Project: Intelligent Social Vehicle

Masters Project 2016
Final Report
11797723, Simone Magri

Introduction:

Intelligent social vehicles are designed to replace or augment human activities, for instance, semi-autonomous cars and autonomous drones. In our case we are using an ambulance scenario to highlight this functionality. Also, drones are also referred to as UAV's(Unmanned Aerial Vehicles), the terms will be used interchangeably in this document.

Our Ambulance Scenario is:

- A number of drones will be mounted on the ambulance in designated areas.
- A ROS/Qt GUI based application will be run on a computer/embedded system within the ambulance.
    - The application is used to control and monitor the flying drones, and also manage their payload delivery
    - Drones will fly off the ambulance to the designated target.
    - One or more drones will fly off the ambulance at any particular time.

Initially, we had intended to use the Hexo+ drones in the research and development of a drone control system for use in the ambulance scenario. More specifically, it was for the development of a Front-end Control System. This would involve working out how to fly a swarm of drones(we have five hexo+ drones) from the ambulance to a particular target location. Also to bear in mind here is that the hexo+ does not have a collision avoidance system. The front-end control system was the focus of the traditional literature review is detailed later. The work of Dr Christian Wietfield was most relevant here[1][2][3][5]. His work indicated that it was quite possible to fly swarms of drones with a wifi network for uav to uav communication and a standard mobile cellular network for uav to the mission control centre(mcc) or base station, the ambulance for us, communications. It also had some details of collision avoidance systems which appear to be sensor based. It it was clear from these articles that one should always get the best sensors one can afford as poor sensor data leads to poor swarm uav flying. Also, an RF(Radio Frequency) link between base station and uav's is required for high quality video data[4]. In briefly reading the work of Dr Sebastien Varrier[2][3][4][5][6], this also supports the use of some form of formal control system for the front-end control system, highlighting the probable use of non-linear tools over linear tools. These papers also cover collision avoidance systems.

However, there were many significant delays in our requests for general information and the acquisition of the SDK(Software Development Kit) for the hexo+ drones, both fundamental to development of the Front-end Control System. This coupled with the fact that hexo+ is no longer the market leader with respect to technical specifications of their drone, we started looking into other drone families. The DJI family of drones were looked into as they are the current commercial market leader amongst their drone offerings, and they have a readily available SDK. An analysis of the DJI family of drones is the focus of the second part of the literature review, with special emphasis on their SDK offerings.

However, we have five existing hexo+ drones and the DJI phantom 4 had not arrived by the time the main development for this project was to take place. Consequently, I decided that I would focus on the development of the Back-end Control System of the Drone Control System, for the Ambulance Scenario. Thus abandoning development on the Front-end Drone Control System for the time being.

Thus, for my masters project I will produce a proof of concept prototype for the the back-end control system using a single Hexo+. This is the main focus of most of the rest of this document.

Drones:

Following is some background on the use of drones as intelligent social vehicles. Drones are also known as UAV's(Unmanned Aerial Vehicles). The term drone is given to a broad category of UAV's that can be used for military, commercial and personal or entertainment applications. Some examples are:

Military applications or high risk situations:

- High risk situations like submerged drones
- High risk situations like locating IED's(Improvised Explosive Device)
- Bridge inspections
- bush fire inspections
- search and rescue for hard to get at areas for humans, eg building collapse
- search and rescue is dangerous or disaster-stricken area
    - smoked out building
    - chemical leaks

Commercial applications:

- cost cutting, like pizza and beer delivery
- cost cutting, like Amazon package delivery
- development of eye-safe lasers for communications between ground stations and UAV's
- vertical take-off requirements
- search and rescue operations
- production of movie s and TV shows
- vineyard mapping for crop planning
- farming, like herding cows or sheep
- waiters to take your order, due to cheap labour shortage, staff now move to making cocktails as its not what a drone can do yet
- wildlife protection, eg rhinos

Personal or entertainment applications:

- drones can fly to places where humans cannot reach and capture breathtaking footage
- fly into the heart of a fireworks displayed for good footage
- selfie shots, that is holiday shots with no stick

Literature Review(Part One):

This was composed of two main sub-parts. With a review of the traditional literature of academic papers[1-12] Two principal candidates, pointed out by mabiclab colleges, were the focus of this preliminary review. Christian Wietfeld and Sebastien Varrier. A list of papers to be read was identified using google scholar searches and reading the abstracts of these papers. Abstract summary notes are provided, as well as summary notes after the paper has been read[1-12] This traditional review of the literature was followed by a review of the technical features, or specification, of the drones offered by the current commercial market leader of drone manufacturers, DJI, for reasons outlined in the introduction. Included in the latter review are comparisons between DJI and the Hexo+ drones. The traditional literature review is partially completed and the technical review is done at this stage.

Project Outline &
further to Literature Review(Part One):

Again, in summary, the use case for this project is the, Ambulance Scenario:

- A number of drones will be mounted on the ambulance in designated areas.
- A ROS/Qt GUI based application will be run on a computer/embedded system within the ambulance.
  - The application is used to control and monitor the flying drones, and also manage their payload delivery if necessary
  - Drones will fly off the ambulance to the designated target.
  - One or more drones will fly off the ambulance at any particular time.

Reasons for evaluating DJI family of drones, in addition to the Hexo+ drones:

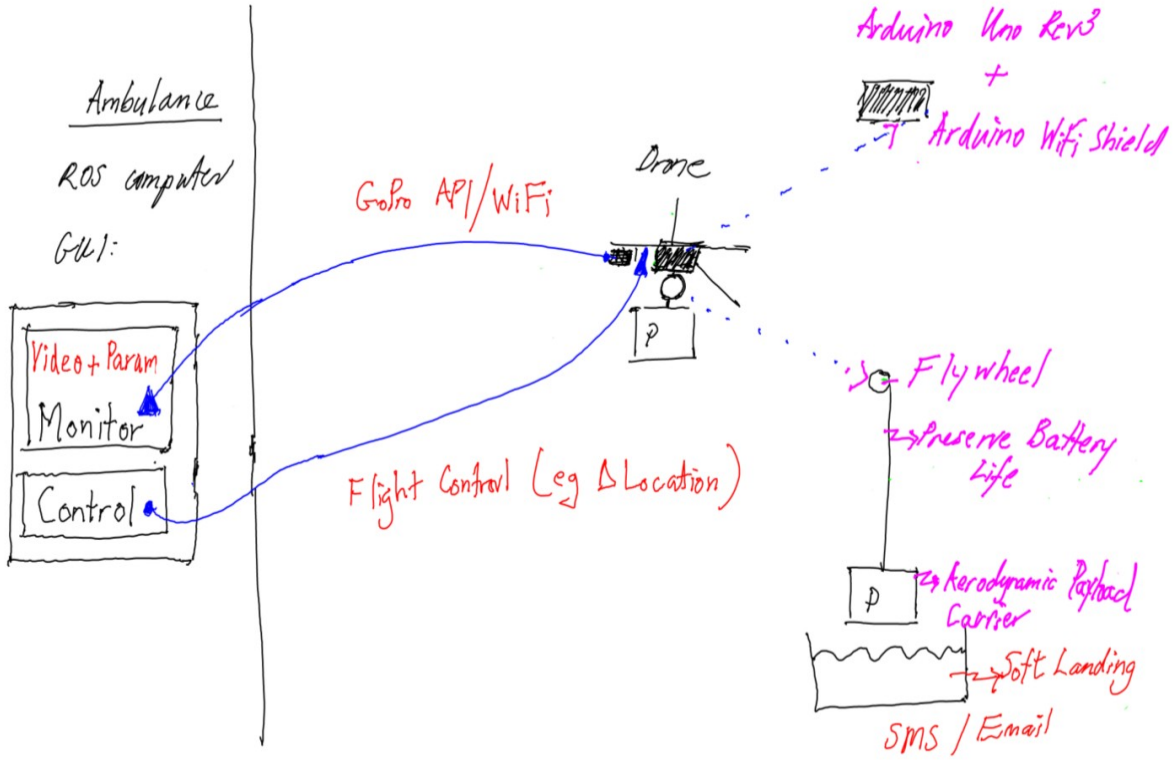
- Initially 5 Hexo+ drones were to be used. But due to lack of SDK arrival and the poor nature of our dealings with Hexo+, for instance:
  - SDK at least a year after it was promised, should be on the way now though(cross fingers)
    - Hexo+ is extremely slow to respond to our requests
    - Hexo+ appear to have cash flow problems(eg battery situation)
- We have ordered a DJI Phantom 4 for prototyping, to hedge our bets.
- Other reason for doing this is that the Hexo+ are now substantially behind the market leaders, feature wise.

Drone Control System

- The ambulance computer and drones form a drone control system
  - It consists of a front-end and back-end control system
- ***For my masters project***
  - ***I will produce a proof of concept prototype for the the back-end control system using a single Hexo+(for reasons outlined in the introduction)***
  - *Front-end work: (described in detail later)*
    - Could not be done since we don't have the Hexo+ SDK

- And as the DJI Phantom 4 has not arrived yet

Figure 1. Back-end Control System Diagram



Back-End Control System(1) Ambulance ROS computer:

- ROS/Qt GUI computer application in the ambulance
  - ***Needs to be there regardless of what drone we are using***
  - Visual Monitoring of drones will be via live video from the GoPro's
    - Using GoPro API and streamed via WiFi
    - This will be part of the ROS Monitoring Node(publisher/subscriber)
    - It appears as though you can mount a GoPro on the DJI Phantom4
      - This would be instead of the native camera
      - so code will immediately be reusable if either the hexo+ or phantom4 is used
- Monitoring of drone flying parameters
  - Will be part of the ROS Monitoring Node also
- Drone flight will be mostly autonomous,
  - any manual flight control will be done from the ROS Control Node(publisher/subscriber)
  - Currently, we do not have the SDK would allow ideal changing of the drone's location,
    - So, Hexo+ flight control, location changing, at this stage:
    - would have to be via a GPS phone application that spoofs it's location, use FakeGPS Free App (Android)
    - Hexo+ will lock onto where it thinks is the GPS location of your phone

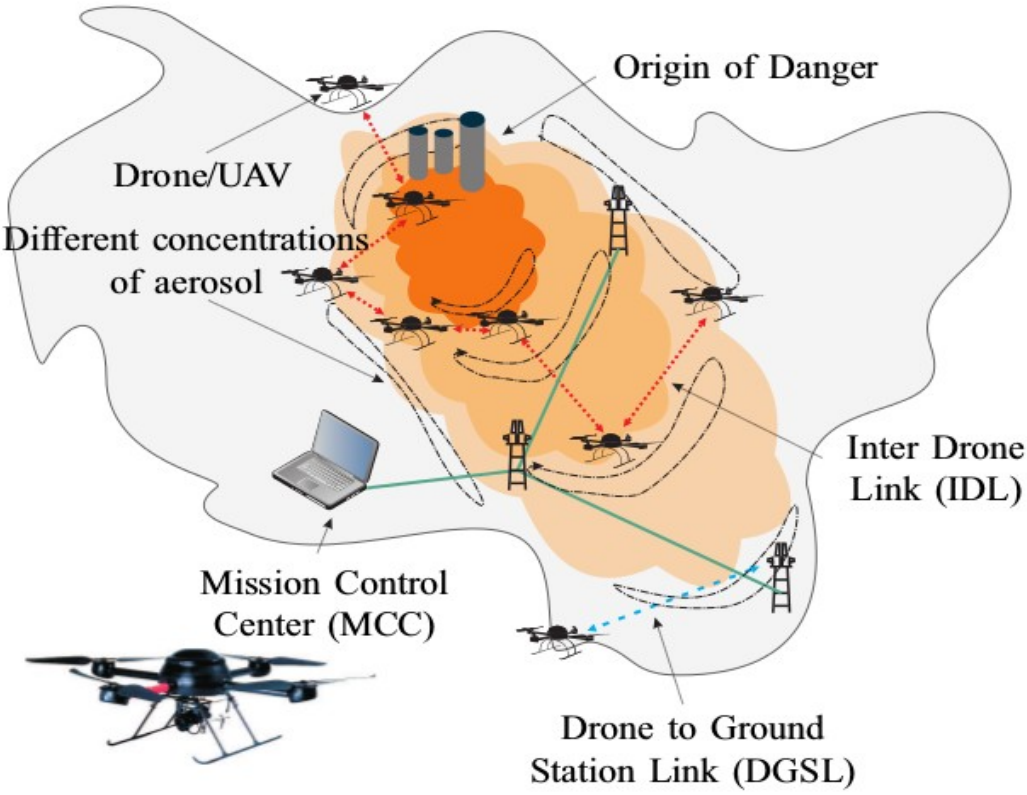
Back-End Control System(2) Arduino Rev 3 & Arduino WiFi Shield:

- An **Arduino** with a **WiFi Shield** attached will be mounted to the **underside** of the Hexo+
  - It will control a **servo** or **stepper motor**
  - And it will form part of a **client**/**server** coding system
  - The **client** code is in the ROS Control Node within the ROS/Qt GUI
    - It will use TCP/IP **socket** connections to communicate with the **arduino via** the **WiFi shield**
  - The **server** code will run on the **arduino** and will **control** the **motor** to ***winch*** *down/up the* ***payload***
  - Arduino will be **powered** by the **existing battery connector** on the **Hexo+**
    - Arduino input voltage needs to be between **7-12Volts**
    - Hexo+ battery is 11.1V from 3S, so 3.7V per cell(**2S=7.4**, 3S=11.1)
      - **I've done** this as, @**home** I use a 9V battery but it's **heavy**, 43g, **weight reduces** my **flight time**
    - **WiFi shield** is **powered** from the Arduino +5V
      - Shield has **same pin-outs** as the **arduino** uno

Back-end Control System(3) Payload Delivery:

- I will **Winch down payload** to **preserve battery power**
  - **Flywheel** on end of motor
  - **Servo *or* Stepper Motor** will be used to **winch down** the **cable.** Both will be evaluated for suitability. So some pros and cons wrt servo vs stepper motor:
    - Rotation angle for servo is normally 180 degrees but for continuous servo is 360 degrees
    - Continuous servo may not perform as well, slippages of wheel etc
    - Servos **easier to power**
    - Stepper motors require an additional circuit(H-Bridge Motor Driver chip to drive high voltage motors from TTL 5V logic)
  - Will have a **3D printed Aerodynamic carrier**
  - **Payload will be delivered** to '**X mat'** or **QR-code** soft target plate, once released from cable
    - **OpenCV *pattern matching*** code will be used to accomplish this
    - **Landing** should be a **soft** one(eg pillow) due to **delicate** nature of **cargo**
  - **SMS** or **email** payload arrival message to official
- **Observation** about one **DJI Phantom 4** payload delivery system I've seen **recently**
  - Overall system design is similar, with a servo used
  - Pin release mechanism; mine is winch down
  - About 85g in total; mine is ~100g but about 60-80g if I can get away with smaller parts
  - Drop/Deliver from a height; would be **dangerous** for us, would need to fly down to **get closer to target first**
  - My system is a little different, it **remains to be seen which is the best method** for this task
  - Developing my own system certainly gives you an appreciation for the DJI Phantom 4 system

Figure 2. Front-end Control System *Diagram ie Flight Control System:*



Drone/Mesh Drone flying may be informed by AI algorithms and methods the sexy part of this project will be in this area

Front-end Control System *Description*:

- Drones are also known as **UAVs**, that is, **U**nmanned **A**erial **V**ehicles
- Drones will fly off the back of the ambulance **individually** or in **unison,** this is yet to be decided
  - Which of those it is, obviously has **implications for the flight control system**...
  - **So** the **ambulance scenario may not require** *flying a group of drones in a mesh system*
  - **However,**
- **Other use cases** may require UAV flying in a **mesh** system:
  - Looking for **lost bush walkers** during the day and night(djiPhantom4)
    - May use facial recognition and audio messages for this task
  - **Fire fighting**, to survey the fire more safely, quickly, and cheaply
  - **Chemical spill fighting**, ascertaining nature of the chemicals and their plumes
- **Literature** I've read[1-12]indicates **control** of **UAV flights** in a **mesh** uses:
  - **WiFi** for **UAV-UAV** communications
  - **Mobile cellular networks** for communications from **UAV to the mission control** centre
  - And that, **Flight Control** is determined by a **feedback** based **control system**
    - Input is taken from the **sensors(ultrasonic, radar, sonar)** located on the drones to provide feedback for the control system
    - **My observation**, your control system is only as good as your sensors, bad sensor data leads to a bad flight path

- ***We now come to an important point:***
- ***Drone/Mesh Drone flying may be informed by AI algorithms and methods***
- ***I think the sexy part of this project will be in this area***

Front-end Control System, *some thoughts on using Radar/Sonar, from the literature(sonar, radar...)*:

- **Hexo+** does **not have** any **obstacle avoidance** system
  - That got me thinking about **obstacle avoidance** systems for a **drone/UAV**
- **I had an idea!** to use **Radar** to avoid **UAV-UAV** collisions and **UAV-Obstacle** collisions
  - **Literature:**
    - **Backed this up**, Christian Wietfield[2](used FMCW(Frequency Modulation Continuous Wave)-radar) is doing one or both of these, it's a little unclear
    - A **quick look** seems to indicate a **low cost radar** system is **limited** by **range**, some **meters** to **tens** of **meters**, these systems can be as **small** as **fingernail**
- **DJI Phantom 4** uses ***Sonar, Cameras,*** *and* ***AI control*** for **obstacle avoidance**
  - **Not sure** how that will **work**:
    - I've seen **videos** that show it avoiding a big **rock formation**
      - however, promotional videos are one thing and real life performance can be another eg battery life specification of 28min, which is about 23min in real life
    - Not sure how it will go with **tree branches, birds**, and even **other drones**...etc
- Thinking about **obstacle avoidance** systems left me with **more Questions**:
  - Why would you use **Radar over Sonar** and visa-versa?
  - *Decided to* ***investigate both a little***

RADAR/Active SONAR basic understanding may be useful:

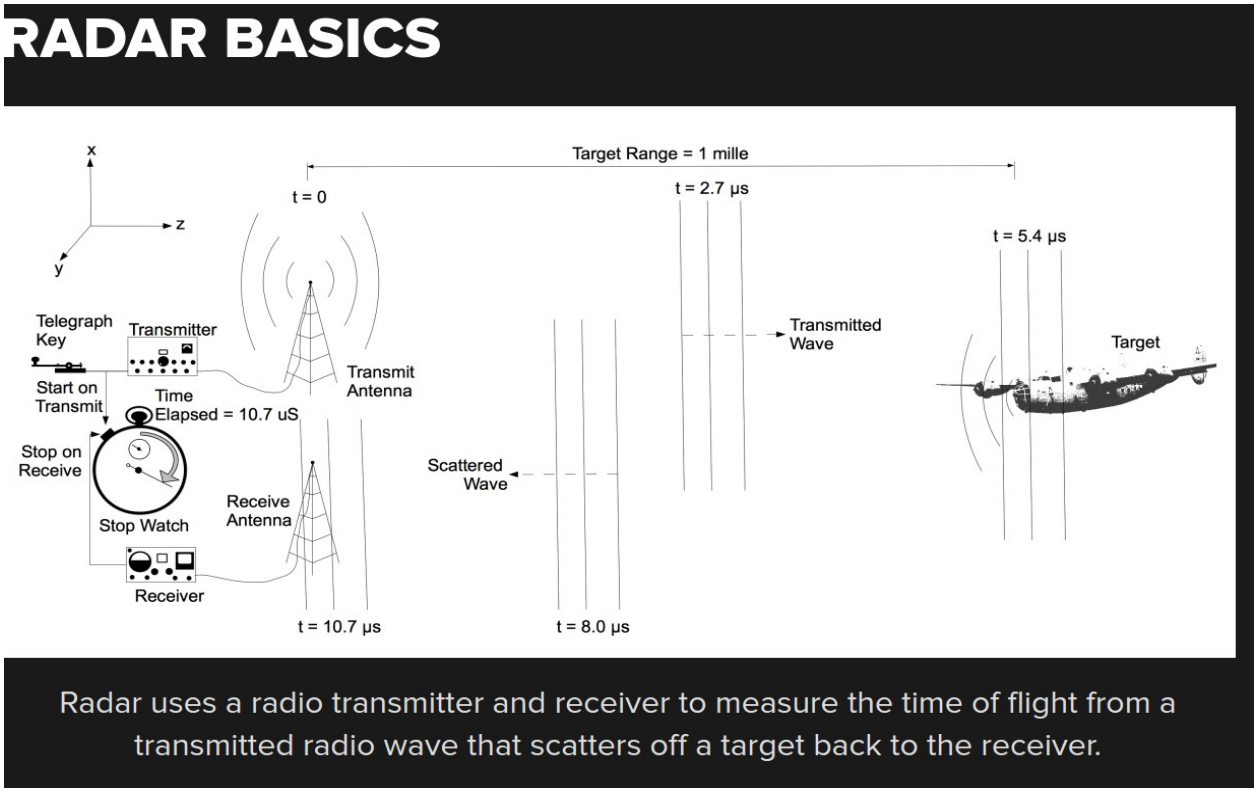


Figure 3. How Radar or Active Sonar Works

*Radar(**RA**dio **D**etection **A**nd **R**anging):*

- Radar works by sending out pulses of **radio waves** from a transmitter which are reflected off the object and detected/measured by the receiver system.
  - Properties of the **received signal** are **analysed** to give us the object's:
  - Presence
  - Direction/Angle
  - Distance/Range
  - Speed/Velocity
- Radar is already used in **aircraft anti-collision** systems and **flight control** systems
  - Thus a **stable and reliable** technology
- Radio waves are **electromagnetic waves** in the radio spectrum
  - They range from <1Hz to about 100GHz(to the end of the microwave range)
  - **Aside**: at radio telescope needed to not use microwaves in tower and quarters when astronomers were observing in certain bands
- **Note of interest: High tech radar** systems are associated with:
  - digital signal processing
  - **machine learning**
  - and are capable of extracting useful information from very high noise levels

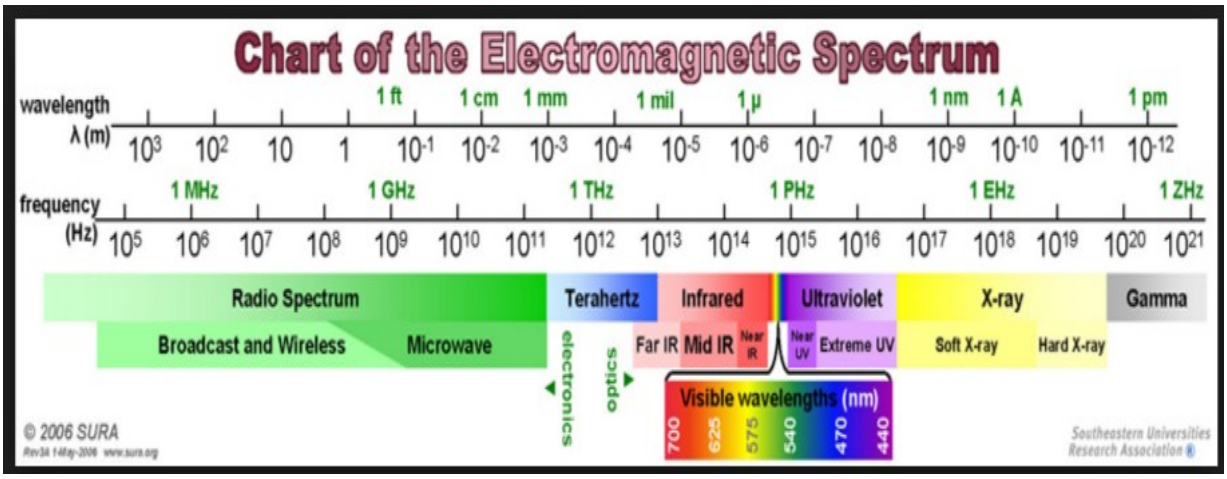


Figure 4. Electromagnetic Spectrum, Radio Wave Frequencies and Wavelengths

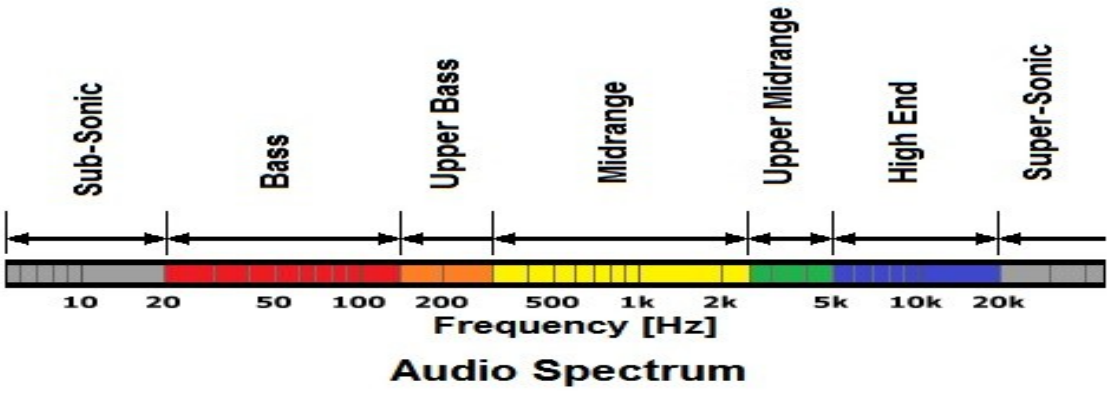


Figure 5. Audio Spectrum, Sound Wave Frequencies and Wavelengths

***Active Sonar**(**So**und **N**avigation **A**nd **R**anging):*

- Active Sonar works **similarly** to **radar** by sending out pulses and analysing the reflected waves
  - but it **uses sound waves** not electromagnetic waves, and thus is a **mechanical** (or pressure) **wave,** so needs a medium to travel in
    - Detects(presence)
    - Gives Bearings(direction/angle)
    - Distance/range
    - Radial speed/velocity
    - Or detect other vessels
  - **Used** as a means of **Active Acoustic location**:
    - It can take place in **gases**(atmosphere), **liquids**(h2o), and **solids**(eg earth)
- The acoustic **frequencies**(sound waves) used vary from the very low **infrasonic** to the extremely high **ultrasonic**
  - **Infrasonic**, low-frequency sound, <20Hz(lower limit of human hearing) and down to about 0.1Hz
  - **Ultrasonic**, high-frequency sound, >20kHZ(the upper limit of human hearing) and upto about several GHz.

In Summary: Radar vs Sonar:

- **Active Sonar** was **used** in **air before** the **introduction** of **Radar**
- **Literature,**
  - **Radar** is said to be a far **more effective** system than Active Sonar
  - **Active Sonar** was rendered as **obsolete *during* world war 2** by the **introduction** of **radar**
- ***Still Not clear*** to me at this stage **why** one would **choose Radar *over* Sonar** or visa-versa, ***requires further investigation***
  - some thoughts that may impact this:
    - Submarines rely on sonar, as at depth radar is highly attenuated in water
    - Sonar is cheaper
    - Radar has a greater range than sonar(in air)?
    - Sonar is used in air for robot navigation

Further to the Literature Review(part two), determining which is the superior Drone?

I'm going to compare a few drones, to determine which is the Superior Drone. These drones are commercial and developer drones currently on the market:

- Hexo+ (commercial drone)
- DJI Phantom 4 (commercial drone)
- DJI Matrice 100 (developer drone)

Table 1. Comparison between Hexo+ and DJI drones

Feature	Hexo+	DJI Phantom4	DJI Matrice 100
Cost Drone	$1307	$1999	$5699
Cost GoPro(Hero4 Black)$679	$679 (Drone + GoPro=$1986)	Inbuilt Camera or GoPro	No Built in camera(SLR)
SDK	August 2016	MobileSDK(App SDK)	MobileSDK, OnboardSDK
SDK ROS integration	Not Sure	No	Yes
Weight[5]	1.7kg(inc batter + gimble)	1.38kg	2.355kg
Flight Time average[1]	8min(15min spec)	23min(28min spec)	40min(spec)(2 batteries)
Payload Delivery System	No	Yes(random drop)	No(make own)
Payload Maximum Weight[2]	(Gimble+Camera=>min)~270g	462g	1kg
Obstacle Avoidance	No	Yes(Sonar)	No(Need Guidance System)
Autonomous Flight	Yes	Yes	No(make own)
Follow Me	Yes	Yes	Yes
Range Maximum(from were you stand)	100m(Follow Me)	5km(Lightbridge)	5km(Lightbridge)
Altitude Maximum(above sea level)	4.8km	6km	5km
Operating Temperature[3]	-10deg-50deg	0deg-40deg	-10deg-40deg
Speed	~75km/hr	~75km/hr	~75km/hr
Live View(what drone camera sees)	No, or 2nd phone's GoPro App	Yes, Phone/Tablet	Yes, Phone/Tablet
Lens Field Of View	0-90deg	0-94deg	No Built in Camera(SLR)
RC Control[4]	PPMSUM transmitter(open drone to install)	Yes, standard	Yes
Vision Positioning System(VPS)	No Indoor Flying	Yes Indoor Flying	Yes
Flight Planning	No	Yes	Yesterday

1, average flight time, specifications never seems attainable
2, increase the payload weight decreases the flight time
3, below zero for bush walkers in mountains, or into a fire
4, non-standard for Hexo+, retrofitted by opening drone
5, weight significant for flight regulations

Which is the Superior Drone?

In summary, if your budget is less than 2k, the **DJI Phantom 4** is the best drone for you. This is the ***current market leader*** in ***consumer drones***. However, if you are a developer, and have a budget greater than 2k the **DJI Matrice 100** is the best drone for you.

Further to the Literature Review(part two), Beyond the Consumer Drones.

The Hexo+ and DJI Phantom 4 are consumer drones, beyond the consumer drones are the Professional Drones & Developer Drones.

Beyond Consumer Drones, within the DJI family:

- **Above** the **consumer drones** are the ***professional drones***, for instance in the DJI family:
  - The **biggest difference** between these models is:
    - **quality** of the cameras and components(eg motors, propellers)
    - some even allow you to **mount** a **digital SLR**
- **Cost** is the **other major difference**:
  - **DJI Phantom 4, $1,999**
  - DJI inspire 1 v2.0, $3,399
  - DJI inspire 1 Pro, $5,899
  - **DJI Inspire 1 Raw, $11,199**
- They **suite**:
  - Those who do budget **add campaigns**
  - **Hollywood** productions
  - **Wedding** Photographers/Video-graphers

- **Also,** a**bove** the **consumer drones** are the ***developer drones***, for instance in the DJI family:
  - The biggest difference between these models is:
  - That come with an **onboardSDK** that gives you access to **flight control** and **other** things
  - Importantly, it **allows** you to **attach** your own **embedded system** via a **serial port**/Uart
  - Cost is the other major difference:
    - ***Matrice 100, $5,699, designed for research***
    - Matrice 600, $7,899, for professional aerial photography and industrial applications

DJI SDK's (Software Development Kits):

Without an SDK we can not develop our **front-end** control system. One of the **big reasons** for looking at the DJI family of drones is because they have readily available SDK's, with a strong user base given they are market leaders, and good documentation available. This is in stark contrast to the Hexo+ SDK, not readily available(NDA needs to be signed), very small user base, and user documentation is very scarce.

- The DJI SDK's are:
  - MobileSDK(Phantom 4 & Matrice 100; Android & IOS)
  - OnboardSDK(Matrice 100; Windows/QT, Linux/ROS, Embedded System)

- GuidenceSDK(Matrice 100; Linux/ROS & Embedded Systems)

MobileSDK*(DJI Phantom 4, DJI Matrice 100):*

- **Mobile SDK**
  - **Allows you to customise your app**
  - ***Platforms Supported: IOS** 8.0+ and **Android** 4.2.2+*
    - *Several tutorials/examples are provided on how to use the SDK for different applications*
    - *Developer guide and API documentation*
    - **Some level of flight control**(High and Low level)
      - **Predefined fight path**
        - **Waypoint,** Hotpoint**, FolowMe**
      - **RC control, manual** flying
        - During which SDK allows monitoring of live video and sensor data
      - Stick commands
        - To simulate pilot
    - Telemetry and **sensor data**
      - **GPS position**
      - Compass**,** barometer**, velocity, altitude**
    - Obstacle avoidance
    - Camera(eg exposure, aperture, ISO speed) and Gimble Control
    - **Live Video Feed** from the **drone's camera**
    - Battery monitoring & control
    - **Flight control with touch screen app**, finger pointing and point of interest

OnboardSDK*(Matrice100):*

- ***Platforms Supported:*** Windows/QT, **Linux**/**ROS** and **Embedded Systems**
- Only **Linux** and **Embedded Systems** are of **interest** to this project
  - **Raw communication wrapped in ROS packages**
  - **Toolchain**:
    - c++ compiler gcc2.8.1/5.3.1
    - Bash shell
    - GNU Make
- **Platform**: Ubuntu 14.04/16.04
- *Is **for C/C++ embedded systems programmers***
- ***Provides control functions(eg flight control) from a C++/ROS environment(eg official ros pkgs)***
  - *And the **Key** ones for **us**:*
    - ***Local/Global Navigation(go to a specified local/global position)***
    - ***Take-off** and **Landing control***
    - ***Photo Taking***
    - ***Start/Stop Video Recording***
  - ***Also** includes a **Simulator***
- Most Importantly, it allows you to integrate your **own** hardware with DJI flying platforms
  - *Using the **API** provided you can, for instance, use a sensor connected to your embedded system(eg arduino) to control the trajectory of the vehicle*
  - *It's **fantastic** for rapid **prototyping***

GuidanceSDK(Matrice 100):

- **GuidanceSDK**
  - ***Platforms Supported:*** **Linux**/**ROS** & **Embedded System** Integration
  - The **SDK for** the **Obstacle Avoidance** System
    - You can use with **Matrice100** and Matrice600
  - Consists of ultrasonic sensors and stereo cameras
    - looks **like** a bunch of **ultrasonic sensors** like I use on **my arduino projects**
    - Gives you full control of your guidance
      - That is, control drone movement WRT **position**, **velocity**, **altitude**, rotation rates etc...s

Results:

All code written for the core and non-core results is provided in my github repository. Namely, https://github.com/smagri/proj30cp.

All code has been developed on an Ubuntu 14.04 linux system.

QtCreator 3.0.1 based on Qt 5.2.1 was the principal GUI development platform, and native ubuntu 14.04 versions. However, QtCreator 3.4.0 based on Qt 5.4.1 was installed and used in one instance specified later.

A lot of non-core results were produced while waiting for the native hexo+ batteries to arrive. Some of these are described in the non-core results section below. In addition, the focus was on getting some front-end control system development done, this is described in the core-results section. Once it was clear that the hexo+ SDK was not going to arrive in time focus switched to developing the back-end control system prototype, also described in the core-results section. Both these parts of the core-results section detail the most important parts of this project.

Core Results:

For those not familiar with ordering electronic parts over the internet, know that it can take quite some time. Starting with the initial large amount of research to determine which parts to buy, followed by peculiarities of ordering from different vendors, then the time for arrival of parts, some weeks if from overseas. So essentially, the thing you must keep in mind is to plan your ordering of parts as early as you can before you will need them. As an example, ordering just the parts for the back-end control system took about 3 weeks of full time effort. A similar effort was required for ordering parts for connectors, and for tracking down appropriate batteries, required for attaching a third party battery to the hexo+.

Front-end control system:

Getting the Hexo+ to fly:

Initially, before the decision was taken to focus on the back-end control system, I was focusing on what was required to develop the front-end control system.

The five hexo+'s that we had sadly arrived without their native batteries. The reason for this is not entirely clear. So after some months of waiting and hexo+ promising they would be on their way soon I decided it was time to act. So I researched and fit a third party battery to a hexo+ to get the birds up in the air! I decided to get the drones flying to facilitate further research. Thus, determining an understanding of what the hexo+ could do or not do, without the control of the SDK. Delays in ordering a third party battery for flying the hexo+ was compounded by regulation changes earlier this year, thus not allowing the transportation of LiPo(Lithium Ion Phosphate) battered by plane. I think this was mainly due to the tendency to catch fire occasionally. Hence a local supplier of batteries needed to be found. After much searching the HobbyCo store in the QVB(Queen Victoria Building) Sydney had a suitable battery, and is pictured below in figure 6.

Figure 6. 5Ah 11.1V 35C Venom Battery

After much research into suitable batteries and power connectors I settled on the above Venom 5.0Ah, 35C, 11.1V battery with the EC3 power connectors to interface the battery to the hexo+ power connector. The hexo+ power connector is hard to find and has part number 9A114251-ND99 and available from Digi-Key. Note also a JST-XH charging connector, four wire(black, blue, yellow, red) with white connector, was used on the battery, as I needed to interface it to existing chargers that came with the hexo+'s. The EC3 battery to hexo+ power connector I soldered up is seen below.

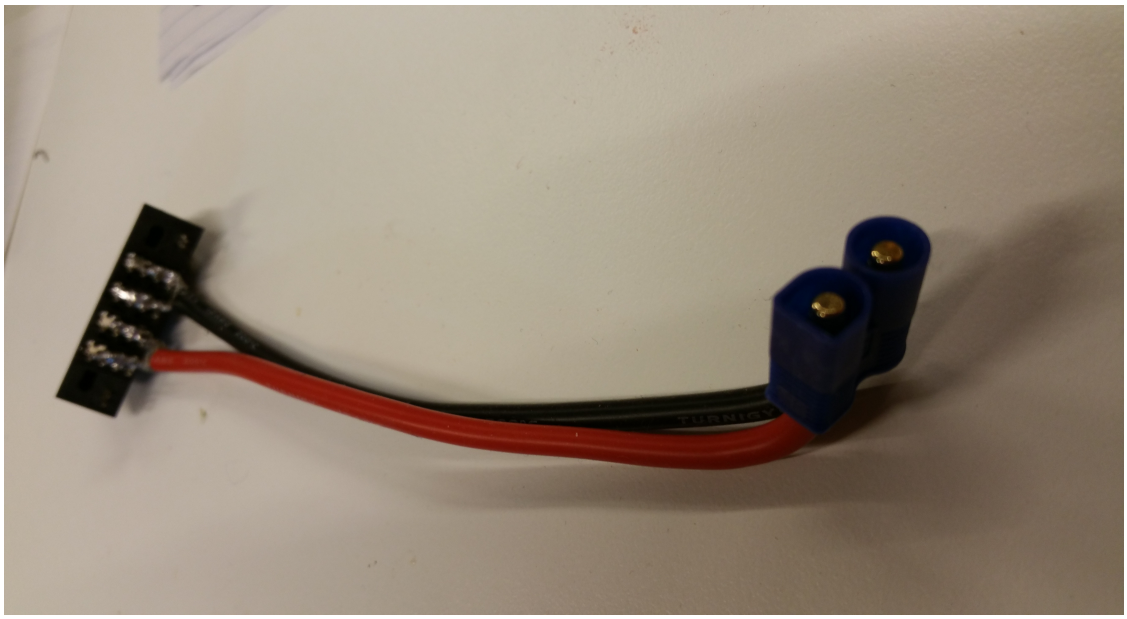


Figure 7. Hexo+ eight pin power connector to gold pin EC3 battery connector

Internet video's were used to work out how to solder the EC3 connectors, the gold two pin and outer shell blue connector in figure 7. Also, various internet sources were used to determine which type of connector was most suitable, like Deans, EC3, EC5, XT60...and many others were evaluated. The EC3's were chosen based on their ease of soldering and robust connection. Note also that 12AWG 200 degree C red and back power wires were purchased to make the hexo+ eight pin power connector connect to the EC3 battery connector.

As can be seen below the third party battery just sits a little outside the main hexo+ enclosure, but you can still put the cap on the end, just.

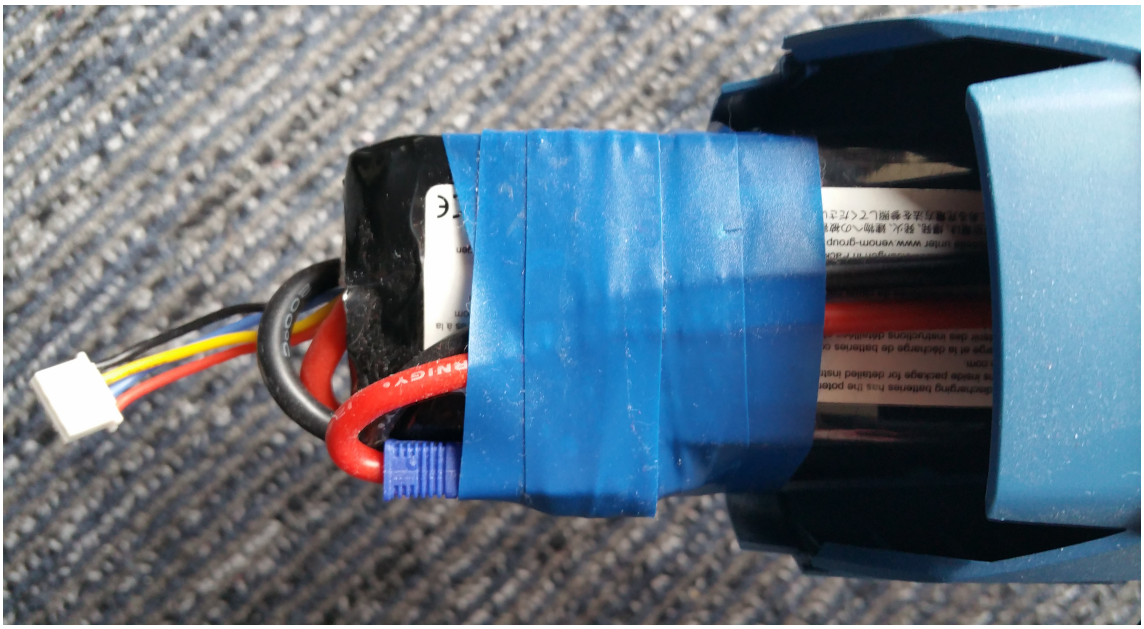


Figure 8. venom battery connected to the hexo+ 8 pin power connector, and installed into the hexo+ body.

Figure 9. overall hexo+ with venom battery connected and mounted

The hexo+ drone has been successfully flown on many test runs with this venom battery.

Also note that I have found that reading the 'hexo+ owners' facebook group very useful for determining which battery would be suitable to fly the hexo+, as well as for other general information about the hexo+. Another additional benefit, I mentioned on the facebook group one time that I had five drones and only one hexo+ power connector, and the lead time for new connectors was about 12 weeks at Digi-Key. Promptly, a person in the group named Joseph Lau, kindly offered to send me four as he

had some surplus ones. It is important to note that the 5.0Ah capacity battery was required to give the hexo+ an equal chance, wrt time in the air, when compared to the native hexo+ battery which is 5Ah also. The hexo+ native battery gives you about 8-9min of flying time, and I am happy to report that I got the 8-9min flying time with my third party battery. For reference, the 35C refers to the amount of charge the hexo+ will draw at any particular time. It was very clear from the facebook group advice that 35C and up was required for the hexo+. The other requirement was for an 11.1volt battery, thus a 3 cell x 3.7 volt battery.

See my GitHub repository for one of the best test flight videos I have of a hexo+, namely gopro0522best7jun2016.mp4.

Some notes on flying the hexo+:

This was done using the hexo+ android app. You needed to turn off wifi on your phone for flying to work. I'm assuming the hexo+ creates its own wifi hotspot for your phone to connect to. The hexo+ is able to cope with the multitude of wifi networks around the normal flying area on different channels. You can't really operate the GoPro app while flying as it requires the wifi network to be on. The hexo+ generally locks onto your phone position to perform its functions like, follow me or hover close and hover far...etc.

Hexo+ Crash:

I was flying in a new location, as it went into the air it became uncontrollable from the phone app, I could not access the main menu for some reason. The phone app was displaying a screen I had never seen before, it was a circle enclosing the words 'RF control'. I didn't know what that meant and was worried about the drone crashing in to surroundings, including people, uncontrollably so I did the only thing it allowed me to do and killed the motors, the drone came crashing down from a height of 10 meters. The drone is still flyable but has sustained some damage, as can be seen in the images below.

In retrospect I probably should have let the drone use up all its battery flying, it was only hovering at the time, and it probably would have landed automatically as it normally does.

Figure 10. hexo+ crash image one

Figure 11. hexo+ crash image two

Switch focus from the Front-end control system design to the Back-end control system design?

At this stage the decision was taken to abandon the investigation of the front-end control system design using the hexo+ drones, principally due to the non-arrival of the hexo+ SDK in time.

Consequently I came up with another avenue to focus my project efforts on. Hence, the design of the back-end drone control system. The rational here was that if I can't work on the front-end control system it would be only natural to focus effort on the back-end control system.

The back-end control system was described earlier in the project outline and in the following sections I present the actual results to date for that system.

Back-end control system:

(Refer to the earlier diagram and notes of this system in the project outline)

Some design decisions explained and to set the picture:

Payload delivery system:

WiFi verses wired ethernet:

The arduino, plus wifi shield plugged in, will be taped to the underside of the drone for testing. The drone could potentially be flying around, so clearly a wired ethernet connection was out of the question. WiFi, typically with a range of about (46m indoors) and 92m outdoors was going to be enough for the prototype back-end control system.

Hollow Oval Dome as the 3D printed payload carrier:

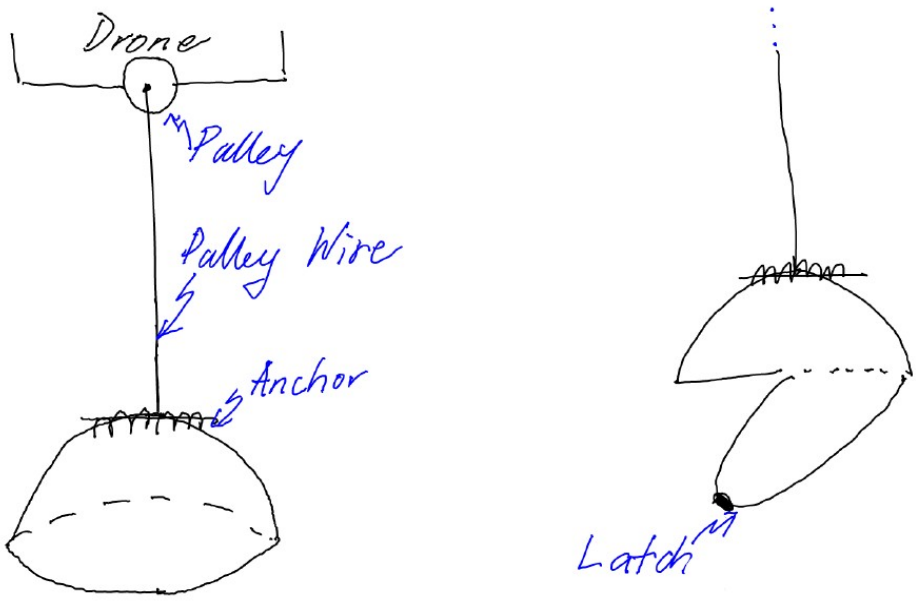


Figure 12. payload carrier drawing

The intention for this is that a hollow oval dome be a compromise between a suitable payload carrier and an aerodynamic system. This was just an idea of mine and has not been tested empirically, or in any wind tunnels as such. The flat bottom of the dome will have a slit with a hinge like system to allow releasing/removal of the payload.

The bottom of the dome needs to be flat for landing on the QR encoded flat landing platform, depicted below.

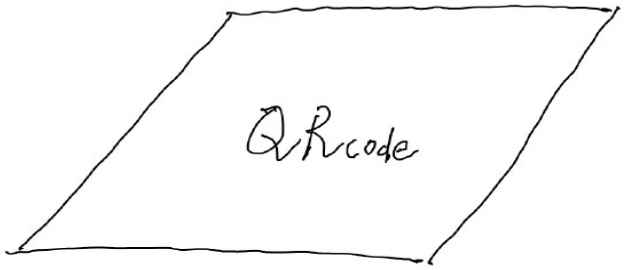


Figure 13. payload enclosure QR coded landing platform

This is essentially a square fitting the outer dimensions of the payload carrier. The platform will sit on level ground and have sizable slits, perhaps even tapered, on the sides so water can drain away. In addition, the level platform will have a QR code painted/printed on it to allow for verification of the landing place/building by the drone camera.

Overall use of motors, pulleys, and decoupling mechanism for the dome enclosure:

Currently, one end of the pulley wire is permanently attached to the top of the dome enclosure, the other end is simply wound onto the pulley. At this stage the idea is to winch down the dome enclosure and land it on the landing platform. There then needs to be sufficient length left in the wire such that by this point the wire has not completely unwound from the pulley and crash landed the do enclosure. So to release the dome enclosure one simply initiates the release sequence, from the gui(yet to be implemented), to winch down the last bit of the wire, thus releasing the enclosure from its coupling with the drone body.

Decision to deliver payload from a hight via motors instead of landing.

The payload needs to be external to the drone body, as we have no control over the drone design that would allow us to put the payload within the drone body. So we can't just land the whole drone straight and flush to the ground in one go, as we have interference from the payload & carrier. Also, we cant just land the enclosure flush on the ground while it is still tightly attached to the pulley mechanism, there is too much instability from the powered on drone. Hence, the payload is decoupled form the drone body/pulley system while the drone at a suitable safe height, perhaps also with some slack in the pulley wire.

***Temporary* Payload Enclosures with sample payloads:**

Due to the complexity of learning how to generate a 3D printed dome, and a very tight project deadline, temporary payload carriers have been sourced, see figures 14 & 15 below. The 3D printing of enclosures is discussed further in future work. The temporary enclosures are just simple décor enclosures, and they will be used for demonstrations.

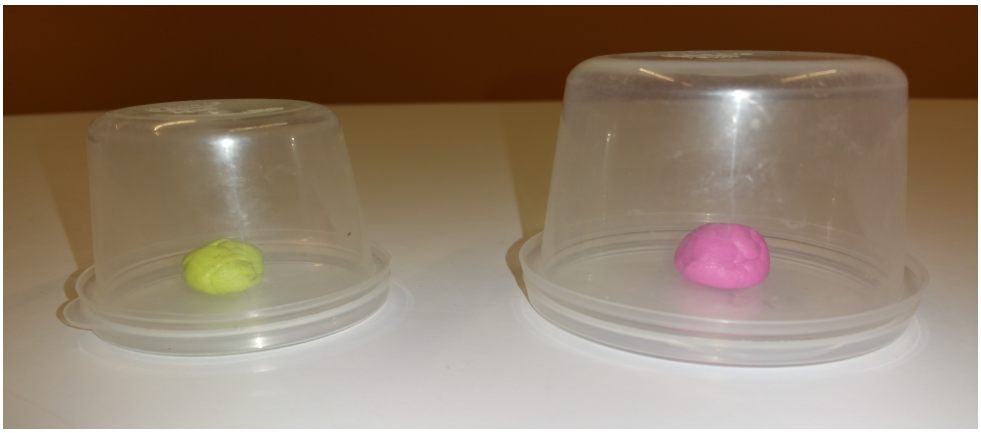


Figure 14. temporary enclosure a and b, side view

Figure 15. temporary enclosure a and b, top view

Servo and Stepper Motors:

Motor control circuits:

An Arduino Uno Rev3 board with a WiFi shield installed into it was used as the embedded development platform that would interface communication between the GUI application commands and control of the servo/stepper motor. Remember that this arduino embedded system will be taped to the underside of the hexo+ drone in the prototype system. Importantly, this means that the GUI motor control occurs over a wifi internet connection between the GUI and the above mentioned embedded system.

Both stepper motor and servo motors were evaluated in this project. Mainly to get a feel for which may be more suitable in a final and finished product, or even for what to focus further development on. At this stage it appears that the servo motor is most easily controlled, and has the simpler powering requirements. Most significantly though, the servo motor does not have the precision of the stepper motor. For instance, in the stepper motor used here, each step is 1.8 degree in movement, you can not control the servo to this degree, see operation details later. There is also greater complexity in driving the stepper motor than driving the servo motor, see this from the circuit diagrams below in figures 16 & 17. The stepper motor may require some tweaking before it operates properly.

Fundamental differences between the servo and stepper motors are in their design and operation. The servo is simply a DC motor and generally only has rotation between 0-180 degrees. However, due to the requirement of contentious rotation, that is 360 degree rotation, for payload delivery, a continuous servo motor was evaluated. The internal design of the stepper motor is much more complicated and involves a bunch of magnets and inductors, refer to the references section for full details.

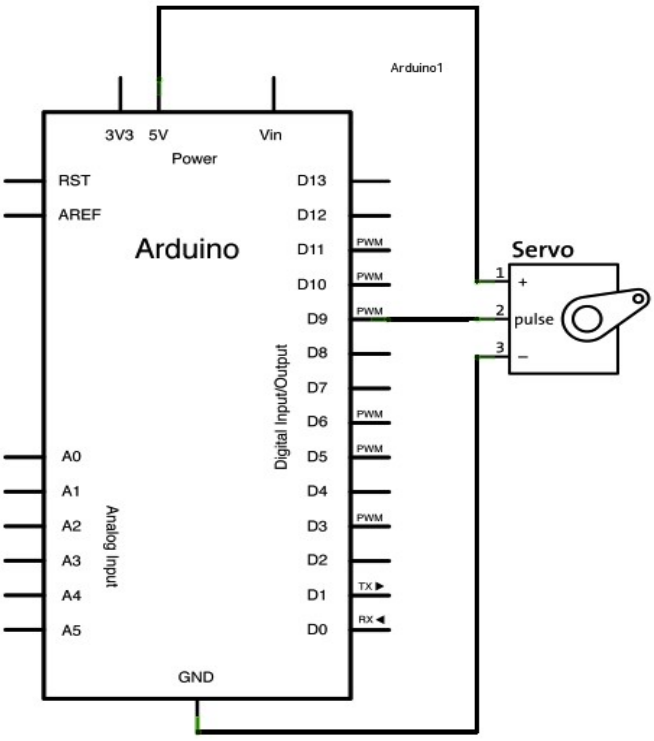


Figure 16. Arduino controlled Servo Motor

Referring to figure 16, the servo can be powered via the +5V signal coming from the arduino uno and controlled via a single arduino digital IO pin, more specifically pin D9. The potentiometer, or variable resistor, between the arduino uno's +5V and ground pins appears to assist in the stability of the circuit, though for the servo this doesn't seem critical. I have done my experiments without the pot for the servo and it works quite well. Refer to figures 16 & 17 for the arduino/servo motor & and arduino/stepper motor control circuit respectively.

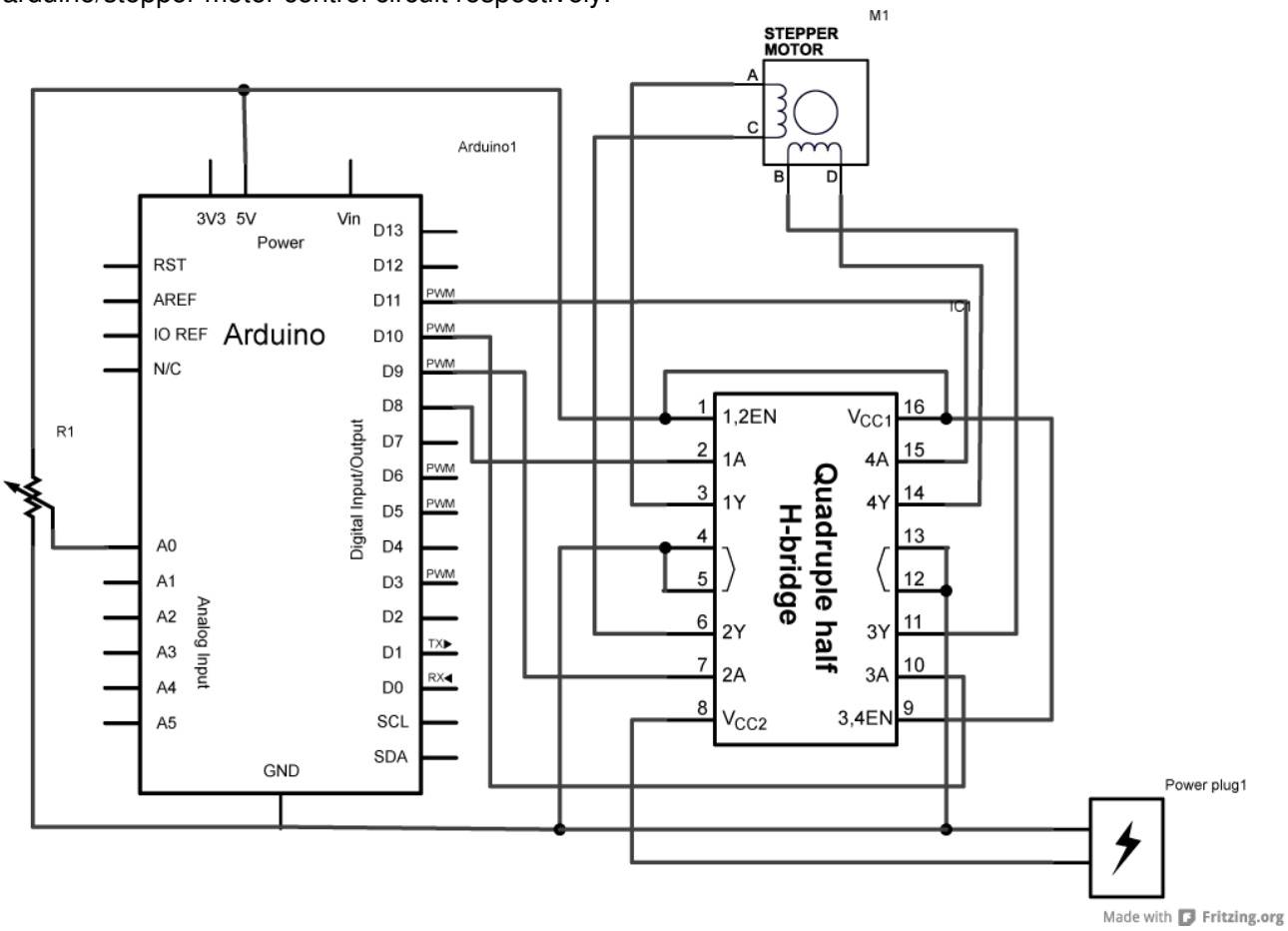


Figure 17. Arduino controlled Stepper Motor

Referring to figure 17, a Quadruple half H-Bridge IC is used to drive each phase of the stepper motor, which is controlled via the PWM(Pulse Width Modulated) digital outputs of the arduino.

Also note that the powering of these circuits was done using either a standard 9V battery, or the hexo+ battery from the charging JST-XH connector(pin 3 is about 8.4V). Hence, not the main EC3 power connector that interfaces with the drone. The hexo+ battery would be the lightest way to power the servo/stepper motor circuits, 9V batteries were used for convenience and in initial testing. The potentiometer, or variable resistor, between the arduino uno's +5V and ground pins appears to assist in the stability. Hence rotation distances appear to be more consistent when it is in place. Also of significance, the H-Bridge IC produces the regulated +5V that the arduino requires.

Motor control using the arduino ***WiFi servers:***

Arduino based Servo Motor Control WiFi Server & the Arduino based Stepper Motor Control WiFi Server:

The arduinoServoServer.ino controls a servo motor, and is thus the arduino servo motor server .
The arduinoStepperServer.ino controls a stepper motor, and is thus the arduino stepper motor server .

Both these servers run on one particular, and separate, arduino/wifi shield combination board. Obviously, this is due to the arduino being a microcontroller, thus only runs one task at a time, as opposed to a microprocessor.

The arduino motor control servers connect to a wifi network for communication with the GUI. More specifically, the GUI communicates via TCP/IP socket connections to the arduino wifi servers, over a wifi internet connection.

The wifi network **SSID** and **password**, are set in each individual server .ino file. The arduino server can be configured with a **SATIC** or **DHCP** IP address. Refer to the server code in-line documentation for how to do this.

Further to this, one will note that the server code is split up into two main sections. One is the **setup()** function and the other is **loop()** function. The setup() function is executed once, on server program startup. However, the loop() loops forever, and is essentially a while(1). These functions are, or structure is, common to all arduino microcontroller code.

Note also that both servers determine and display what version of firmware they have. If the firmware is less than 1.1.0 then the firmware of the wifi shield needs to be updated. I had to update the firmware on both wifi shields that I had in my possession before any of my code would work. A tedious but essential process, refer to my document describing this procedure called README.upgradeWiFiShieldFirmware.

Significant, *only* in the code of *arduinoServoServer.ino:*

The direction of motor rotation is required by the arduino motor server, and is sent from the gui(or client code) via the xfer2as() function. So firstly, the motor server reads this value. This value will be '1' for clockwise and '0' for anti-clockwise. The length of time the rotations continue for, in a particular direction, is determined by the length of time in the delay() function, or sleep function, between turning the servo on and turning the servo off. At this stage the delay() is nominal, for demonstration purposes only. However, in practice it will have to be the estimation of the time the servo needs to run for to deliver the payload to the delivery platform. This is the mode of operation I expect for the servo, as the winching process is quite smooth.

Alternatively, I can foresee perhaps a different mode of operation for the servo in production or in the engineering testing process. This would be when perhaps the total time the servo needs to be on is not known. Then one can imagine that you may want to winch down with a small number of rotations at a time. Which may require a different delay() value foreach group of rotations. The gui already has a stub for this delay time value. Small modifications would need to be made to xfer2as.cpp and arduinoServoServer.ino in this scenario. Time did not permit to implement every desired feature.

Significant, *only* in the code of *arduinoStepperServer.ino*:

The stepper motor used has 200 steps per revolution, this value must be set in the server code. Four pins from the arduino, linked to the arduino stepper motor library, control the operation of the stepper motor. These signals are input to the half-h driver circuit, which in turn is drive the phases of the stepper motor. These four pins have to be specified in the server code as well.

This server requires the rotation direction and the total number of rotations required, be sent to it from the client code. Thus the gui sends these via the xfer2as() function. Again here, as for the servo server, the rotation direction will be '1' for clockwise and '0' for anti-clockwise. In the section describing **xfer2as()** and **xfer2as.cpp** the reasons for reading the next four bytes, the number of rotations required in a four byte integer, from the client code are described in detail. But essentially, it is due to the

arduino server code only being able to read one byte at a time, and we have four bytes to the integer value that dictates the number of rotations. This appears to be a limitation with the arduino microcontroller.

Currently, the gui caters for the specification of the rotation direction and the number of rotations in two separate fields. This would normally be the 'engineering mode' of operation. As once the total number of rotations to deliver a payload has been determined empirically, you can have just one button delivering both winch down and winch up functions.

This is not the only scenario possible, one may want to winch down with some fraction of 200 steps at the one time, thus less than one full revolution. For this case the the server code would need to be modified slightly. Time did not permit to implement every desired feature.

Significant issues with the *development and operation* of the arduino motor control servers:

It's still not clear at this stage whether the stepper or servo motor will be used in production, there are still a few things to be worked out before that decision is made. For instance, some empirical tests need to be run. Such as testing what sort of torque is required in these motors to carry a typical payload weight. Also to be decided is the accuracy in rotation of the motor that is required. There are 200 steps, each step travels 1.8 degrees, for the stepper motor, so it is incredibly accurate. For the servo motor you just turn off the power to the motor when you want it to stop, and the turn the power on again when you want it to move, or rotate. Significantly, each time the servo is stopped it continues to travel just a little beyond the time that power was cut to it from the code, it travels just a few degrees extra, but is noticeable.

Also note that I have chosen a continuous rotation servo which has 360 degree rotation, as opposed to a traditional servo which has 180 degree rotation. However, continuous rotation servos can have inaccuracies and limitations, hence the decision to evaluate a stepper motor as well. Further tests need to be carried out to determine these shortcomings and limitations, and allow a proper evaluation between the motors.

Also, some thought should be given to the 'ease of use' with either type of motor. The servo motor powering and control is just a simple circuit, see figure 16. While the stepper motor is more complex in it's powering and control circuitry, see figure 17. The servo motor control works quite well, and is easy to get the desired operation from. However, there appears to be issues with the stepper motor circuit, with reasons not so obvious at this stage, so it's proving a little more difficult to use. So initially, the servo motor appears the simplest to operate. See the following paragraphs in this section for details of the stepper motor issues that require further investigation.

Testing revealed that the best operating voltage for the H-Bridge SN754410 driver circuit is about 9 volts at the Vcc2 input, even though it has a range of 4.5 to 36 volts. Outside of these voltage ranges the stepper would not operate properly. One symptom was a rotation of less than 200 steps, when it was programmed to do one full rotation of 200 steps. The movement, or rotation, of the motor sounded very laboured in these instances, probably not a good sign. From memory this would happen most often in the clockwise direction than in the anti-clockwise direction.

The performance of the stepper motor also seemed coupled with the speed at which you set it to operate at. That is, when you set the speed in the server code there seems to be a sweet point in its value, see the code for details. So a speed, above the sweet point speed and below the sweet point speed, the motor would either not turn at all or turn in a laboured fashion, as described in the previous section.

There are a few other things that are worrisome in the stepper motor control circuit. One being the motor gets a bit too hot for my liking(ie hot to touch), as does the H-Bridge IC, when in operation, and

especially when the input control voltage is is increased from ~9V to ~12V. This situation needs to be investigated further. As a starting point one needs to look at the PWM(Pulse Width Modulation) output signals from the arduino, with a CRO(Cathode Ray Oscilloscope), and look at how they are driving the H-bridge. This may require an understanding of how the arduino library functions are working in driving the motor. The other point of investigation would be what I think may be the reference voltages for the arduino, or R1 values. I initially used a simple voltage divider circuit to try to solve the problem. I have made some initial experiments as to this voltage divider circuit, currently a R1=1k and R2=1k stabilises the the stepper motor. By stabilises the motor I mean that the clockwise rotation doesn't travel a lot more, say a few degrees only, beyond the 200 steps per rotation, for each full rotation. The anti-clockwise direction seems to be at 200 steps, to the naked eye. Before using the voltage divider circuit the extra travel, beyond 200 steps, was more pronounced. I have used various R1 and R2 voltage divider values and the ratio doesn't seem to matter too much, just having a resistor between +5V and ground is useful, with the resistance tapped as input to A0(analog input zero) of the arduino.

Arduino IDE and the impact on the choice of board to provide the needed wifi facility:

To write, build and download the above mentioned server code to the arduino uno board one requires the use of the arduino IDE(Integrated Development Environment). On my development system, ubuntu 14.04, the arduino IDE version is 1.0.5.

Writing code for the arduino is very similar to writing in c or c++, but not exactly. For instance, you write to the arduino serial console in your code with special arduino functions, but fprintf()'s will not work. It also does seem to be built with classes underneath, given the documentation. Being a c coder myself it was quite easy to adapt to coding for the arduino. To the credit of the arduino community there are may examples of how to write code to control various sensors and shields. This readily available help contributed to my decision to use it for prototyping, especially when compared to the beaglebone black. The beaglebone black has a much smaller user base than the arduino community, but it does have more functionality. The other contributing factor was that I had used the arduino IDE for a home based project before.

The version of the IDE played a critical part in the choosing of which particular arduino, or arduino shield to use. In the 1.0.5 version of the arduino IDE there is no default support for the Arduino Yun. The yun has wifi included on-board, without requiring a separate shield. The cost of the yun was also less than the arduino uno plus wifi shield. The latest version of the arduino IDE is version 1.6.12 and it clearly and natively has support for the arduino yun. So in future, the arduino yun may be a better choice than the arduino uno/wifi shield combination.

In addition to the situation with the yun, the choice of wifi shield was also impacted by the support or otherwise of the shield in version 1.0.5 of the the arduino IDE. The wifi 101 shield is now supported in the 1.6.12 version of the IDE, but is not supported in the 1.0.5 version. Hence the use of the arduino wifi shield, which is defiantly supported in the older version of the IDE. The one downside to choosing the arduino wifi shield instead of the arduino wifi shield 101 is that the arduino wifi shield has now been retired, so they will be hard to get. Also of significance here, given that, when one gives a brief look at the WiFi Chat Server Code(the code my arduino servers were build from) for the 101 vs wifi shield, it looks about identical. To me it looks like the 101 library is now native in the latest arduino IDE and thus its wifi code uses the same function calls on the outside when compared with the wifi shield, with the underlying class function having the 101 specific code.

For reference, at this time and from sparkfun, which is generally good value and a good benchmark, the relative costs of the arduino boards discussed above are:

- Arduino uno r3 is $34.49 AUD
- Arduino Yun is $103.60 AUD
- Arduino WiFi Shield 101 is $97.41 AUD at LBE(Little Bird Electronics) – sparkfun appeared not

to stock this one.
- Arduino WiFi Shield $117.43 AUD
- Beaglebone Black Rev C $75.96AUD at sparkfun with about $10 AUD for a wifi dongle.

Thus if one could choose the Yun you probably would, if it came down to just cost and amongst the arduino based boards. However, the Beaglebone black(+wifi dongle) is even cheaper than the Yun. See the 'Beyond the back-end control system as described in this document:' section within the 'Future Work:' section for a discussion of how the Beaglebone Black is the next choice for the control system development.

GUI application for the ambulance(running on a ROS computer):

GUI Overview:

Library Function xfer2as() in xfer2as.cpp:

This function name stands for 'transfer(X) to Arduino Server'. It is the library function that allows client applications to communicate with the respective arduino motor control server.

In this function TCP/IP socket code is used to connect to the respective arduino server, arduinoServoServer or arduinoStepperServer. This function is called from the client application, at one or more times, to initiate requests to an arduino motor control server.

In the first parameter you pass the rotation direction, as a character '1' for clockwise and '0' for anti-clockwise . This parameter if for either the servo motor or the stepper motor.

Some interesting, or important, parts of the code in this function are described in the following. The arduino appears to only be able to read one byte, or one character, at a time. So for instance, to transfer an integer from the client/gui, say the number of rotations for the stepper motor, one has to convert this integer value to four separate bytes, and then send one byte at a time, to the relevant arduino server. The arduino server would then need to reconstruct the integer from the four bytes. See the code for the details of how all this was done. One more thing is of note on this issue, that is the endianess of both sides of this four byte transfer. Endianness is the order of the bytes that comprise a digital word in computer memory. Words may be represented in big-endian or little-endian format, thus indicating where their LSB(Least Significant Byte and MSB(Most Significant Byte) is. I have developed this client/gui software on an Intel processor and they are little-endian processors. With little-endian the LSB is the right most byte in the word, thus at the lowest address value of the four bytes. For big-endian it is the opposite, the MSB is at the lowest address value of the four bytes. So I have little-endian on the client side and as it so happens also the server side, or code running on the arduino. Clearly then, you can see that there was no need to cater for different endianess of the client and server code.

Library Function sendSMS() in sendSMS.cpp:

I think relying on humans to pick-up the payload in a timely manner, before the drone leaves the destination sight, is unreliable. For instance, that human could be in the toilet , not on site that day, in a meeting...etc. Consequently, code in the form of a library function has been written, that sends an SMS to a relevant official, once payload had been delivered to destination.

At this stage I have taken the decision that we would loose too much flying time, or battery energy, having the drone land, release payload, and take-off at each destination; remember the flying time of the hexo+ is only 8-9minutes. So instead, once the drone is over the QR pad, the payload is winched down and released, with the payload enclosure to be collected later by other methods yet to be determined. However, if empirical results show the energy lost in landing and take-off is not that significant I may change to landing the drone for payload delivery.

Ambulance GUI in general, and its source files and binary files:

Initially the GUI was to be suitable as a final piece to be used by the ambulance staff. However, during development it became clear that two versions of the GUI needed to be developed, a user version and an engineering version. The engineering version would be the most feature rich as it will facilitate the testing and development of the the back-end drone control system. The user version will have a subset of the engineering controls. A similar distinction will probably be made between the user an engineering versions of the front-end software control system. The distinction has also been drawn due

to not being able to easily foresee what the final requirements for the actual end user are at this stage. In addition, this system could be adapted to other use case scenarios with different but similar requirements. At this stage I would imagine the engineering and user version code base will be the same. With different options being enabled or configured for each respective system build.

The GUI was developed using the opensource, cross-platform, widget library called Qt. Development with the Qt widget toolkit is most easily facilitated with QtCreator, it is essentially the Qt development IDE.

The GUI source files encompass xfer2as.cpp & sendSMS.cpp described above. In addition, includes ambulance.hpp, ambulance.cpp, ambulance.ui and main.cpp outlined below.

All source code for the gui is in the **ambulance** directory in the git repository. However, the build files are in **build-ambulance-Desktop-Debug**. The command line tools, or programs, that lead up to the final files in the ambulance development project are in the **cli** directory. QtDesigner, a tool within QtCreator makes creating the gui interface **ambulance.ui**, it's signals and slots, or callback functions, relatively easy. To use QtDesigner one just needs a general conceptual understanding of container widgets, or layouts; and of course callback functions, or slots.

ambulance.pro is the project build file,
ambulance.cpp contains the class implementation,
ambulance.hpp contains the class declaration,
ambulance.ui is the user interface built with QtDesigner, a tool within QtCreator.
main.cpp *is the main application file where the ambulance class widget instance is created and realised.*

StartStreaming, csh script facilitating video streaming
GoProStreamKeepAlive.py, python script facilitating video streaming

Using the GUI:

While describing the following sections of the GUI the term 'group box' is mentioned often. It simply refers to a titled bordered area enclosing a group of widgets. If you refer to figure 18 below, a screenshot of the current GUI, you will be able to see the 'Send SMS group box'; the 'Motor Control' group box with 'Automated Control group box' and 'Manual Control group box' as sub group boxes...etc.

Default constructor:

Initialises some variables and attempts to start playing the video stream. The variables set are mentioned in the following sections.

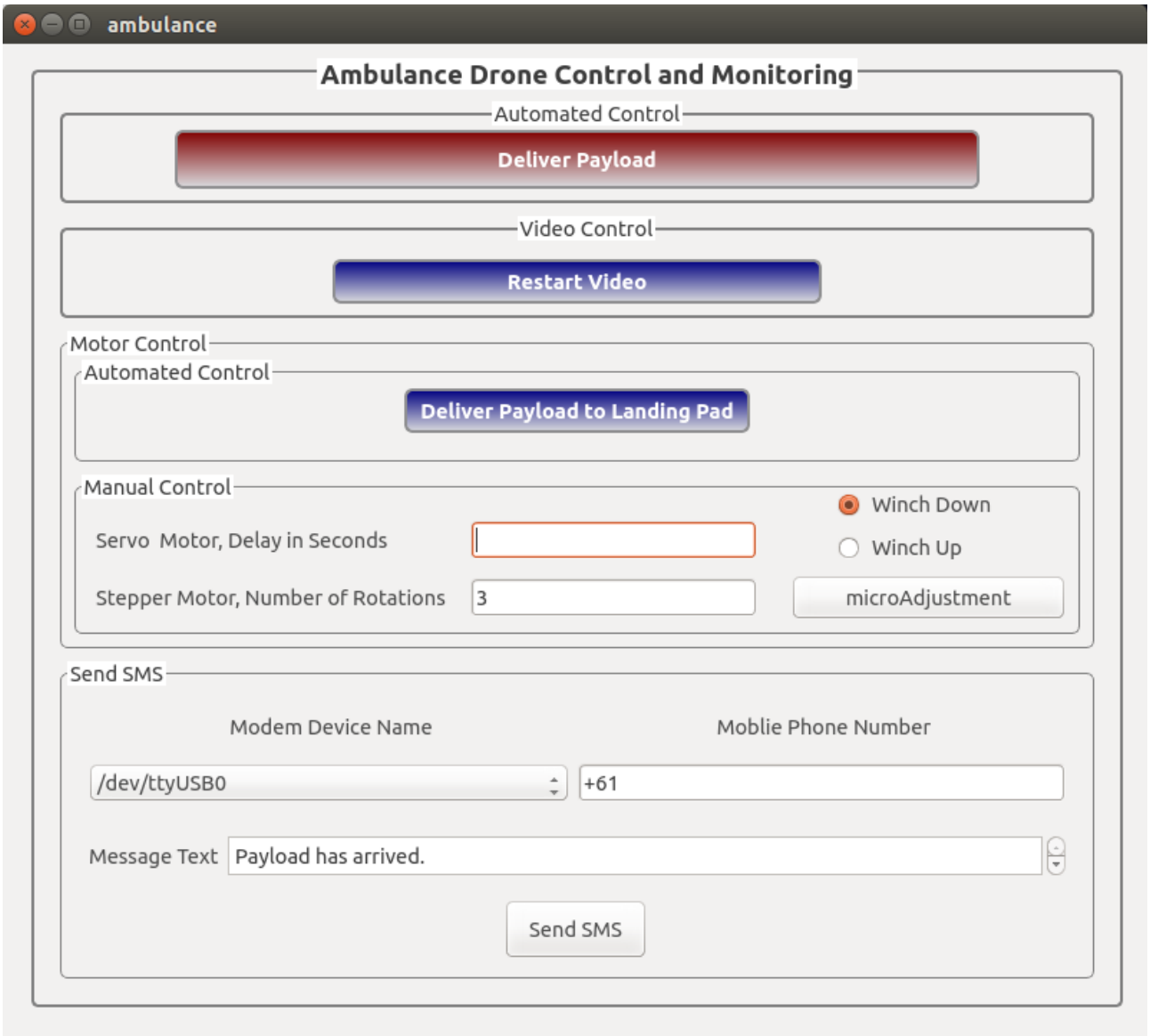


Figure 18. screenshot of current GUI.

Send SMS group box:

I have used a 3G usb modem for sending SMS's. It will be plugged into the ambulance ROS based computer. A post paid SIM card has been inserted and is used. An arduino shield was contemplated for sending the SMS from the arduino. However, the cost of a good 3G shield was close to $200 AUD, my USB modem is about $15 AUD. Since in Australia the 2G network is to be decommissioned soon the decision was taken to send the SMS via the 3G network. Also, when looking at the chip sets on these modems make sure that they cover all the Australian 3G networks, and look at the actual frequency bands, as sometimes they are misnamed. In summary, note that in the city of Sydney the 2100MHz band is the most used, followed by the 850/900 frequencies. So any device that can't transmit at one or more of those frequencies is useless.

The name of the modem device needs to be chosen from the drop down list. Looking at /var/log/kern.log will help you deduce the correct device name, if you watch this file as you insert the 3G usb modem into the computer's usb port.

The mobile phone number to send the SMS too, currently by default it is sent to my mobile number but any world wide number can be used, as it accepts any *+Country_Code* number. The default number

can be set in the default constructor.

The current default message text is 'Payload has arrived', which may be sufficient in most cases. To change the default text edit the default constructor. You can send a different message text, which can be longer than one line, from the field adjacent to the 'Message Text' field in the gui.

When the 'Send SMS' button is pressed the required AT* modem commands, to send the SMS with the details specified in the SMS group box, are issued to the modem, see the code in sendSMS() for specifics.

Motor Control group box:

The current functionality of the motor control group box is described here. The functionality possibly required into the future is covered in the previous sections on 'Servo and Stepper Motors: ***Motor control*** using the arduino ***WiFi servers:'.*** Also note that for the purposes of development I had for circuit one: arduino uno + wifi shield + servo motor connected; and for circuit two I had an: arduino uno + wifi shield + stepper motor connected. So essentially, each system had a separate WiFi IP address.

Motor Control, Manual Control group box:

Stepper Motor:

Firstly select whether you want to **winch** the payload down or up, via the radio buttons. These cause the motor to rotate in a clockwise or anti-clockwise direction respectively.

Next, the stepper Motor, number of **rotations field**. One rotation is considered one full revolution of the motor. The motor I have used has 200 steps per revolution, thus each step is 1.8 degree of rotation.

Then finally press the **microAdjustment** button to activate the motor with the previously chosen settings.

Servo Motor:

Currently, the servo motor delay in seconds **field** merely indicates to the rest of the program that servo motor control is required, once you hit return in the field.

Then finally press **microAdjustment** button to activate the motor. This will cause the motor to firstly rotate in the clockwise direction for a specific length of time, stops the servo momentarily, then rotates the servo motor in an anti-clockwise direction for roughly the same amount of time.

Both the servo and stepper motor controls need to be refined though further testing and development phases. Some possible future scenarios are outlined in the sections of, 'Servo and Stepper Motors: *Motor control* using the arduino *WiFi servers:'.*

Motor Control, Automated Control group box:

This is just a programming stub at this stage. The idea is that firstly the motor control is, refined and finalised, using the 'Manual Controls' for both the stepper and the servo motors. Then the sum of all these motions is put into one function that is activated when 'Deliver Payload to Landing Pad' button is clicked. Here we would assume that the drone is located roughly where the landing pad is, otherwise the action wouldn't make any sense.

Video Control group box:

Visual Feedback from the drones is designed to facilitate control and monitoring of the drones. For instance, determining if the drone/s may have crashed. Using an image of the QR coded landing pad to determine if the drone is at the correct payload delivery location. This image can also generate localisation data to help us land the payload on the landing pad.

Activate the 'Restart Video' push button, or click this push button, to make an attempt to restart the video stream. This simply calls the playStreamedCameraVideo() function, as is done in the default constructor for the ambulance class, see a description of this function below.

However, for streamed GoPro video you firstly need to put the GoPro into the correct mode. That is, put the GoPro into 'WiFi on', and 'GoPro App' Mode. The GoPro will then create it's own wifi hotspot. You need to connect your computer to this wifi hotspot. For the voyager1 drone's GoPro voyager1hero4black is the name of this hotspot, with password voyagerUTS.

Video Control group box: function playStreamedCameraVideo():

There are a few significant things to point out in the code for this function. The most important being, that an external system program, mplayer, is principally used to facilitate streaming video from the the GoPro.

The mplayer command to start the video stream is:

mplayer -vf screenshot -nosound -geometry 532x340-16+65 udp://:8554";

- **-vf screenshot** allows you to press s on the keyboard to take a screenshot.
- **-nosound** is optional, I just used to prevent feedback from my desktop speakers to the GoPro.
- **-geometry** is obvious, and as per any x based program.
- the GoPro video stream is on **port 8554** of a **UDP socket.**

mplayer was chosen here principally due to me having a little bit of familiarity with the program, one may have used the ffmpeg suite of executables, vlc...etc. However, it appears that mplayer is using ffmpeg executable suite underneath. In addition, using an external program like mplayer was the first method of streaming video from the GoPro that I could get to work. A number of other methods were tried before the mplayer winning method was found, these being:

- using QMediaPlayer and QVideoWidget Qt widget classes
  - firstly, using these classes required the install of QtCreator 3.4.0 based on Qt 5.4.1 from source, these were not the standard versions on ubuntu 14.04.
  - the main problem here was that these Qt widget classes appeared to use gstreamer media player underneath which came up with errors that weren't easily solved at the time and after a few quick attempts. Perhaps my, now old, version of ubuntu 14.04 did not help the resolution of these errors, as ubuntu 16.04 has been released this year.
- using the external widget library class qmpwidget for embedding mplayer into Qt
  - installation of this widget class was straight forward but implementation quickly became difficult
- c language system() call was used but this does not allow you to return to the gui event loop

However, before the mplayer command is called some stream maintenance needs to be performed. This is all initiated from the execution of the csh script called **startStreaming**. In summary, what startStreaming does: at various stages it cleans up the old invocations of this script; initiates the **GoProStreamKeepAlive.py** python script which keeps the stream operating beyond its timeout period of just a few seconds; and initiates a non-interactive download from the web by connecting to a URL,

using wget.

Video Control group box: function playStreamedCameraVideo():
How to run external programs, mplayer and the startStreaming csh script, in a Qt gui:

The way to do this in Qt is to use the **QProcess** class member functions.

To execute the mplayer command stated earlier, the **start(program_name, argument_list),** QProcess member function is used. This ensures the persistence of the mplayer command beyond the duration of the callback function and default class constructor. Essentially, mplayer is put into a separate process from the gui process. In addition, with the start() member function, mplayer will close when the gui or calling process exits, which is the desired outcome in this case anyhow.

For reasons which are still not entirely clear at this stage, a different QProcess member function is required to execute the startStreaming csh script. This being, **startDetached(program_name, argument_list).** It is important to note one main difference between the start() and startDetached() methods. This being, that when the gui or calling process exits, the program started with startDetached() continues to ***exist***, where as a program started with start() ***exits***. As a result it is clear why processes started within startStreaming need to be cleaned up, as stated earlier, before a new set of these processes are started with the next invocation of startStreaming.

Video Control group box: implementation details:

At UTS(University of Technology Sydney) there is a saturation of wifi networks within building 11 such that streaming wifi from the GoPro is not practicably possible in this building. One needs to go to areas of the building where the UTS wifi networks do not have, any or much, reach, such as: in a microwave oven(unplugged of course); very close to a corner of a concreted area; or in one of the lifts at UTS building 11. The way I was able to portably ascertain the ability to stream wifi from the GoPro while roaming the university grounds was by running the GoPro wifi streaming app on my mobile phone, and carried this in one hand, while holding the GoPro in the other hand. However, as described earlier the GoPro needs to be in the *'WiFi on', and 'GoPro App' Mode* for this to work. With this method, when I lost, or never gained, a picture on my phone it meant you can't stream wifi from the GoPro in that area.

I was able to confirm my suspicions about a saturated UTS wifi network using the 'WiFi eye' app on my android phone. 'WiFi eye' clearly shows all the wifi networks within range and their signal strengths. And fully confirm my suspicions buy walking out of building 11 and within about 100m from the door the wifi stream on the GoPro app on my phone returned.

The other implementation detail of consequence is what appears to be a two minute timeout when the video has previously been started, and a restart attempt is being made. For reasons that are currently illusive you can stop and restart the video two times(including the initial default constructor starting of the video) without too much delay on start-up. On the third or further invocations of the video, there will be a large delay before the video starts, if there hasn't been about two minutes of clock time elapse between restarts. This delay is about two minutes at maximum, in practice you need to wait a little less than two minutes. Some future work needs to occur here to resolve this problem.

QR code detection on payload landing mat:

At this stage I have come part way to implementing this feature. Where an image from the GoPro of the landing mat is compared to an existing Qr-code image identifying a particular payload receiving facility. The idea is to make sure we are delivering the correct payload to the correct location. Following, are the results so far. For reference, Qr-codes are machine readable codes consisting of black and white squares, typically used for storing URL's or other information. One requires software to encode and decode a QR code.

Firstly, I generated a number of Qr-codes from text strings. I used the **qtqr** Qt gui tool for this and the results are in my git repository, namely, bad.png, helloWorld.png uts.png...etc. In practice these would be the particular payload receiving facility's Qr-codes, thus unique Qr-codes. Following is a procedure for obtaining an image being seen by the GoPro whilst over the landing mat, and with the lens facing the landing mat(would need access to the gimble control from the on-board embedded drone system/SDK, which is for future work).

We need to refer to the earlier mentioned mplayer command used for streaming GoPro video, this is:

mplayer -vf screenshot -nosound -geometry 532x340-16+65 udp://:8554";

- **-vf screenshot** allows you to press s on the keyboard to take a screenshot.
- **-nosound** is optional, I just used to prevent feedback from my desktop speakers to the GoPro.
- **-geometry** is obvious, and as per any x based program.
- the GoPro video stream is on **port 8554** of a **UDP socket.**

Of significance here is the first option **-vf screenshot.** As the note says, the user, or ambulance computer controller, would have to manually intervene here, pressing the s button on the keyboard to take a screenshot of what the GoPro is currently looking at. With each press of the s keyboard button mplayer will save the screenshot as a file named shot????.png, it will start with shot0001.png, shot0002.png, shot0003.png...etc.

Then to determine if the drone is at the correct facility I need to decode the shot*.png file landing mat Qr-code to a text string. Then compare this text string with the corresponding Qr-code text string given to me from the expected facility, if both strings match we are at the correct facility and should land the payload.

To facilitate this final stage string matching I have written a csh script called **png2qrCode.argv1**, also on the git repository. As input it takes the name of the shot*.png file and it will output the Qr-code text string or indicate if there is an error in decoding the code. Of significance in this script is the use of the **zbarimg** binary to accomplish the decoding of the .png image to the text string.

The overall functionality required for determining if the drone is at the correct location, using Qr-codes, has just been described. See the 'Future Work' section for how I might integrate this functionality, and some more, into the gui.

Automated Control group box:

Activating the 'Deliver Payload' push button will most likely be an end user control feature but may still be useful as an engineering tool. Essentially, this button will be the one and only button press or action that needs to take place for the payload to be delivered, when it is activated from the ambulance or during and engineering test. The aim is for it to combine all the functionality and field values of all the group box's in the gui.

Non-Core Results:

The first non-core result was to gain familiarity with ROS(Robot Operating System) programming. This was mainly to fulfil robot programming milestones that are required for future work with the drones. In addition, it was to provide a personal milestone of relevance to my professional development, that being C++ and ROS programming experience.

Using the Microcone to be strapped to the PR2 robot:

JSK software:

Initially, it was thought that using the JSK software could be used to control the Microcone, thus being used to distinguish the seven angles/voices talking into the Microcone. Hence, it was hoped this would be a task to help me get familiar with ROS. However, after the initial investigation into the JSK software it proved the path to using it for such a task was going to be too time consuming, especially when compared with other ways to get upto speed with ROS.

ALSA Linux Drivers:

After abandoning my attempt to use the JSK software to control the microcone, I made an attempt to use the ALSA linux drivers to control the microcone. This was simply another avenue to follow and the code for this is in the git repository, in the **microcone** directory. With this C code I produced a set of *.raw files of recordings from the microcone, standard laptop microphone, and a professional voice microphone. However, this data wasn't telling me a lot on its own. So I looked into other methods of recording and viewing sound file data.

For instance, the linux program called audacity. For this experiment I again have used as input the microcone, standard laptop microphone, and a professional voice microphone. Audacity was able to display all seven channels of data from the microcone. From looking at this data, it was evident that a lot of signal processing needs to take place to decipher what is in the signal, what is being said, and who it belongs too. The audacity recordings for each microphone are the *.mp3 files in the above mentioned git repository.

ROS tutorials:

So by this stage getting upto speed with ROS programming had not occurred. After a discussion with work colleges they suggested various ROS tutorials that would be of help to me in getting upto speed with ROS. Consequently, I decided to do some introductory ROS tutorials, involving simple publisher and subscriber nodes.

The following ROS tutorials were completed, namely:

- simple publisher subscriber for a ROS indigo system, from within the indigo workspace
- simple publisher subscriber for a ROS catkin system, from within the catkin workspace
- creating a ROS first package with the indigo and catkin workspaces
- ROS catkin node_example, listner.cpp & talker.cpp, and, listner_node.cpp & talker_node.cpp

All the lessons learnt from the above mentioned ROS tutorials culminated in the development of my ROS catkin voice_node, namely: simplePub.cpp & simpleSub.cpp, and simplePub_node.cpp & simpleSub_node.cpp.

All of the above mentioned code is available on my git repository under the repositories rosVoiceNode for the ROS indigo work and catkin.voice_node for the ROS catkin work. There is a description on how to run my voice_node publisher & subscriber nodes and some example output in

catkin.voice_node/src/README. Also, note that there is a small bug in the voice node software, again refer to the catkin.voice_node/src/README for specific details. The problem seems to be with soundplay_node.py, and has been verified to occur by other members of the MagicLab group.

The catkin voice node I have written will be used as a template for my ROS Control/Monitor Nodes, as it is a good example on how to publish to multiple topics and subscribe to multiple topics within ROS. At this stage it is clear that the monitoring node will need to :

- publish data related to the visual monitoring of the drones(back-end control system)
- publish drone flying parameters(front-end control system)

The control node will need to subscribe to topics that relate to the manual controlled flying of the drones, and publish data with respect to this. However, note that drone flying should be mostly autonomous.

Why go from a ROS indigo workspace(indigo_workspace) to a ROS catkin workspace(a catkin_ws)?

It eventually became clear that a catkin workspace was being used in the ROS programming documentation, not the indigo workspace. Thus it was thought best to create this workspace for the neophyte to follow the examples more easily. Note I had a ROS indigo system already installed on my development system.

Why work through the ROS example called node_example & create my voice_node ROS code?

At this stage it is thought that the GUI front-end would interface to a ROS node on the ambulance ROS based computer. Hence, in going through the most obvious helpful example, being node_example, it was hoped that it would facilitate the writing of my own ROS node code.

So after completing the ROS catkin example called node_example I was able to successfully write my voice_node ROS node code. My voice node allows you to change the voice of a ROS catkin system and pronounce whatever words you like. It publishes on two topics and subscribes to those two topics, namely, txt4TTStopic and voiceNameTopic. The former topic facilitates converting text to speech and the later topic facilitates selection of the voice you want to use to speak the text.

Future Work:

In my estimation, there is about four months worth of work required to develop the back-end system alone. This would get it to a state were you could make judgements, of success or otherwise, about different design decisions. I have had about two months worth of work on the back-end system, three months would have produced a more useful result.

So what is left to do for the back-end control system as described in this document:

- QR code script integration into main gui
  - To take the QR code image of the landing mat: we need to learn how to control the camera of the drone to face the ground. That is, we need to work out how to control the gimble of the hexo+ and the phantom4. To execute this, access to the on-board drone embedded system, via the SDK, is required.
  - Run the **png2qrCode.argv1** script using QProcess member function startDetached()
    - compare the output string from this script with the given string from the delivery facility; a positive match indicates we are at the correct location.
  - For instance, the above functionality can be put into a 'QR code match push button pressed' callback/slot.
  - Or perhaps, implementing all the **png2qrCode.argv1** functionality using C or C++ and the zbar library.
- Perhaps, the of FakeGPS android app to spoof the location of a mobile phone, then attempt to get the hexo+ to lock onto it; a crude form of changing the position of the drone, while without SDK control.
  - Inhibitors to making this happen are:
    - Is hard to access a suitable park for test runs
    - Inaccuracies of GPS positions could cause obvious problems
    - Most of these location spoofing app's appear **not** to allow you to set an offset to the current GPS position; and require you to specify an absolute GPS position or to point to a position on a map. Both these methods are not so conducive to drone flying.
  - *Really the best solution here is to use the SDK to change drone location, or modify an existing app to allow for offset positions.*
- mplayer, video restart time-out problem needs to be solved; or mplayer functionality needs to be integrated into the gui using Qt classes, as described earlier in section 'Using the GUI: Video Control group box:'
- Investigation into why the stepper motor circuitry doesn't *appear* to be very stable, for details refer to the section called '*Motor control* using the arduino *WiFi servers:* Significant issues with the *development and operation* of the arduino motor control servers: '.
- 3D printing of payload dome enclosure, thus one needs to spend the time, probably about a week, to learn a 3D printing design tool
- Not a lot is required, but a bit more robust error handling in the odd sport.

Beyond the back-end control system as described in this document:

One could investigate porting the arduino code to a Beaglebone Black embedded system. It clearly is the next step up in development platforms with the needs of this project. There you would be able to use a full OS like ubuntu on your system, making your client-server server code more portable than the arduino servers. The Beaglebone Black is probably all you need, with respect to CPU speed and GPIO ports, for prototyping the back-end and front-end control system for this project. The Beaglebone

community support is not as large are the Arduino community but is probably large enough for the expertise of the people within the UTS MagicLab group.

Moving to the Beaglebone would also reduce the cost of prototyping of this system, which if used on multiple drones becomes significant, as:

- Arduino + WiFi shield is about $150AUD
- Arduino Yun is about $105AUD
- Beaglebone Black + WiFi dongle is about $80AUD

Note, the Raspberry Pi is not suitable due to lack of sufficient GPIO ports and their inability to drive motors; it has no analog inputs too, which are required for lots of different sensors, eg temperature, moisture...etc.

*As a side benefit, or spin-off to this project, we may end up developing a low cost opensource payload delivery system, attachable to **any** drone! Which may have social benefits for commercial and personal applications.*

Drones like the hexo+ and phantom4 could easily be used for commercial and personal applications, with a cost that is bearable by their users. One example that comes to mind is for farms or large scale properties, starting at about the 5 acres mark in size. Where they could be used for sending tools, getting the day's lunch, fetching any other household item...etc, between the house and shed/s or other farm locations.

The drone should be able to be locked to commands from only one person(or only from His Masters Voice, ie HMV), or a small group of persons, via a voice recognition key. The drone will accept voice commands to location landing pads, fixed or otherwise. Perhaps machine learning could be used for voice recognition.

However, I suspect there are many groups that could benefit from low cost drone applications to augment their activities and improve productivity. Thus many 'use case' scenarios exist. To bring these scenarios to light there probably needs to be some wide scale surveys and discussions had between myself and various consumer and government service groups. See the next section for how this affects the 'ambulance scenario' presented in this document.

Consultation with government services or industry bodies:

The other important work that must take place soon is with contacting hospitals and ambulance groups for their input, to better inform the system design of the back-end and front-end drone control system. These is no point in building a system that isn't useful to someone.

Integration of various GUI functionality with ROS control and monitoring nodes(using publisher/subscriber code).

Next main technical milestone for this project:

Once the back-end control system future work is sorted out the next logical step would be development of the front-end control system. However, front-end development may be done concurrently with back-end development.

At this stage it appears that whatever application is chosen for the drones the first step would be to workout how to change the location of the drone from the SDK's we have, giving us ultimate control on where the drone is. This would apply both to the Hexo+ and Phantom4 drones that we now have.

Hexo+ SDK:

As of August 2016 the Hexo+ SDK, after much negotiation and signing of multiple NDA's(Non-Disclosure Agreement), was finally provided to us, in a rudimentary form and with little documentation. However at that stage the development of the back-end control system was already under way so no time could be devoted to the front-end control system that requires it. One embedded software engineer, now needs to sit with the SDK for a time to come to grips on how to use it, it is a non-trivial exercise. Once this is done one can start looking for ways to change the location of the drone by programming the on-board embedded system.

Also of note, we were given a full working IOS app for the hexo+, and Android libraries. Possibly, the position of the drone can be modified from the IOS or Android app code. This needs to be determined.

DJI Phantom4 SDK:

As mentioned earlier in Table 1., the phantom4 has a mobile SDK, so essentially an SDK for the IOS and Android Apps. Similarly, there is a possibility of being able to modify the position of the drone within these SDK's. As of a few weeks ago the DJI Phantom4 has arrived in the lab. Of course I need to learn to fly this bird before all of this SDK work.

Front-end control system:

The front-end control system will either consist of one drone flying, or more than one drone flying in what is known as a swarm. Firstly, one would need to investigate the use of the DJI mobileSDK for how much control this gives you over the position of the drone. Perhaps as a temporary measure, would be to use position spoofing with a phone app like FakeGPS , an android app, where drone will lock onto where it thinks is the GPS location of your phone is. However, read section 'So what is left to do for the back-end control system as described in this document:' above for as to why this may not be a good idea. Investigation of the mobileSDK should probably done before advancing to the the DJI onboardSDK of the Matrice 100 drone. In the long term, interface the DJI Matrice 100 to existing front-end and back-end systems is probably going to be required.

For the hexo+ the best option is going to be to use the on-board SDK we now have. The mobile app we have may also be of use here.

Upgrade of the current temporary payload enclosures to 3D printed enclosures:

3D printed payload enclosures would allow for fast prototyping if any changes need to be made in design. Also of note, 3D enclosures are not water proof, but I suspect there is some sealant that can be used to make them water proof.

Note that the advice I received from an experienced designer of 3D printed enclosures is that it takes about a week to get upto speed with using a 3D printed CAD tool. So this task was prioritised out of this project for the time being, the timeline was too tight. It appears that there are some non-propitiatory CAD packages around that I could try, so they would be the first port of call. However, SolidWorks for windows appears to be widely used.

Qr-code detection on landing mat:

In the back-end system design description, section 'Back-end Control System(3) Payload Delivery', it was stated that pattern matching with OpenCV was to be used to detect the Qr-code landing mat. This method has been forfeited for the simpler QR-code method. Also stated in this section is that a soft landing pad was required, I have since revised my design to be that described in section 'Payload Delivery System: Hollow Oval Dome as the 3D printed payload carrier:', within the 'Back-end Control

System: Some design decisions explained and to set the picture:' section.

To execute this fully one would need to gain control of the gimble via the SDK of a particular drone. As the camera lens needs to be parallel to the landing mat to take an image. The hexo+ uses a GoPro gimble. For the phantom4 you would either use the native camera and gimble or fit a GoPro gimble to it, which should be possible.

Also for future work, we may be able to extract localisation data from the Qr-code to help land the drone's payload on the mat.

Refer to section 'So what is left to do for the back-end control system as described in this document:' above for how to integrate **png2qrCode.argv1** script functionality into the gui.

Long term future, beyond the prototype, to a final product:

If the talks with the hospital or ambulance staff prove fruitful, or with any other group for that matter, one would hope to iterate front-end and back-end control system prototype development to a final usable product for the emergency services or any other group.

Scaling the prototype:

- need to solve the problem of <100m range for wifi, for visual monitoring of drones from base station
- need to solve the problem of TCP/IP control of back-end system over a <100m range wifi, from base station to drone.
    - may need wifi repeater stations to do this

However, one can use a system suggested by the literature, mentioned earlier, to solve the base station to drone communication range problem, where:

- wifi can be used for uav-uav communications
- mobile cellular network used for uav to base station communications

Arduino wifi motor servers & motor development and operation:

Variable delay value for the servo motor server operation, as described earlier in section titled 'Significant, *only* in the code of *arduinoServoServer.ino:'.*

Other features to be developed, as described earlier in section titled '*Significant, only in the code of arduinoStepperServer.ino:.*

Also, the need for a lot of other features in these servers are probably not defined yet.

Other motor related future work is discussed in the section titled 'Significant issues with the *development and operation* of the arduino motor control servers:'

Conclusion:

The aim of this project was to deliver a back-end control system for an 'ambulance scenario', which is as such:

Our Ambulance Scenario is:

- A number of drones will be mounted on the ambulance in designated areas.
- A ROS/Qt GUI based application will be run on a computer/embedded system within the ambulance.
  - The application is used to control and monitor the flying drones, and also manage their payload delivery
  - Drones will fly off the ambulance to the designated target.
  - One or more drones will fly off the ambulance at any particular time.

Due to the late arrival of the Hexo+ SDK and the DJI Phantom4 drone, work on the front-end control system, or drone *flying* system, was not possible. So in the above scenario all that is left is to work on the back-end control system consisting of:

- A ROS/Qt GUI based application will be run on a computer/embedded system within the ambulance.
  - For the back-end system this application is used to monitor the flying drones only, and also manage their payload delivery

The 'Results:' section of this document details the prototype back-end control system I have developed. All *functionality,* aside from changing the location of the drone has been achieved, with some polishing to go as detailed in the 'Future work:' section(such as QR code matching integration with the GUI, mplayer two minute timeout...etc).

From the traditional literature review some insights have been gleaned. Namely, that uav to uav communications are good enough over wifi and that uav to base station control needs to occur over another network, the mobile cellular network being the obvious choice, especially given the handy repeaters everywhere. And the control systems are sensor based. This essentially solves the problems that would be associated with scaling the prototype, namely:

- need to solve the problem of <100m range for wifi, for visual monitoring of drones from base station
- need to solve the problem of TCP/IP control of back-end system over a <100m range wifi, from base station to drone.

The next stage of development will focus on the front-end control system, or flying control system. Starting by using the respective SDK's to change the location of the drone. I consider this the next main *technical* milestone for this project. Both mobile(phantom4) and on-board SDK's(hexo+) are now available to be looked at. I suspect using GPS location spoofing on a mobile phone to change the drone's location is not going to be any quicker than learning how to use the SDK's. This is because it appears as though one would need to expend effort in modifying an existing app, for instance, FakeGPS.

However, consultation with interested parties needs to take place to inform this process. And perhaps approaching more groups than the 'ambulance scenario' involves, like farmers. This process could potentially identify many commercial and personal applications for a drone control system. I would also expect some form of AI(Artificial Intelligence) needs to be used within such a system, the guise of which is yet to be determined. However, it is clear from the literature that AI is used in the flying of drones.

As a side benefit, or spin-off to this project, we may end up developing a low cost opensource payload

delivery system, attachable to any drone! Which may have social benefits for commercial and personal applications.

So what lies beyond this project for the back-end control system. One could investigate porting the arduino code to a Beaglebone Black embedded system. This would give us more portable code and a cheaper back-end control system.

And perhaps machine learning may be used to have a drone only respond to HMV!

Also, integration of various GUI functionality with ROS control and monitoring nodes(using publisher/subscriber code).

The drone that is going to be the long term solution, hexo+, DJI Phantom4, DJI Matrice100, or some other drone is yet to be determined. And the move from prototype to final product is quite a huge step too.

References: Section Heading followed by references.

Traditional Literature Review:

[1] googleDocs
https://docs.google.com/spreadsheets/d/1tPDRtMtJsARVJXLwWscXz21GMHTQGDOUXUfQ9vrUwdQ/edit#gid=0

Traditional Literature Review, specifics:

Christian Wietfeld, most relevant to least relevant:

[2] http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4815797
[3] http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5624356
[4] http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5624356
[5] http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5506428
[6] http://dl.acm.org/citation.cfm?id=2750678
[7] http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6998482&tag=1

Sebastien Varrier, most relevant to least relevant:

[8] https://hal.archives-ouvertes.fr/hal-00640404/document
[9] http://www.tandfonline.com/doi/abs/10.1080/00207179.2014.986201
[10] http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6862445&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6862445
[11] http://www.ifac-papersonline.net/Detailed/55593.html
[12] https://www.researchgate.net/profile/Alexandre_Seuret/publication/48414390_FeedNetBack-D04.03_-_Design_of_Robust_Variable_Rate_Controllers/links/53e081be0cf27a7b830a4390.pdf

ROS documentation
http://wiki.ros.org/ROS/Tutorials

Jsk and Microcone links.
https://github.com/jsk-ros-pkg
https://github.com/jsk-ros-pkg/jsk_3rdparty

Upgrade WiFi Shield Firmware:
https://www.arduino.cc/en/Main/ArduinoWiFiShield
https://www.arduino.cc/en/Hacking/WiFiShieldFirmwareUpgrading
https://github.com/arduino/Arduino/tree/master/hardware/arduino/avr/firmwares
https://www.arduino.cc/en/Main/ArduinoWiFiShield
https://www.arduino.cc/en/Guide/ArduinoWiFiShield
http://katrinaeg.com/arduino-wifi-firmware-upgrade.html

Sonar
https://en.wikipedia.org/wiki/Sonar
https://en.wikipedia.org/wiki/Acoustic_location

Radar
https://en.wikipedia.org/wiki/Radar

Sonar vs Radar
https://www.google.com.au/webhp?sourceid=chrome-instant&ion=1&espv=2&ie=UTF-8#q=differnce%20between%20radar%20and%20sonar
http://www.differencebetween.com/difference-between-radar-and-vs-sonar/
https://answers.yahoo.com/question/index?qid=20080512154609AAqMFOl
http://www.physlink.com/education/askexperts/ae456.cfm
http://www.encyclopedia.com/doc/1G2-3437800496.html
http://teacher.scholastic.com/activities/explorations/bats/libraryarticle.aspItemID=234&SubjectID=110&categoryID=3
http://www.venompower.com/products/traxxas-aton-rc-drone-quadcopter-35c-3s-5000mah-11-1v-lipo-battery-by-venom
https://hexoplus.com/product/hexo_drone_3d
https://www.jbhifi.com.au/cameras/video-cameras/gopro/gopro-hero4-action-video-camera-4k-black-edition/644300/

Drones
https://www.dji.com/product/phantom-4/info
https://www.youtube.com/watch?v=-ZJJf93vrFA
http://www.phantompilots.com/threads/payload-on-phantom-4.79433/
http://www.rcgroups.com/forums/showthread.php?t=2670690
http://fpvlab.com/forums/showthread.php?31833-Pushing-the-DJI-Phantom-to-the-limit

http://dji-phantom4.com/dji-phantom-4-vs-inspire-1/
http://drones.specout.com/l/452/DJI-Phantom-4
https://www.aus-cameras.com/dji-phantom-4-with-3-batteries-charging-hub-64gb-drone-quadcopter

Hexo+
https://www.digikey.com/product-detail/en/6450543-5/A114251-ND

SDK's
https://developer.dji.com/mobile-sdk/documentation/introduction/mobile_sdk_introduction.html
https://developer.dji.com/mobile-sdk/documentation/introduction/index.html
https://developer.dji.com/onboard-sdk/documentation/hardware-setup/index.html
https://developer.dji.com/onboard-sdk/documentation/github-platform-docs/Linux/README.html
https://developer.dji.com/onboard-sdk/documentation/github-platform-docs/ROS/README.html
https://developer.dji.com/onboard-sdk/
https://github.com/dji-sdk/Onboard-SDK-ROS
https://developer.dji.com/onboard-sdk/documentation/github-platform-docs/ROS/README.html

Matrice
https://developer.dji.com/products/#!/mobile
https://developer.dji.com/products/#!/mobile

Payload
https://www.djivideos.com/video_play/3b7b0411-37ff-4323-b4a5-04756db05b8

Matrice 100
http://www.dji.com/product/matrice100?site=developer
https://www.djivideos.com/video_play/3b7b0411-37ff-4323-b4a5-04756db05b8b

http://store.dji.com/product/matrice-100?site=brandsite&from=buy_now_bar
https://www.youtube.com/watch?v=Ii9xS6iRR0o
https://www.youtube.com/watch?v=_eL8rmjV870

Matrice 600
http://www.dji.com/product/matrice600?site=developer
https://www.djivideos.com/video_play/dff97db5-6530-4e71-be6b-7a4a98a81a75?autoplay=1

Arduino
https://www.arduino.cc/en/Main/ArduinoBoardUno
https://www.sparkfun.com/products/11287
https://www.arduino.cc/en/Main/ArduinoWiFiShield
https://www.arduino.cc/en/Reference/StepperBipolarCircuit

QR codes
https://apps.ubuntu.com/cat/applications/qtqr/
http://manpages.ubuntu.com/manpages/wily/man1/zbarimg.1.html

Video
http://www.mplayerhq.hu/design7/news.html

Coding:

QProcess and detached:

https://www.cs.umd.edu/class/sum2003/cmsc311/Notes/Data/endian.html

Servo & Stepper motor:

LBE stepper motor and servo motor specs.
https://littlebirdelectronics.com.au/products/stepper-motor-bipolar-200-steps-rev-20x30mm-3-9v-0-6-a-phase
https://littlebirdelectronics.com.au/products/springrc-sm-s4303r-continuous-rotation-servo
https://littlebirdelectronics.com.au/products/easydriver

Wikipedia servo and stepper pages.
https://en.wikipedia.org/wiki/Servomotor
https://en.wikipedia.org/wiki/Stepper_motor

Voltage Divider circuit.
https://en.wikipedia.org/wiki/Voltage_divider

Arduino IDE 1.0.5 and 1.6.0 links
https://www.arduino.cc/en/Main/ArduinoBoardYun
https://www.arduino.cc/en/Main/ArduinoWiFiShield

https://www.arduino.cc/en/Main/ArduinoWiFiShield101
https://www.arduino.cc/en/Main/ArduinoWiFiShield
https://www.lifewire.com/range-of-typical-wifi-network-816564

Arduino IDE 1.6.0
https://www.arduino.cc/en/Guide/HomePage
https://www.arduino.cc/en/Guide/Linux
https://www.arduino.cc/en/Main/Software
https://www.arduino.cc/en/Tutorial/Wifi101WiFiChatServer

Costs of arduino boards:
https://www.sparkfun.com/products/11021
https://www.sparkfun.com/products/13787
https://www.sparkfun.com/products/12053
http://www.pakronics.com.au/products/arduino-wifi-shield-101-ada2891?gclid=CjwKEAjws5zABRDqkoOniLqfywESJACjdoiG_g8Jx5DWUcTX34vBn4QvOoDR2LiQSeRllDekWL3zuBoCtHLw_wcB
https://littlebirdelectronics.com.au/search?type=product&q=arduino+wifi+shield

Arduino vs Raspberry Pi 2 and Beaglebone Black:
http://makezine.com/2013/04/15/arduino-uno-vs-beaglebone-vs-raspberry-pi/
http://makezine.com/2014/02/25/how-to-choose-the-right-platform-raspberry-pi-or-beaglebone-black/
https://www.quora.com/What-are-the-major-differences-between-BeagleBone-Raspberry-Pi-and-Arduino
https://www.youtube.com/watch?v=xg3i1o5aQRY
https://pimylifeup.com/beaglebone-vs-raspberry-pi/
http://lifehacker.com/how-to-pick-the-right-electronics-board-for-your-diy-pr-742869540
https://www.linkedin.com/pulse/iot-devices-arduino-vs-raspberry-pi-beaglebone-which-kithion
http://www.thewindowsclub.com/single-board-computers
http://au.element14.com/element14/bbone-black-4g/beaglebone-black-rev-c-cortex-a8/dp/2422228

Hexo+ batteries:
https://hobbyco.com.au/
Queen Victoria Building, 455 George St, Sydney NSW 2000
https://www.venompower.com/products/axial-scx10-deadbolt-35c-11-1v-5000mah-lipo-battery-by-venom-1
http://www.hobbyking.com/hobbyking/store/__9625__EC3_Plugs_10pairs_set_.html?gclid=Cj0KEQjwnKzABRDy2pb7nPSazdsBEiQAI4lZQJY2-h1Ut-T_45c8gPkjwuF8Txrea4JZX2-oaBNMgbkaAtri8P8HAQ
https://www.youtube.com/watch?v=u8aF3-4uBkQ
https://www.digikey.com/product-detail/en/6450543-5/A114251-ND

Sending an SMS:
https://www.cooking-hacks.com/documentation/tutorials/3g-gps-shield-arduino-raspberry-pi-tutorial/
http://whirlpool.net.au/wiki/mobile_phone_frequencies
http://www.simcom.eu/index.php?m=termekek&prime=1&sub=38&id=0000000159
https://www.cooking-hacks.com/forum/viewtopic.php?f=20&t=7470
http://www.linksprite.com/wiki/index.php5?title=3G_%2B_GPS_Shield_for_Arduino#Sending_and_Receiving_SMS

http://www.linksprite.com/wiki/index.php5?title=3G_%2B_GPS_Shield_for_Arduino#SMS
https://blog.arduino.cc/2012/05/29/arduino-gets-3g-connectivity/
https://www.cooking-hacks.com/3g-gprs-shield-for-arduino-3g-gps
https://littlebirdelectronics.com.au/products/itead-3g-shield
http://www.simcom.eu/index.php?m=termekek&prime=1&sub=38&id=0000000159
https://www.itead.cc/wiki/ITEAD_3G_Shield
ftp://imall.iteadstudio.com/Shield/IM121026002/DC_IM121026002_3G_Shield.zip
http://www.simcom.eu/index.php?m=termekek&prime=1&sub=38&id=0000000159
http://forum.arduino.cc/index.php?topic=140526.0
https://littlebirdelectronics.com.au/products/itead-3g-shield
https://www.itead.cc/itead-3g-shield.html
ftp://imall.iteadstudio.com/Shield/IM121026002/DS_IM121026002_3G_Shield.pdf
ftp://imall.iteadstudio.com/Shield/IM121026002/SCH_IM121026002_3G_Shield.pdf
https://www.diafaan.com/sms-tutorials/gsm-modem-tutorial/at-cmgs-text-mode/
http://www.activexperts.com/sms-component/at/commands/?at=E0
http://www.edaboard.com/thread250174.html
https://en.wikibooks.org/wiki/Serial_Programming/Modems_and_AT_Commands
https://www.sparkfun.com/datasheets/Cellular%20Modules/AT_Commands_Reference_Guide_r0.pdf
http://electronics.stackexchange.com/questions/26238/why-do-people-use-at-commands-in-serial-communication
http://www.forensicswiki.org/wiki/AT_Commands

GUI development:

playing mplayer, Qt QProcess:
http://www.mplayerhq.hu/MPlayer/DOCS/man/en/mplayer.1.txt
http://www.qtcentre.org/threads/13043-Playing-mplayer-inside-QWidget
http://www.bogotobogo.com/Qt/Qt5_QProcess_QFileDialog_QTextEdit_FFmpeg.php
http://www.qtforum.org/article/27971/run-a-shell-script-from-qt.html
http://www.qtcentre.org/threads/44398-How-execute-a-script-in-a-Qt-aplication
http://stackoverflow.com/questions/23322739/how-to-execute-complex-linux-commands-in-qt
https://forum.qt.io/topic/25976/qprocess-not-work-with-start-but-work-with-startdetached
http://stackoverflow.com/questions/23263805/what-is-the-difference-between-qprocessstart-and-qprocessstartdetached
http://doc.qt.io/qt-5/qprocess.html#readAllStandardOutput
http://www.qtforum.org/article/25844/using-mplayer-in-qt4-4.html

embedding mplayer into Qt via qmpwidget
http://qmpwidget.sourceforge.net/
http://stackoverflow.com/questions/24931792/qmp-widget-fails-to-load-video-in-qt5-3

embedding mplayer into Qt via QWebview widget class
http://www.bogotobogo.com/Qt/Qt5_QWebView_FFmpeg_Converter_Media_Player.php

Qt QProcess debug output:
http://stackoverflow.com/questions/10098980/real-time-display-of-qprocess-output-in-a-textbrowser
http://stackoverflow.com/questions/15614927/how-do-i-read-from-qprocess
http://www.qtcentre.org/archive/index.php/t-28406.html

startStreaming:
https://www.youtube.com/watch?annotation_id=annotation_2812730463&feature=iv&src_vid=fsweAm8NMeg&v=LqPP3FsAdhM
https://www.youtube.com/watch?v=WqEzyWMFQtg
https://www.youtube.com/watch?v=fsweAm8NMeg

other methods for streaming video in qt:
https://www.youtube.com/watch?v=ZoFm_Mznq1M

run a script from qt:
http://www.qtforum.org/article/27971/run-a-shell-script-from-qt.html
http://www.qtcentre.org/threads/44398-How-execute-a-script-in-a-Qt-aplication
http://stackoverflow.com/questions/23322739/how-to-execute-complex-linux-commands-in-qt

QR code:
http://www.linux-magazine.com/Online/Features/Generating-QR-Codes-in-Linux