about the intricacy involved in multi-copter construction, nor were deadlines of other commitments published. These deadlines were not assumed to be non-existent however, with the period of March 2018 through to the start of April 2018 defined as covering them within the Gantt chart produced in the dissertation proposal document. With the industrial action

throughout the latter semester of the dissertation[78] came adjusted external deadlines, leading to the balance of workload also shifting to suit. This lead to greater time being spent on courseworks to take advantage of these adjusted deadlines, in lieu of the dissertation's software project.

7.2.2 External Sponsor

As outlined in Section 1.5, Dr Michael Pound of the University of Nottingham was contacted to be an external sponsor to the Project. Initially, his role was to be advisory, providing input as to the most appropriate approach in which to develop the ideas of the dissertation for the hobbyist multi-copter community.

However, due to his expertise in the construction and flight of aerial hobbyist vehicles, it was found his role became that of an instructor, providing useful insights into improving the construction of the multi-copter. Furthermore, he provided the transmitter mentioned in Section 6.3, allowing for improved testing of the multi-copter. This has proved instrumental in ensuring the hardware constructed was tested to a suitable degree.

7.2.3 Supervisory Meetings

Throughout the duration of the dissertation, supervisory meetings with Dr Steven Bagley have been held weekly, except during the month of January. As stated in Section 3.1, these meetings were typically treated as end-of-sprint meetings, and as such revolved around progress updates given to the supervisor. During the final weeks of the dissertation, meetings typically focused upon a holistic overview of the implemented software project. This allowed for a discussion on the focus given by the dissertation document to varying aspects of the software project, alongside the recommendation of useful references.

7.3 Future Development

Due to the unfinished state of the dissertation's software project, more work post-submission must be undertaken before it is of full utility to the general hobbyist multi-copter community. To foster a collaborative approach to this post-submission work, the author has published the software produced under the GNU Public License version 2 (GPLv2), chosen due to `MAVProxy` also being licensed under the GPLv2 and thus restricting choice of licensure. The URL to the published `git` repository can be found in Section 8.1.

To ease future development efforts, multiple measures have been taken. The first is that of autonomous waypoint generation. Stubs for its corresponding subclass of `BaseWaypointGenerator` have been provided, removing the need to design the subclass before its implementation, and thus accelerating its future development. In addition, the build process for the entire project has been automated. This comprises of two shell scripts: the first automatically installs any required dependencies such as `wxPython` and `gstreamer`, albeit for macOS only. The second automates the building of each discrete system component. First, the `CMake` build system for

the localisation subprocess is undertaken, with the resulting binary packaged as the localisation subprocess. Next, the Python setup script for `MAVProxy` is ran, producing a runnable software package containing all implemented components. As a result, any changes to the implemented software project can be tested by executing this second shell script.

By publishing source code, some utility is afforded to the hobbyist multi-copter community even without a full working system. Work had been begun by Dudley (2017)[79] to integrate `ORB-SLAM` into APM Planner 2.0, with the reference implementation achieved by the author providing a beneficial springboard into completing this work. Some modification will be required to integrate into APM Planner 2.0 itself, though is not an impossible task if so desired.

Furthermore, at the time of implementation `ArduPilot` did not support receiving raw visual position estimates, leading to the usage of faux GPS co-ordinates as described in Section 3.6. As of March 8th 2018 however, support for the `GLOBAL_VISION_POSITION_ESTIMATE` MAVLink message[80] was added to the flight stack. This accepts raw position estimates in even an arbitrary reference frame, removing the need for faux GPS co-ordinates entirely. The author recommends that future work should transition to sending positioning data via this message, though the included `ArduPilot` binary in the OS image for the multi-copter will need updating.

Regarding multi-copter construction, a viable alternative for future work would be to purchase a small WiFi-controlled multi-copter. This will require bridging `ArduPilot` over a reverse-engineered connection to such a multi-copter, though would likely consume less time than the approach taken in this dissertation. As a result, a greater focus on GCS implementation would be possible. Furthermore, this addresses the issue of the constructed multi-copter being overly powerful as found in Section 6.1, with a smaller hardware platform posing a lesser personal safety risk. Unfortunately, this was not a viable solution for this dissertation, as the safety issue with the constructed multi-copter was discovered in the final weeks of development.

7.4 Achievements and Contributions

In the course of such an overly ambitious software project, much has been achieved. A holistic achievement is that of the implemented GCS; even with adversity in the form of time discrepancy, it provides functional indoor localisation without GPS, and a manual approach to photogrammetric indoor mapping. To achieve this in part, a deep understanding of the interplay between the hardware and software of a modern multi-copter has been required of the author. This understanding lead to a greater appreciation of Linux systems, such as that of providing new functionality to be started on system boot through `systemd`. Regarding hardware specifically, the author has learnt to solder components such as GPIO pins and wire connectors to produce effective electrical contacts. Amusingly, adeptness in soldering only arose after a heat-related injury was sustained from the soldering iron.

Aside from the multi-copter's OS image, the majority of the GCS has been implemented in Python. Prior to the commencement of this dissertation, the author had not utilised this

language other than for small learning exercises in 2010. The resultant system architecture of the GCS is a testament to the rapid familiarity gained by the author for Python. The full gamut of object-orientated paradigms are present, alongside usage of language-specific features such as pickling and slicing, for object serialisation and for simplified substring obtaining respectively.

In terms of the discrete components comprising the GCS, the implemented localisation component provides the greatest contribution to the wider world. By adhering to existing standards for video stream input, and utilising cross-platform technologies such as `gstreamer` and indeed Python, the author has produced a reference implementation for providing visual position estimates to `ArduPilot`. The solution to the arbitrary reference frame is of note, due to its simple calibration procedure. Even in the case no other component is utilised by the multi-copter community, this reference implementation can be ported to other software such as APM Planner 2.0 with little effort.

Aside from the merits of the implemented GCS, further appreciation for time management has been gained. Due to the mis-estimated initial milestone for multi-copter construction, an effort has been made to analyse its root cause. As Section 7.2.1 notes, this was as a result of not anticipating the need for additional re-tuning of the multi-copter to allow indoor flight. Ergo, it was driven by a lack of prior knowledge. Future projects can combat this through researching alternative approaches to tasks relying on newly gained knowledge. This research can be combined with a time assessment analysis, undertaken perhaps weekly, to ascertain whether these alternative approaches should be utilised in lieu of the current approach. In the case of this dissertation, an alternative approach would be to reverse-engineer a small WiFi-connected multi-copter rather than constructing the multi-copter, as suggested in Section 7.3.

7.5 Final Remarks

In summary, every effort has been made to meet the objectives defined by this dissertation throughout the implementation of the software project. In the cases of the fourth and fifth objectives, adaptions were undertaken to provide an end software package that can be extended to meet the full project aim. Whilst it is unfortunate that this end package could not be tested with the constructed multi-copter due to safety concerns, measures have been put into motion to ensure future development can meet the original aim of autonomous indoor mapping.

At the commencement of the dissertation, the author would have found this document wholly invaluable, detailing potential issues that must be overcome. Perhaps a different approach would be taken regarding hardware, though without failure no experience can be gained. Regarding experience, this has entailed the learning of a new programming language, understanding of hardware specific to the domain of aerial vehicles, alongside the most current work in robotics research. It is upon this breadth of experience the author is able to state this dissertation has been rewarding, giving a greater appreciation for both self-management, and for the undertaking of large-scale software projects.

8 Bibliography

8.1 Further Reading

MAVLink supported messages
Meier, L. (Updated 2018). *MAVLINK Common Message Set*. Available at: http://mavlink.org/messages/common. [Accessed 18th April 2018].

Published `git` repository
Clarke, M. (2018). *Indoor 3D Mapping*. Available at: https://github.com/Matchstic/indoor-3d-mapping. [Accessed 22nd April 2018].

8.2 References

1. Google Inc. (2005). *Google Maps*. Available at: https://www.google.co.uk/maps. [Accessed 16th April 2018].

2. Schenk, T. (2005). *Introduction to Photogrammetry*. Available at: http://www.mat.uc.pt/~gil/downloads/IntroPhoto.pdf. [Accessed 16th April 2018].

3. Al Khalil, O., Grussenmeyer, P., and Nour El Din, M. (2001). *3D Indoor Modelling of Buildings Based on Photogrammetry and Topologic Approaches*. In: XVIII CIPA International Symposium. p1-7.

4. Pix4D SA. (2017). *Indoor Mapping Game Plan*. Available at: https://pix4d.com/indoor-mapping-game-plan/. [Accessed 12th October 2017].

5. Du, H., Henry, P., Ren, X., Cheng, M., Goldman, D., Seitz, S. M., and Fox, D. (2011). *Interactive 3D Modelling of Indoor Environments with a Consumer Depth Camera*. In: Proceedings of the 2011 ACM Conference on Ubiquitous Computing. p75-84.

6. Microsoft Robotics. (2012). *Kinect Sensor*. Available at: https://msdn.microsoft.com/en-gb/library/hh438998.aspx. [Accessed 16th April 2018].

7. Díaz-Vilariño, L., Khoshelham, K., Martínez-Sánchez, J., and Arias, P. (2015). *3D Modelling of Building Indoor Spaces and Closed Doors from Imagery and Point Clouds*. In: Sensors. 15 (2), p3491-3512.

8. Budroni, A. and Boehm, J. (2010). *Automated 3D Reconstruction of Interiors from Point Clouds*. In: International Journal of Architectural Computing. 8 (1), p55-73.

9. LiDAR UK. (2011). *How does LiDAR work?* Available at: http://www.lidar-uk.com/how-lidar-works/. [Accessed 16th April 2018].

10. Lehtola, V. V., Kaartinen, H., Nüchter, A., Kaijaluoto, R., Kukko, A., Litkey, P., Honkavaara, E., Rosnell, T., Vaaja, M. T., Virtanen, J.-P., Kurkela, M., El Issaoui, A., Zhu, L., Jaakkola, A., and Hyyppä, J. (2017). *Comparison of the Selected State-Of-The-Art 3D Indoor Scanning and Point Cloud Generation Methods*. In: Remote Sensing. 9 (8), 796.

11. Durrant-Whyte, H. and Bailey, T. (2006). *Simultaneous localization and mapping: part I*. In: IEEE Robotics & Automation Magazine. 13 (2), p99-110.

12. Skydio, Inc. (2018). *Technology*. Available at: https://www.skydio.com/technology/. [Accessed 16th April 2018].

13. Franklin, D. (Updated 2018). *Jetson TX2*. Available at: https://elinux.org/Jetson_TX2. [Accessed 16th April 2018].

14. Liang, O. (Updated 2017). *Complete Mini Quad Parts List – FPV Quadcopter Component Choice*. Available at: https://oscarliang.com/250-mini-quad-part-list-fpv/. [Accessed 16th April 2018].

15. Upton, E. (2017). *New product! Raspberry Pi Zero W joins the family*. Available at: https://www.raspberrypi.org/blog/raspberry-pi-zero-w-joins-family/. [Accessed 12th October 2017].

16. Emlid Ltd. (Updated 2018). *Navio2*. Available at: https://emlid.com/navio/. [Accessed 17th April 2018].

17. Erle Robotics. (Updated 2017). *An open autopilot daughter-board for the Raspberry Pi Zero.* Available at: http://erlerobotics.com/blog/pxfmini/. [Accessed 7th December 2017].

18. Scanse LLC. (2017). *Affordable Scanning LiDAR for Everyone*. Available at: http://scanse.io/. [Accessed 17th April 2018].

19. Intel Corporation. (2018). *Intel® RealSense™ Depth Camera D435*. Available at: https://click.intel.com/intelr-realsensetm-depth-camera-d435.html. [Accessed 17th April 2018].

20. Intel Corporation. (2016). *Intel® RealSense™ Camera SR300*. Available at: https://software.intel.com/en-us/realsense/sr300. [Accessed 17th April 2018].

21. Raspberry Pi Foundation. (2016). *Camera Module V2*. Available at: https://www.raspberrypi.org/products/camera-module-v2/. [Accessed 17th April 2018].

22. Pagnutti, M. A., Ryan, R. E., Cazenavette, G. J., Gold, M. J., Ryan Harlan, R., Leggett, E., and Pagnutti, J. F. (2017). *Laying the foundation to use Raspberry Pi 3 V2 camera module imagery for scientific and engineering purposes*. In: Electronic Imaging. 26 (1).

23. ArduPilot. (2016). *ArduPilot :: About*. Available at: http://ardupilot.org/about. [Accessed 7th December 2017].

24. PX4 Dev Team. (Updated 2017). *PX4: The Professional Autopilot*. Available at: http://px4.io/. [Accessed 17th April 2018].

25. Dronesmith Technologies. (2016). *Why We Chose PX4 (vs APM) as Luci's Default Firmware.* Available at: https://medium.com/@Dronesmith/why-we-chose-px4-vs-apm-as-lucis-default-firmware-ea39f4514bef. [Accessed 17th April 2018].

26. The Linux Foundation. (2017). *HOWTO setup Linux with PREEMPT_RT properly*. Available at: https://wiki.linuxfoundation.org/realtime/documentation/howto/applications/preemptrt_setup. [Accessed 7th December 2017].

27. Meier, L. (2009). MAVLink Developer Guide. Available at: https://mavlink.io/en/. [Accessed 7th December 2017].

28. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., and Ng, A. (2009). *ROS: an open-source Robot Operating System*. In: ICRA Workshop on Open Source Software. 3.

29. International Organization for Standardization. (2015). *ISO 11898-1:2015 – Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling*. 2nd Edition.

30. The Internet Engineering Task Force. (2016). *RFC 7826 – Real-Time Streaming Protocol Version 2.0*. Available at: https://tools.ietf.org/html/rfc7826. [Accessed 17th April 2016].

31. PX4 Dev Team. (Updated 2018). *uORB Messaging*. Available at: https://dev.px4.io/en/middleware/uorb.html. [Accessed 17th April 2018].

32. Dronecode Project, Inc. (Updated 2018). *What is Dronecode?* Available at: https://www.dronecode.org/about/. [Accessed 17th April 2018].

33. Misra, P. and Enge, P. (2011). *GPS: Signals, Measurements, and Performance*. 2nd Edition. Massachusetts, US: Ganga-Jamuna Press.

34. Mertikas, S. P. (1983). *Differential Global Positioning System Navigation: A Geometrical Analysis*. MSc Thesis. University of New Brunswick, Department of Surveying Engineering. Canada.

35. Nistér, D., Naroditsky, O., and Bergen, J. (2004). *Visual Odometry*. In: Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. 1.

36. Cheng, Y., Maimone, M., and Matthies, L. (2005). *Visual odometry on the Mars exploration rovers*. In: 2005 IEEE International Conference on Systems, Man and Cybernetics. 1, p903-910.

37. Yousif, K., Bab-Hadiashar, A., and Hoseinnezhad, R. (2015). *An Overview to Visual Odometry and Visual SLAM: Applications to Mobile Robotics*. In: Intelligent Industrial Systems. 1 (4), p289-311.

38. Mur-Artal, R., Montiel, J. M. M., and Tard'os, J. D. (2015). *ORB-SLAM: A Versatile and Accurate Monocular SLAM System*. In: IEEE Transactions on Robotics. 31 (5). p1147-1163.

39. Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). *ORB: an efficient alternative to SIFT or SURF*. In: IEEE International Conference on Computer Vision. p2564-2571.

40. Engel, J., Schöps, T., and Cremers, D. (2014). *LSD-SLAM: Large-Scale Direct Monocular SLAM*. In: Proceedings of the 2014 European Conference on Computer Vision. p834-849.

41. GStreamer Team. (Updated 2018). *GStreamer: open source multimedia framework*. Available at: https://gstreamer.freedesktop.org/. [Accessed 17th April 2018].

42. Bala, S. (2016). *H.264 is Magic*. Available at: https://sidbala.com/h-264-is-magic/. [Accessed 17th April 2018].

43. Library of Congress. (2016). *MJPEG (Motion JPEG) Video Codec*. Available at: https://www.loc.gov/preservation/digital/formats/fdd/fdd000063.shtml. [Accessed 17th April 2018].

44. Micheletti, N., Chandler, J. H., and Lane, S. N. (2015). *Structure from Motion (SfM) Photogrammetry. In: Geomorphological Techniques*. Chapter 2 (Sec. 2.2), p1-12.

45. Furukawa, Y. and Hernández, C. (2015). *Multi-View Stereo: A Tutorial*. In: Foundations and Trends in Computer Graphics and Vision. 9 (1-2), p1-148.

46. Falkingham, P. (2017). *Free photogrammetry software review: 2017*. Available at: https://pfalkingham.wordpress.com/2017/12/17/free-photogrammetry-software-review-2017/. [Accessed 18th April 2018].

47. Schönberger, J. L. and Frahm, J.-M. (2016). *Structure-from-Motion Revisited*. In: Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition. p4104-4113.

48. OpenMVS authors. (Updated 2017). *Open Multi-View Stereo Reconstruction Library*. Available at: https://cdcseacave.github.io/openMVS/. [Accessed 18th April 2018].

49. Scrum Alliance. (2013). *What Is Scrum?* Available at: https://www.scrumalliance.org/why-scrum. [Accessed 7th December 2016].

50. ZenHub. (Updated 2017). *A better way to manage your GitHub Issues*. Available at: https://www.zenhub.com/. [Accessed 7th December 2017].

51. Erle Robotics S.L. (Updated 2018). *Erle Robotics | Bringing the next generation of robotic brains*. Available at: http://erlerobotics.com/blog/. [Accessed 18th April 2018].

52. Mojang Synergies AB. (Updated 2018). *Official site | Minecraft*. Available at: https://minecraft.net/en-us/. [Accessed 18th April 2018].

53. freedesktop.org. (Updated 2017). *systemd System and Service Manager*. Available at: https://www.freedesktop.org/wiki/Software/systemd/. [Accessed 18th April 2018].

54. CanberraUAV. (2012). *A UAV ground station software package for MAVLink based systems.* Available at: http://ardupilot.github.io/MAVProxy/html/index.html. [Accessed 7th December 2017].

55. FFmpeg Developers. (Updated 2018). *A complete, cross-platform solution to record, convert and stream audio and video.* Available at: http://ffmpeg.org/. [Accessed 18th April 2018].

56. QGroundControl. (Updated 2017). *Intuitive and Powerful Ground Control Station for PX4 and ArduPilot UAVs.* Available at: http://qgroundcontrol.com/. [Accessed 18th April 2018].

57. ArduPilot Dev Team. (2016). *EKF2 Estimation System*. Available at: http://ardupilot.org/dev/docs/ekf2-estimation-system.html. [Accessed 18th April 2018].

58. Meier, L. (Updated 2018). *GPS_INJECT_DATA*. Available at: http://mavlink.org/messages/common#GPS_INJECT_DATA. [Accessed 18th April 2018].

59. fileformat.info. (2013). *Wavefront OBJ File Format Summary*. Available at: http://www.fileformat.info/format/wavefrontobj/egff.htm. [Accessed 18th April 2018].

60. Jacobs, A. (2017). *How to choose the right size motors & ESCs for your Drone, quadcopter, or Multirotor build*. Available at: https://quadquestions.com/blog/2017/02/22/choose-right-size-motors-drone/. [Accessed 7th December 2017].

61. Parrot SA. (Updated 2017). *Parrot Bebop 2.* Available at: https://www.parrot.com/global/drones/parrot-bebop-2. [Accessed 7th December 2017].

62. NXP Semiconductors. (2014). *I²C-bus specification and user manual.* 6th Edition. Available at: https://www.nxp.com/docs/en/user-guide/UM10204.pdf. [Accessed 18th April 2018].

63. Suchanek, M. (2012). *Debian Repository Format.* Available at: https://wiki.debian.org/DebianRepository/Format. [Accessed 7th December 2017].

64. USB Implementers Forum (2001). *On-The-Go Supplement to the USB 2.0 Specification*. Rev. 1. Available at: http://www.usb.org/developers/onthego/otg1_0.pdf. [Accessed 18th April 2018].

65. Chaharbakhshi, A. (2017). *Raspberry Pi Zero W Simultaneous AP and Managed Mode Wifi*. Available at: https://albeec13.github.io/2017/09/26/raspberry-pi-zero-w-simultaneous-ap-and-managed-mode-wifi/. [Accessed 19th April 2018].

66. Mitchell, M. (2016). *hostapd: execute a command when there is new connection established*. Available at: https://superuser.com/a/1152298. [Accessed 19th April 2018].

67. Internet Engineering Task Force. (2011). *The WebSocket Protocol*. Available at: https://tools.ietf.org/html/rfc6455. [Accessed 7th December 2017].

68. Pagnutti, M. A., Ryan, R. E., Cazenavette, G. J., Gold, M. J., Ryan Harlan, R., Leggett, E., and Pagnutti, J. F. (2017). *Laying the foundation to use Raspberry Pi 3 V2 camera module imagery for scientific and engineering purposes*. In: Electronic Imaging. 26 (1).

69. Raspberry Pi Foundation. (2016). *Camera Module*. Available at: https://www.raspberrypi.org/documentation/hardware/camera/README.md. [Accessed 19th April 2018].

70. wxPython Team. (Updated 2018). *Welcome to wxPython!* Available at: https://www.wxpython.org/. [Accessed 19th April 2018].

71. Craig, I (1995). *Blackboard Systems.* New Jersey, US: Ablex Publishing Corporation. p15-51.

72. The GTK+ Team. (Updated 2017). *GdkPixbuf.Pixbuf.* Available at: https://lazka.github.io/pgi-docs/GdkPixbuf-2.0/classes/Pixbuf.html. [Accessed 19th April 2018].

73. Bagley, S. (2018). Private Conversation with Matthew Clarke. University of Nottingham, Department of Computer Science. UK.

74. ArduPilot Dev Team. (2016). *Failsafe.* Available at: http://ardupilot.org/copter/docs/failsafe-landing-page.html. [Accessed 20th April 2018].

75. UK Civil Aviation Authority. (Updated 2017). *The Drone Cod*e. Available: http://dronesafe.uk/wp-content/uploads/2016/11/Dronecode.pdf. [Accessed 20th April 2018].

76. UK Government. (2017). *Unlocking the UK's High-Tech Economy: Consultation on the Safe Use of Drones in the UK.* United Kingdom, London. Available at: https://www.gov.uk/government/uploads/system/uploads/attachment_data/file/631638/unlocking-the-uks-high-tech-economy-consultation-on-the-safe-use-of-drones-in-the-uk-government-response.pdf. [Accessed 8th December 2017].

77. Clarke, M. (2017). *Remote class sharing over process and machine boundaries*. MSc Dissertation. University of Nottingham, Department of Computer Science. UK.

78. University and College Union. (2018). *UCU announces 14 strike dates at 61 universities in pensions row*. Available at: https://www.ucu.org.uk/14-strike-dates. [Accessed 21st April 2018].

79. Dudley, S. (2017). *ORB-SLAM2 for Ardupilot*. Available at: https://github.com/SamuelDudley/ORB_SLAM2. [Accessed 21st April 2018].

80. Barker, P. (2018). *GCS_MAVLink: feed vision position data into AHRS*. Available at: https://github.com/ArduPilot/ardupilot/commit/a5a36c04d1056a2864267e9a84dc8edc90d9aa84. [Accessed 22nd April 2018].

9 Appendices

9.1 Appendix A – Multi-copter Bill of Materials

Raspberry Pi Zero W
Link: https://thepihut.com/collections/raspberry-pi-store/products/raspberry-pi-zero-w
Cost: £15.60
Weight: 9g

Raspberry Pi Camera Module
Link: https://thepihut.com/collections/raspberry-pi-camera/products/raspberry-pi-camera-module
Cost: £24.00
Weight: 3.4g

Erle Robotics PXFmini
Link: http://erlerobotics.com/blog/product/pxfmini/
Cost: €69.00 (around £60.00)
Weight: 83g

Erle Robotics Power Module
Link:
https://erlerobotics.com/blog/product/power_mod/
Cost: €30.00 (around £26.00)
Weight: 50g

Martian II 250 (Frame)
Link: https://www.banggood.com/Martian-250-250mm-4mm-Arm-Thickness-Carbon-Fiber-Frame-Kit-w-PDB-For-FPV-Racing-p-1064323.html?rmmds=category
Cost: £21.60
Weight: 130g

Total cost: £235.48
Total weight: 618.54g

Battery (LiPo, 3S, 2200mAh, 40C)
Link: https://hobbyking.com/en_us/multistar-racer-2200mah-3s-40c.html
Cost: £13.60
Weight: 195g

4x Racerstar RS30A V2 30A (ESCs)
Link: https://www.banggood.com/4X-Racerstar-RS30A-V2-30A-Blheli_S-ESC-OPTO-2-4S-Support-Oneshot42-Multishot-for-FPV-Racer-p-1074733.html?rmmds=category
Cost: £35.90 (around £9 each)
Weight: 25.04g (6.26g each)

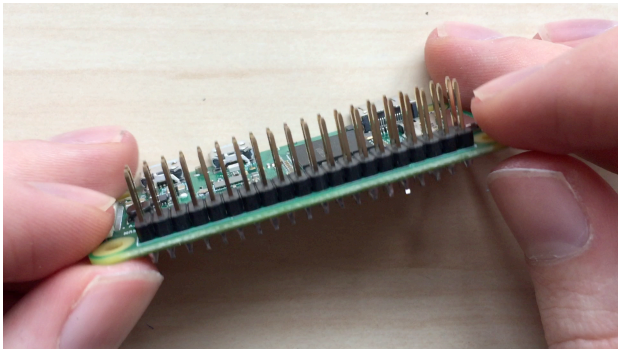
4x MultiStar V-Spec 2205 (Motors)
Link: https://hobbyking.com/en_us/2205-2350kv-ccw-v-spec-mongoose.html
Cost: £35.68 (£8.92 each)
Weight: 120g (30g each)

4x Gemfan Multirotor CRP Propeller 6x4.5
Link: https://hobbyking.com/en_us/gemfan-multirotor-crp-propeller-6x4-5-black-cw-ccw-2pcs.html
Cost: £3.10 (2x packs for 4 blades in total)
Weight: negligible

9.2 Appendix B – Photography of Construction Process

1. Soldering GPIO pins onto the Raspberry Pi Zero W

The Pi by default does not ship with GPIO pins pre-attached to save on cost. These are required for connection to the PXFmini.

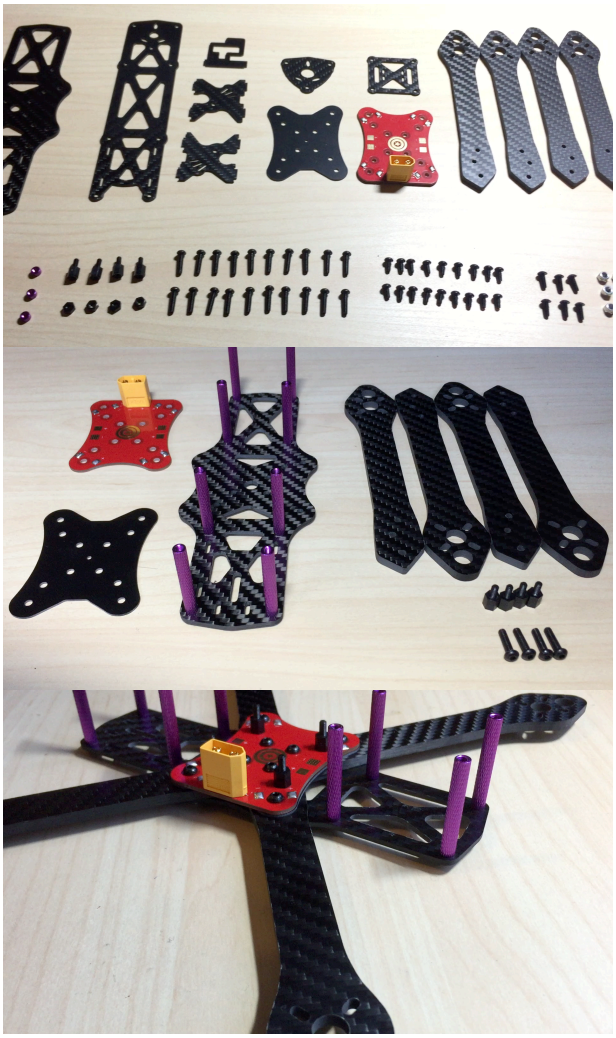


2. Soldering bullet connectors to the ESCs

The ESCs purchased only have a solder point to connect the motors to them, yet the wires on the motors bought terminate in 2mm bullet connector plugs.

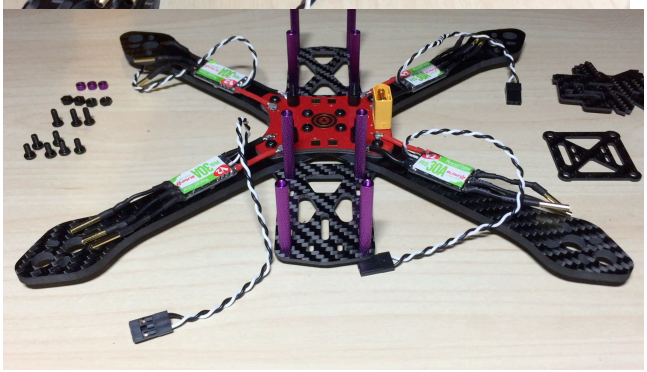
3. Partial construction of the frame

After soldering on the battery connector to the Power Distribution Board (PDB) – which provides power from the battery to the ESCs and motors – it needed to be installed in the frame to progress.



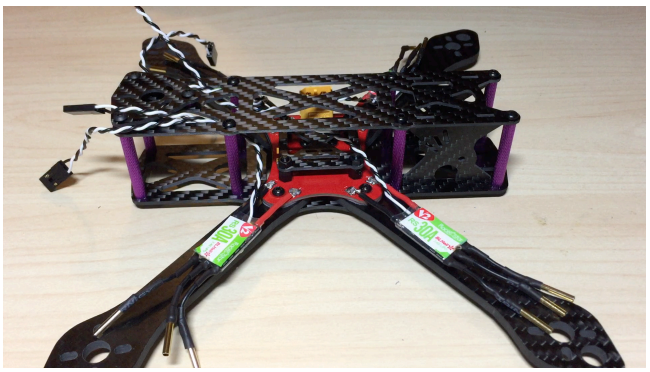
4. Solder ESCs to the PDB

The ESCs needed to be soldered to the PDB to make an electrical connection. Care needed to be taken to match up positive and negative leads correctly.



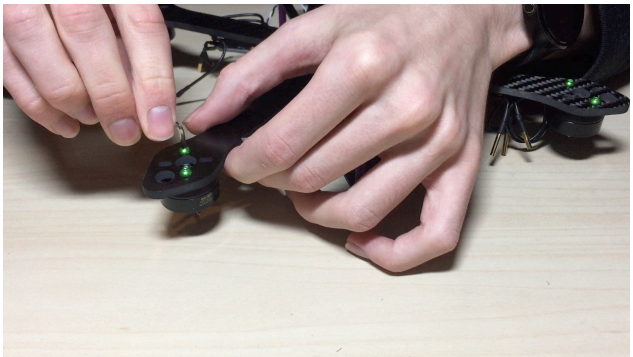
5. Construct the rest of the frame

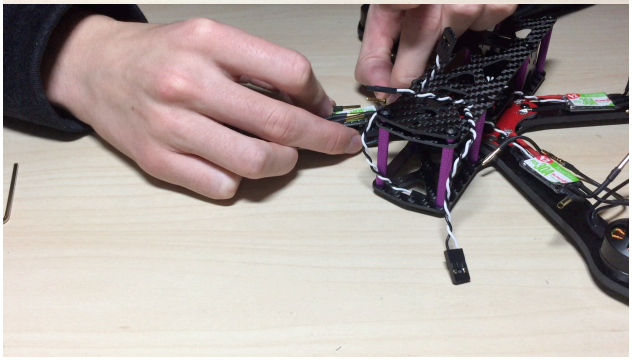
Self-explanatory, with the signal wires from the ESCs also being routed through the frame.



6. Screw on the motors and attach to the ESCs

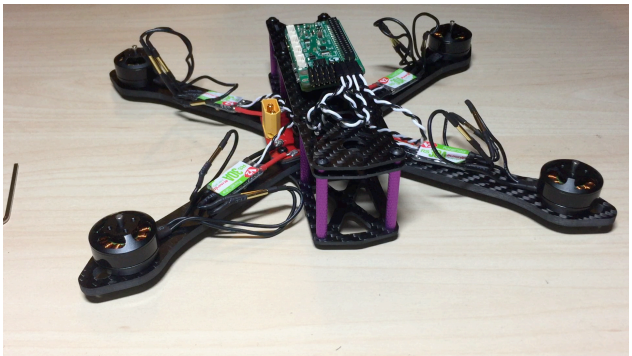
Each motor was screwed onto an arm of the frame, and then plugged into the bullet connectors prior soldered to the ESCs.

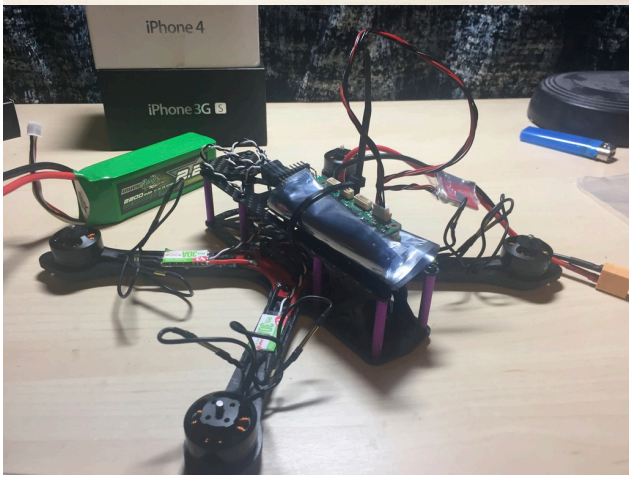




7. Wire up everything to the PXFmini

The Raspberry Pi and PXFmini were situated on the top of the frame, as they did not fit inside the frame. The frame was later modified to fit them correctly. The signal wires were attached to the appropriate output pins on the PXFmini; care was taken to match counter-clockwise and clockwise spinning motors to the correct outputs.





8. Attach propellers

The propellers were attached to the motors. These needed to be matched to the rotation of the motor; those marked with an "R" are to be clockwise spinning. If incorrectly matched, thrust will be directed upwards instead.

9. Tidy up all connections

A small amount of electrical tape and zip ties were used to secure cables out of the way of the propellers.