AvaDrone: An Autonomous Drone for Avalanche Victim Recovery

AvaDrone:
An Autonomous Drone for Avalanche Victim Recovery

by

Benjamin D. Dickensheets

ES 100 Senior Capstone Engineering Project Report submitted to the School of Engineering and Applied Sciences in partial fulfillment of the requirements of

Bachelor of Science (S. B.) in Electrical Engineering

at

Harvard University

Spring 2015

Submitted by ....

Benjamin D. Dickensheets
School of Engineering and Applied Sciences
April 2, 2015

Supervised and Evaluated by ....

Robert J. Wood
Charles River Professor of Engineering and Applied Sciences

Introduction

For the 179 Americans that are caught in avalanches each year, timely recovery often means the difference between life and death. The goal of this project was to design and build a prototype drone for a system to quickly and automatically locate a buried victim, using an on-board antenna to receive a signal from industry standard transmitting beacons. The design was based on a quad-rotor platform and uses Arduino hardware to receive a beacon signal and navigate the craft.

In broad strokes, this project is an effort to apply the new and exciting technology of hobby drones to the well-established application of avalanche victim recovery. Current avalanche beacon technologies suffer from challenges associated with user operation. Slow or untrained human searchers are poorly equipped to handle the challenges of a fast-paced search. The vision of an entirely autonomous solution to this problem has guided the project from its inception.

This idea has been little explored despite a proliferation of drone technology in recent years. On one hand, all of the pieces of the project already exist in one form or another. Avalanche beacon technologies continue to mature, as do hobby drones and their application. This project builds on precisely these preexisting pieces, to ask whether they can effectively work together to create something new.

Throughout the project, I ran into challenges and roadblocks of all kinds. Whenever possible, I looked toward existing solutions to guide my design decisions or to justify admitting defeat on a particular difficulty, in order to maintain my focus on the larger questions of *how all of the pieces will work together.* As I hope I have conveyed, the real contribution of this project is located at the intersection of these technologies, and it is there that I have focused my energies.

While much remains to be done on this project, the results that I have found all point to the viability of this project. This project isn't close to being ready to actually rescue someone, but the pieces are all in place and ready for further development. More questions remain, but I hope that this work will help to propel avalanche recovery technologies into the future.

Contents

Background

2.1 Motivation: Avalanches in the U.S.

Last year alone, 38 Americans were buried and killed in avalanche accidents[1]. Worldwide, these numbers are even higher, with approximately 45% of victims caught in an avalanche surviving the experience[2][3]. Backcountry recreationalists have traditionally had to simply accept the risks associated with winter sports as part of the package. However, with the advent of avalanche beacon technologies in the 1960's risks have been reduced and increasing numbers of people flock to avalanche terrain in the winter. As beacon technologies have matured, trends of decreasing rates of avalanche fatalities have been countered by a massive influx of recreationalists. Figure 2.1 shows historic data, as compiled by the Colorado Avalanche Information Center (CAIC).

While it is not important to go into a detailed discussion about how and why avalanches occur, it is useful to frame the problem in terms of a shared basic understanding of the mechanics of an avalanche. When we talk about avalanches, we generally mean a specific type known as a slab avalanche. During normal winter conditions, snow bonds together with its neighboring snow into the slab that gives these slides their name. In a slab avalanche, this cohesive mass breaks loose from the substrate it is resting on and accelerates downhill, sweeping anything in its path along with it.

In 94% of fatal avalanches, the slide is triggered by the victim or a member of the victim's party[4]. Once triggered, there is no escaping the downhill tumble. Recreationalists are advised to do their best to stay afloat by "swimming" through the slide, made difficult because of the low density of snow. Once caught, a victim faces two main perils. Trauma accounts for around a quarter of fatalities, while the remaining are due to asphyxiation[5]. In the case where the buried victim is uninjured when the avalanche comes to rest, time determines whether he or she will survive. Survival rates plummet in the first 30 minutes of burial, to less than 20%[2]. It is clear then, that minimizing the time necessary to find the victim is crucial and the defining element of current rescue techniques.

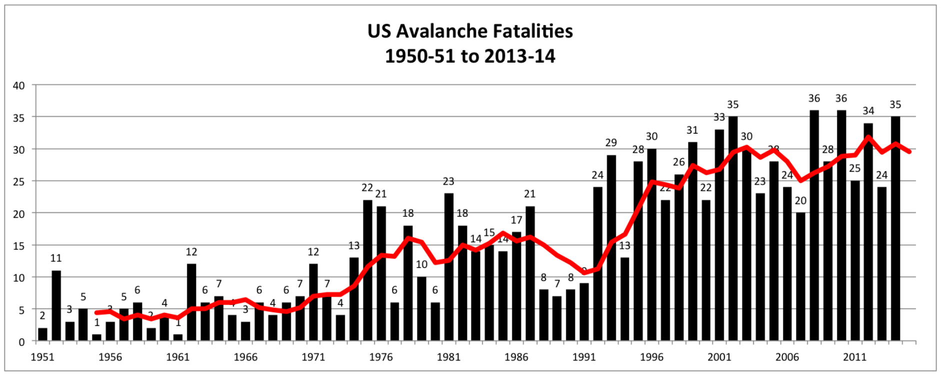


Figure 2.1: Trends of avalanche fatalities from 1950 to today[1]. The red line represents the 5-year moving average.

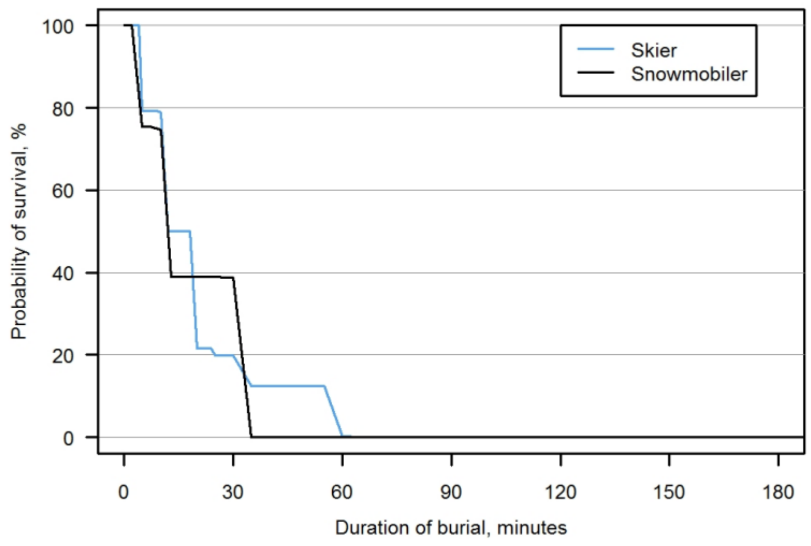


Figure 2.2: Relationship between duration of avalanche burial and chances of survival of winter recreationalists in Canada[2].

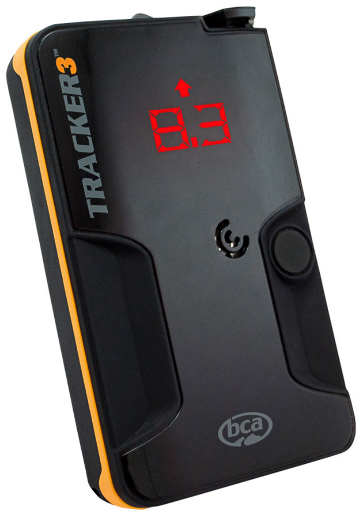


Figure 2.3: This is an example of a standard avalanche beacon in search mode. The direction indicator and distance (signal strength) readout are clearly visible.

2.2 Current Rescue Techniques

The invention and proliferation of avalanche beacons was an incredible turning point in the history of avalanche precautions. A small, portable radio-frequency transmitter worn on the body could be used to guide searchers to the location of burial and turn an avalanche into an event that is actually survivable. Today, beacons function both as transmitters and, in the case of an avalanche, receiver units for a search party.

Because of the tight timeframe for recovery, searchers nearly always have to be part of the victim's party. This requires every member of a group to wear a beacon while out recreating. Beacons cost between \$250 and \$500. It is also imperative that each person also have a shovel and a long (collapsible) metal probe to determine the depth of burial. Once the beacon has guided searchers to the burial location, these tools are used to actually extract the victim.

Beacons transmit a pulsed signal at 457 kHz, which can be received by a different beacon operating in "search" mode, up to 30 or 40 meters away. The first step of any search is traversing the slide path while looking for a beacon signal. Once the initial signal has been obtained, the beacon indicates

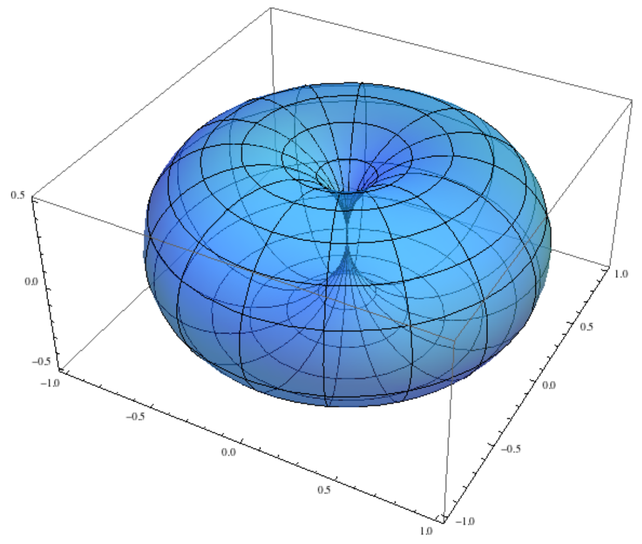


Figure 2.4: Magnetic radiation pattern of a loop antenna, like the one in a transmitting avalanche beacon. The transmitting antenna is oriented vertically.

a direction of travel and a distance, based on signal strength. Traveling in the prescribed direction, the searcher knows that they have reached the victim when a digital readout of signal strength starts to decrease. At this point, the searcher uses the avalanche probe to physically locate the victim and begins to dig he or she out.

Modern beacons are complex affairs, with multiple antennas and sophisticated signal processing techniques to handle situations where more than one person is buried or where the transmitting antenna is oriented sub-optimally. Despite the various complexities, the underlying principles are relatively simple.

2.3 Electromagnetic Radiation at 457 kHz

Waves with a frequency of 457 kHz have a corresponding wavelength of $\lambda = 650$ m. Because the boundary between near-field and far-field radiation patterns is approximately given by $\lambda/2\pi \approx 100$ m, and the average range of a commercial avalanche beacon is between 30 m and 60 m, the received signal exhibits a complicated near-field radiation pattern, such as the

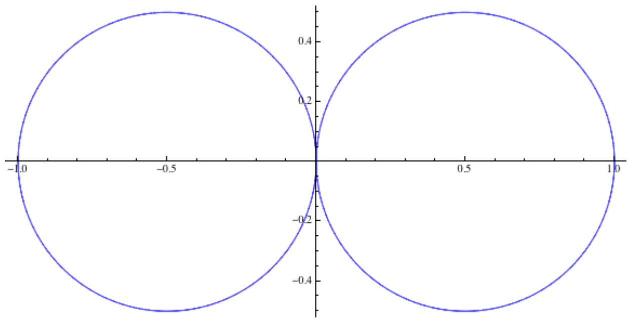


Figure 2.5: 2-dimensional slice of the radiation pattern in Figure 2.4. Imagine this plot as a vertical view of the ground. The antenna is oriented horizontally in the y-direction. These are the magnetic field lines seen by the receiving antenna.

one shown in Figure 2.4. This plot, as well as the one in Figure 2.5, depict magnetic field lines. The transmitting and receiving antennas in commercial beacons are loop antennas, usually constructed by wrapping wire around a ferrite core.

The receiving antenna receives a maximum signal when it is oriented parallel to the line of magnetic flux at its particular location. The beacon leverages this information to point a searcher along the flux line, allowing the searcher to "follow" the curved path in toward the buried victim.

Because of the apparent difficulties associated with a vertically oriented transmitting antenna like we see in Figure 2.4, (imagine taking a horizontal slice of this field pattern—the flux lines only travel in a circle and don't travel in toward the transmitter!) new beacons actually have multiple antennas and choose which one to transmit from based on measured orientation of the transmitter unit.

In the near-field regime, the field amplitude falls off with a $\frac{1}{r^3}$ relationship, while the power falls off with $\frac{1}{r^3}$. Moreover, the transmitted power is on the order of ∼0.1 W. This points to the intricacies of commercial solutions, and also foreshadows some of the difficulties I faced later in the project.

2.4 The Hobby Quad-Copter

On the other side of the project, the remote-controlled drone is becoming increasingly ubiquitous with every passing year. Specialized drones dot the skies, taking video, mapping, or simply flying. Drones follow complicated, autonomous flight plans, or follow a transmitted signal from a cell phone.

Proprietary and open-source platforms have both achieved substantial success in the market. DIY kits capable of autonomous flight are comparable in price to a commercial solution, between $800 and $1,200. Notable work has been done in the open-source arena by developers of the ArduCopter software, as well as hardware developers such as 3DRobotics[6][7].

Looking at hobby quads on the market today, they certainly appear ripe for applications related to avalanche victim recovery. As one might expect, this project is not the first attempt to combine drone technology and avalanche applications. However, some notable differences exist, as I describe next.

2.5 Prior Art

Most of the work that has been done at the intersection between drones and avalanches has been conducted in the Alps, where slightly different recreational practices have led to somewhat different approaches. One major difference is, with a single notable exception, the use of a different beacon technology[8][9]. Additionally, because of the concentrated activity in the Alps, these platforms are designed to be located a base stations and take off in response to a report of an avalanche in the vicinity. In the United States, backcountry use is much more spread out and would require any technology to be carried with recreationalists. Effectively, these designs solve a fundamentally different problem. However, aspects of their design could be used to motivate the current project.

In addition to these projects, a group in switzerland is reported to have worked on a project very similar to the present one in 2011[10]. Unfortunately, few details are available about their success or any continued work, so it is unclear how far the project progressed and if it was ever built.

2.6 Defining a Successful Project

With this background, we now approach the challenge of defining the scope and objectives of this project. The problem we face is that of designing and building an *automated* search platform to locate a buried avalanche victim more *efficiently* and *reliably* than current methods. As demonstrated by the prior art in this area, this task is more suited to a multiple-year development timeline with a team of engineers.

Consequently, the goal of this current project was to demonstrate the viability of a quad-rotor platform as an autonomous receiver. The project is based on the assumptions that the victim is wearing a 457 kHz industry-standard avalanche beacon. Additionally, I make the crucial assumption that the receiver is already within the signal range of the transmitting antenna. While this is only part of the search process, I restrict my attention to this part because the conclusions are largely generalizable to the acquisition phase of the search. In fact, the acquisition phase is actually much closer to standard autonomous quad-copter flight, where the craft simply follows a pre-determined path until the signal is picked up.

With this in mind, it is important that a successful project demonstrate the craft's ability to sense a signal, show directional sensitivity to the signal, and be able to direct movement in response to the signal. Additionally, the project should demonstrate that the functions of flying and of detecting a signal do not interfere with one another. Ideally, all of these steps take place while the Quadcopter is under its own power, but given the difficulties associated with developing an autonomous craft, this should not be a strict requirement for the project.

Design and Implementation

Considerations while designing this project ranged from price, to weight, to electromagnetic signal interference from various sources. In order to describe and work on each constraint, I broke the problem into discrete sections, looking first at the quad-copter, signal processing, and algorithm modules separately, and then considering constraints that are implied at the intersection of these different parts. Because my designs changed throughout the project in response to unanticipated complications, the project design and building process were intimately entwined. This section will walk through each iteration of the project, describing design and fabrication decisions and the influences each had on the other.

3.1 Quad-Rotor Platform

I will begin, as I did during the design phase of this project, by considering the quad-copter as a platform that would help to achieve the stated goals of the project.

3.1.1 Broad Platform Considerations

A successful platform design should address the following aspects of the broad problem statement: Automation, Efficiency, and Reliability. While many different platforms exist that satisfy these three tenants, secondary considerations of ease of use and broad-based support for the platform suggest the viability of the quad-copter platform. Certainly the sheer variety of different applications shows the versatility of the quad-copter. It also easily satisfies the requirements for Automation, Efficiency, and Reliability.

Automation is important because it allows a searcher to be uninvolved in the search process, leaving their attention focused on other aspects of the rescue. It also may eliminate operator errors that cost precious time. The quad-copter, I have stated previously, has a demonstrated track record for autonomous flight and in many cases is easily programmed to follow new direction sets, an important aspect of any iterative design process.

Efficiency takes into account the time needed to located an avalanche victim as compared to traditional search methods. One method for optimizing this time is to increase the search speed. A searcher traveling on foot over uneven avalanche debris will be much slower than an aerial craft.

Reliability is immensely important for a system that is involved in rescue work. Any final design must be incredibly robust. Here again, the quad-copter, with redundant thrust and demonstrated stability is an excellent choice for an autonomous craft.

With the decision made to use a quad-rotor platform for the rest of the project, the first task was to build and demonstrate a working quad-copter.

3.1.2 Initial Platform

Figure 3.1 depicts the platform that I started with. This was a project that a friend[1] and I worked on a few years ago. The platform I inherited consisted of a frame, the motor electronics, an Arduino microcontroller to run the motors, and a mess of broken code.

There were a few considerations that prompted me to start with this frame. The most obvious was related to budget. With at $500 operating budget, the price of existing quad-copters was too high to consider. Even purchasing a simple frame would have proved cost-intensive and inefficient. On top of that, the decision to purchase a different frame would have been accompanied with a period of discovering how best to control the motors with my custom software. Since this challenge is peripheral to the end-goal of the project, I decided to stick with a frame that I was familiar with and that I knew could physically sustain flight.

3.1.3 Flight Control Board

On the subject of flight, the inherited platform certainly had the physical capabilities to fly, but it lacked the precise control algorithms necessary to actually maintain flight. Again, because the challenges associated with writing and testing these control algorithms is entirely peripheral to the central problem, I decided to use a good portion of my budget to purchase a commercial flight control board that is designed to plug into a custom frame

[1]A huge thanks to Akeo Maifeld-Carucci for allowing me to use our platform for this project!

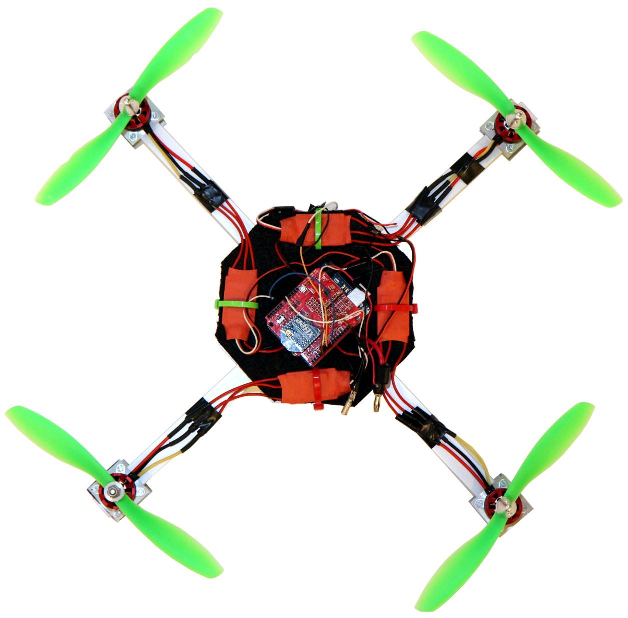


Figure 3.1: The initial quad-rotor platform.

and control the craft's attitude and throttle based on inputs from integrated sensors and information from a hobby remote control receiver.

The PixHawk[11] flight control board that I purchased is an open-source product that runs similarly open-sourced ArduCopter[6] software. It includes built-in GPS functionality and a variety of other features, which I hoped to leverage for my project. An example of the configuration interface is found in Figure 3.3.

3.1.4 Remote Control Transmitter and Receiver

In order to function the Pixhawk expects an input signal from a remote control receiver. It takes this signal and processes it to determine the control loop setpoint, which allows a user to remotely control the vehicle. I used a hobby RC transmitter/receiver pair that was readily available. The output signal from the receiver was a 4-channel parallel pulse width modulated (PWM) signal, for the 4 standard controls of roll, pitch, throttle, and yaw. The difference between these controls is shown in Figure 3.6. However, the Pixhawk expected a single input signal, a so-called pulse position modulated

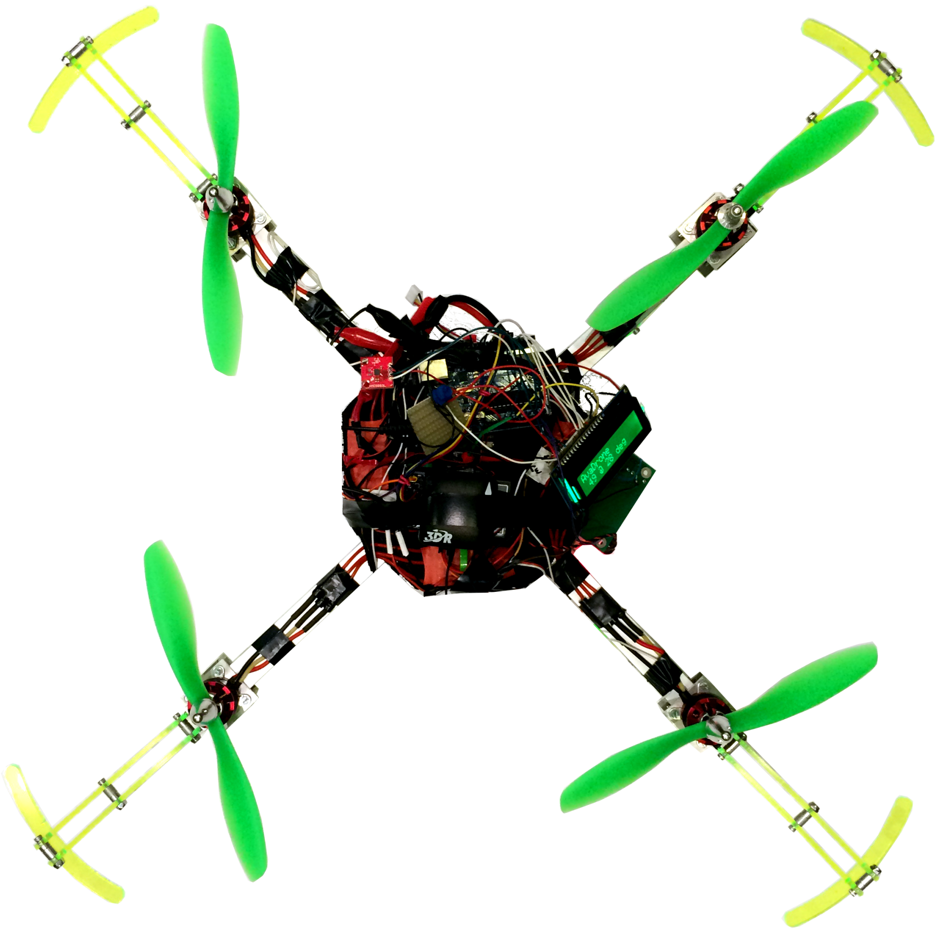


Figure 3.2: The final quad-rotor platform.

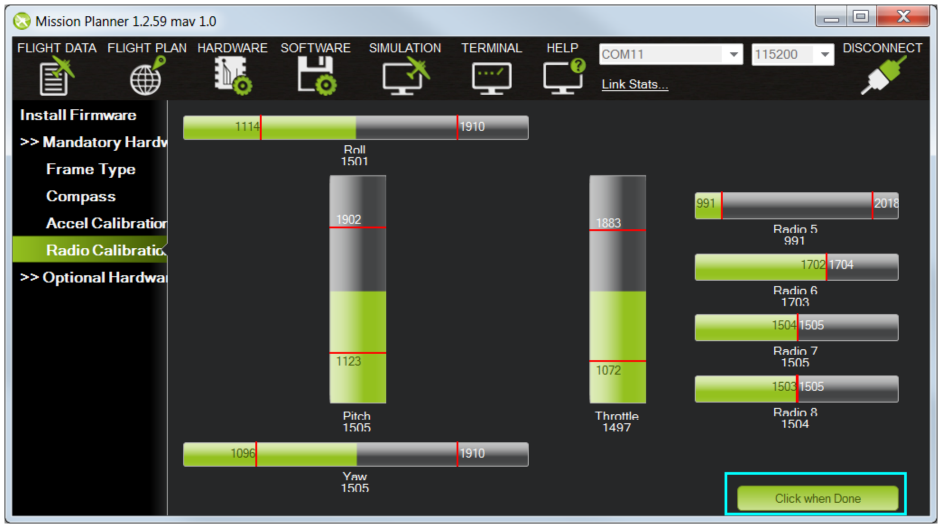


Figure 3.3: An example of the ArduCopter interface[12]. This screen allows a user to configure the remote control transmitter/receiver pair, setting maximum and minimum control values.

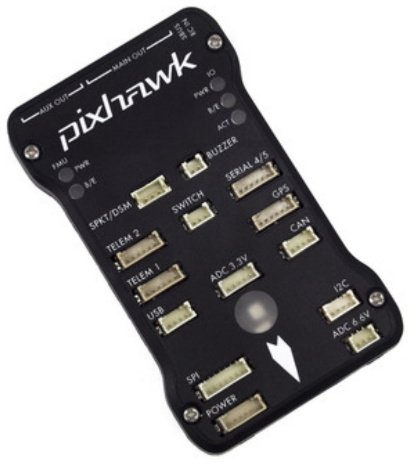


Figure 3.4: I used the Pixhawk flight controller for low-level stabilization of the AvaDrone. The controller took inputs from an Arduino and provided outputs to each motor controller[11].

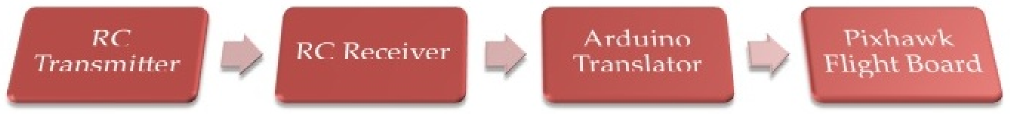


Figure 3.5: The flow of information from remote control transmitter to flight controller.

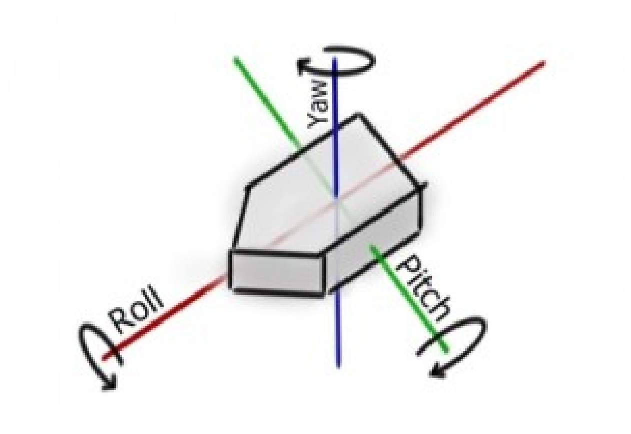


Figure 3.6: Definition of the rotational controls, as used in this document[13].

(PPM) signal. I therefore connected an Arduino between these two signals, and modified a library to enable the Arduino to translate between these two protocols.

3.1.5 Arduino Functionality

While the Arduino was initially included as a simple translator, it later became clear that placing the Arduino in the signal path would have other benefits. Initially however, the Arduino passed the signal from input to output with a minimal amount of processing or analysis. The code for this translation step is contained within `Comm_Test_1.ino`, in Appendix B.

In essence, this code takes and reads in 4 different PWM signals, looping through and storing them in an array. The output signal is controlled by an interrupt timer, which periodically checks the value stored in the array, and outputs an 8-channel PPM signal.

I found that this process in the Arduino was somewhat noisy, so as an added step of signal processing, I added a low-pass filter, averaging N number

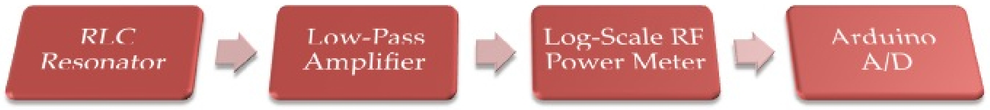


Figure 3.7: The flow of information in the final design. The inductor in the RLC resonator also acts as the receiving antenna. See also Figure 3.14 for schematic-level detail.

of input samples together before updating the expected output value. The trade-off comes at the expense of speed, but this scheme was able to clean up the signal without a noticeable loss in responsiveness.

The elegance of this implementation is clear when we consider the ease by which autonomous control can be asserted in the control loop. Signal processing in the digital regime is relatively low-cost and incredibly easy to modify, allowing for a large number of design iterations in a short period of time. I will discuss the algorithms I adopted in Section 3.3.

3.2 Antenna Design and Signal Processing

The other major portion of the project was an antenna to detect an avalanche beacon signal at 457 kHz and pass that signal into an Arduino to interpret and process. The basic flow of information that I ended up with is shown in Figure 3.7. The corresponding schematic is shown later in Figure 3.14. Each module was designed separately, usually in response to a problem that I encountered along the building process.

Indeed, the initial design involved only an RLC Resonator and the analog to digital conversion. I'll explore the particular limitations and challenges associated with this method in Section 3.2.2. However, my design began with the instrument designed to take the transmitted signal and turn it into useful information, the antenna.

3.2.1 Ferrite Rod Antenna

A signal with a frequency of 457 kHz has a wavelength of 656 meters. At this low frequency, traditional half-wavelength dipole antennas are impractical and not typically used. The loop antenna is the usual choice for a

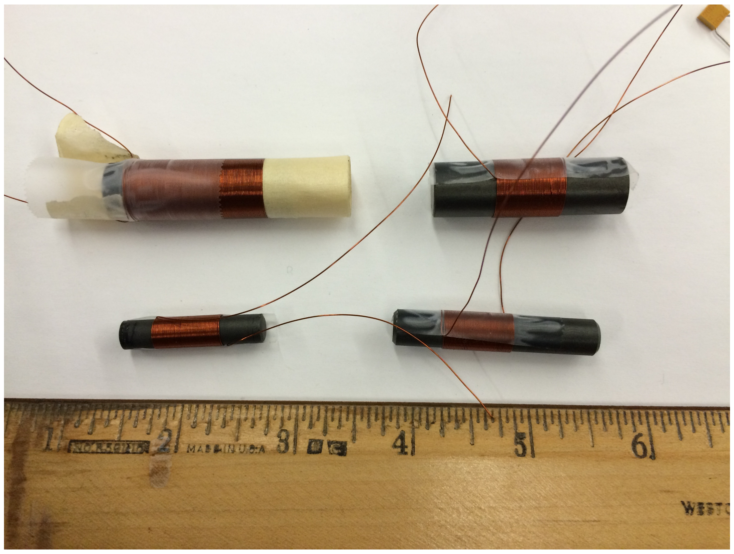


Figure 3.8: I tested three different types of rod geometries, with various wrapping geometries. The final design selected for high inductance and minimal parasitic capacitance.

receiving antenna in this regime. One common example is the oft-seen loop antenna for receiving AM radio. One technique that can be used to increase the sensitivity of the loop antenna is to wrap wire around a magnetically susceptible core, either ferrite or similar. In fact, commercial avalanche beacons also use this technique for their antennas, because it allows the receiver to be much more compact for a given sensitivity.

I began with three different rod geometries to compare. They were between 1 and 2 in. long, and varied from 8 mm to 34 mm in diameter. I closely examined two different geometries, and iterated through a number of different versions, looking to maximize inductance and the resultant sensitivity of the antenna. Some of these iterations are shown in Figure 3.8. My findings are summarized in Table 3.1, which lists measured parameters for various geometries.

Of the different antennas I measured, I was able to maximize inductance while keeping parasitic capacitance to a reasonable level by selecting the 100-wrap, 560 μH antenna for testing. It is important to note that more investigation could go into selecting an appropriate antenna design. This is a topic that I will address again in Chapter 5.

Table 3.1: Measured inductance and parasitic capacitance of various rod geometries. The missing values are a result of not taking parasitic capacitance into account in initial tests. Based on initial results, I opted not to retest each geometry.

Rod Length (cm)	Rod Radius (cm)	Rod Weight (g)	Initial Perme-ability	Wire Wraps	Induc-tance (uH)	Parasitic Capacitance (pF)
4.13	0.616	27.0	2000	25		
4.13	0.616	27.0	2000	50	176	
4.13	0.616	27.0	2000	100	560	64.6
4.13	0.616	27.0	2000	200	2200	187
4.50	0.400	11.9	2300	25		
4.50	0.400	11.9	2300	50	140	
4.50	0.400	11.9	2300	100		

For a video showing an oscilloscope trace of the induced EMF in the inductor from an avalanche beacon, see `Inductor_Induced_EMF.MOV` in the submitted Supplemental Material. In this short clip, you can clearly see the pulsed nature of the transmitter and the sensitivity to relative orientation. The receiving antenna is located ~3m away from the beacon, on top of the breadboard which can be seen to the left of the oscilloscope.

Parallel vs. Serial RLC Filter

There was a decision made somewhere along the line to switch from a parallel LC tank topography to a serial topography. The decision was made on the basis of examining the input impedance of the subsequent power-meter stage. In order to generate a high-Q filter, the input impedance of the next stage should be large. Unfortunately, it happens to be a small, 50 Ω affair. It is possible to achieve a large R_{IN} by adding a large input resistance in serial but this necessarily attenuates the signal, and so this seems to call for a serial topography.

However, the LC Tank has the virtue of being able to store energy, so it has the potential to actually provide gain when driven at resonance. These competing factors, the necessary attenuation of the signal and the stored energy seem to essentially cancel one another, as I discovered while testing.

I kept the serial topography for the remainder of the testing, as you can see in Figure 4.4.

3.2.2 Sampling with Arduino

My initial design involved directly sampling the signal from the antenna by the same Arduino already located on the quad-copter platform. However, I quickly discovered that the sampling frequency of the Arduino Uno that I was currently using was nowhere near fast enough. By the Nyquist criterion, any sampling would need to happen at twice the received frequency, around 1 MHz for this particular application. Moreover, we need additional room to reconstruct a usable signal, even with a very steep anti-aliasing filter. This suggests a goal sampling frequency of 2 MHz or higher.

I conducted tests with the much more powerful Arduino Due in order to determine the maximum sampling frequency. What I discovered was that the on-board A/D converter couldn't run faster than about 800 kHz, despite various optimizations and running a script that did nothing but take a sample and store it in an array.

A quick discussion about the Arduino A/D is perhaps in order. The A/D is clocked by a division of the system clock. It can be put into *continuous* mode, which starts a new sample as soon as the last sample is complete[14]. In addition, the sampling rate can theoretically be increased by decreasing the quantization. The Due generally supports 12 bit conversions, but can also be told to ignore the two least significant bits. However, this mode showed no appreciable improvement to speed.

Facing these limitations, I was left with the decision to either locate a more robust microcontroller, a daunting task, or continue with the Arduino and add an analog front-end to process the signal prior to sampling. With the relatively straightforward solution that I discuss in the next section, I decided against choosing a new microcontroller and all of the overhead work that would be required to learn a new programming environment and particulars of the microcontroller instruction set. In the future, it will likely be necessary to revisit this design decision and update the microcontroller. See Chapter 5 for more details.

3.2.3 An Analog Solution

What became apparent was the necessity of designing a solution that converted the amplitude of a low-frequency RF signal into a DC (or at least a much lower-frequency) signal. One simple solution for this problem comes in the form of a common envelope-detector. This circuitry basically amounts to an AM demodulator, and is relatively easy to implement. However, because the transmitted signal falls off like $\frac{1}{r^3}$ as you move away from the transmitter ($\frac{1}{r^6}$ when considering power), the receiver circuitry needs a large dynamic range. One solution to this problem is to use a logarithmic-scale RF power meter, like the AD8362 from Analog Devices[15].

In addition to its logarithmic output, the AD8362 has a variety of characteristics well-matched for this application. It is a wideband part, taking inputs from 50 Hz to 3.8 GHz. This clearly includes the range around 0.5 MHz where the beacon signal is located, but unfortunately it also introduced problems down the road. Without a pre-amplifier of any kind, the AD8362 can understand inputs down to $\sim$1 mV. At further distances, the signal from the antenna would need to be amplified, but when located close to the transmitter, no additional amplification circuitry is needed.

The output signal of this board is a DC signal that maps to a logarithmic scale of 50 mV/dB. An input power of -52 dBm is mapped to 100 mV output. This scale allows users to easily convert from raw voltages to signal power.

I was able to procure[2] a printed circuit board with the AD8362 and the auxiliary parts described in the Typical Use section of the datasheet[15] (see Figure 3.9. From this building block, I was able to modify the circuitry to suit my particular needs. One extra word about this particular board is in regards to its included voltage regulator, which allowed me to easily power it from the 12 V supply.

Board Modifications

Looking at Figure 3.9, we see that our input signal, in addition to being amplified by a factor of 4, is passed through a high pass filter defined by the value of C6 and C5. The differential input impedance is 200 Ω, so that after the signal passes through the transformer it sees a standard 50 Ω. What this means, though, is that the cutoff frequency for this high pass is given by the

[2]Thank you to Jim MacArthur for helping me track down this part and helping me to work through the details of this particular sensing scheme.

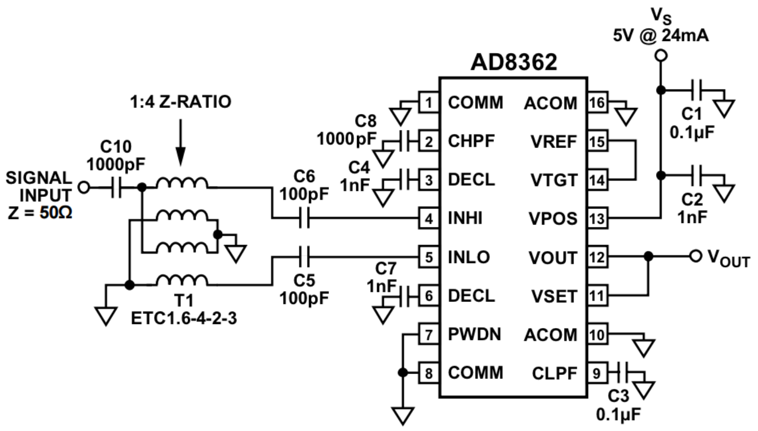


Figure 3.9: This is the approximate layout of my power meter circuit[15], though some component values have been modified (see section on Board Modifications for details).

formula,

$$f_{HP} = \frac{1}{200\ \pi\ C5}, \text{assuming } C5 = C6, \tag{3.1}$$

and for the values listed in Figure 3.9 $f_{HP} = 15$ MHz. I replaced the values of C5 and C6 with 0.1 μF capacitors to get $f_{HP} = 15$ kHz, allowing the transmitter signal to pass through unattenuated.

3.2.4 Improved Filtering

After an initial round of testing, it became clear that the power meter was receiving much more noise than it should. Indeed, one clear indicator was the significant interference caused by the remote control transmitter which operates at 72 MHz. These pointed to a problem with the assumed band-pass capabilities of the RLC circuit, namely that at high enough frequencies the inductor looks like a capacitor and the whole circuit resembles a high-pass filter. Because I had not previously included any low-pass elements and because the power meter has such a large bandwidth, these high frequency signals contributed a large portion of the measured power.

To solve this problem, I designed the low-pass filter shown in Figure 3.11. It has the dual property of amplifying signals in the passband and attenuating signals above the f_{3dB} cutoff. I designed a single-supply amplifier

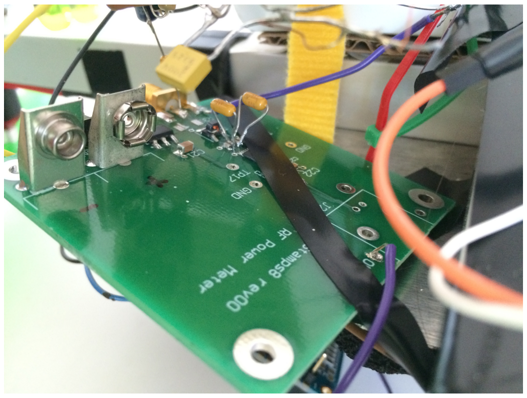


Figure 3.10: This is a picture of the AD8362 circuit board, as mounted on the underside of the quad-copter. You can clearly see the modified capacitors and the purple output signal wire in the picture foreground. Power is provided by the global 12 V supply, rather than the optional 9 V battery.

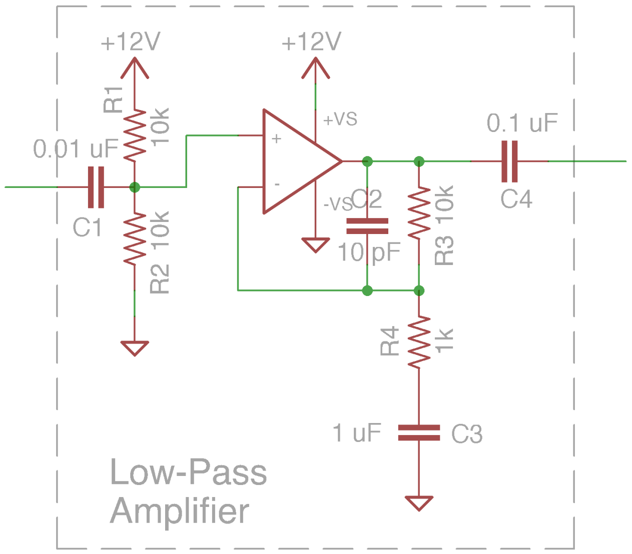


Figure 3.11: The single-pole low-pass amplifier I designed, with a gain of 10 in the passband.

and biased my input at half of the 12 V supply. Here are a few other key design considerations:

- Input impedance is 5 kΩ, easily high enough for our 50 Ω input.
- The input stage is a high-pass filter with $f_{3dB} = 3$ kHz. While it would be possible to set this pass frequency higher, the observed operation of the band-pass was that it is effective at blocking low-frequency signals and therefore any additional filtering is largely redundant[3]
- Resistors R3 and R4 define a gain of 10 in the passband.
- The capacitor C2 in parallel with resistor R3 defines the low-pass filter. $f_{3dB} = 1.5$ MHz, so we maintain less than 10% attenuation in the passband.
- The capacitor C3 functions to cut DC gain to 1, biasing the output at half of the 12 V supply. In series with resistor R4, it creates a high-pass filter with $f_{3dB} = 150$ Hz.
- Output impedance is low, so that it can drive the 50 Ω power meter.

The actual implementation was somewhat different because of a limited selection of parts, but achieved similar functionality nonetheless. The main restriction was a dearth of operational amplifiers with a Gain Bandwidth Product larger than 4 MHz. In order to account for this limitation, I redesigned the amplifier, making explicit use of the decreasing op-amp gain. Figures 3.12 and 3.13 show the updated schematic and the open-loop gain bode plot for the LF412. As you can see, I entirely removed the low-pass elements because of the gain compensation of the op-amp. The marginally reduced complexity drove this particular design decision.

With this design, it was possible to see the amplification step as well as the -6 dB/octave fall off of a single pole filter. More discussion on the filter design as well as future improvements to the design are addressed in Sections 4.3 and 5.

3.2.5 Putting It All Together

The overall schematic is shown in Figure 3.14. While many improvements

[3]This is in contrast to the noise above resonance, which dominates.

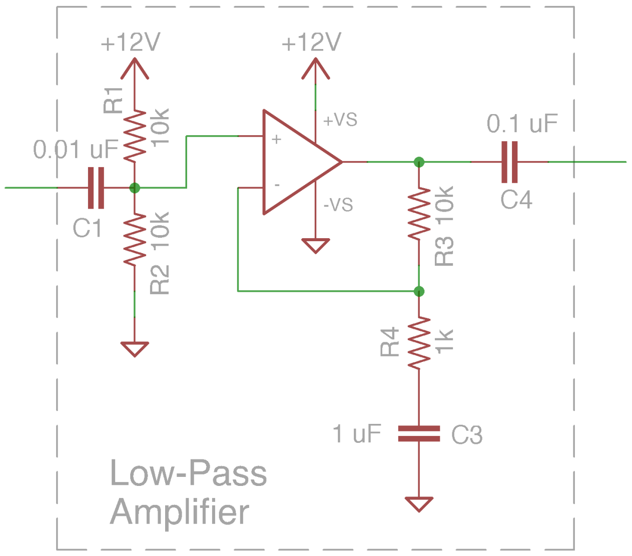


Figure 3.12: The low pass amplifier as implemented, with a gain of ∼6 in the passband. The op-amp is the LF412, a JFET input low offset op-amp[16].

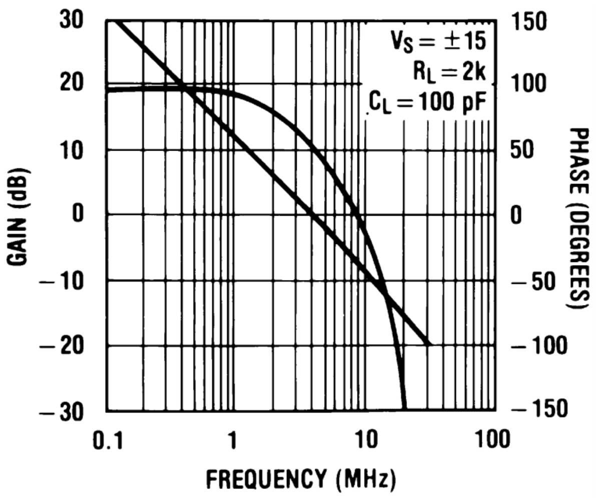


Figure 3.13: Open-loop gain as a function of frequency for the LF412[16].

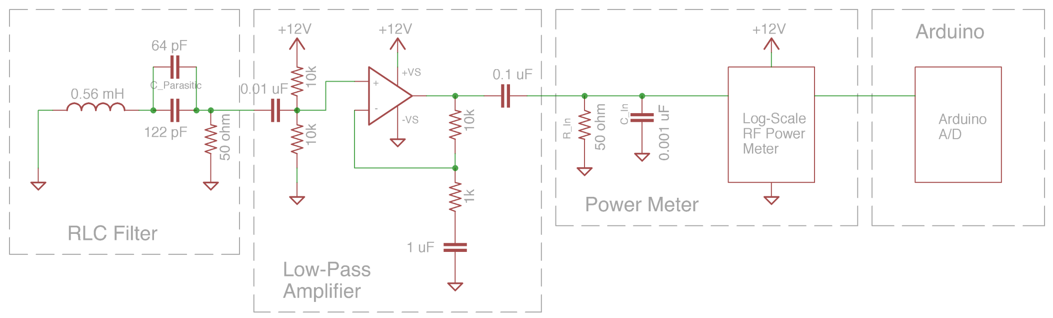


Figure 3.14: The full schematic for the receiver architecture used in this project, as suggested by Figure 3.7.

remain to be made, This circuit enabled me to detect a transmitted signal and investigate the feasibility of the overall technology.

3.3 Algorithms and Integration

The standard search method for avalanche victim rescue is described in Section 2.2. Because of its historic success as a method, I opted to closely mimic this algorithm for the project.

In terms of a high-level approach, the algorithm can be expressed in the following manner:

1. Take sample
2. Rotate by a small, defined amount
3. Take another sample
4. If the signal is smaller, change the direction of rotation. Otherwise maintain the same direction
5. Continue to rotate and take samples until the signal strength starts to go back down. Take note of maximum signal strength
6. Rotate back to the strongest signal direction
7. Move forward by a defined amount

8. Repeat Steps 1–6. If the maximum signal strength is smaller at the new location, reverse direction of travel
9. Repeat Steps 1–7 until maximum signal strength is smaller at the new location
10. Notify searchers that victim has been located

The project did not reach the stage of autonomous flight, so the algorithm has not yet been translated into code. However, I was able to test a few key steps, in particular Step 5. I will discuss the results of my testing in the next chapter, as well as important considerations for future work.

In order to do these tests, I designed a system by which I could insert a human tester into the control loop. I added an LCD display to which the Arduino could pass information about the received signal strength. The code I ran to make this work is found in Appendix B.10. This code made use of the filtering discussed in Section 3.1.5 and displayed it on the attached LCD screen. This particular step allowed me to test more important aspects of the project despite the challenges presented by quad-copter flight.

In the next chapter, I will detail all of the different testing I completed and discuss my results.

Testing and Results

This phase was roughly split between testing indoors and testing outdoors. Each had its own benefits and challenges, which I will enumerate more fully throughout this chapter. Testing a project is intimately related to the project design, in the way it is not only driven by design, but also validates past designs and inspires new ones. Much of this chapter has already been discussed in the previous one, but the emphasis will be somewhat more focused on what the results imply for the broad project goals.

4.1 Flight Testing

The first step in testing the quad-copter was to determine whether it would fly. Before any flight testing could be done, I had to accomplish the design work discussed previously and a somewhat arduous debugging process to make sure that each piece of the remote control signal pathway spoke to its neighbors. Accomplishing this, the craft was ready to fly.

Flight testing was initially done outdoors, at the Harvard athletic fields, so as to test in as deserted an environment as possible. This was largely a safety concern, both for surrounding buildings and people, and for the craft itself.

The first tests demonstrated that the architecture of using the Arduino as a translator was highly successful. The craft exhibited a constant drift, however this was easily corrected by the operator. Subsequent tests probed the functionality of the GPS hold mode, a function designed to keep the craft at a fixed GPS position without any user input. This was unfortunately unsuccessful and prompted the design changes described in Section 3.3.

I believe that there are a few different reasons that the GPS functionality wasn't able to produce the desired results. For one, the behavior that I observed consisted largely of a constant drift. While one would expect a closed-loop control algorithm to correct for whatever drift the craft was experiencing, the precision of a GPS locating signal is likely not fine enough to avoid any drift. The other observed behavior was occasional glitches in the aircraft attitude, where the quad-copter would suddenly pitch or roll

Figure 4.1: Testing at the Harvard Athletic Fields while snow flies.

before recovering. This behavior could be explained in terms of the GPS symptom by hypothesizing that the glitch was the control algorithm's ill-fated attempt to correct for a slowly accumulating drift. However, the fact that this glitching also occurred during normal flight points to a problem with the transmitter/receiver pair, or more likely with the Arduino software loop. Indeed, I believe that these glitches are caused by the low-level timing interrupts that drive both PWM input and PPM output coinciding and disrupting each other's timing. The Arduino can only support one interrupt at a time, so this seems to be a likely candidate for the glitches.

Further testing with the code could confirm or disprove this particular hypothesis. As I mentioned in the Design chapter, one way to address this problem would be to upgrade to an industrial microprocessor that is better equipped to handle the sorts of time-sensitive tasks that we require.

In the absence of this upgrade, and in coming to the realization that I would be unable to produce autonomous flight, some of the initial goals and design considerations had to be reevaluated and somewhat relaxed. It was at this stage that I determined that I could still assess the viability of the project without explicitly demonstrating autonomous flight. Autonomous

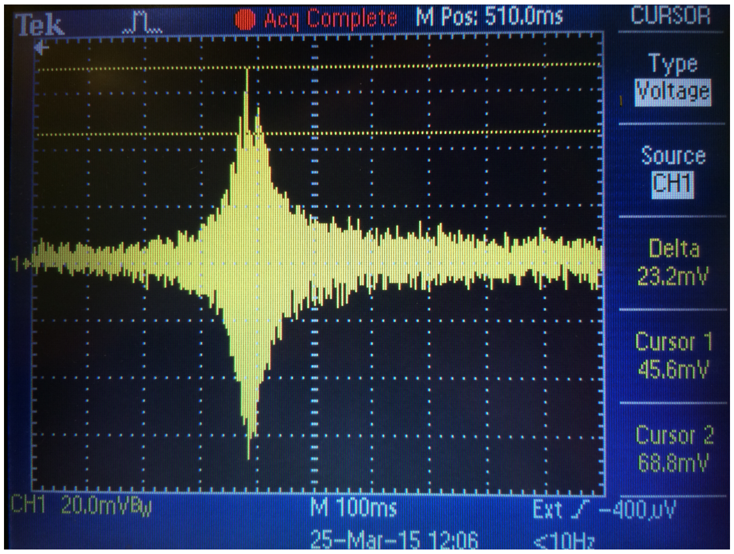


Figure 4.2: Frequency response plot showing one RLC test configuration with a measured quality factor of 38.

flight is a solved problem and the failure of my design represents simply that, my design. It does not, however, imply the failure of a more rigorously designed and tested flight module. With this in mind, I determined that simply approximating the sort of autonomous flight required by the algorithm by physically carrying the craft would be sufficient to demonstrate either success or failure of the global project design.

4.2 Antenna Characterization

Characterizing the antenna and resonator was done in a couple of different steps. The first consisted of driving the antenna with a function generator and characterizing the frequency response. Two of these configurations are shown in Figures 4.2 and 4.3. Both of these were tested in the series configuration (see Section 3.2.1 for a discussion of this design), with two different input impedances. The circuit being tested is shown in Figure 4.4

The calculation used for the quality factor was

$$Q = \frac{2\pi f_0 L}{R}.$$

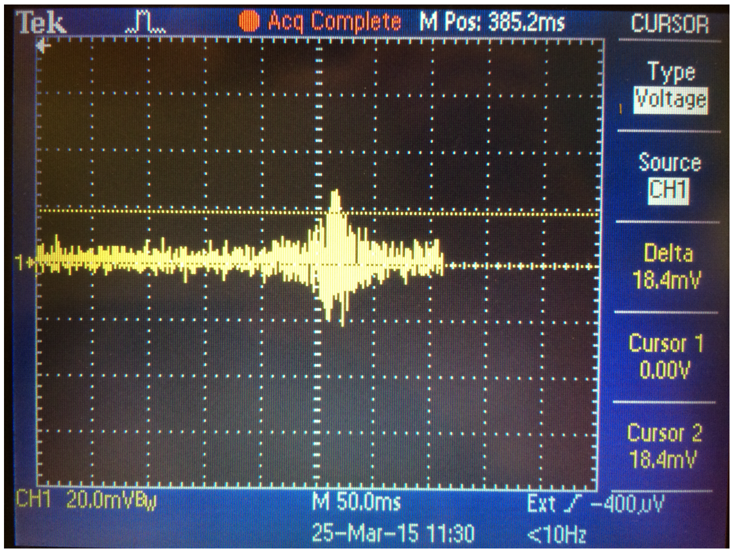


Figure 4.3: Frequency response plot showing one RLC test configuration with a measured quality factor of 5.

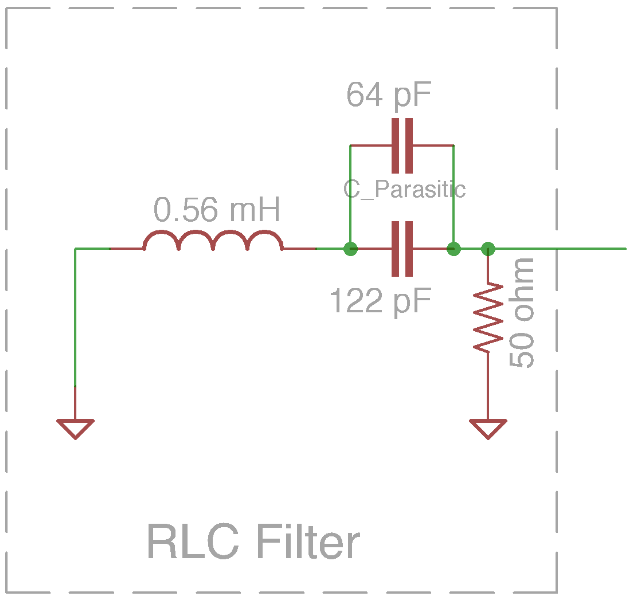


Figure 4.4: Series band-pass filter configuration. R was varied and the resultant quality factor recorded in Table 4.1.

Table 4.1: Measured and calculated quality factor for different resistive values in the series RLC configuration shown in Figure 4.4.

R_{IN} (Ω)	Calculated Q	Measured Q
5000	0.34	5
50	34	38

Table 4.1 displays the my results for three different parallel configurations. What I found was, as predicted by the design step, a lower parallel resistance generated the higher quality filter, so I selected my final resistance of 50 Ω based on these results. I tested an additional configuration, $R = 10\ \Omega$ and found the quality factor to be roughly the same as $R = 50\ \Omega$. However, because the measured series resistance of the inductor was $\sim 1\ \Omega$, the 50 Ω resistor resulted in a lower attenuation of the signal.

4.3 Ambient Noise

As mentioned in the Design Chapter, I faced challenges related to unusually high background noise while testing the quad-copter receiver circuitry. One strong indication of the interference was the tendency of the ambient, background noise level to increase with altitude. Figure 4.5 clearly displays this phenomenon. While this phenomenon does not go away after the addition of a low-pass filter, it is greatly reduced, as shown in Figure 4.6, a similar reading taken in the Law School Quad. One other noticeable aspect of this particular figure is the continued prevalence of strong, intermittent noise in the data, a characteristic found when testing near Harvard's science buildings but not in the residential neighborhood. This points to a further need to filter effectively.

Despite the challenges associated with elevated background noise, I was able to conduct a modified version of the orientation tests what were critical to defining the success of the project.

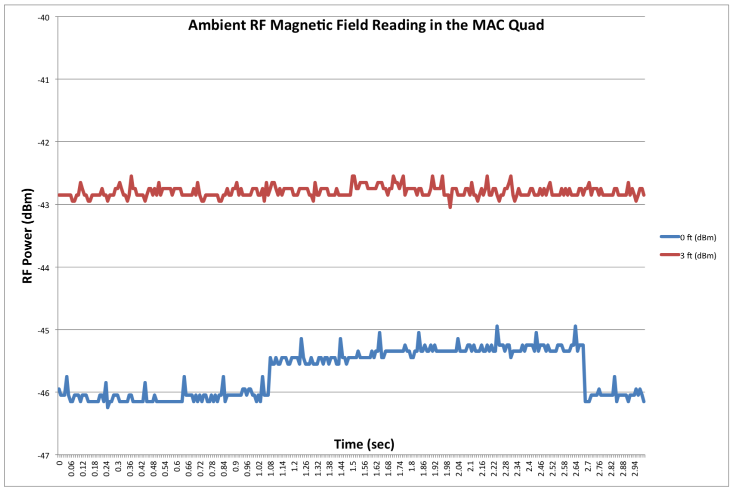


Figure 4.5: Background noise measured over time (no transmitter) as a function of altitude.

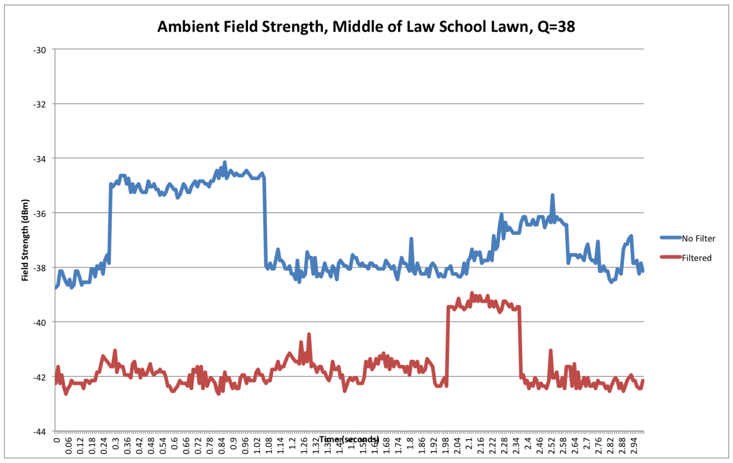


Figure 4.6: Effect of low-pass filter on background noise measured in the Law School Quadrangle.

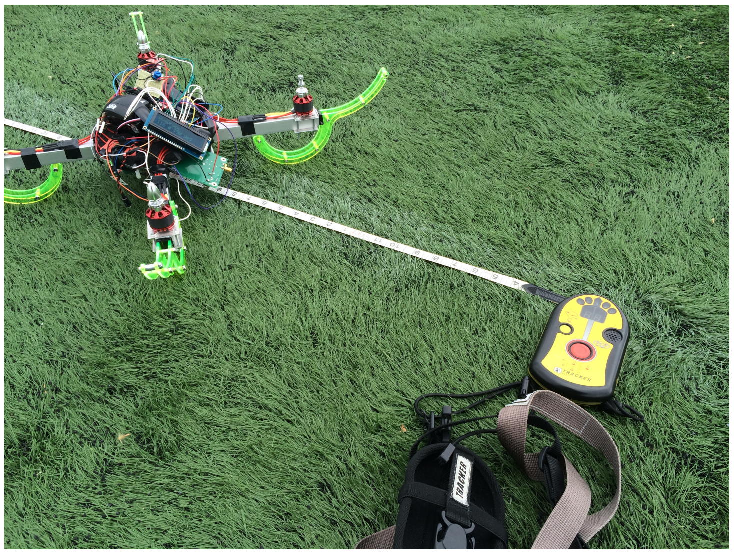


Figure 4.7: The basic arrangement for doing orientation testing. The receiver antenna is 1 ft. from the transmitting beacon.

4.4 Orientation Tests

One of the key results of the project was to demonstrate that the antenna displayed directional sensitivity to a transmitted beacon signal. In order to test this, I transported the quad-copter and all of the testing materials to the Harvard Athletic Fields. As I addressed in Section 4.3, these fields have a much lower noise level than the possible testing areas near the engineering buildings, closer to what would be found in a true backcountry setting. All of the orientation testing was conducted before the addition of the crucial low-pass filter, so noise levels were higher than what would have been observed with the filter.

Because of the high background noise present while testing, the signal became lost in the noise when the transmitter and receiving antenna were only 2 ft. apart. Consequently, all orientation data was collected from a 1 ft. radius around the transmitter. The test setup is depicted in Figures 4.7 and 4.8.

Data were collected by reading signal strength and compass orientation from the LCD display. The location of the receiver was described by a measurement "heading," which is the compass direction from the transmitter

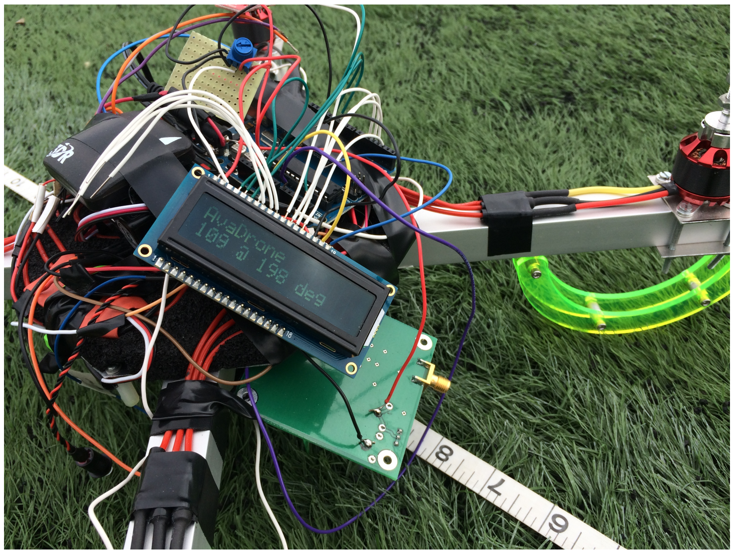


Figure 4.8: The LCD screen attached to the quad-copter reads out signal strength and compass orientation.

to the receiver. At each heading, the compass direction faced by the quad-copter was recorded as the measurement "orientation." 8 samples were taken at each heading, one for each orientation, equally spaced at 45 degrees. The transmitting antenna was oriented at approximately 240 degrees.

Figure 4.9 shows representative data from a single heading. Notice the double peak, characteristic of the fact that the receiving antenna is oriented parallel to the flux lines twice during any given rotation. Data from other headings looks qualitatively similar, though the maximum signal strength varies among different headings.

What is noticeable in this plot, and critical to the success of this project, is the measured difference between the signal maximum and the signal minimum. The difference ranges from 4 dB to 10 dB, but is obvious at all headings. This fact demonstrates very clearly the requirement that the processed signal display directional sensitivity.

In addition to the basic orientation tests, measurements were taken both before and after the switch to a series RLC configuration, so I was able to compare the two topographies and their effects on the signal. Figure 4.10 depicts my results for this comparison. The figure shows that the energy-storing effects of the parallel topography outweighs the signal attenuating

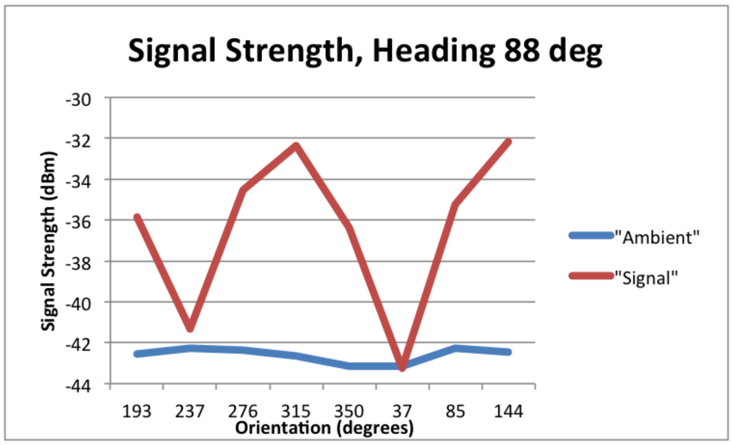


Figure 4.9: Orientation data from a single heading. These data are qualitatively representative of data taken at any other heading.

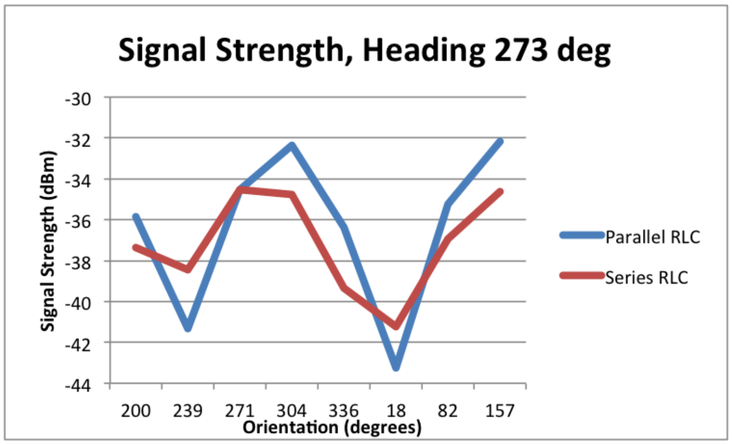


Figure 4.10: This schematic shows one selected test, comparing the parallel and series RLC topologies.

effects I calculated when designing the series topography.

While orientation testing was necessarily conducted against non-ideal background noise levels, I was able to definitively show that this signal processing scheme is capable of giving information about the relative orientation of the quad-copter and the received flux lines.

A complete set of plots for the orientation testing is found in Appendix A.

4.5 Motor Noise

In addition to the central measurement of the orientation sensitivity of the receiver, it was critical do demonstrate that the motors did not produce any detectable noise by the receiver circuitry. This is an entirely unique challenge to this particular combination of quad-copter and avalanche detector, so there was no real way to predict whether it would appear or not.

Early in the design stages I tested to see if a commercial avalanche beacon receiver would register the motor noise as a false signal and I found that it did not. That finding was confirmed during motor noise testing. It was simply not possible to detect any difference in measured signal whether the motors were off or running at full speed.

Future Work

Most of the items that I would like to mention in the Future Work section came up elsewhere in this report. I'll start by mentioning those, and then moving to some of the last-minute considerations that are necessary for a viable product.

The first improvement has to come in the form of a commercial drone that is well-matched between flight controller and body. While the prototype I used was adequate for simple RC-controlled flight, a more robust system is clearly necessary for the sorts of autonomous control that the project requires.

I mention the "necessity" of a commercial solution not because it would be impossible to design a working system from scratch, but rather because this project relies so heavily on next generation functionality of these systems and it makes little sense to reinvent the wheel, so to speak.

While it makes sense to continue the process of prototyping with a commercial product, any eventual, marketable product really does have to be designed more carefully. This step would also be associated with another fundamental change, combining the Flight Controller with the Arduino Navigational Computer.

This step would also require a significant amount of re-working existing code and preparing everything to run simultaneously. The obvious benefit is that at the end of the day one has full control over everything that the quad-copter does.

Up to this point, these projects are obviously multiple-year projects that would likely take a team to develop. However, there are a number of other improvements, largely on the signal processing side, that could be worked on immediately.

Designing a better band-pass system to keep harmful noise out of the receiver is one such design problem. Also important and only moderately more difficult would be to continue testing different antenna geometries for directional sensitivity. For this project, I optimized for signal strength, but other characteristics like directionality could be chosen for and attenuated signals compensated for.

My final concerns and suggestions for future work have not yet been mentioned, but they are related to battery life and frame design. The current battery is likely to last for $\sim$10 min. of flight time. This is probably not sufficient, so one would need to increase carried charge or decrease power consumption. One obvious way to do this is to cut weight. And in fact, this brings me to my final suggestions for work.

This product needs to be portable. It needs to be comparable in size to the gear (beacon, shovel, and probe) that a recreationalist is already carrying. A quad-copter will likely have to collapse in some way to accommodate. This is an entirely separate project that a senior could probably work on for their entire thesis. And while we're making the body portable, the current prototype is strangely non-waterproofed for an outdoor tool. A minor point, but important nonetheless.

There are any number of ways that this project could continue into the future, but the framework that I have described in this document demonstrates that it *can* continue.

Conclusions

Throughout this project, I have focused on the fundamental questions of whether a quad-copter platform can be a viable extension of current avalanche recovery technology. It is a steep road ahead, but what I conclude from the results that I found is that this intersection represents a new and exciting route to help save lives. The tragedy of avalanches in the backcountry is that anyone, regardless of how competent or intelligent, can get swept away. A project like this shows that we can continue to do better.

A homemade frame, though satisfying to work with, is clearly sub-optimal for this sort of work. In order to leverage previous work done by so many other hobbyists, it makes sense to use the right tools for the job. And despite the shortcomings of this particular frame, it was possible to verify the minimum requirements needed to consider this framework a success and worth pursuing.

On the other side of the spectrum, the signal processing that we see in machines today points to the almost limitless possibilities for improved signal detection. With better, smarter filtering and perhaps a different measurement scheme, it should be easy to continue to improve design and performance.

Finally, regarding integration, this project has demonstrated with a resounding show that these two different technologies pose no fundamental threat to one another. There remain many difficulties surrounding the integration step, but the possibility is clear.

Acknowledgments

There are far too many people to thank, all of my mentors and teachers who helped me throughout this process and in everything that led up to it.

I would like to first thank my Project Advisor and Section Leader, Prof. Rob Wood and Prof. Woody Yang. Your guidance throughout this project has been invaluable.

I would also like to thank the ES100 teaching staff, particularly Avi Uttamchandani and Jordan Stephens for their tireless help and patience, and for seemingly putting in more hours into our projects than the rest of us combined.

Thank you to my friends and classmates for your support, insight, and camaraderie. The late nights and early mornings have been all the sweeter for your constant presence.

And finally, thank you to my parents and family. Without your undying support, none of this would have been possible. Thank you to my father for the midnight debugging sessions, to my mother for the midday emotional support, and to my incredible siblings for everything in between.

Bibliography

[1] Colorado Avalanche Information Center. (2014). "Statistics and Reporting," *Colorado Avalanche Information Center* [Online]. Avaliable: http://www.avalanche.state.co.us

[2] P. Haegeli, M. Falk, H. Brugger, H-J. Etter, and J. Boyd, "Comparison of avalanche survival patterns in Canada and Switzerland," *Canadian Med. Assoc. J.*, vol. 183, no. 7, pp. 789-795, Apr. 2011. doi: [10.1503/cmaj.101435]

[3] M. Falk, H. Brugger, and L. Adler-Kastner, "Avalanche Survival Chances," *Avalanche.org*, 2001 [Online]. Available: www.avalanche.org/moonstone/rescue/avalanche%20survival%20chances.htm

[4] D. Chabot. (2009, Nov. 21). "The facts and figures of avalanche fatalities," *Global Rescue News* [Online]. Available: http://www.blog.globalrescue.com

[5] J. Boyd, P. Haegeli, R. Abu-Laban, M. Shuster, and J. Butt, "Patterns of death among avalanche fatalities: a 21-year review," *Canadian Med. Assoc. J.*, vol. 180, no. 5, pp. 507-512, Mar. 2009. doi: [10.1503/cmaj.081327]

[6] "ArduCopter," *APM Copter* [Online]. Available: http://copter.ardupilot.com

[7] "Iris$^+$," *3DRobotics* [Online]. Available: http://www.3drobotics.com

[8] "AARA_Airborne Avalanche Rescue Assistant," *Tatjana Rolle* [Online]. Available: http://tatjanarolle.com

[9] *Delta Drone* [Online]. Available: http://www.deltadrone.com

[10] "alcedo - the flying avalanche transceiver," *ETH Zurich* [Online]. Available: http://www.ion-ch.ch/media/navigare11/pdf/06-Grauwiler_Oth_Navigare2011.pdf

[11] "Pixhawk Autopilot System," *3DRobotics* [Online]. Available: http://www.3drobotics.com

[12] "Mandatory Hardware Configuration," *APM Copter* [Online]. Available: http://copter.ardupilot.com/wiki/initial-setup/configuring-hardware/

[13] S. Hatfield, "The Physics of Quadcopter Flight," *Black Tie Arial* [Online]. Available: http://www.blacktiearial.com

[14] Atmel Corp., "SAM3X Datasheet," [Online]. Available: http://www.atmel.com

[15] Analog Devices Inc., "AD8362 Data Sheet," [Online]. Available: http://www.analog.com

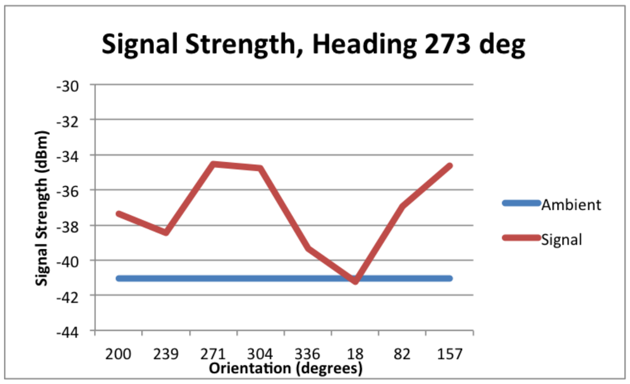
[16] Texas Instruments Corp., "LF412-N Low Offset, Low Drift Dual JFET Input Operational Amplifier," [Online]. Available: www.ti.com

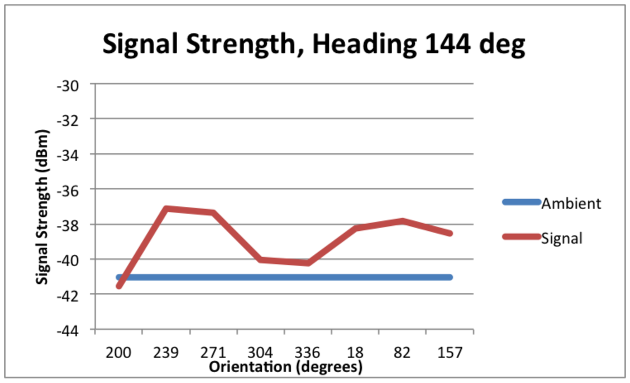
Orientation Data

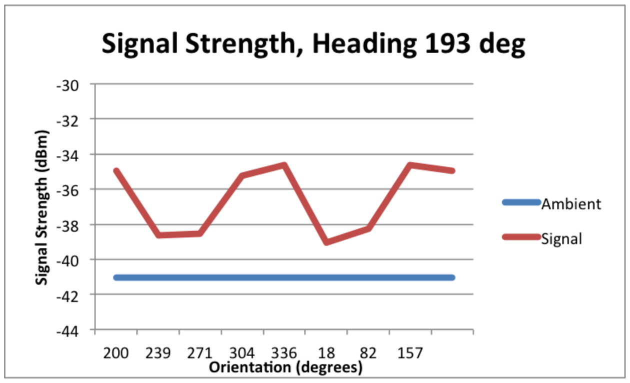
A.1 Athletic Fields, Series Filter

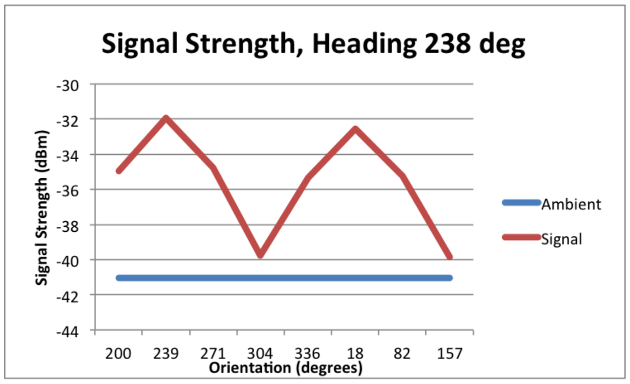
A.2 Athletic Fields, Parallel Filter

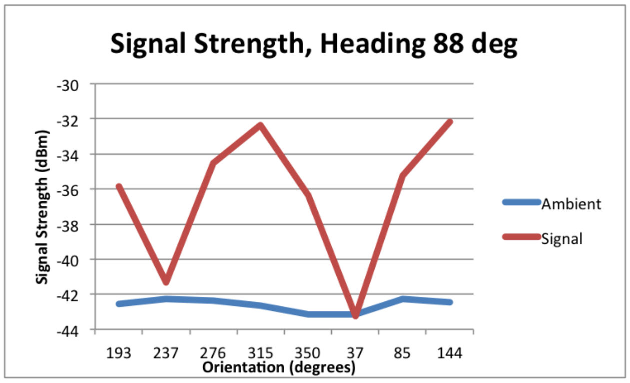
sectionAthletic Fields, Parallel Filter

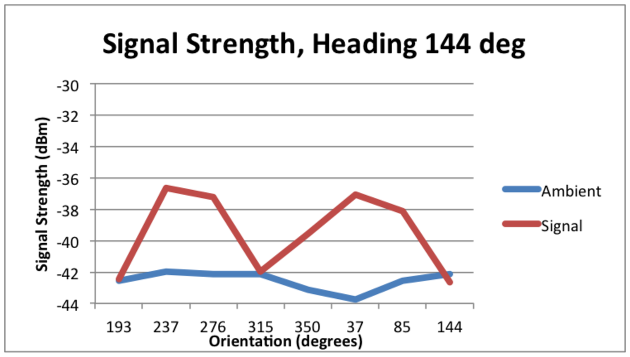


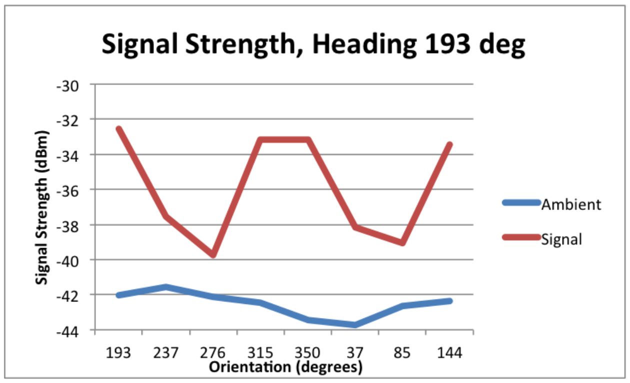

Signal Strength, Heading 144 deg
Signal Strength (dBm)
-30
-32
-34
-36
-38
-40
-42
-44
200
239
271
304
336
18
82
157
Orientation (degrees)
Ambient
Signal

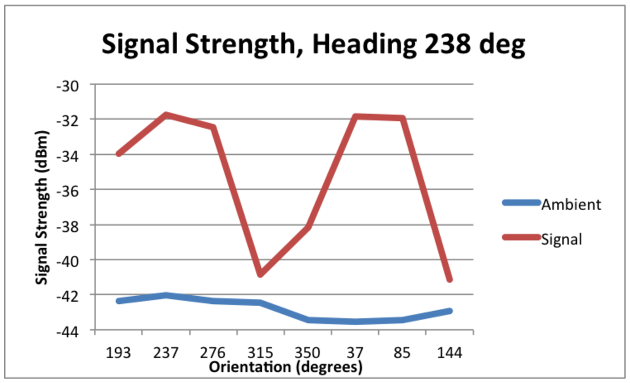

Signal Strength, Heading 193 deg
Signal Strength (dBm)
-30
-32
-34
-36
-38
-40
-42
-44
200
239
271
304
336
18
82
157
Orientation (degrees)
Ambient
Signal

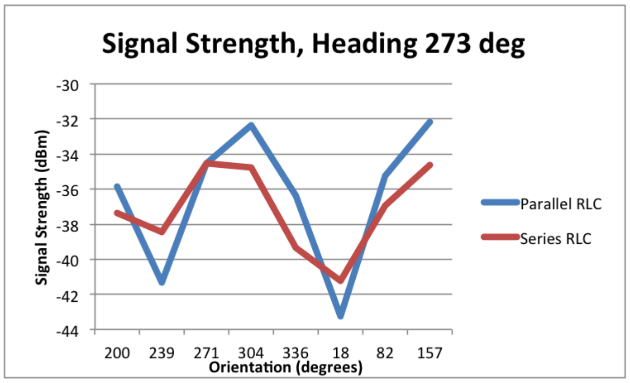

Signal Strength, Heading 238 deg
Signal Strength (dBm)
-30
-32
-34
-36
-38
-40
-42
-44
200
239
271
304
336
18
82
157
Orientation (degrees)
Ambient
Signal

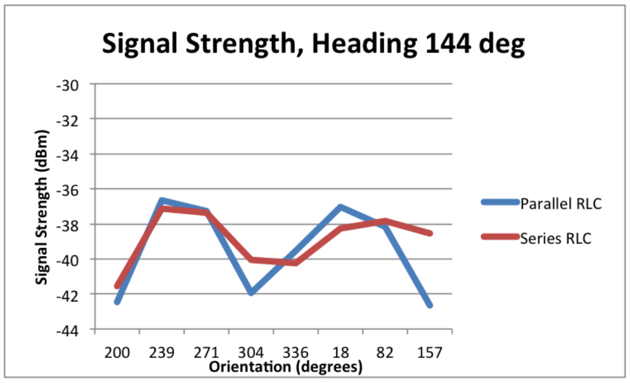

Signal Strength, Heading 88 deg
Signal Strength (dBm)
-30
-32
-34
-36
-38
-40
-42
-44
193
237
276
315
350
37
85
144
Orientation (degrees)
Ambient
Signal

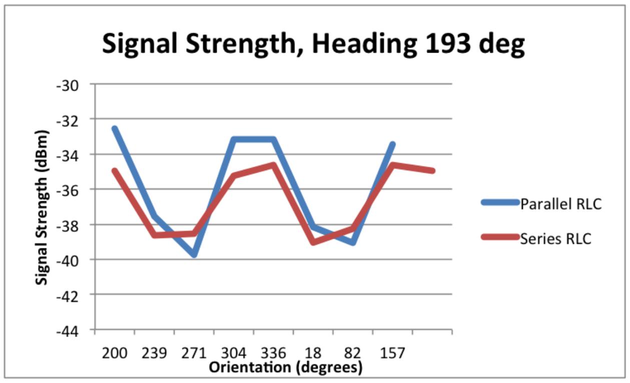

Signal Strength, Heading 144 deg
Signal Strength (dBm)
-30
-32
-34
-36
-38
-40
-42
-44
193
237
276
315
350
37
85
144
Orientation (degrees)
Ambient
Signal

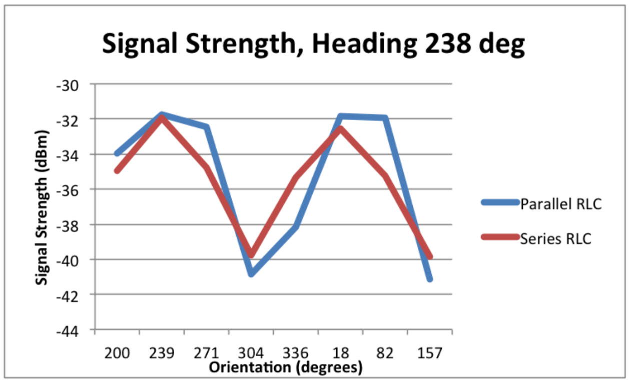

Signal Strength, Heading 193 deg
Signal Strength (dBm)
-30
-32
-34
-36
-38
-40
-42
-44
193
237
276
315
350
37
85
144
Orientation (degrees)
Ambient
Signal


Signal Strength, Heading 238 deg
Signal Strength (dBm)
-30
-32
-34
-36
-38
-40
-42
-44
193
237
276
315
350
37
85
144
Orientation (degrees)
Ambient
Signal


Signal Strength, Heading 273 deg
Signal Strength (dBm)
-30
-32
-34
-36
-38
-40
-42
-44
200
239
271
304
336
18
82
157
Orientation (degrees)
Parallel RLC
Series RLC


Signal Strength, Heading 144 deg
Signal Strength (dBm)
-30
-32
-34
-36
-38
-40
-42
-44
200
239
271
304
336
18
82
157
Orientation (degrees)
Parallel RLC
Series RLC


Signal Strength, Heading 193 deg
-30
-32
-34
-36
-38
-40
-42
-44
Signal Strength (dBm)
200
239
271
304
336
18
82
157
Orientation (degrees)
Parallel RLC
Series RLC


Signal Strength, Heading 238 deg
-30
-32
-34
-36
-38
-40
-42
-44
Signal Strength (dBm)
200
239
271
304
336
18
82
157
Orientation (degrees)
Parallel RLC
Series RLC

Code

B.1 Comm_Test_1.ino

```
/*****************************************************************
* Testing for wireless control using Arduino as a ppm encoder.
*
* This program is adapted from an open source project located at
* https://code.google.com/p/generate-ppm-signal/
*
* Adapted by Ben Dickensheets on 1/4/15 for a senior design
* project for SB degree requirements in Electrical Engineering
* at Harvard University
*
* Arm with throttle low and yaw high
* Disarm with throttle low and yaw low
*****************************************************************/

//////////////////////CONFIGURATION///////////////////////////////
//#include <eRCaGuy_Timer2_counter.h>

#define chanel_number 8  //set the number of chanels
#define default_servo_value 1500  //set the default servo value
#define max_servo_value 1900
#define min_servo_value 1100
#define default_throttle_value 1000 //set the default throttle value
#define max_throttle 1900 //set the max throttle value
#define PPM_FrLen 22500  //set the PPM frame length in microseconds
//(1ms = 1000us)
#define PPM_PulseLen 300  //set the pulse length
#define onState 1  //set polarity of the pulses: 1 is positive, 0 is
  //negative
#define sigPin 10  //set PPM signal output pin on the arduino
```

```
#define ROLL_PIN 6 //RC channel 1
#define PITCH_PIN 7 //RC channel 2
#define THROTTLE_PIN 8 //RC channel 3
#define YAW_PIN 9 //RC channel 4

#define ROLL 0
#define PITCH 1
#define THROTTLE 2
#define YAW 3

const int ROLL_OFFSET = 0;
const int PITCH_OFFSET = 35;
const int THROTTLE_OFFSET = 0;
const int YAW_OFFSET = 0;
//////////////////////////////////////////////////////////////////


/*this array holds the servo values for the ppm signal
 change theese values in your code (usually servo values move between
 1000 and 2000)*/
int ppm[chanel_number];
int throttle = default_throttle_value;
int last_throttle = throttle;
int temp_throttle = throttle;

void setup(){
  //initialize serial communication
  Serial.begin(9600);

  //initiallize default ppm values
  for(int i=0; i<chanel_number; i++){
    ppm[i]= default_servo_value;       // [roll, pitch, throttle,
     // yaw, Radio 5, Radio 6, Radio 7, Radio 8]
  }
  ppm[2] = default_throttle_value;     // low throttle instead of
     // midpoint

  pinMode(sigPin, OUTPUT);
  pinMode(13, OUTPUT);    // led indicator on board
  digitalWrite(sigPin, !onState);  //set the PPM signal pin to the
     //default state (off)

  cli();
  TCCR1A = 0; // set entire TCCR1 register to 0
  TCCR1B = 0;

  OCR1A = 100;  // compare match register, change this
  TCCR1B |= (1 << WGM12);  // turn on CTC mode
  TCCR1B |= (1 << CS11);  // 8 prescaler: 0,5 microseconds at 16mhz
  TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt
  sei();
}

void loop(){
  unsigned long start = micros();
  for(int i=0; i<200; i++){
  if(!(ppm[ROLL] = pulseIn(ROLL_PIN, HIGH) + ROLL_OFFSET)){
    Serial.println("Error: no roll value read.");}
  if(!(ppm[PITCH] = pulseIn(PITCH_PIN, HIGH) + PITCH_OFFSET)){
    Serial.println("Error: no pitch value read.");}
  if(!(throttle = pulseIn(THROTTLE_PIN, HIGH) + THROTTLE_OFFSET)){
    Serial.println("Error: no throttle value read.");}
  if(!(ppm[YAW] = pulseIn(YAW_PIN, HIGH) + YAW_OFFSET)){
    Serial.println("Error: no yaw value read.");}

  if((throttle - last_throttle < 150) && (last_throttle -
   throttle < 150)) {
    ppm[THROTTLE] = throttle;
    last_throttle = throttle;
  }
  else if ((throttle - temp_throttle < 150) && (temp_throttle -
   throttle < 150)) {
    ppm[THROTTLE] = throttle;
    last_throttle = throttle;
  }
  temp_throttle = throttle;
  Serial.print(ppm[0]);
  Serial.print('\t');
  Serial.print(ppm[1]);
  Serial.print('\t');
  Serial.print(ppm[2]);
  Serial.print('\t');
  Serial.println(ppm[3]);
  }
  unsigned long elapsed = micros() - start; //in microseconds
  float avg = ((float) elapsed) / 200 / 1000000; //in seconds

  Serial.print(elapsed);
  Serial.print(" elapsed total; \tAverage time per loop:");
  Serial.println(avg, 5);

  //ppm[4] = 1300; //auto-land after 1 cycle (about 15 seconds)
}

ISR(TIMER1_COMPA_vect){  //leave this alone
  static boolean state = true;

  TCNT1 = 0;

  if(state) {  //start pulse
    digitalWrite(sigPin, onState);
    OCR1A = PPM_PulseLen * 2;
    state = false;
  }
  else{  //end pulse and calculate when to start the next pulse
    static byte cur_chan_numb;
    static unsigned int calc_rest;

    digitalWrite(sigPin, !onState);
    state = true;

    if(cur_chan_numb >= chanel_number){
      cur_chan_numb = 0;
      calc_rest = calc_rest + PPM_PulseLen;//
      OCR1A = (PPM_FrLen - calc_rest) * 2;
      calc_rest = 0;
    }
    else{
      OCR1A = (ppm[cur_chan_numb] - PPM_PulseLen) * 2;
      calc_rest = calc_rest + ppm[cur_chan_numb];
      cur_chan_numb++;
    }
  }
}
```

B.2 Flight_Test_1.ino

```
/*******************************************************************
* This program is adapted from an open source project located at
* https://code.google.com/p/generate-ppm-signal/
*
* Adapted by Ben Dickensheets on 12/18/14 for a senior design
* project for SB degree requirements in Electrical Engineering
* at Harvard University
*
* Arm with throttle low and yaw high
* Disarm with
*******************************************************************/

//////////////////////CONFIGURATION///////////////////////////////
#define chanel_number 8  //set the number of chanels
#define default_servo_value 1500  //set the default servo value
#define max_servo_value 1900
#define min_servo_value 1100
#define default_throttle_value 1100 //set the default throttle value
#define max_throttle 1900 //set the max throttle value
#define PPM_FrLen 22500  //set the PPM frame length in microseconds
//(1ms = 1000us)
#define PPM_PulseLen 300  //set the pulse length
#define onState 1  //set polarity of the pulses: 1 is positive, 0 is
```

```
  //negative
#define sigPin 10  //set PPM signal output pin on the arduino
//////////////////////////////////////////////////////////////////


/*this array holds the servo values for the ppm signal
 change theese values in your code (usually servo values move between
 1000 and 2000)*/
int ppm[chanel_number];
int throttle = default_throttle_value;

void setup(){
  //initialize serial communication
  Serial.begin(9600);

  //initiallize default ppm values
  for(int i=0; i<chanel_number; i++){
    ppm[i]= default_servo_value;       // [roll, pitch, throttle,
     // yaw, Radio 5, Radio 6, Radio 7, Radio 8]
  }
  ppm[2] = default_throttle_value;     // low throttle instead of
     // midpoint

  pinMode(sigPin, OUTPUT);
  pinMode(13, OUTPUT);    // led indicator on board
  digitalWrite(sigPin, !onState);  //set the PPM signal pin to the
     //default state (off)

  cli();
  TCCR1A = 0; // set entire TCCR1 register to 0
  TCCR1B = 0;

  OCR1A = 100;  // compare match register, change this
  TCCR1B |= (1 << WGM12);  // turn on CTC mode
  TCCR1B |= (1 << CS11);  // 8 prescaler: 0,5 microseconds at 16mhz
  TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt
  sei();
}
```

```
void loop(){
  // throttle output
  //Serial.println(throttle);
  ppm[2] = throttle;                  //  update throttle output

  // update throttle level
  if(Serial.available()) {
    int incomingByte = Serial.read();

    // kill switch
    if(incomingByte == 'k') {
      ppm[2] = default_throttle_value;
      delay(2000);
      while(true){}
    }

    // throttle up
    else if(incomingByte == 'u')
      throttle = constrain(throttle + 50, default_throttle_value,
       max_throttle);
    // throttle down
    else if(incomingByte == 'j')
      throttle = constrain(throttle - 50, default_throttle_value,
       max_throttle);

    // arm
    else if(incomingByte == 'z') {
      throttle = default_throttle_value;
      ppm[2] = throttle;
      ppm[3] = max_servo_value;
      delay(2000);
      ppm[3] = default_servo_value;
    }
    //disarm
    else if(incomingByte == 'x') {
      throttle = default_throttle_value;
      ppm[2] = throttle;
      ppm[3] = min_servo_value;
      delay(2000);
      ppm[3] = default_servo_value;
    }
  }

  /*
  digitalWrite(13, HIGH);
  ppm[0] = 1900;                  //  max roll
  delay(3000);
  ppm[0] = 1100;                  //  min roll
  delay(3000);
  ppm[0] = default_servo_value;
  digitalWrite(13, LOW);
  delay(1000);

  digitalWrite(13, HIGH);
  ppm[1] = 1900;                  //  max pitch
  delay(3000);
  ppm[1] = 1100;                  //  min pitch
  delay(3000);
  ppm[1] = default_servo_value;
  digitalWrite(13, LOW);
  delay(1000);

  digitalWrite(13, HIGH);
  ppm[3] = 1900;                  //  max yaw
  delay(3000);
  ppm[3] = 1100;                  //  min yaw
  delay(3000);
  ppm[3] = default_servo_value;
  digitalWrite(13, LOW);
  delay(1000);

 */
}

ISR(TIMER1_COMPA_vect){  //leave this alone
```

```
  static boolean state = true;

  TCNT1 = 0;

  if(state) {  //start pulse
    digitalWrite(sigPin, onState);
    OCR1A = PPM_PulseLen * 2;
    state = false;
  }
  else{  //end pulse and calculate when to start the next pulse
    static byte cur_chan_numb;
    static unsigned int calc_rest;

    digitalWrite(sigPin, !onState);
    state = true;

    if(cur_chan_numb >= chanel_number){
      cur_chan_numb = 0;
      calc_rest = calc_rest + PPM_PulseLen;//
      OCR1A = (PPM_FrLen - calc_rest) * 2;
      calc_rest = 0;
    }
    else{
      OCR1A = (ppm[cur_chan_numb] - PPM_PulseLen) * 2;
      calc_rest = calc_rest + ppm[cur_chan_numb];
      cur_chan_numb++;
    }
  }
}
```

B.3 Flight_Test_2.ino

```
/*****************************************************************
* Modules for autonomous flight
*
* This program is adapted from an open source project located at
* https://code.google.com/p/generate-ppm-signal/
```

```
*
* Adapted by Ben Dickensheets on 1/4/15 for a senior design
* project for SB degree requirements in Electrical Engineering
* at Harvard University
*
* Arm with throttle low and yaw high
* Disarm with throttle low and yaw low
******************************************************************/

//////////////////////CONFIGURATION///////////////////////////////
//#include <eRCaGuy_Timer2_counter.h>

#define chanel_number 8  //set the number of chanels
#define default_servo_value 1500  //set the default servo value
#define max_servo_value 1900
#define min_servo_value 1100
#define default_throttle_value 1000 //set the default throttle value
#define max_throttle 1900 //set the max throttle value
#define PPM_FrLen 22500  //set the PPM frame length in microseconds
//(1ms = 1000us)
#define PPM_PulseLen 300  //set the pulse length
#define onState 1  //set polarity of the pulses: 1 is positive, 0 is
  //negative

#define sigPin 9  //set PPM signal output pin on the arduino

#define LAND_MODE 1300

#define ROLL_PIN 4 //RC channel 1
#define PITCH_PIN 11 //RC channel 2
#define THROTTLE_PIN 6 //RC channel 3
#define YAW_PIN 7 //RC channel 4

#define ROLL 0
#define PITCH 1
#define THROTTLE 2
#define YAW 3
#define FLIGHT_MODE 4
```

```
//////////////////////////////////////////////////////////////////


/*this array holds the servo values for the ppm signal
 change theese values in your code (usually servo values move
 between 1000 and 2000)*/
int ppm[chanel_number];
int throttle = default_throttle_value;

void setup(){
  //initialize serial communication
  Serial.begin(9600);

  //initiallize default ppm values
  for(int i=0; i<chanel_number; i++){
    ppm[i]= default_servo_value;       // [roll, pitch, throttle,
    yaw, Radio 5, Radio 6, Radio 7, Radio 8]
  }
  ppm[2] = default_throttle_value;     // low throttle instead of
     // midpoint

  pinMode(sigPin, OUTPUT);
  pinMode(13, OUTPUT);    // led indicator on board
  digitalWrite(sigPin, !onState);  //set the PPM signal pin to the
     //default state (off)

  cli();
  TCCR1A = 0; // set entire TCCR1 register to 0
  TCCR1B = 0;

  OCR1A = 100;  // compare match register, change this
  TCCR1B |= (1 << WGM12);  // turn on CTC mode
  TCCR1B |= (1 << CS11);  // 8 prescaler: 0,5 microseconds at 16mhz
  TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt
  sei();
}

void loop(){
  fly(75);

  arm();
  delay(3000);

  take_off();
  delay(5000);

  land();
}

ISR(TIMER1_COMPA_vect){  //leave this alone
  static boolean state = true;

  TCNT1 = 0;

  if(state) {  //start pulse
    digitalWrite(sigPin, onState);
    OCR1A = PPM_PulseLen * 2;
    state = false;
  }
  else{  //end pulse and calculate when to start the next pulse
    static byte cur_chan_numb;
    static unsigned int calc_rest;

    digitalWrite(sigPin, !onState);
    state = true;

    if(cur_chan_numb >= chanel_number){
      cur_chan_numb = 0;
      calc_rest = calc_rest + PPM_PulseLen;//
      OCR1A = (PPM_FrLen - calc_rest) * 2;
      calc_rest = 0;
    }
    else{
      OCR1A = (ppm[cur_chan_numb] - PPM_PulseLen) * 2;
      calc_rest = calc_rest + ppm[cur_chan_numb];
      cur_chan_numb++;
    }
  }
}
void fly(int cycles){ // about 5 seconds for 75 cycles
  unsigned long start = micros();
  for(int i=0; i<cycles; i++){ //about 5 seconds
  if(!(ppm[ROLL] = pulseIn(ROLL_PIN, HIGH))){
    Serial.println("Error: no roll value read.");}
  if(!(ppm[PITCH] = pulseIn(PITCH_PIN, HIGH))){
    Serial.println("Error: no pitch value read.");}
  if(!(ppm[THROTTLE] = pulseIn(THROTTLE_PIN, HIGH))){
    Serial.println("Error: no throttle value read.");}
  if(!(ppm[YAW] = pulseIn(YAW_PIN, HIGH))){
    Serial.println("Error: no yaw value read.");}

  Serial.print(ppm[0]);
  Serial.print('\t');
  Serial.print(ppm[1]);
  Serial.print('\t');
  Serial.print(ppm[2]);
  Serial.print('\t');
  Serial.println(ppm[3]);
  }
  unsigned long elapsed = micros() - start; //in microseconds
  float avg = ((float) elapsed) / 200 / 1000000; //in seconds

  Serial.print(elapsed);
  Serial.print(" elapsed total; \tAverage time per loop:");
  Serial.println(avg, 5);
}

void land(){
  ppm[FLIGHT_MODE] = LAND_MODE;

  //initiallize default ppm values
  for(int i=0; i<4; i++){
    ppm[i]= default_servo_value;       // [roll, pitch, throttle,
        // yaw, Radio 5, Radio 6, Radio 7, Radio 8]
  }
  ppm[THROTTLE] = default_throttle_value;     // low throttle
             // instead of midpoint

  delay(30000); // 30 second timeout
  //while(true){}
}

void take_off(){
  ppm[THROTTLE] = 1500;
}

void arm(){
  ppm[THROTTLE] = default_throttle_value;
  ppm[YAW] = max_servo_value;
  delay(3000);
  ppm[YAW] = default_servo_value;
}

void disarm(){
  ppm[THROTTLE] = default_throttle_value;
  ppm[YAW] = min_servo_value;
  delay(3000);
  ppm[YAW] = default_servo_value;
}
```

B.4 Flight_Test_2.ino

```
/*******************************************************************
* Testing for wireless control using Arduino as a ppm encoder.
*
* This test script is for the ultrasonic sensor to maintain a
* stable altitude.
*
* Parts of this program were adapted from an open source project
* located at https://code.google.com/p/generate-ppm-signal/
*
```

```
* Adapted by Ben Dickensheets on 3/17/15 for a senior design
* project for SB degree requirements in Electrical Engineering
* at Harvard University
*
* Arm with throttle low and yaw high
* Disarm with throttle low and yaw low
*****************************************************************/

//////////////////////CONFIGURATION///////////////////////////////
//#include <eRCaGuy_Timer2_counter.h>

#define chanel_number 8  //set the number of chanels
#define default_servo_value 1500  //set the default servo value
#define max_servo_value 1900
#define min_servo_value 1100
#define default_throttle_value 1000 //set the default throttle value
#define max_throttle 1900 //set the max throttle value
#define PPM_FrLen 22500  //set the PPM frame length in microseconds
//(1ms = 1000us)
#define PPM_PulseLen 300  //set the pulse length
#define onState 1  //set polarity of the pulses: 1 is positive, 0 is
  //negative
#define sigPin 9  //set PPM signal output pin on the arduino

#define ROLL_PIN 4 //RC channel 1
#define PITCH_PIN 11 //RC channel 2
#define THROTTLE_PIN 6 //RC channel 3
#define YAW_PIN 7 //RC channel 4

#define ROLL 0
#define PITCH 1
#define THROTTLE 2
#define YAW 3

const int ROLL_OFFSET = 20;
const int PITCH_OFFSET = 35;
const int THROTTLE_OFFSET = 0;
const int YAW_OFFSET = 0;
```

```
// ultrasonic definitions
#define ultraPinIn 8
#define ultraPinOut 5
#define SPEED_OF_SOUND 340 // m/s

const float SET_H = 15; //in cm
const float PROP_H = 0.1;
const int INT_H = 0;
const float DERIV_H = 0.1;
//////////////////////////////////////////////////////////////////


/*this array holds the servo values for the ppm signal
 change theese values in your code (usually servo values move
 between 1000 and 2000)*/
int ppm[chanel_number];
int throttle = default_throttle_value;

//store last height error for derivative calculation
int last_error_h = 0;

void setup(){
  //initialize serial communication
  Serial.begin(9600);

  //initiallize default ppm values
  for(int i=0; i<chanel_number; i++){
    ppm[i]= default_servo_value;       // [roll, pitch, throttle,
     // yaw, Radio 5, Radio 6, Radio 7, Radio 8]
  }
  ppm[2] = default_throttle_value;     // low throttle instead of
     // midpoint

  pinMode(sigPin, OUTPUT);
  pinMode(13, OUTPUT);    // led indicator on board
  digitalWrite(sigPin, !onState);  //set the PPM signal pin to the
     //default state (off)
  cli();
  TCCR1A = 0; // set entire TCCR1 register to 0
  TCCR1B = 0;

  OCR1A = 100;  // compare match register, change this
  TCCR1B |= (1 << WGM12);  // turn on CTC mode
  TCCR1B |= (1 << CS11);  // 8 prescaler: 0,5 microseconds at 16mhz
  TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt
  sei();

  //ultrasonic setup
  pinMode(ultraPinIn, INPUT);
  pinMode(ultraPinOut, OUTPUT);
  digitalWrite(ultraPinOut, LOW);
}

void loop(){
  unsigned long start = micros();
  //unsigned long loop_start_t = start;

  for(int i=0; i<200; i++){
    if(!(ppm[ROLL] = pulseIn(ROLL_PIN, HIGH) + ROLL_OFFSET)){
      Serial.println("Error: no roll value read.");}
    if(!(ppm[PITCH] = pulseIn(PITCH_PIN, HIGH) + PITCH_OFFSET)){
      Serial.println("Error: no pitch value read.");}
    if(!(ppm[YAW] = pulseIn(YAW_PIN, HIGH) + YAW_OFFSET)){
      Serial.println("Error: no yaw value read.");}

    if(!(throttle = pulseIn(THROTTLE_PIN, HIGH) + THROTTLE_OFFSET)){
      Serial.println("Error: no throttle value read.");}
    if(throttle < 1600)
      ppm[THROTTLE] = throttle;
    else {
      float height = getHeight();
      float error_h = SET_H - height;
      ppm[THROTTLE] = constrain(ppm[THROTTLE] + (int) (PROP_H *
      error_h), 1100, 1900);// +
      //  (int) (DERIV_H * (error_h - last_error_h)), 1100, 1900);
      last_error_h = error_h;
      /*unsigned long temp_t = micros();
      float dt = ((float) (temp_t - loop_start_t)) / 1000000;
       //in seconds
      loop_start_t = temp_t;*/
    }

    Serial.print(ppm[0]);
    Serial.print('\t');
    Serial.print(ppm[1]);
    Serial.print('\t');
    Serial.print(ppm[2]);
    Serial.print('\t');
    Serial.print(ppm[3]);
    if(ppm[2] < 1200){
      Serial.print('\t');
      Serial.print("Oops!");
    }
    Serial.print('\n');
  }
  /*unsigned long elapsed = micros() - start; //in microseconds
  float avg = ((float) elapsed) / 200 / 1000000; //in seconds

  Serial.print(elapsed);
  Serial.print(" elapsed total; \tAverage time per loop:");
  Serial.println(avg, 5);*/

  //ppm[4] = 1300; //auto-land after 1 cycle (about 15 seconds)
}

ISR(TIMER1_COMPA_vect){  //leave this alone
  static boolean state = true;

  TCNT1 = 0;

  if(state) {  //start pulse
    digitalWrite(sigPin, onState);
    OCR1A = PPM_PulseLen * 2;
    state = false;
  }
  else{  //end pulse and calculate when to start the next pulse
    static byte cur_chan_numb;
    static unsigned int calc_rest;

    digitalWrite(sigPin, !onState);
    state = true;

    if(cur_chan_numb >= chanel_number){
      cur_chan_numb = 0;
      calc_rest = calc_rest + PPM_PulseLen;//
      OCR1A = (PPM_FrLen - calc_rest) * 2;
      calc_rest = 0;
    }
    else{
      OCR1A = (ppm[cur_chan_numb] - PPM_PulseLen) * 2;
      calc_rest = calc_rest + ppm[cur_chan_numb];
      cur_chan_numb++;
    }
  }
}


// returns height from ultrasonic rangefinder in cm
int getHeight(void){
  digitalWrite(ultraPinOut, HIGH);
  delayMicroseconds(15);
  digitalWrite(ultraPinOut, LOW);
  float raw = (float) pulseIn(ultraPinIn, HIGH);
   // returns microseconds
  // microseconds->s, m->cm, corrects for constant 2cm offset
  return (int) (raw * SPEED_OF_SOUND * 100 / 2 / 1000000) - 2;
}
```

B.5 Motor_Test.ino

```
#include <Servo.h>

#define FRONT 4  //CW
#define BACK 6   //CW
#define LEFT 5   //CCW
#define RIGHT 7  //CCW

Servo front;
Servo back;
Servo left;
Servo right;

void setup() {
  front.attach(FRONT);
  back.attach(BACK);
  left.attach(LEFT);
  right.attach(RIGHT);

  allOff();
  delay(4000);
  frontOn(1300);
  delay(2000);
  allOff();
  delay(1000);
  rightOn(1300);
  delay(2000);
  allOff();
  delay(1000);
  backOn(1300);
  delay(2000);
  allOff();
  delay(1000);
  leftOn(1300);
  delay(2000);
  allOff();
  delay(1000);
  allOn(1400);
  delay(10000);
  allOff();
}

void loop() {}

int frontOn(int setpoint) {
  front.writeMicroseconds(setpoint);
  return 0;
}

int backOn(int setpoint) {
  back.writeMicroseconds(setpoint);
  return 0;
}

int leftOn(int setpoint) {
  left.writeMicroseconds(setpoint);
  return 0;
}

int rightOn(int setpoint) {
  right.writeMicroseconds(setpoint);
  return 0;
}

int allOn(int setpoint) {
  frontOn(setpoint);
  backOn(setpoint);
  leftOn(setpoint);
  rightOn(setpoint);
  return 0;
}

int allOff(void) {
  allOn(1000);
  return 0;
}
```

B.6 Receiver_Config.ino

```
/*****************************************************************
* This program is adapted from an open source project located at
* https://code.google.com/p/generate-ppm-signal/
*
* Adapted by Ben Dickensheets on 11/29/14 for a senior design
* project for SB degree requirements in Electrical Engineering
* at Harvard University
*
* Arm with throttle low and yaw high
* Disarm with
*****************************************************************/

//////////////////////CONFIGURATION///////////////////////////////
#define chanel_number 8  //set the number of chanels
#define default_servo_value 1500  //set the default servo value
#define default_throttle_value 1100 //set the default throttle value
#define PPM_FrLen 22500  //set the PPM frame length in microseconds
//(1ms = 1000us)
#define PPM_PulseLen 300  //set the pulse length
#define onState 1  //set polarity of the pulses: 1 is positive, 0 is
  //negative
#define sigPin 10  //set PPM signal output pin on the arduino
//////////////////////////////////////////////////////////////////


/*this array holds the servo values for the ppm signal
 change theese values in your code (usually servo values move
 between 1000 and 2000)*/
int ppm[chanel_number];

void setup(){
  //initiallize default ppm values
  for(int i=0; i<chanel_number; i++){
    ppm[i]= default_servo_value;       // [roll, pitch, throttle,
     // yaw, Radio 5, Radio 6, Radio 7, Radio 8]
  }
  ppm[2] = default_throttle_value;     // low throttle instead of
     // midpoint

  pinMode(sigPin, OUTPUT);
  pinMode(13, OUTPUT);    // led indicator on board
  digitalWrite(sigPin, !onState);  //set the PPM signal pin to the
     //default state (off)

  cli();
  TCCR1A = 0; // set entire TCCR1 register to 0
  TCCR1B = 0;

  OCR1A = 100;  // compare match register, change this
  TCCR1B |= (1 << WGM12);  // turn on CTC mode
  TCCR1B |= (1 << CS11);  // 8 prescaler: 0,5 microseconds at 16mhz
  TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt
  sei();
}

void loop(){
  //put main code here
  digitalWrite(13, LOW);
  delay(2000);                    // wait a couple of seconds to
    // start config

  digitalWrite(13, HIGH);
  ppm[2] = 1900;                  //  max throttle
  delay(3000);
  ppm[2] = default_throttle_value;   //  min throttle
  digitalWrite(13, LOW);
  delay(1000);

  digitalWrite(13, HIGH);
  ppm[0] = 1900;                  //  max roll
  delay(3000);
  ppm[0] = 1100;                  //  min roll
  delay(3000);
  ppm[0] = default_servo_value;
  digitalWrite(13, LOW);
  delay(1000);

  digitalWrite(13, HIGH);
  ppm[1] = 1900;                  //  max pitch
  delay(3000);
  ppm[1] = 1100;                  //  min pitch
  delay(3000);
  ppm[1] = default_servo_value;
  digitalWrite(13, LOW);
  delay(1000);

  digitalWrite(13, HIGH);
  ppm[3] = 1900;                  //  max yaw
  delay(3000);
  ppm[3] = 1100;                  //  min yaw
  delay(3000);
  ppm[3] = default_servo_value;
  digitalWrite(13, LOW);
  delay(1000);

  while(true) {}
}

ISR(TIMER1_COMPA_vect){  //leave this alone
  static boolean state = true;

  TCNT1 = 0;

  if(state) {  //start pulse
    digitalWrite(sigPin, onState);
    OCR1A = PPM_PulseLen * 2;
    state = false;
  }
  else{  //end pulse and calculate when to start the next pulse
    static byte cur_chan_numb;
    static unsigned int calc_rest;

    digitalWrite(sigPin, !onState);
    state = true;

    if(cur_chan_numb >= chanel_number){
      cur_chan_numb = 0;
      calc_rest = calc_rest + PPM_PulseLen;//
      OCR1A = (PPM_FrLen - calc_rest) * 2;
      calc_rest = 0;
    }
    else{
      OCR1A = (ppm[cur_chan_numb] - PPM_PulseLen) * 2;
      calc_rest = calc_rest + ppm[cur_chan_numb];
      cur_chan_numb++;
    }
  }
}
```

B.7 Sig_and_Motor_Test.ino

```
/*****************************************************************
* Testing for wireless control using Arduino as a ppm encoder.
*
* This program is adapted from an open source project located at
* https://code.google.com/p/generate-ppm-signal/
*
* Adapted by Ben Dickensheets on 1/4/15 for a senior design
* project for SB degree requirements in Electrical Engineering
* at Harvard University
*
* Arm with throttle low and yaw high
* Disarm with throttle low and yaw low
*****************************************************************/
```

```
//////////////////////MOTOR CONFIGURATION/////////////////////////
//#include <eRCaGuy_Timer2_counter.h>

#define chanel_number 8  //set the number of chanels
#define default_servo_value 1500  //set the default servo value
#define max_servo_value 1900
#define min_servo_value 1100
#define default_throttle_value 1000 //set the default throttle value
#define max_throttle 1900 //set the max throttle value
#define PPM_FrLen 22500  //set the PPM frame length in microseconds
//(1ms = 1000us)
#define PPM_PulseLen 300  //set the pulse length
#define onState 1  //set polarity of the pulses: 1 is positive, 0 is
  //negative
#define sigPin 10  //set PPM signal output pin on the arduino

#define ROLL_PIN 6 //RC channel 1
#define PITCH_PIN 7 //RC channel 2
#define THROTTLE_PIN 8 //RC channel 3
#define YAW_PIN 9 //RC channel 4

#define ROLL 0
#define PITCH 1
#define THROTTLE 2
#define YAW 3

const int ROLL_OFFSET = 0;
const int PITCH_OFFSET = 35;
const int THROTTLE_OFFSET = 0;
const int YAW_OFFSET = 0;

/*this array holds the servo values for the ppm signal
 change theese values in your code (usually servo values move between
 1000 and 2000)*/
int ppm[chanel_number];
int throttle = default_throttle_value;
int last_throttle = throttle;
int temp_throttle = throttle;
```

```
////////////////////////LCD CONFIGURATION/////////////////////////
#include <LiquidCrystal.h>
#include <Wire.h>
#include <HMC5883L.h>

// Store our compass as a variable.
HMC5883L compass;
// Record any errors that may occur in the compass.
int error = 0;

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);


//////////////////////SIGNAL CONFIGURATION////////////////////////
const int antennaPin = A0;
int max_sample = 0;



void setup(){
  //initialize serial communication
  Serial.begin(9600);

  //initiallize default ppm values
  for(int i=0; i<chanel_number; i++){
    ppm[i]= default_servo_value;       // [roll, pitch, throttle,
     //yaw, Radio 5, Radio 6, Radio 7, Radio 8]
  }
  ppm[2] = default_throttle_value;     // low throttle instead of
     // midpoint

  pinMode(sigPin, OUTPUT);
  pinMode(13, OUTPUT);    // led indicator on board
  digitalWrite(sigPin, !onState);  //set the PPM signal pin to the
     //default state (off)
  cli();
  TCCR1A = 0; // set entire TCCR1 register to 0
  TCCR1B = 0;

  OCR1A = 100;  // compare match register, change this
  TCCR1B |= (1 << WGM12);  // turn on CTC mode
  TCCR1B |= (1 << CS11);  // 8 prescaler: 0,5 microseconds at 16mhz
  TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt
  sei();

  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("AvaDrone");

  // Digital compass setup
  Wire.begin(); // Start the I2C interface.
  compass = HMC5883L(); // Construct a new HMC5883 compass.

  error = compass.SetScale(1.3); // Set the scale of the compass.
  if(error != 0) // If there is an error, print it out.
    Serial.println(compass.GetErrorText(error));

  error = compass.SetMeasurementMode(Measurement_Continuous);
  // Set the measurement mode to Continuous

  if(error != 0) // If there is an error, print it out.
    Serial.println(compass.GetErrorText(error));

  lcd.clear();
  lcd.print("AvaDrone");

}

void loop(){
  unsigned long start = micros();
    if(!(ppm[ROLL] = pulseIn(ROLL_PIN, HIGH) + ROLL_OFFSET)){
      Serial.println("Error: no roll value read.");}
    if(!(ppm[PITCH] = pulseIn(PITCH_PIN, HIGH) + PITCH_OFFSET)){
      Serial.println("Error: no pitch value read.");}
    if(!(throttle = pulseIn(THROTTLE_PIN, HIGH) + THROTTLE_OFFSET)){
      Serial.println("Error: no throttle value read.");}
    if(!(ppm[YAW] = pulseIn(YAW_PIN, HIGH) + YAW_OFFSET)){
      Serial.println("Error: no yaw value read.");}

    if((throttle - last_throttle < 150) && (last_throttle -
     throttle < 150)) {
      ppm[THROTTLE] = throttle;
      last_throttle = throttle;
    }
    else if ((throttle - temp_throttle < 150) && (temp_throttle -
     throttle < 150)) {
      ppm[THROTTLE] = throttle;
      last_throttle = throttle;
    }
    temp_throttle = throttle;

    /*Serial.print(ppm[0]);
    Serial.print('\t');
    Serial.print(ppm[1]);
    Serial.print('\t');
    Serial.print(ppm[2]);
    Serial.print('\t');
    Serial.println(ppm[3]);*/

  unsigned long elapsed = micros() - start; //in microseconds
  float avg = ((float) elapsed) / 200 / 1000000; //in seconds

  /*Serial.print(elapsed);
  Serial.print(" elapsed total; \tAverage time per loop:");
  Serial.println(avg, 5);*/

  //ppm[4] = 1300; //auto-land after 1 cycle (about 15 seconds)
```

```
  //low-pass filter
  int new_sample = 0;
  int avg_num = 10;
  for(int j=0; j<avg_num; j++)
    new_sample += analogRead(antennaPin);
  new_sample /= avg_num;

  //comment out if using low-pass filter
  //int new_sample = analogRead(sigpin);

  //track high signal level
  if(new_sample > max_sample) {
    max_sample = new_sample;
    float heading = readCompass();
    lcd.setCursor(0,1);
    if(max_sample < 100)
      lcd.print(' ');
    lcd.print(max_sample);
    lcd.print(" @ ");
    lcd.print((int) heading);
    lcd.print(" deg");

    Serial.print(max_sample);
    Serial.print('\t');
    Serial.println(heading);
  }
  max_sample = 0;

}



ISR(TIMER1_COMPA_vect){  //leave this alone
  static boolean state = true;

  TCNT1 = 0;
  if(state) {  //start pulse
    digitalWrite(sigPin, onState);
    OCR1A = PPM_PulseLen * 2;
    state = false;
  }
  else{  //end pulse and calculate when to start the next pulse
    static byte cur_chan_numb;
    static unsigned int calc_rest;

    digitalWrite(sigPin, !onState);
    state = true;

    if(cur_chan_numb >= chanel_number){
      cur_chan_numb = 0;
      calc_rest = calc_rest + PPM_PulseLen;//
      OCR1A = (PPM_FrLen - calc_rest) * 2;
      calc_rest = 0;
    }
    else{
      OCR1A = (ppm[cur_chan_numb] - PPM_PulseLen) * 2;
      calc_rest = calc_rest + ppm[cur_chan_numb];
      cur_chan_numb++;
    }
  }
}



float readCompass(void){
   // Retrived the scaled values from the compass
   // (scaled to the configured scale).
  MagnetometerScaled scaled = compass.ReadScaledAxis();

    // Calculate heading when the magnetometer is level,
    // then correct for signs of axis.
  float heading = atan2(scaled.YAxis, scaled.XAxis);

  // Once you have your heading, you must then add your
  // 'Declination Angle', which is the 'Error' of the
  // magnetic field in your location.
  // Find yours here: http://www.magnetic-declination.com/
  // Mine is: 2deg 37' W, which is 2.617 Degrees, or
  // (which we need) 0.0456752665 radians, I will use 0.0457
  // If you cannot find your Declination, comment out these
  // two lines, your compass will be slightly off.
  float declinationAngle = 0.0457;
  heading += declinationAngle;

  // Correct for when signs are reversed.
  if(heading < 0)
    heading += 2*PI;

  // Check for wrap due to addition of declination.
  if(heading > 2*PI)
    heading -= 2*PI;

  // Convert radians to degrees for readability.
  float headingDegrees = heading * 180/M_PI;

  return headingDegrees;
}
```

B.8 Sig_Test_1.ino

```
/*****************************************************************
* Test script for transceiver signal acquisition.
*
* Live time update, output on serial monitor.
*
* Written by Ben Dickensheets on 3/19/15 for a senior design
* project for SB degree requirements in Electrical Engineering
* at Harvard University
*
* Written for Arduino Uno.
*****************************************************************/
```

```
const int sigpin = A0;
const long samples = 500;
int signal_array[samples];

void setup(){
  Serial.begin(9600);
}

void loop(){
  for(int i=0; i<samples; i++) {
    signal_array[i] = analogRead(sigpin);
    Serial.println(signal_array[i]);
  }
}
```

B.9 Sig_Test_2.ino

```
/*******************************************************************
* Test script for transceiver signal acquisition.
*
* Tracks maximum value received and outputs this value over serial.
*
* Written by Ben Dickensheets on 3/19/15 for a senior design
* project for SB degree requirements in Electrical Engineering
* at Harvard University
*
* Written for Arduino Uno.
*******************************************************************/
#include <LiquidCrystal.h>

/*
  LiquidCrystal Library - Hello World

 Demonstrates the use a 16x2 LCD display.  The LiquidCrystal
 library works with all LCD displays that are compatible with the
 Hitachi HD44780 driver. There are many of them out there, and you
 can usually tell them by the 16-pin interface.

 This sketch prints "Hello World!" to the LCD
 and shows the time.

  The circuit:
 * LCD RS pin to digital pin 12
 * LCD Enable pin to digital pin 11
 * LCD D4 pin to digital pin 5
 * LCD D5 pin to digital pin 4
 * LCD D6 pin to digital pin 3
 * LCD D7 pin to digital pin 2
 * LCD R/W pin to ground
 * 10K resistor:
 * ends to +5V and ground
 * wiper to LCD VO pin (pin 3)

 Library originally added 18 Apr 2008
 by David A. Mellis
 library modified 5 Jul 2009
 by Limor Fried (http://www.ladyada.net)
 example added 9 Jul 2009
 by Tom Igoe
 modified 22 Nov 2010
 by Tom Igoe

 This example code is in the public domain.

 http://www.arduino.cc/en/Tutorial/LiquidCrystal
 */

// include the library code:
#include <LiquidCrystal.h>
#include <Wire.h>
#include <HMC5883L.h>

// Store our compass as a variable.
HMC5883L compass;
```

```
// Record any errors that may occur in the compass.
int error = 0;

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

const int sigpin = A0;
int max_sample = 0;

void setup(){
  Serial.begin(9600);

  // set up the LCD’s number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("AvaDrone");

  // Digital compass setup
  Wire.begin(); // Start the I2C interface.
  compass = HMC5883L(); // Construct a new HMC5883 compass.

  error = compass.SetScale(1.3); // Set the scale of the compass.
  if(error != 0) // If there is an error, print it out.
    Serial.println(compass.GetErrorText(error));

  error = compass.SetMeasurementMode(Measurement_Continuous);
  // Set the measurement mode to Continuous
  if(error != 0) // If there is an error, print it out.
    Serial.println(compass.GetErrorText(error));
}

void loop(){
  for(int i=0; i<1000; i++){


    //low-pass filter
    int new_sample = 0;
    int avg_num = 10;
    for(int j=0; j<avg_num; j++)
      new_sample += analogRead(sigpin);
    new_sample /= avg_num;

    //comment out if using low-pass filter
    //int new_sample = analogRead(sigpin);

    //track high signal level
    if(new_sample > max_sample) {
      max_sample = new_sample;
      float heading = readCompass();
      lcd.setCursor(0,1);
      if(max_sample < 100)
        lcd.print(' ');
      lcd.print(max_sample);
      lcd.print(" @ ");
      lcd.print((int) heading);
      lcd.print(" deg");

      Serial.print(max_sample);
      Serial.print('\t');
      Serial.println(heading);
    }
  }
  max_sample = 0;
  // set the cursor to column 0, line 1
  // (note: line 1 is the second row, since counting begins with 0):
  //lcd.setCursor(0, 1);
  // print the number of seconds since reset:
  //lcd.print(millis()/1000);
}

float readCompass(void){
   // Retrived the scaled values from the compass
   // (scaled to the configured scale).
  MagnetometerScaled scaled = compass.ReadScaledAxis();

    // Calculate heading when the magnetometer is level,
    // then correct for signs of axis.
  float heading = atan2(scaled.YAxis, scaled.XAxis);

  // Once you have your heading, you must then add your
  // 'Declination Angle', which is the 'Error' of the
  // magnetic field in your location.
  // Find yours here: http://www.magnetic-declination.com/
  // Mine is: 2deg 37' W, which is 2.617 Degrees, or (which
  // we need) 0.0456752665 radians, I will use 0.0457
  // If you cannot find your Declination, comment out these
  // two lines, your compass will be slightly off.
  float declinationAngle = 0.0457;
  heading += declinationAngle;

  // Correct for when signs are reversed.
  if(heading < 0)
    heading += 2*PI;

  // Check for wrap due to addition of declination.
  if(heading > 2*PI)
    heading -= 2*PI;

  // Convert radians to degrees for readability.
  float headingDegrees = heading * 180/M_PI;

  return headingDegrees;
}
```

B.10 Sig_Test_3.ino

```
/*****************************************************************
* Test script for transceiver signal acquisition.
*
* Takes 2 time-series samples, then continuously displays them over
* serial.
*
* Testing for strange observed differences between signal at ground
* level and at 1 foot (or higher) elevation.
*
* Written by Ben Dickensheets on 3/20/15 for a senior design
* project for SB degree requirements in Electrical Engineering
* at Harvard University
*
* Written for Arduino Uno.
*******************************************************************/
#include <LiquidCrystal.h>

/*
  LiquidCrystal Library - Hello World

 Demonstrates the use a 16x2 LCD display.  The LiquidCrystal
 library works with all LCD displays that are compatible with the
 Hitachi HD44780 driver. There are many of them out there, and you
 can usually tell them by the 16-pin interface.

 This sketch prints "Hello World!" to the LCD
 and shows the time.

  The circuit:
 * LCD RS pin to digital pin 12
 * LCD Enable pin to digital pin 11
 * LCD D4 pin to digital pin 5
 * LCD D5 pin to digital pin 4
 * LCD D6 pin to digital pin 3
 * LCD D7 pin to digital pin 2
 * LCD R/W pin to ground
 * 10K resistor:
 * ends to +5V and ground
 * wiper to LCD VO pin (pin 3)

 Library originally added 18 Apr 2008
 by David A. Mellis
 library modified 5 Jul 2009
 by Limor Fried (http://www.ladyada.net)
 example added 9 Jul 2009
 by Tom Igoe
 modified 22 Nov 2010
 by Tom Igoe

 This example code is in the public domain.

 http://www.arduino.cc/en/Tutorial/LiquidCrystal
 */

// include the library code:
#include <LiquidCrystal.h>
#include <Wire.h>
#include <HMC5883L.h>

// Store our compass as a variable.
HMC5883L compass;
// Record any errors that may occur in the compass.
int error = 0;

// initialize the library with the numbers of the interface pins
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);

const int sigpin = A0;
const int sample_size = 300;
int series_1[sample_size];
int series_2[sample_size];

void setup(){
  Serial.begin(9600);

  // set up the LCD's number of columns and rows:
  lcd.begin(16, 2);
  // Print a message to the LCD.
  lcd.print("AvaDrone");

  // Digital compass setup
  Wire.begin(); // Start the I2C interface.
  compass = HMC5883L(); // Construct a new HMC5883 compass.
```

```
error = compass.SetScale(1.3); // Set the scale of the compass.
if(error != 0) // If there is an error, print it out.
  Serial.println(compass.GetErrorText(error));

error = compass.SetMeasurementMode(Measurement_Continuous);
// Set the measurement mode to Continuous
if(error != 0) // If there is an error, print it out.
  Serial.println(compass.GetErrorText(error));

//test statement
Serial.println("Reset :(");

// Begin countdown until first sample series
for(int i=5; i>-1; i--) {
  lcd.setCursor(0,1);
  lcd.print("Series 1 in ");
  lcd.print(i);
  delay(1000);
}
lcd.setCursor(0,1);
lcd.print("Reading Series 1");

// Store data points for Series 1
for(int i=0; i<sample_size; i++) {
  series_1[i] = analogRead(sigpin);
  delay(10);
}

// Begin countdown until second sample series
for(int i=5; i>-1; i--) {
  lcd.clear();
  lcd.print("AvaDrone");
  lcd.setCursor(0,1);
  lcd.print("Series 2 in ");
  lcd.print(i);
  delay(1000);
}
```

```
  lcd.setCursor(0,1);
  lcd.print("Reading Series 2");

  // Store data points for Series 2
  for(int i=0; i<sample_size; i++) {
    series_2[i] = analogRead(sigpin);
    delay(10);
  }

  lcd.clear();
  lcd.print("AvaDrone");
  lcd.setCursor(0,1);
  lcd.print("Printing data");

}

void loop(){
  Serial.println("\nSeries 1:");
  for(int i=0; i<sample_size; i++)
    Serial.println(series_1[i]);
  Serial.println("\nSeries 2:");
  for(int i=0; i<sample_size; i++)
    Serial.println(series_2[i]);
}

float readCompass(void){
   // Retrived the scaled values from the compass
   // (scaled to the configured scale).
  MagnetometerScaled scaled = compass.ReadScaledAxis();

    // Calculate heading when the magnetometer is level,
    // then correct for signs of axis.
  float heading = atan2(scaled.YAxis, scaled.XAxis);

  // Once you have your heading, you must then add your
  // 'Declination Angle', which is the 'Error' of the
  // magnetic field in your location.
  // Find yours here: http://www.magnetic-declination.com/
  // Mine is: 2deg 37' W, which is 2.617 Degrees, or
  // (which we need) 0.0456752665 radians, I will use 0.0457
  // If you cannot find your Declination, comment out
  // these two lines, your compass will be slightly off.
  float declinationAngle = 0.0457;
  heading += declinationAngle;

  // Correct for when signs are reversed.
  if(heading < 0)
    heading += 2*PI;

  // Check for wrap due to addition of declination.
  if(heading > 2*PI)
    heading -= 2*PI;

  // Convert radians to degrees for readability.
  float headingDegrees = heading * 180/M_PI;

  return headingDegrees;
}
```

B.11 Sig_Test_Due.ino

```
const int sigpin = A0;
const long samples = 10000;
int signal_array[samples];

void setup(){
  SerialUSB.begin(9600);
}

void loop(){
  for(int i=0; i<samples; i++) {
    signal_array[i] = analogRead(sigpin);
    SerialUSB.println(signal_array[i]);
  }
}
```

B.12 Ultrasonic_Test.ino

```
/*****************************************************************
* Testing for an ultrasonic rangefinder
*
* Written by Ben Dickensheets on 3/17/15 for a senior design
* project for SB degree requirements in Electrical Engineering
* at Harvard University
*
* Rangefinder should be given +5V at pin V_cc.
*****************************************************************/

#define ultraPinIn 8
#define ultraPinOut 5
#define SPEED_OF_SOUND 340 // m/s

void setup() {
  pinMode(ultraPinIn, INPUT);
  pinMode(ultraPinOut, OUTPUT);
  digitalWrite(ultraPinOut, LOW);

  Serial.begin(9600);
}

void loop() {
  unsigned long start_time = micros();
  for(int i=0; i<10000; i++){
    int height = getHeight();
  }
  unsigned long stop_time = micros();
  Serial.println((stop_time - start_time)/10000);
  delay(10);
}

// returns height from ultrasonic rangefinder in cm
int getHeight(void){
  digitalWrite(ultraPinOut, HIGH);
  delayMicroseconds(15);
  digitalWrite(ultraPinOut, LOW);
  float raw = (float) pulseIn(ultraPinIn, HIGH); // returns microseconds
  return (int) (raw * SPEED_OF_SOUND * 100 / 2 / 1000000);
  // microseconds->s, m->cm
}
```