

Analysis

Project Background

In the last few months, I have been getting into an advanced computing and electronic engineering hobby: high altitude ballooning. This involves sending a balloon with a payload attached containing a microcontroller (in my case, a Raspberry Pi Zero), a low power UHF radio, a GPS module and a camera. The Pi is programmed to take pictures at regular intervals and transmit both the images and the GPS data (telemetry) down in order to aid tracking and recovery of the payload. The software I developed and used for my first 3 flights can be seen here: <https://github.com/Abrasam/SKIPI-Launch-1> and here: <https://github.com/Abrasam/SKIPI2>. Below is a data-flow diagram for the current system.

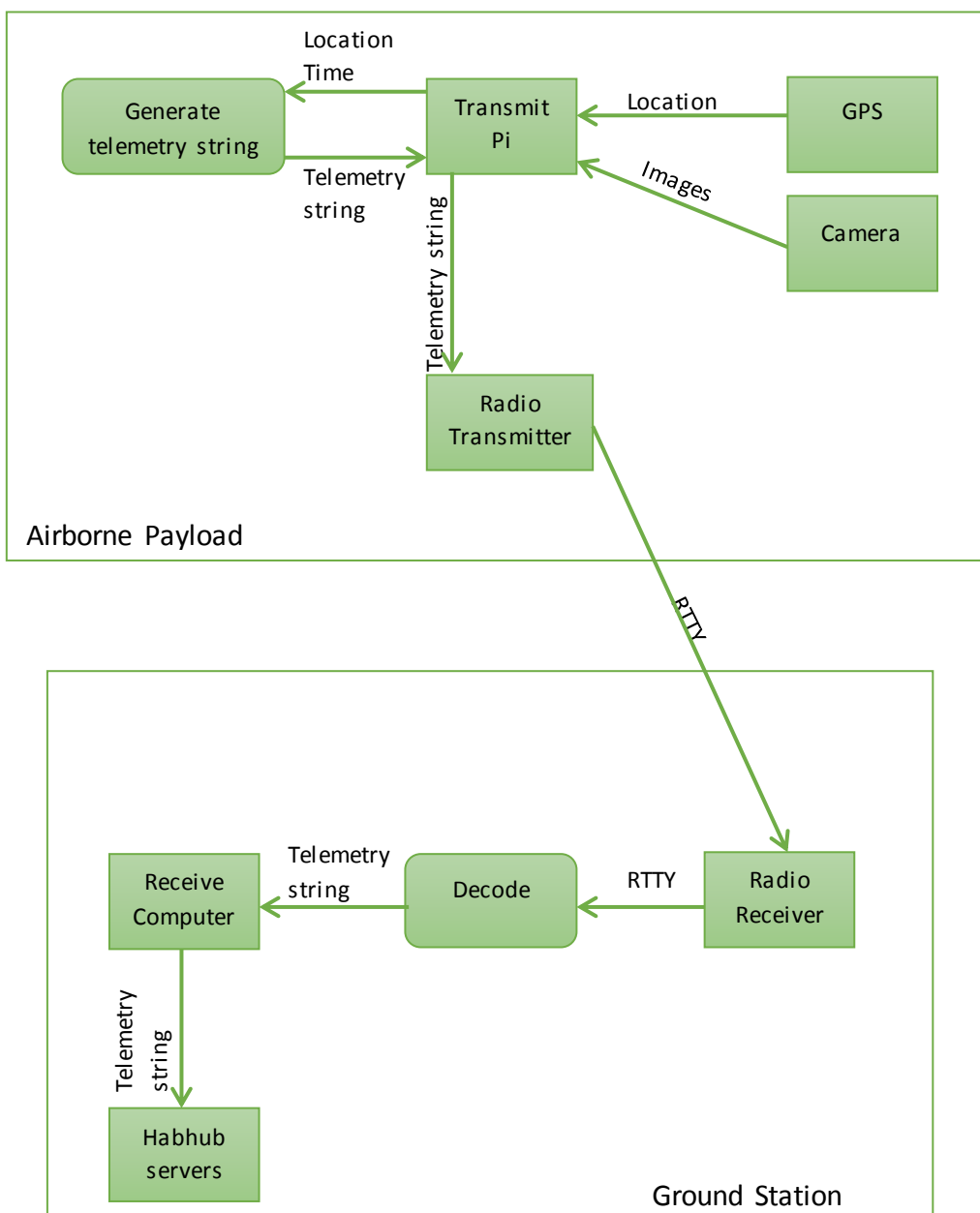


Figure 1 - DFD diagram for the current system, note the lone one-way RTTY link between the airborne payload and the ground-based station; there is no capacity for uplink.

Project Outline

As described previously, there is only one-way communication occurring between the payload and the receiver and if something goes wrong, then there is very little that can be done about it, furthermore, it means that the information which comes down from the payload is very specific and limited and all diagnostic data must be predefined when creating the payload's telemetry format. For my project, I want to create a prototype piece of software which allows two-way communication with and control of an airborne high altitude balloon payload for use by UKHAS (UK High Altitude Society) members, myself included. I have used two methods of communicating with the receiver on the ground, they are described in the research section. My clients for this project are members of the UKHAS who fly High Altitude Balloons (HABs) as a hobby. We have several pieces of software which are used by members, and this software is intended to be an addition to our suite of HAB utilities. I intend to use a Raspberry Pi as the microcontroller for my payload, I could use an Arduino, however, the Raspberry Pi camera module makes the Pi an attractive piece of hardware for HAB, as well, it gives me more freedom of programming language.

Client

As noted above, my intended clients are members of the UKHAS who will be the principle users of this software. One member of the UKHAS, [\[Name\]](#), has agreed to answer questions via email and provide technical advice with regards to the hardware implementation.

This is an interesting project. As far as I am aware, previous amateur balloon flights with 2-way communications have used a single transceiver at each end of the link, with some arrangement to ensure that only one of the two transmits at any given moment, whilst the other side of the link listens.

Some factors and possible issues that you should consider and find solutions for are:

1. Once launched, balloons have a large listening footprint (area on the ground that they can potentially receive radio signals from) and this of course increases with altitude. So a transceiver attached to a balloon will hear a lot of random signals on the frequency that it is tuned to. This reduces the link budget from ground to balloon, thus limiting the range at which the balloon can receive messages. You should thus take care to choose a frequency band (see IR2030) that provides the best link budget, taking into account the noise level whilst airborne (which increases linearly with bandwidth), and the maximum transmitted power from the ground. Having done so, consider the best set of LoRa parameters that will give you the best range.
2. The downlink is less problematic provided there is no local (to the receiver) interference on the downlink frequency.
3. Leading on from (1), consider the data throughput of different LoRa modes and bandwidths, both for the uplink and downlink, and the response time (ping time) for those modes/bandwidths with different packet sizes.
4. Consider if you want to use fixed-sized packets with no header (giving a higher potential data rate) or variable sized packets (giving lower ping times). The best options will depend on the application. For example, a simple uplink "please cutdown now" does not need a rapid response, but a remote telnet-style session would.
5. Be careful that the 1st harmonic of the 434 frequency is not within the 868 listener's bandwidth.
6. Even with carefully chosen frequencies to negate (5), be aware that filtering on the LoRa transceivers, or any other transceiver, is not perfect and there will be local crosstalk from the each transmitter to its adjacent receiver. This is something that you should measure, by running a test with the 434 and 868 aerials close together (see (7) below), with one listening and the other transmitting. Plot the noise level at the receiver with the transmitter on and off, and for different Tx frequencies. Do this test for 434 Tx and 868 Rx, and vice versa.
7. Consider placement of aerials on the payload, as your scheme needs 2 of them.

I wish you good luck with your project and please do ask again if you have more questions.

Best Regards,

Figure 2 - Email from [\[Name\]](#), member of the UKHAS, amateur radio operator and embedded systems programmer. IR2030 is the document published by Ofcom noting what radio frequency bands are available for license-free use.

[\[Name\]](#) comments on the fact that the payload will have a large listening footprint and as such will be receiving lots of other transmissions, meaning that our transmissions may require higher power in order to be received over the noise. I will discuss potential frequencies to use which are license-exempt in the next section. [\[Name\]](#) also suggests that I use a LoRa transceiver (see research for more information) and that I choose my parameters wisely to maximise range. [\[Name\]](#) points out that 434MHz and 868MHz are harmonically related (these

are the two frequencies frequently used by HAB enthusiasts and are the two frequencies which LoRa modules operate on). I will discuss points more in research.

Research

Radio Communications

The first method of communication I have used is RTTY or Radio Teletype which is an old protocol initially developed for the teleprinter, which works by simply shifting frequency up and down to correspond to binary 1 and binary 0, this is done by applying a small voltage to one of the pins on the radio. These voltage changes must be timed accurately, but as the Pi doesn't have a real time OS, the best way to achieve this was using the Pi's RS-232 (a standard for asynchronous serial communications) serial interface (and thus connecting the Pi's Tx (see fig. 3) pin to the radio's pin). This runs at 75-100 baud robustly and could be pushed to 300 but is then significantly susceptible to interference, so this means a maximum usable downlink bitrate of 300bps, as I have only one bit encoded per state change. This rate is unfeasible for two way communications and additionally, automatic detection and decoding of RTTY is difficult to achieve reliably, even when receiving transmissions during a normal flight I typically receive errors on at least 20% of packets, this is significantly too high to be useful for 2-way communications, particularly of telnet style communication is desired.

Raspberry Pi GPIO Header A+, B+, Zero, Pi2				
Pin#	NAME		NAME	Pin#
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I ² C)		DC Power 5v	04
05	GPIO03 (SCL1 , I ² C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I ² C ID EEPROM)		(I ² C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

Rev. 1.5
09/02/2016 www.element14.com/RaspberryPi

Figure 3 - Image showing the pin layout of a Raspberry Pi Zero, the TXD0 pin is used to communicate with the NTX2B (RTTY) radio as well as the GPS, the RXD0 is also used to communicate with the GPS. The SPI_MOSI, SPI_MISO, SPI_CLK and SPI_CE0_N pins are used for the LoRa radios, described in the paragraph below.

The second method of communication is LoRa which is a long range, low power, high data rate radio solution also running at a license exempt frequency. The LoRa modules are low cost and are controlled with an SPI (Serial Peripheral Interface, a synchronous serial communication protocol) interface and provide state change notifications via DIO (Digital Inout/Output, a simple protocol whereby a line can have either a HIGH or a LOW signal to denote binary flags). Their modulation and demodulation is handled internally as the modulation scheme is patented. The modules are capable of achieving an equivalent of 17,000 baud RTTY (see figure 4), making them ideal for long range 2-way communications. Their range is somewhere in the region of 60-100km, which is perfectly adequate for high altitude ballooning, this is somewhat lower than the RTTY which can reach 600km and more with perfect conditions, however, we do not need that extra range.

Bandwidth (kHz)	Spreading Factor	Coding rate	Nominal Rb (bps)
7.8	12	4/5	18
10.4	12	4/5	24
15.6	12	4/5	37
20.8	12	4/5	49
31.2	12	4/5	73
41.7	12	4/5	98
62.5	12	4/5	146
125	12	4/5	293
250	12	4/5	586
500	12	4/5	1172

Figure 4 - Shows nominal bitrate vs bandwidth. Bitrate can be further optimised with modification to spreading factor and error coding rate. Taken from the LoRa module datasheet.

Thus far I have flown three flights, all of which have used the RTTY and two of which have used the LoRa radios, in order to use the RTTY I used a cheap Radiometrix NTX2B radio (see: <http://www.radiometrix.com/files/additional/NTX2B.pdf>) which shifts frequency as a result of a voltage applied to one of its pins, however, the Pi outputs 3.3v which would result in a frequency shift of about 7kHz, this is far too large as it is outside the range of typical SDR (software defined radio, a USB radio receiver) receivers, so a potential divider was needed to lower the voltage to about 0.2v-0.3v to acquire a shift of around 400-500Hz, a graph of frequency shift against voltage applied to TXD pin is shown in fig. 4. Furthermore, the LoRa modules were very successful, I used a Python library designed for a similar module which worked well, however, in this project, I will want to develop my own wrapper and API for the LoRa radios which will be more robust than the library I used before, and will handle the SPI and DIO interfaces itself, while providing me with a self-documenting API for use throughout the rest of the programming and possibly publication as a stand-alone LoRa API.

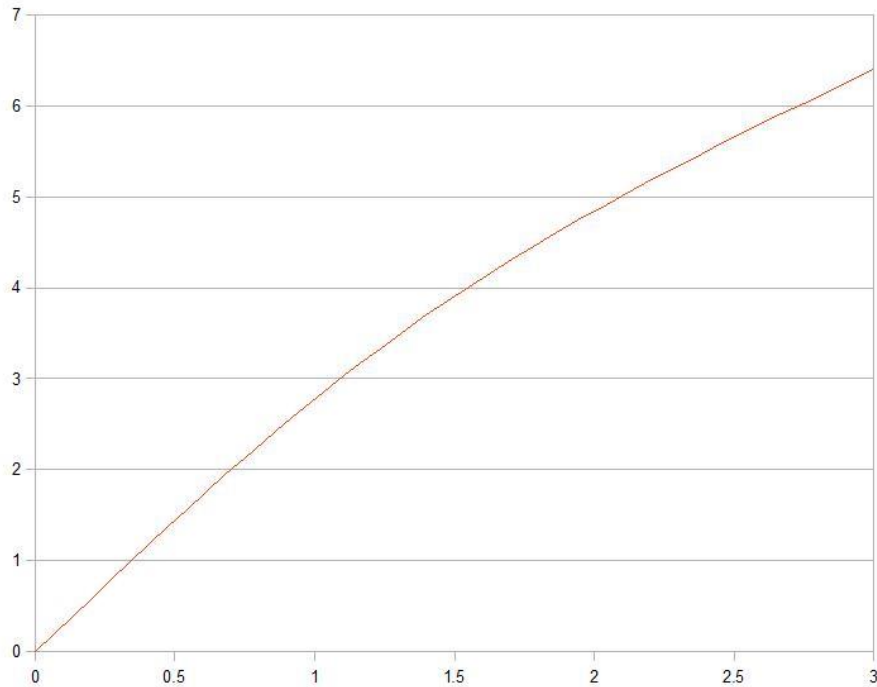


Figure 5 - The above graph shows the frequency shift (kHz, y-axis) vs the voltage applied to the radio's Tx pin (V, x-axis). Sourced from the UKHAS website.

Due to Ofcom regulations, I am restricted to specific power outputs and frequency bands as I do not hold an amateur or commercial radio license. Therefore, I must use license-exempt bands and adhere by any specific rules in those bands. There is a 434.04-434.79MHz band which I have used for RTTY, this has a power output limit of 10mW E.R.P. (effective radiated power – this takes into account the antenna gain (amplification by the antenna)); a band at 869.70-870.00MHz which has a power output limit of 5mW E.R.P. which I will use for the 868MHz LoRa downlink from the payload. These bands are useful due to their 100% duty cycle, meaning I can transmit continuously, and their lack of requirement for techniques to mitigate interference such as “Listen Before Talk” which could inhibit some of my communications. However, there is another band, 869.40-869.65MHz which allows 500mW transmissions with a duty cycle limit of 10%, this could be used for the uplink as I am only likely to be transmitting to the payload for a small amount of time compared to the time during which the payload will be transmitting to me and thus I could easily manage with 10% duty cycle, this would provide a much greater signal strength for transmissions to the airborne payload, it would be much less susceptible to interference. The table in figure 5 shows the available bands that are suitable for this project.

Frequency (MHz)	Power Limits (mW)	Other Requirements
869.40-869.65	500	Duty cycle limit of 10%.
869.70-870.00	5	None.
434.04-434.79	10	Channel spacing ≤ 25kHz

Figure 6 - Summary of frequency bands which could be useful for my project. Sourced from the Ofcom IR2030 table of license-exempt frequency bands in the radio spectrum, available at: https://www.ofcom.org.uk/__data/assets/pdf_file/0028/84970/ir_2030-june2014.pdf

The LoRa module requires a detailed understanding of its interface. The datasheet (http://www.hoperf.com/upload/rf/RFM95_96_97_98W.pdf) describes the SPI and DIO (Digital Input/Output) interface it uses. SPI is a synchronous serial communication interface used primarily in embedded systems; it has a master-slave architecture with a single master and can operate in full-duplex mode; it supports multiple slaves when separate slave-selection lines are used. The SPI is used to modify the registers on the radio, in order to configure the radio (e.g. frequency, spreading factor, operation mode etc.) or, indeed, to write the packet that is to be transmitted to the appropriate register or to read a received packet from the appropriate register. Modulation, demodulation, receiving and transmission of packets is handled internally as the methods used are patented. The DIO is used to notify the interfacing device, in my case a Raspberry Pi, when specific events occur (see fig. 6 for functionality of each pin), in my case I will be using the DIO0 and DIO5 pin, they function as follows: the DIO0 pin can be configured to change state to HIGH (1) when a transmission has finished sending or receiving (TxDone or RxDone), while the DIO5 pin can be configured to change state to HIGH (1) when the radio has changed operation mode (ModeReady, i.e. changing from transmit to standby mode). My software will need to use both SPI and DIO so I propose that I use a library such as WiringPi or Pi4J to do this. WiringPi has implementations in many languages and Pi4J is a Java based library, as the name 'Pi 4 Java' suggests; however, there are many other options such as spidev in Python which is an excellent library which I have experience developing with.

Operating Mode	DIOx Mapping	DIO5	DIO4	DIO3	DIO2	DIO1	DIO0
ALL	00	ModeReady	CadDetected	CadDone	FhssChangeChannel	RxTimeout	RxDone
	01	ClkOut	PllLock	ValidHeader	FhssChangeChannel	FhssChangeChannel	TxDone
	10	ClkOut	PllLock	PayloadCrcError	FhssChangeChannel	CadDetected	CadDone
	11	-	-	-	-	-	-

Figure 7 - Table showing the functions of the individual DIO pins. Taken from the datasheet.

The LoRa SPI interface works in the following way: each register on the LoRa module is assigned a unique address of up to 7 bits, to write to or read from a register you must send a sequence of bytes via SPI to the radio, the least significant bits of the first byte will be the register's address, the most significant bit is 1 when writing and 0 when reading. Then, if writing, you send the bytes you wish to write in the order you wish then to be written (if you send more bytes than the register can hold it will ignore the excess); if reading you must send the n 0x00 bytes where n is the number of bytes you wish to read from the specified register, again, if you specify more bytes than the register contains it'll just stop once it's read the whole register. This gives a sufficient knowledge of the SPI interface for this project, for more information see the aforementioned datasheet.

GPS

I will need a GPS module on my payload, however, due to the COCOM limits (see <https://en.wikipedia.org/wiki/CoCom>), which prevent a GPS from functioning if it is above 18,000m altitude or travelling faster than 1,000 knots in order to prevent GPS technology being used to guide inter-continental ballistic missiles (this limitation is an artefact of the Cold War), I have to purchase a specialist GPS module which instead requires both high

altitude and high speed to shut down rather than just one of the two. There are several options and all have a serial interface running at 96,000 baud by default, this will require usage of the Pi's RS232 serial connection to read continuously from the GPS. Alternatively, I could use an I²C interface which many of the GPS modules provide; however, the Ublox GPS modules (which is the most popular type with HAB enthusiasts) perform clock stretching at arbitrary times and the Pi I²C driver simply cannot handle this. I should note that the GPS also outputs several different types of location strings in a looping sequence, all the output types can be seen on <http://www.gpsinformation.org/dale/nmea.htm> but we are only interested in GGA strings, which give a 3D position (i.e. including altitude) and the number of satellites the GPS is currently in contact with. These give latitude and longitude in the form ddmm.mmmm where dd is the number of degrees and mm.mmmm is the number of arc minutes as a decimal. So, to convert to degrees correctly, which most mapping systems use you must use the following equation:

$$\text{coordinate degrees} = dd + \frac{mm.mmmm}{60}$$

Additionally, the GPS doesn't give the correct negative values, it instead gives another field saying whether the position given is North/South of the equator for latitude or West/East of the Greenwich Meridian for longitude so these fields will need to be checked and the correct negative sign will need to be applied to the latitude and longitude values.

I should also note that the GPS doesn't by default work at high altitudes, by default it adheres to the standard CoCom limits, a sequence of bytes must be sent to it in order to switch to 'Airborne Mode', the required bytes are as follows:

```
[0xB5, 0x62, 0x06, 0x24, 0x24, 0x00, 0xFF, 0xFF, 0x06, 0x03,
0x00, 0x00, 0x00, 0x00, 0x10, 0x27, 0x00, 0x00, 0x05, 0x00,
0xFA, 0x00, 0xFA, 0x00, 0x64, 0x00, 0x2C, 0x01, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x16, 0xDC]
```

Server and Slave Network

As I have stated, I have flown flights using both LoRa and RTTY transmission methods and have gained an understanding of their function and implementation through building and programming the payloads for these flights. I've also seen that when chasing a payload, it is quite easy to lose contact with it temporarily due to, for example, an inconvenient road route or a building breaking line of sight, and at these times I've previously relied on other enthusiasts (members of the UK High Altitude Society) to receive the transmissions using their high gain stationary antennas based around the country. In order for my 2-way communication system to function consistently, through signal dropouts with the payload, I suggest that I need to harness the many willing enthusiasts across the country who would be happy to assist in tracking. So, I propose the development of a slave tracking software that acts to rebroadcast transmissions that do not successfully reach the payload when transmitted by the main controller and to forward any transmissions received by the payload to a central server (the function of this server is described below). Figure 7 below shows a map of UKHAS listening stations recently active, as you can see there are many

receivers across the country, massively increasing listening and transmitting capacity if they can be harnessed! I will suggest this to members of the UKHAS.

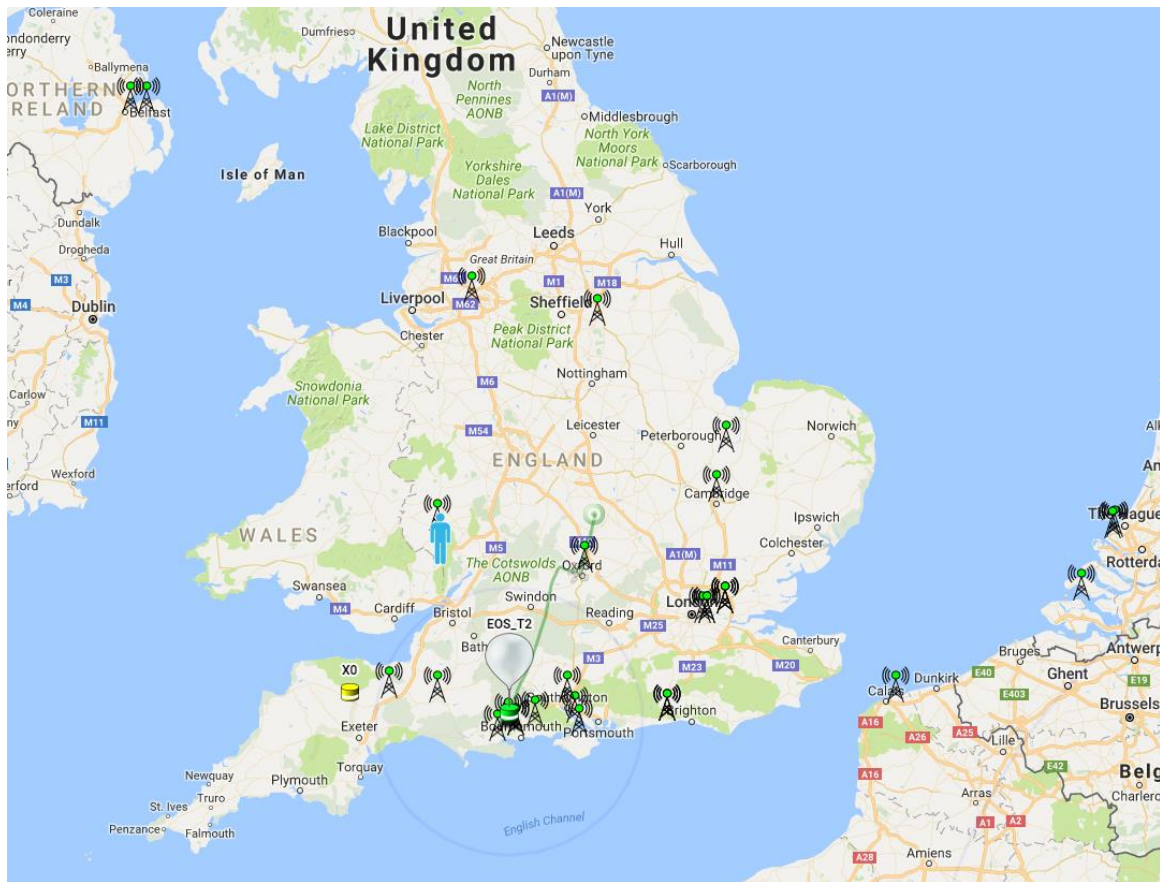


Figure 8 - Map of UKHAS receiving nodes active recently. Screenshot from the UKHAS tracker (<https://tracker.habhub.org/>).

Potential Radio Solutions

Parallel

My software could work my transmitting packets to the payload on one LoRa radio, and from the payload on another LoRa radio running at a different frequency. Having two radios operating simultaneously at different frequencies would allow us to maintain 100% duty cycle on both transmit and receive operations in order to maximize efficiency, rather than having to wait for a cycle to complete. Note that although in my implementation I would most likely be using 434MHz for uplink and 868MHz for downlink, the frequencies should be configurable by the end user in a configuration file, so if they wanted to use two 868MHz radios with lower bandwidths or one 868MHz and one 434MHz then they should be able to by modifying a configuration file. However, there are problems with having two radios operating simultaneously, particularly as the two frequencies available to the LoRa radios are harmonically related which would result in significant interference, potentially preventing any communication from functioning correctly.

Cycled

Alternatively, I could use only one transceiver and switch between receive and transmit regularly, this would reduce responsiveness for the 2-way communications, but make it

more robust and less likely to be affected by interference. Note that my implementation should be fully configurable allowing users to use different frequencies and different cycles of uplink and downlink. This would mean that I would need to configure a cycle where the airborne payload spends some time transmitting, then transmits a packet informing the ground-based device that it is now accepting packets, waits to receive packets, and then if no packets are received for some defined period of time, begin transmitting for a while again. The ground based receiver would need to be configured to receive packets and then upon receiving a packet that states that the transmitter is going into receive mode, transmit any packets that are queued for transmission.

Further Client Discussions

Radio Solutions

As mentioned the UK High Altitude Society (UKHAS) is a group of enthusiasts who fly high altitude balloons, of which I am a member. We have a suite of software called habitat which allows data received by numerous receivers to be forwarded to a central server and displayed on a map. I would like for location data received from my payload to be relayed to habhub. My project will be targeted at myself and other members of the UKHAS. Further, in order for the slave transceiver network to function correctly, I would need to develop my own central server software which coordinates the slave node network and logs flight data for my own records, this would be developed with the overall aim in mind for it to be eventually integrated into the habhub habitat software suite. The central server could also function to ensure that the main controller is forwarded any packets from the payload which it does not receive itself which are, however, received by the army of slaves.

I have been discussing this project with members of the UKHAS and have made many of the decisions noted in this section and below in the specification with their interests in mind, as they are my principle clients. I had some discussions with members of the UKHAS on the #highaltitude IRC channel on Freenode, I discussed the advantages and disadvantages of having two LoRa radios in parallel and of using just one with a cycle of transmit and receive. I have reached the conclusion that it would be more suitable to use just one radio with a cycle of transmit and receive, with both queuing packets for transmission while unable to transmit (as they're in receive mode). Several members of the UKHAS were concerned about the interference due to having the two radios simultaneously transmitting and receiving while adjacent to each other and most were unconcerned about the slight delay that would be caused by having to queue packets for transmitting while in receive mode. See fig. 9 for transcript of IRC.

```

[08:27] jakeio (~jakeio_@149.254.183.73) joined #highaltitude.
[08:29] <jakeio> Hello, my A-Level Computing project is to create a software suit that allows 2-way communications with airborne payloads. It will use two LoRa radios in parallel to maintain 100% duty cycle transmit/receive. Can I ask what people think of this, and if they have any suggestions for features. This is all for my 'research and analysis'.
[08:33] fab4space-f4hhv (~Fabrice@109.237.242.98) left irc: Ping timeout: 248 seconds
[08:34] fab4space-f4hhv (~Fabrice@109.237.242.98) joined #highaltitude.
[08:34] <Geoff-G8DHE-M> Will the receiver front end work happily when your transmitting (in the same band ?) next too it ?
[08:36] <jakeio> I wasn't necessarily going to use the same band, as many people have one 434MHz LoRa and one 868MHz LoRa on one Pi I thought I might use one of those for transmission and another for receiving. Should be less interference that way shouldn't there?
[08:36] <mfa298> some analysis of ir2030 might be useful to demonstrate you've looked at what's allowed (or not)
[08:37] <Geoff-G8DHE-M> Possibly, the front ends have little to no filtering so can still be swamped by other large signals.
[08:40] <jakeio> mfa298, I've commented on the main bands suitable for LoRa, aren't these the 434.04-434.79 and the two 868ish ones, one with the 500mw limit but duty cycle limit of 10% or "techniques to mitigate interference" and the other being the 5mw one we usually use? Geoff-G8DHE-M would you suggest it would be better to use just one transceiver to prevent this? Slightly reducing responsiveness in favour of robustness?
[08:40] <SpacenearUS> New position from 03HIRFW-6 after 0313 hours silence - 12https://tracker.habhub.org/#!qm=all&q=HIRFW-6
[08:41] <Geoff-G8DHE-M> It certainly makes for a simpler and lightweight system. Unless you really need the bandwidth/time then its a lot simpler.
[08:42] <gonzo.> if you are planning those for flight, double check the power limits for airborne kit
[08:43] <Geoff-G8DHE-M> remember 434 and 868 are harmonically related so there is always going to be greater residual signals from one band to the other.
[08:46] <mfa298> if you don't need 100% duty cycle on uplink then the higher power 868 option could be good for ground to balloon.
[08:47] <jakeio> Thanks, I'll probably do a table comparing both methods and then decide on the one radio one! And mfa298 thanks, I shall include this in my write up!
[08:47] <Geoff-G8DHE-M> You might also want to read up and comment on intermodulation and mixing products of radio signals as its these that normally limit such systems.
[08:48] <mfa298> A Level computing sounds a lot more interesting than when I did it.

```

Figure 9 - Transcript of IRC chat with members of the UKHAS.

Two Way Protocol

I discussed my concept for my protocol specification with the UKHAS on IRC, the log is in figure 10, we discussed the network of slave receivers and it was suggested that the system be coordinated so that only one node is transmitting at any given time, this is what I had suggested above in my research. When discussing the protocol, I suggested a system where the payload transmits to say it is entering into receive mode and then begins to listen, then the ground station begins transmitting and when the ground station has not transmitted anything for a given time the payload returns to transmit mode and then processes the received packets; members of the UKHAS suggested that I expand upon this with something similar to the sliding window protocol used in TCP, this is where each packet has a consecutive number and the receiver uses the numbers to put the packets in the correct order, detect duplicates and detect missing packets; the sliding window protocol puts a limit on the number of packets that can be transmitted in a given time by limiting the number of packets that are sent before waiting for an acknowledgement (source: https://en.wikipedia.org/wiki/Sliding_window_protocol). Although this will not exactly be used in my implementation, it would be suitable to limit to a set number of transmissions per transmit cycle to prevent any cycle continuing for too long. I do not think that sending NACK (negative-acknowledgement) packets or ACK (acknowledgement) packets will be appropriate for my implementation, ACK packets would be inappropriate because I am running on an unreliable and slow medium and this would result in having to send an extra packet for each packet, NACKs would still be inappropriate because the number of packets transmitted in a cycle will not necessarily be fixed.

Members of the UKHAS expressed concern that others could attempt to enact remote control operations on the payload, a proposed solution to this is described later (see 'Authentication').

```

[15:51:28] jakeio My A-Level computing project is to develop a software suite for 2-way communications with airborne HAB payload.
[15:52:02] jakeio I've settled on a system with one LoRa radio on each end with a cycle of transmit and receive with packets being
queued for transmission while in receive mode.
[15:52:46] Ian_ Define A=Ground B=Air
[15:52:51] jakeio OK.
[15:53:35] *urchin_ has quit (Ping timeout: 252 seconds)
[15:53:43] Ian_ What protocol, GPS time slots?
[15:54:10] jakeio Would it be useful to have a system where multiple users can contribute to the receiving and transmitting of
packets to the payload, in a similar way to how we all contribute to tracking at present, however, with the added
ability of helping with transmitting.
[15:54:12] SpacenearUS New vehicle on the map: KF6RFX-3 - https://tracker.habhub.org/#!qm=All&q=KF6RFX-3
[15:54:12] SpacenearUS New vehicle on the map: KF6RFX-4 - https://tracker.habhub.org/#!qm=All&q=KF6RFX-4
[15:54:15] jakeio And protocol will be...
[15:55:08] Ian_ I would have a single A node. Any messages by internet to A for transmission or you risk collisions
[15:55:47] jakeio B sends a packet that says "I'm listening" and then switches to receive mode, A then transmits an acknowledgement
packet and begins transmitting it's queued data, when A has finished and no packets are received for a short
while by B then B returns to transmit mode and A receives.
[15:57:25] jakeio Ah, I've come up with a way to coordinate the multiple transceivers, in that the other transceivers will only be
used instead of the main one if the main one fails to communicate with the payload for a given time.
[15:57:40] *pcall has quit (Ping timeout: 260 seconds)
[15:58:50] Ian_ If you have a network of potential ground station txrs then I would be inclined to coordinate them via internet,
with only one having the permission at any given time.
[15:59:36] Ian_ So communication is node to node LoRa but communications management is via internet
[16:00:17] Ian_ The airborne node merely needs to let you know that it has GPS lock to be in business of receiving and sending
acks
[16:00:30] jakeio Yes, if A doesn't get through to B for two "cycles" then A sends packets to the server to distribute to other
receivers.
[16:00:40] jakeio transceivers*
[16:00:49] Ian_ Sounds sound!
[16:01:21] jakeio The reason I opted for a non-timed option, i.e. B has to initiate receive mode, is so as to have flexibility in
how long those cycles last.
[16:01:33] jakeio This is in order to maximize duty cycle while airborne.
[16:01:40] Geoff-G8DHE-Lap Radio links are unreliable - using Ack's on unreliable links is bad, better to use packet numbering and then
NACK's when sequence is missing a packet.
[16:01:55] Ian_ That makes good sense
[16:02:25] jakeio OK, so each packet has an ID and if the sequence is interrupted then request retransmission?
[16:02:29] jakeio Or have I misunderstood.
[16:02:47] *Guest46323 is now known as richardeoin
[16:02:59] Geoff-G8DHE-Lap That's right, otherwise you can spend more time chasing missing Acks then sending useful data
[16:03:14] jakeio OK, that makes sense. Thanks Geoff-G8DHE-Lap.
[16:04:00] Ian_ No, that's right. Personally I would have fixed time slots up and down to keep it simple. You can always refine
that later otherwise you are mixing communications and communications management on an unreliable medium
[16:04:14] Geoff-G8DHE-Lap The TCP sliding window concept is a good idea for this
[16:05:17] Geoff-G8DHE-Lap https://en.wikipedia.org/wiki/Sliding_window_protocol
[16:06:38] jakeio OK, so the sliding window is used to prevent extremely large packet numbers?
[16:06:59] Geoff-G8DHE-Lap One aspect of it yes,
[16:07:25] Ian_ It saves things getting too far out of real time I guess
[16:07:56] Geoff-G8DHE-Lap all the time there is data being exchanged you keep going, only go back and resend if NACK, but you can't send
for ever as the other end may not be receiving anything
[16:08:37] Ian_ So telemetry on the down leg would be uncontrolled and optimistic
[16:12:23] jakeio Right, thanks for the info on the sliding window protocol.

```

Figure 10 - IRC log of discussions about the network of slave receivers and the 2-way protocol. Note that 'ACK' means acknowledgement packet and 'NACK' means negative-acknowledgement packet.

Character Set

When transmitting data via the LoRa radios, it needs to be encoded into bytes, so I will have to use a specific chosen character set. The chosen character set needs to be 8 bit because the SSDV program encodes data into packets of 256 bytes and I wish to transmit one packet per transmission. Additionally, the character set needs to be compatible with existing HAB software which use extended-ASCII. The character set I shall use will be ISO-8859-1 for it fulfils all the requirements and is included on most systems with most languages having built-in functions that can encode strings in it. This is effectively extended ASCII so is compatible with most existing HAB software.

DI-Fldigi Interface

Further discussions with members of the UKHAS led me to the conclusion that it would be a suitable extension to interface with the already commonly used (by UKHAS members) dl-fldigi software which is used to decode RTTY. This results in my software effectively being an all-round HAB toolkit, making it very useful to an enthusiast. I did, however, decide that this should only be an extension as most enthusiasts are happy to have both programs open simultaneously without an overall wrapper and this project is at its current stage a prototype.

Backup Tracker

Additionally, members of the UKHAS who I asked all wanted to have RTTY transmitted by the payload as well as an emergency backup tracking method due to it being so robust and reliable. Because of this, as the RTTY typically runs at 434MHz, I would thus need to use 868MHz for the LoRa. The IRC log for this is shown in figure 11. It might be sensible to disable the RTTY for the duration of the payload's receive cycle as although the RTTY is running at a different frequency than the LoRa the two frequencies are indeed harmonically related, resulting in possible interference. Having done some research and testing with an RTTY tracker running from the same payload, I have decided to instead run a separate payload separated by several metres from the 2-way payload in order to limit interference. For this I will use my well-tested tracker from my previous flights which I know to work well. Hence, the two-way payload software will not need to transmit RTTY as well as LoRa.

```
[16:04:03] jakeio Would you think it important to have a backup standard RTTY tracker setup also, in case the new 2-way LoRa system fails?
[16:04:30] gonzo_ a backup is always going to be a good idea if weight will allow
[16:06:12] gonzo_ if you are running something experimental, and you need to get it back, or need to know where it is for part of the experiment. Then a backup known reliable tracker is a good ide
[16:06:25] gonzo_ a
[16:09:31] mfa298 In terms of a backup transmitter I think today probably demonstrates why a backup is good!
```

Figure 11 - IRC log showing discussion of backup RTTY tracker. Note that the final comment by mfa298 is regarding a payload that was lost due to the failure of its single tracker.

Maximizing Link Budget

In the past, members of the UKHAS have managed only limited 2-way communications, they have successfully transmitted a packet to the payload in the event that image packets are missed requesting that they are retransmitted. These used high gain transmitting antennas and the software was written in Python. I spoke to [\[redacted\]](#), who wrote this software and he has confirmed that he has had reasonable success with it in the past using a relatively simple antenna as well as with a high gain directional antenna.

A screenshot of an email from [\[redacted\]](#) is shown in figure 2, following this and discussion on the IRC I have concluded that I shall use the 500mW license-exempt band (see fig. 6) to transmit up to my payload as this will provide a greater link budget and this will make the 2-way communications much stronger, however, the LoRa module is only 100mW at its maximum so this will be my maximum output, however, the 500mW band is still the only band in which I can use this power output on the transmitter (using the 500mW band was suggested by a user on the IRC, see fig. 9). Furthermore, [\[redacted\]](#) suggests in his email to optimise the LoRa parameters for best link budget, typically a spreading factor of 7 has been used with a 250kHz bandwidth for high data-rate long-range communications from HAB to payload, however, for payload to HAB I intend to use a lower bandwidth to maximize resistance to interference and increase link budget, a lower bandwidth means that the total power output is spread over a smaller section of the radio spectrum. This should help alleviate the issues created due to the large listening footprint of the airborne payload that [\[redacted\]](#) mentions in his email.

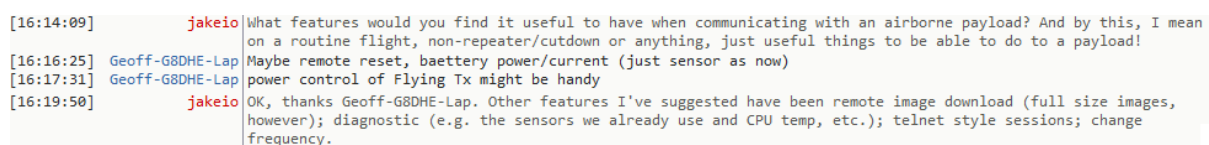
Authentication

I will be sending transmissions from the ground station that could, if used incorrectly, sabotage my flight, for example, sending the console command `sudo halt` would achieve

that. Because of this, I feel that my packets should be some way secured, encrypting would be one method of doing this, however, I am not sending sensitive data so it seems unnecessary. An alternative would be using a sort of 'salt' like those used in salted hashes when storing passwords in databases, by this I mean that when configuring the payload the user would need to set a 'salt' or key which will be included in the calculation of the checksum but not actually transmitted, that way, on the receiving end the software could append the same key to the received packet and calculate the checksum, if the checksums match then we know both that the data is correct and has come from an authorised source.

Conclusion

In conclusion, as discussed, my target users are members of the UKHAS who would find it useful to see basic diagnostic information about an airborne payload such as battery voltage and internal temperature in order to detect issues, without having to include that in their standard telemetry format; members also expressed the desire to have the ability to reboot the entire Pi, making the point that the software should be configurable to start when the Pi boots, this is all in order to fix potential runtime issues; it was also pointed out that the payload software should not continue to wait if no packets are received from the ground as this could result in the payload going silent, it should time out, this is to prevent lost flights; several members also expressed the need to have multiple receivers and transmitters to maximize the range of the 2-way communications as discussed above, however, as this project is a prototype I think this should be considered as a future extension to the main project, simply achieving two way communications with the payload from a single ground station will be sufficient proof of concept. It was also noted that if there were no 2-way communications packets to send then the payload should just transmit standard telemetry strings following the standard UKHAS format (see figure 13 or for more information see: <https://ukhas.org.uk/communication:protocol>), 2-way packets should be in some way distinct from standard telemetry strings, perhaps a specific prefix should be used, rather than the standard \$\$ used by the UKHAS strings.



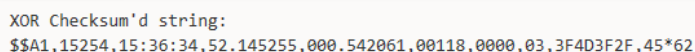
[16:14:09] jakeio What features would you find it useful to have when communicating with an airborne payload? And by this, I mean on a routine flight, non-repeater/cutdown or anything, just useful things to be able to do to a payload!

[16:16:25] Geoff-G8DHE-Lap Maybe remote reset, baattery power/current (just sensor as now)

[16:17:31] Geoff-G8DHE-Lap power control of Flying Tx might be handy

[16:19:50] jakeio OK, thanks Geoff-G8DHE-Lap. Other features I've suggested have been remote image download (full size images, however); diagnostic (e.g. the sensors we already use and CPU temp, etc.); telnet style sessions; change frequency.

Figure 12 - Showing a small snippet of my discussion with members of the UKHAS about potential features for the system.



XOR Checksum'd string:
 \$\$A1,15254,15:36:34,52.145255,000.542061,00118,0000,03,3F4D3F2F,45*62

Figure 13 - Image showing UKHAS protocol standards.

Specification

Ground Station Controller

1. The transceiver controller should:
 - a. Provide the user with a clear front-end.

- i. This need not be usable by a naïve user, as this software is aimed at experienced radio operators and high altitude balloon enthusiasts; this hobby requires a certain amount of technical knowledge on the subject matter.
 - ii. The UI should function on a touchscreen, as the Pi used will be using a touchscreen for the interface.
 - iii. The UI should provide the following features: current telemetry display; current image display; transmission log; remote console; control interface; configuration tab.
- b. Provide a configuration file which allows the user to set the callsign, transmission frequency and bandwidth, the receiving frequency and bandwidth, the error coding rate, the spreading factor, the transmission power and whether the payload is using explicit packet headers. The user should also be able to set the key for 2-way packet authentication.
- c. Allow normal LoRa receiving using an 868MHz module or a 434MHz module.
 - i. This is done by awaiting the DIO0 pin to go high, then reading the contents of the LoRa FIFO.
 - ii. LoRa modules are interfaced with SPI so this will require an SPI wrapper to be developed which handles all LoRa functions needed for the project.
- d. Decode SSDV image packets and display on the graphical interface, also forward SSDV image packets to the habhub servers.
 - i. SSDV can be decoded using a freely available library developed by a member of the high altitude ballooning community called fsphil.
 - ii. SSDV is encoded by the library into 256 byte packets, these can be decoded individually so if a packet is lost most of the image can still be seen.
- e. Parse telemetry, ignoring those with failed checksums or incomplete data and showing the relevant data on the telemetry display.
- f. Log all received packets to a file with a timestamp.
- g. Switch to transmit mode after receiving a packet from the payload stating that it is entering listening mode and send any 2-way communication packets that are queued.
 - i. Packets will be queued by the user when they request that an action be completed by the payload.
- h. Allow the user to queue packets for transmission by either clicking one of the command buttons in the control menu or by sending a remote console command.
- i. Allow two-way communication with the airborne payload as described below.
 - i. Two-way communication should be initialised by the sending of a packet by the payload stating that it has begun waiting for transmissions from the ground.

- ii. The ground station should then switch to transmit mode and move to the transmit frequency and bandwidth.
 - iii. The ground station should then send a fixed number of its queued packets.
 - iv. The payload should return to transmit mode after not receiving any packets for a set duration of time, or after receiving the correct number of packets.
 - v. Each packet should have a consecutive ID; the ID will reset to zero at the start of each transmit cycle.
 - vi. All packets involved in two-way communications should be prefixed with an identifier so that they are not mistaken for standard telemetry or SSDV.
 - vii. The user should be able to control basic airborne operation such as toggling image transmission.
 - viii. The user should have remote shell/telnet style access and be able to reboot the payload in an emergency debugging attempt.
 - ix. The user should be able to request diagnostics of the payload e.g. number of pictures stored or output from sensors, there should also be a telemetry log, as telemetry data will be transmitted regularly as part of 2-way communications to maintain accurate location.
 - x. The ground station should return to the receiving frequency and bandwidth once the transmit cycle is over.
- j. If the user has configured their payload on the web portal, the ground station should be able to, given the payload callsign, download the payload configuration and write the configuration file for the user. The user will, of course, have to enter the key themselves.
 - k. Allow the user to toggle uploading data to the server (effectively allow 'offline mode' for testing).

Server

- 2. The server should:
 - a. Wait for telemetry and SSDV data to be received and then forward this data to habhub.
 - b. Provide a means to export 2-way communications data in CSV format from the web portal.
 - c. Allow users to view a live log of telemetry packets and 2-way packets received by the server, also indicating when an image packet is received.
 - d. Provide a web portal to allow users to configure their payload and add it to the database.
 - i. The web portal should have a similar graphical design to the habhub website.

Payload Software

- 3. The payload software should:

- a. Transmit standard telemetry and SSDV on LoRa.
 - i. In that the LoRa should transmit standard telemetry and SSDV for the duration of its transmit cycle which is not taken up by 2-way packets.
 - ii. This is again going to require an SPI interface with the LoRa radio.
- b. Have a configuration file functioning in the same way as that of the ground station.
- c. Allow the 2-way communications to function as described in section 1, allowing the user to view diagnostics, access shell remotely, reboot, control transmission mode, etc.
 - i. Remote mode is, as described, using LoRa in a cycle of transmit and receive.
 - ii. A standardised packet format will be designed in the design section of the project.
 - iii. The LoRa radio should be used to transmit for a given number of 2-way packets, followed by the telemetry and SSDV packets and ending with a packet stating that the payload is about to begin receiving instructing the ground station to transmit queued packets.
 - iv. It should then wait to receive the fixed number of packets that will be send by the ground station. It should time out after a short while if it receives nothing.
 - v. It should then return to transmission.
 - vi. The payload should be able to handle any 2-way packets sent to it from the ground station, and it should queue appropriate responses to its transmit queue. For example, it should execute the command given in a shell command packet and transmit the resulting output.
- d. Add all packets which are required to be transmitted to a queue so that they can be sent in required order.
- e. Read from the GPS regularly, updating the current telemetry data so that the most up-to-date telemetry is transmitted each cycle. Additionally, it should clear the serial cache after reading a location fix from the radio as otherwise the buffer will fill up with old location fixes.
- f. Take images at fixed intervals using the Raspberry Pi camera module.
- g. Should be designed to continue functioning without failure under unforeseen circumstances.

So, to summarise, the project should encompass three modules: the controller module, responsible for both receiving standard telemetry from the payload and transmitting and receiving 2-way communication packets, as well as communicating with the central server to handle logging and packet forwarding (to habhub); the server which is responsible for logging all data and storing in a database configuration data for payloads; and the payload software responsible for operating the payload and also transmitting to and receiving from the ground station(s) as well as taking pictures and maintaining an accurate location fix for the telemetry.

Potential Programming Solutions:

Java

I could use Java with the Pi4J library which provides SPI, DIO and RS232 APIs. This would mean I only have to use a single library. Additionally, an Oracle Java Runtime Environment is available in the Pi repositories by default (see: <https://www.raspberrypi.org/blog/oracle-java-on-raspberry-pi/>) and is installed by default on the latest version of Raspbian.

Additionally, this allows me to use Java Swing or JavaFX for the development of the GUI which would provide excellent ease of development, though I am aware that there can be some issues with JavaFX on Raspbian. Additionally, an object-oriented approach will be sensible as I'm writing an interactive program with mutable states. Additionally, as habitat uses an HTTP interface, I could use any of the myriad of HTTP libraries available for Java, for example the Apache HTTP Client library or the built-in HttpURLConnection library.

Python

I could use Java with spidev for the SPI interface, the integrated GPIO library for DIO and pycserial for the serial interface. These are both very easy to use, however, pycserial has a few known bugs which can result in a read operation hanging indefinitely. The 'Requests' (see: <http://docs.python-requests.org/en/master/>) library is an extremely easy to use HTTP library. Python is also a very easy programming language to use and is often far more flexible than other languages. Additionally I could use Tkinter to develop my GUI. Python is also the most used language on Raspberry Pis.

C

I could use C and use the WiringPi library to access the serial, DIO and SPI interfaces. Then I could use a library like cURL to for networking. Furthermore, I could use ncurses to develop a command-line GUI. C is a more complex programming language but would provide a significant performance gain. However, as this performance gain will be unnecessary using C would be unnecessary.

VB.NET

VB.NET would probably not be suitable for this project because although it is possible to access the I/O devices of the Pi, VB.NET is certainly not designed with these in mind.

Chosen Solution

I will use Java and Pi4J to develop my project because Java's in-built Java Swing graphics libraries will make the development of my GUI simpler; the Pi4J library provides all the I/O capabilities that I need for my project. I will not use Python due to the potential issues with pycserial; I will not use C as the only reason to do so would be for performance which is not necessary here and adds unnecessary complexity; VB.NET is, as I have mentioned, unsuitable for this project. Additionally, I should note that Java has easy built-in functions for converting strings to and from ISO-8859-1 byte arrays.