Design of peripheral devices to augment use of unmanned aerial systems in agriculture

by

Phillip Dix

B.S., Kansas State University, 2012

A THESIS

submitted in partial fulfillment of the
requirements for the degree

MASTER OF SCIENCE

Department of Biological and Agricultural Engineering
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2018

Approved by:

Major Professor
Daniel Flippo

Copyright

Abstract

This document presents two distinct designs for tools with potential to improve the efficacy of using multirotor aerial systems in agricultural research. I show design methods and results for constructing electrical, mechanical, and software subsystems capable of working in concert to achieve functional results in each design case. The first presented design is a device capable of remotely collecting pest samples directly from a multirotor to improve the speed and efficiency with which researchers and observe and respond to insect infestations. Design concepts, potential improvements, and construction methods are discussed culminating in the presentation of a prototype. The second design presented here is a printed circuit board for integration of a GNSS receiver with Real Time Kinematic correction capability, an IMU for orientation estimation, and a microcontroller with firmware to support, condition, and log data outputs. The purpose of this design is to provide precise logs of position and orientation of an aircraft and attached camera while collecting images of cropland. This reference data allows precise and accurate geolocation of the images and permits them to be stitched together into a composite map of cropland without requirements for overlap in the content of each individual image. Reduction in required image overlap allows composite aerial images of cropland to be constructed with far less flight time and research expenditure. The development and basic functionality of the device is discussed here. Deeper analysis of performance and applications of this technology is reserved for future publications.

Table of Contents

List of Figures

Acknowledgments

I'd like to thank Daniel Flippo, Brian McCornack, and Ajay Sharda for making this research possible. I'd also like to thank Devin Schottler for his assistance in late night (and early morning) firmware testing and debugging during the development of this paper.

Dedication

To Soflax, the kindest resto shaman in all of Azeroth. I hope I've made you proud.

Chapter 1

Applications of Unmanned Aerial Systems in Automated Collection of Insect Samples

1.1 Introduction

Crop losses due to disease and pests have been the subject of research for decades. These losses have economic and ecological impacts that vary with differing locations and crops, but the impacts are universally negative. Over the years many attempts have been made to estimate the true global impact of these losses. One 2003 review by Thomas Henneberry concluded that the approximate losses to arthropod pests at the time were in the range of ten to fifteen percent of total production with worldwide costs of over 120 billion dollars.[1] Another review from 2005 provides similar estimates but also makes a worrying observation that while pesticide use has increased dramatically, pest losses had not decreased significantly in the previous forty years. Instead, increasing pesticide use seemed to only allow agriculture to sustain the steady increase in production to meet demand while keeping loss percentages constant.[2] These observations are particularly worrying in the context of other challenges faced by modern agriculture. There is significant political pressure on farmers

in some areas, especially in Europe, to become more environmentally viable by reducing or eliminating use of certain pesticides. This political pressure is backed by a significant body of research suggesting that current levels of pesticide use can contaminate water and soil, leading to negative environmental impacts.[3][4] In the face of this challenge, communities are looking to develop and adopt more holistic and targeted pest management schemes.[5] Part of this initiative is the search to improve chemical delivery techniques in a way that will more specifically target the pest infestations and reduce the necessary dose.[4] Research in these areas presents an opportunity to both reduce crop loss and chemical pesticide use, and could be the key to sustainably increasing yields and improving the viability of agriculture in the long term.

In the course of searching for an agricultural pest management solution, it is useful to look at concurrent research to identify similar challenges and successful solutions presented throughout the field of applied ecology. Detection and control of invasive or destructive species is a common subject in the study of many ecosystems, and focuses on the same problems faced by modern agriculture in different settings. Researchers have found that in pest management operations, one of the main factors that constrain a solution is the availability of time and human resources to effectively search for and identify infestations.[6] One example of an extremely successful response to an invasive species can be found in California in the Agua Hedionda Lagoon where *Caulerpa taxifolia* posed a severe threat to the ecosystem. The response to this particular infestation has become an example of what is required to effectively suppress the appearance of an invasive pest infestation. One of the key elements that made this response so successful was the speed with which the infestation was identified. Researchers familiar with the project have suggested that similarly rapid responses are necessary for successful invasive pest control in the future.[7] In the spirit of this example, some researchers have begun to investigate ways to leverage new technologies to reduce the effort required to scout and identify infestations. During this same time period, we have seen the popularity and availability of unmanned aerial systems (UAS) for both commercial and recreational uses increase significantly. This has led to many projects attempting to use

UAS to scout for information related to pest infestations. One ecology team successfully used UAS with visual sensor and near infrared reflection systems to identify oak trees with oak splendour beetle infestations.[8] Researchers have outfitted UAS with a wide variety of sensor systems depending on the intended applications, varying from basic visual cameras to highly specialized sensor arrays. Agricultural researchers are no exception to this pattern, with UAS being routinely used for a number of agricultural research applications. In the field of precision agriculture, UAS are now being widely used as a method to gather information in a lower cost, higher resolution, and more flexible process than using satellite imagery[9]. The images collected using UAS can be used for any application where a spatial map of crop properties is needed, including cursory evaluations of possible insect infestations.

At Kansas State University, the entomology department routinely uses DJI S1000 octorotor systems to collect images of cropland. Using these photos, the team is able to detect locations where crops have potentially been damaged by insect infestations. Unfortunately, this is where the usefulness of the UAS in this system ends. After the UAS imagery locates areas of potential damage, researchers must manually walk to each of these locations and visually assess damaging pests. With this approach, navigating fields to find the correct location can be challenging, and assessing the severity of infestations can be labor intensive. Despite our efforts to leverage technology to visually locate infestations, we find that this workflow is still constrained by the previously discussed limitations of time and human resources for sampling. This publication presents an alternative method by which a sample can be collected directly from the UAS. This improvement has potential to improve the speed at which researchers can identify threats to crop health, allowing more prompt and precise intervention to prevent crop damage and reduce use of chemical insecticides.

1.2 Requirements

Designing an airborne apparatus with the capability to capture and hold captured insects presents many challenges. I determined two main requirements that I had to adhere

to in the design process in order to build a successful prototype. One of the main limitations was the weight restriction inherent to any airborne design. The maximum takeoff weight of the S100 is 11 kg, of which 4 kg is used by the multirotor, according to the user manual provided by DJI[10]. With the addition of a camera load (two Sony a5100 cameras), the entire system weighed 8.6 kg. I tested the system by flying it with different payload weights and evaluating battery performance. I found that at this weight, the S1000 could fly for 29 minutes prior to running out of battery. Thus, according to the results of this flight test and the manufacturer specifications, the absolute maximum allowable weight of our sampler design was 2.4 kg.

Another challenge faced in the design process was the air currents that are produced by the S1000 rotors. I tested this by flying the S1000 over crops and visually examining the crop canopy, and found that if the S1000 was operated within roughly two meters of the canopy, there was significant disturbance of the crop canopy by these air currents. Additionally, the operator of the UAS was uncomfortable flying any closer to the ground than two meters because of concerns with flight stability at very low altitudes. As a result, another major design requirement was that the apparatus must be capable of sampling from heights greater than two meters, meaning it must also be able to recognize and regulate its distance from the canopy.

1.3 System Design

During the early stages of the design process, I determined that our solution would need to be physically separated into two parts. The first component was a suction device referred to as the "sampler" that lowered into the canopy of the crop to trap insects. The second component connected directly to the S1000 and controlled a spool to raise and lower the sampler as needed, it was known as the "base". These two components each needed their own power supply, processor, and a method for wirelessly communicating with each other. The base lowered the sampler from the S1000 into the canopy, where it opened a

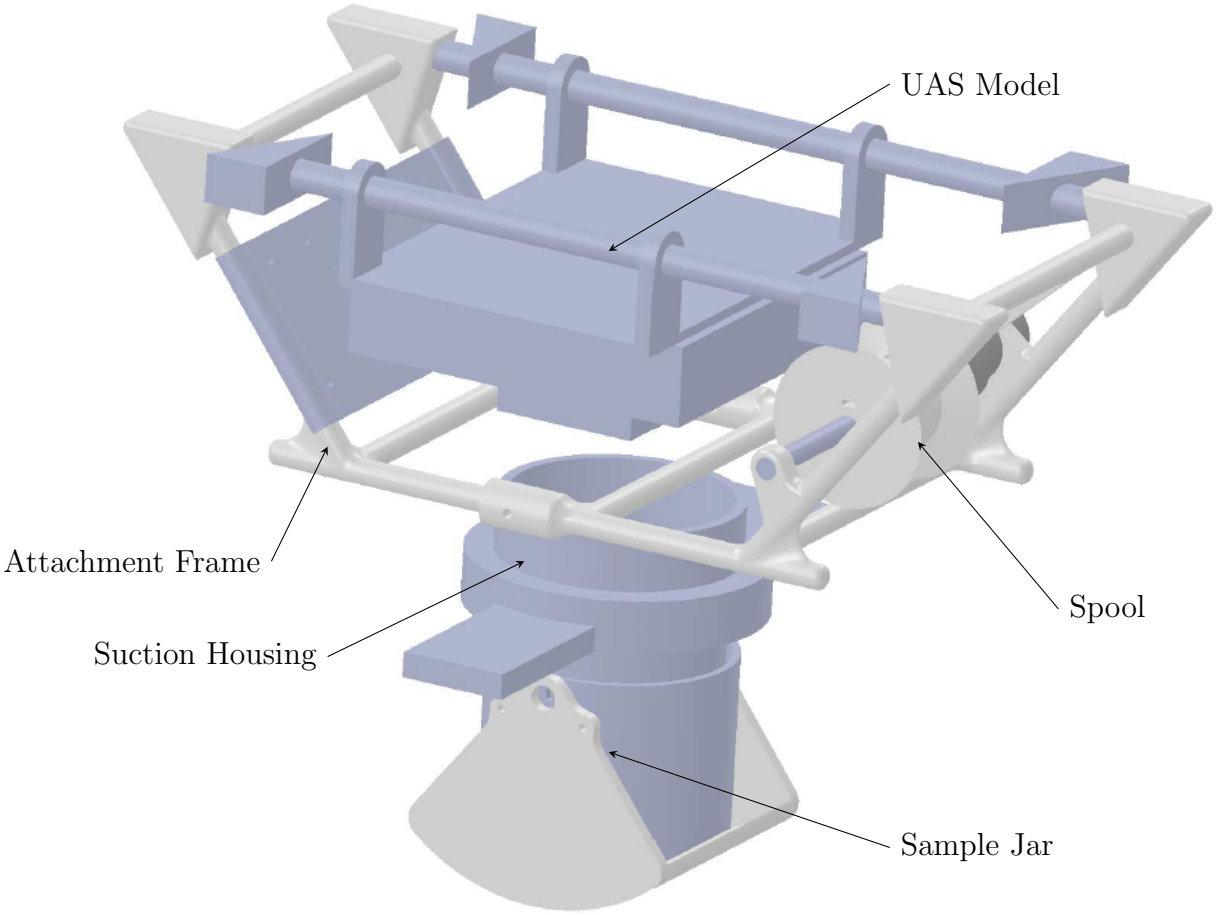
sampling jar backed with an aluminum mesh and apply suction. This trapped any insect in the vicinity in the jar, which then closed automatically after a predetermined amount of time. The base then hoisted the sampler back up to the S1000 to be carried back to the entomologist for analysis.

1.4 Physical Design

In my design for the mechanical components of the base station, the primary design goals were ease of use, security, and compatibility with other mounting attachments. The mounting platform on the underside of the S1000 is flanked by two carbon fiber rods that serve as the connection points for most payloads that would be attached to the system. Typically, a device will have a series of mounting brackets that are attached near the midpoint of these rods. I noticed that the ends of these rods are exposed and rarely used for anything, so I designed a two piece frame that would fit on over the ends of the rods and connect in the middle. This method creates an extremely secure connection using only two screws, and it can be mounted without interfering with any payload attachments connected near the middle of the mounting rods. Figure 1.1 shows a labeled CAD model of the initial design of the physical system including a rough model of the underside of the S1000. The mounting frame discussed here can be identified as the white part labeled "Attachment Frame". The frame was 3D printed from "ngen" copolyester for temperature resistance and toughness[11], and includes mounting locations for the motor and spool components. I added press fit needle bearings to the frame as support for an anodized aluminum shaft coupled to a motor, and friction fit a 3D printed ABS spool onto the shaft. This apparatus was functional in raising and lowering objects to and from the S1000, but I observed over time that the motor mounting locations were flexing from load, and the midpoint connections for the two pieces of the attachment frame were difficult to connect and disconnect. I addressed these two weaknesses in the second design iteration.

To help ease the integration of my device into the existing workflow used to collect field samples, I decided to incorporate a readily available mesh backed sample jar that was already being used in ground-based samplin g into my design as a removable part. Through careful measurement and iterative design, I was able to create a part with threads that mate with the sample jar. Careful design of the thread geometry allows this part to be 3D printed on most FDM machines with no support structure for rapid adjustment and replacement. I used this design to create a motorized lid for the sample jars that can be actuated electronically. I was also able to create a removable fixture for the back side of the sample jar that housed a motorized fan to create suction, and also served as a connection point to the base station. These two elements are also labeled in figure 1.1.

Figure 1.1: *Physical System Design: First Iteration*



Once I assembled an initial prototype of this design, I began testing it for basic functionality. It performed well at low fan speeds up to 80% power, but when the fan was run at higher than 80% power the system experienced a catastrophic failure. The aluminum mesh on the backside of our sample jar impeded the flow of air and created a pressure drop. This caused a partial vacuum to form between the fan and the sample jar. As a result, the fan blades bent inward and made contact with the housing at full speed, causing the ducted fan unit to fracture.

This failure resulted in a complete redesign of the suction system using two horizontal fans. The new design addressed the previously observed design flaws related to fan interference and reduced the down force created by the airflow through the apparatus. The redesign introduced more weight by adding a second fan, but remained within our design limitations. Figure 1.2 is an image of the updated sampler design, and Figure 1.3 is an exploded view of the same updated design.

Figure 1.2: *Physical System Design: Second Iteration*

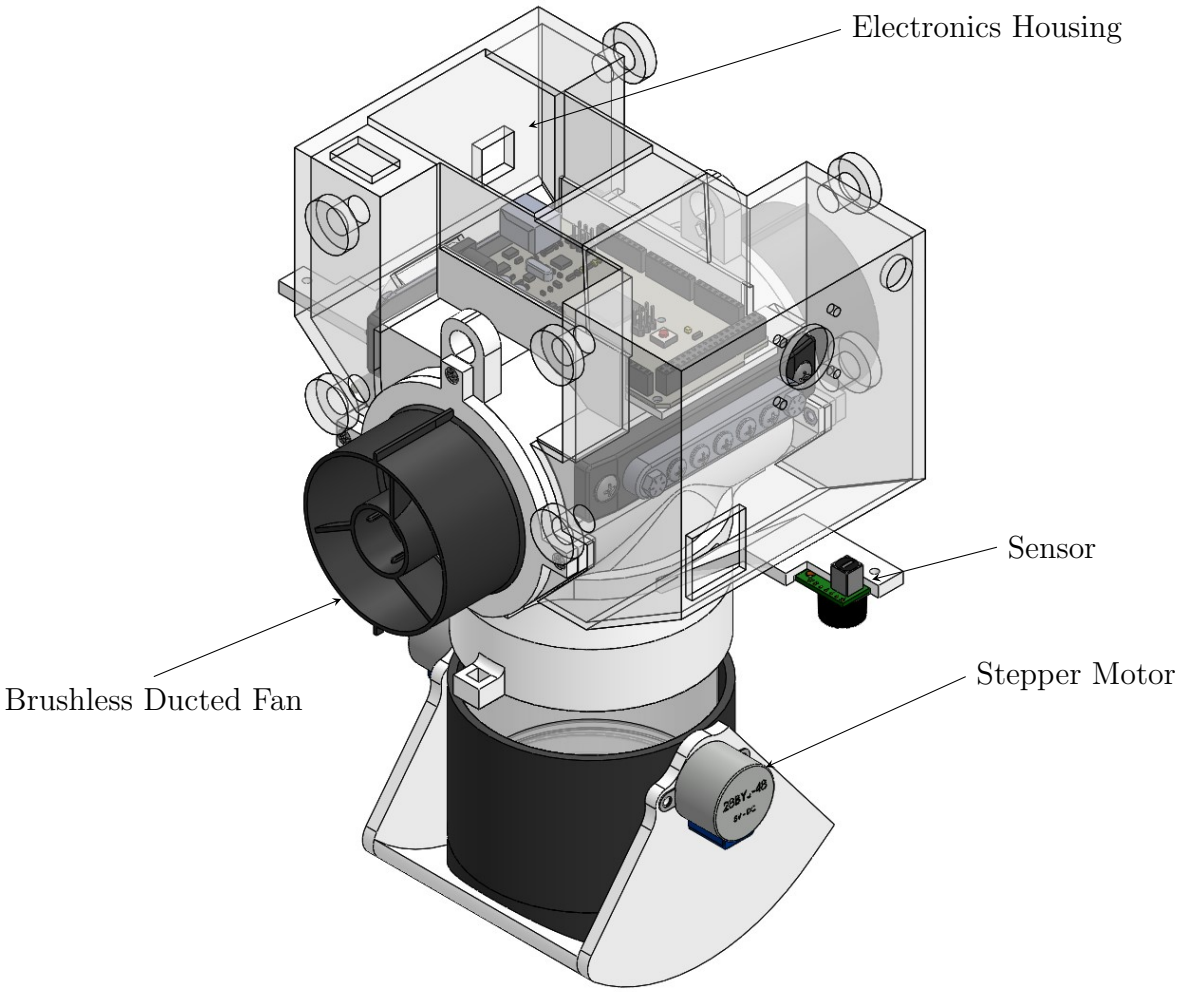
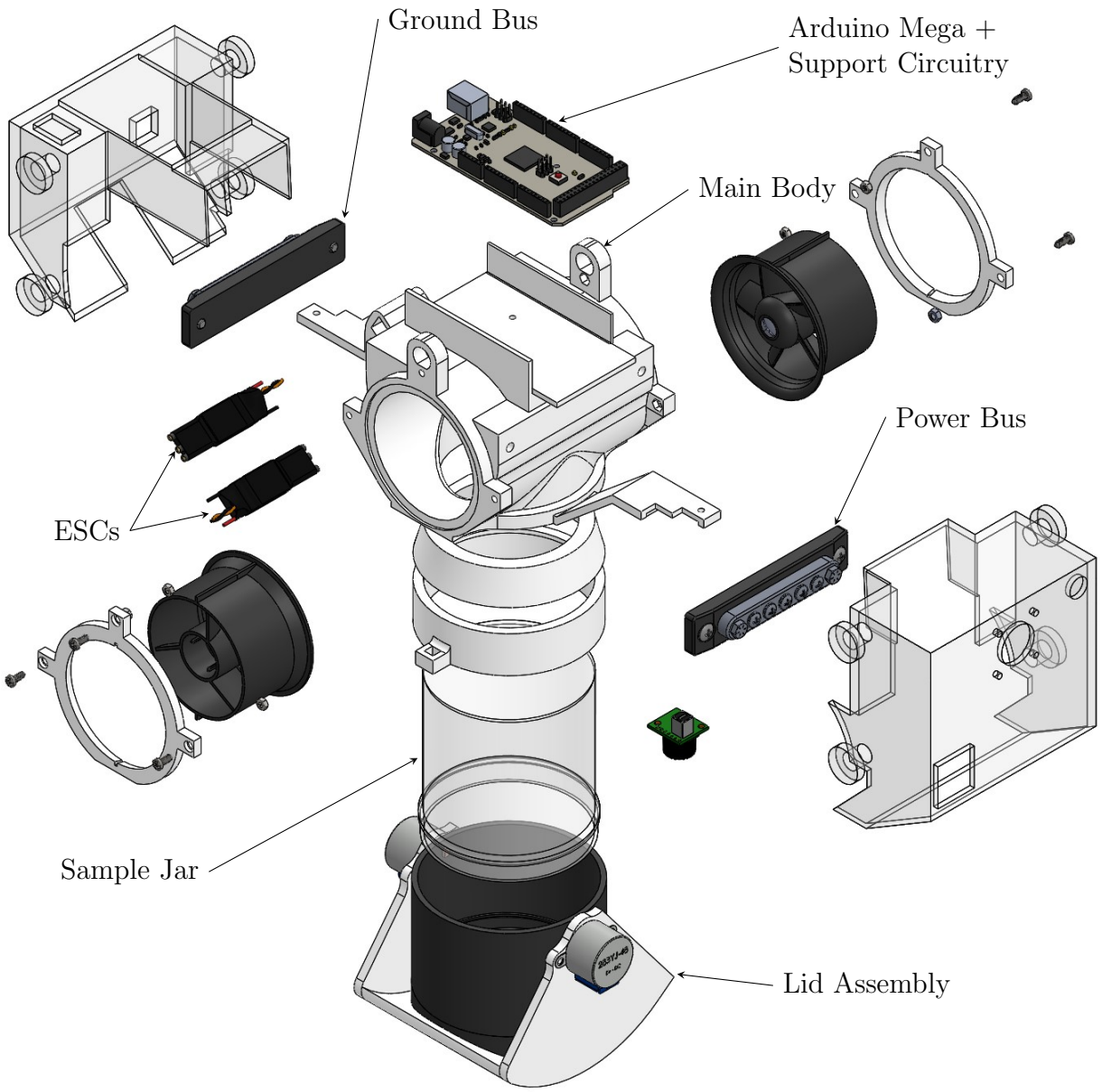


Figure 1.3: *Physical System Design: Exploded View*



1.4.1 Detailed Component Selection

1.4.1.1 Hoist Motor Selection

My application requires high torque but has no special environmental or lifetime constraints. I also have no need for precise measurements of motor position. This is because the relevant controllable parameter in this system is the distance between the sampler module and the ground, which varies independently of motor position. This leads to the conclusion that there is no need for the precision of a stepper motor or the durability of a brushless motor, and the most cost effective choice is a simple brushed DC motor equipped with a speed reducing gearbox. Choosing the correct speed reduction and motor power requires some basic engineering analysis of the system.

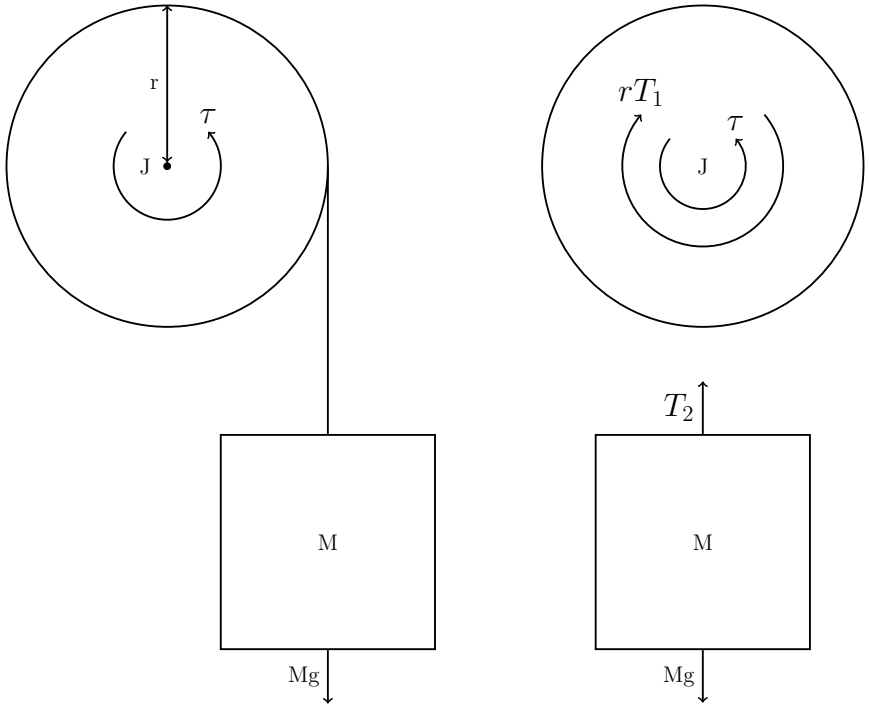


Figure 1.4: *Spool Free Body Diagrams*

Figure 1.4 shows a simplified diagram of the physics that describes the spool raising the sampler. By Newton's second law, the equations of motion for this system are:

$$\ddot{\theta} = \frac{\tau}{J} - \frac{rT_1}{J} \tag{1.1}$$

$$\ddot{x} = \frac{T_2}{M} - g \tag{1.2}$$

I ignore viscous friction here because I are choosing this motor for an application that calls for high torque and low velocity. Because viscous friction is typically modeled as either a linear or exponential function of velocity, I can conclude that it will be negligible within the desired operating range. The system can be simplified to a single equation by considering the following:

$$x = r\theta \tag{1.3}$$

$$T_1 = -T_2 \tag{1.4}$$

$$J = \frac{M_1 r^2}{2} \tag{1.5}$$

$$M_1 << M \tag{1.6}$$

Using these properties, I can solve for the following by substitution:

$$\ddot{x} = \frac{2\tau + 2Mgr}{M_1 r + 2Mr} \tag{1.7}$$

$$\tau \gtrapprox \ddot{x}_d Mr - Mgr \tag{1.8}$$

In equation (1.8) I have an approximate design constraint where $\ddot{x}_d$ is our desired upward acceleration capability. Some parameters are attained by estimation based on an intuitive understanding of the system. For instance, the drone will lower the sampler about two meters, and the maximum reasonable time to hoist that distance is about five seconds. This translates to a velocity of about 0.4 meters per second. Because I chose a design velocity of

0.4 m/s and reasonably I want to reach that velocity in 1 second, our desired steady state acceleration is $\frac{\Delta v}{\Delta t} = 0.4\frac{m}{s^2}$. I must account for a maximum sampler mass of four kilograms and spatial considerations constrain $r \lessapprox 6mm$. Plugging these values in to equation (1.8) I find a minimum torque rating of 0.2448 kg*m for our motor selection.

Ultimately, this analysis lead me to select the RKI-1420 DC motor. This cost effective motor advertises the capability to generate 0.32kg*m of torque with a maximum speed of 200 RPM and a stall current of 8 A[12]. The torque delivered by this motor is well within our calculated specification and the current demand can easily be supplied by a typical lithium-polymer battery.

1.4.1.2 Stepper Motor Selection

The sampler lid assembly calls for two motors to open and close the lid. The movement of these motors only require them to rotate 70 degrees one direction, and then the same distance in the opposite direction. Because this motion calls for some degree of precision, a stepper motor is the most viable product for this application. There are no special constraints on the power output of the motor because the lid can open or close as slowly as it needs to without significantly affecting the performance of the system. This means that I can introduce any gear reduction necessary to extract the appropriate amount of torque from the motor I use. The 28BYJ-48 unipolar stepper motor is a very small, light, and cheap product that can be easily driven by four simple darlington pair circuits. While by many criteria this stepper motor is an inferior product to most bipolar stepper motors, it meets the design needs of this application while offering a more simple driving circuit scheme and ready availability. Based on this reasoning, I selected the 28BYJ-48 stepper motor to actuate the lid of the sampler.

1.4.1.3 Ducted Fan Selection

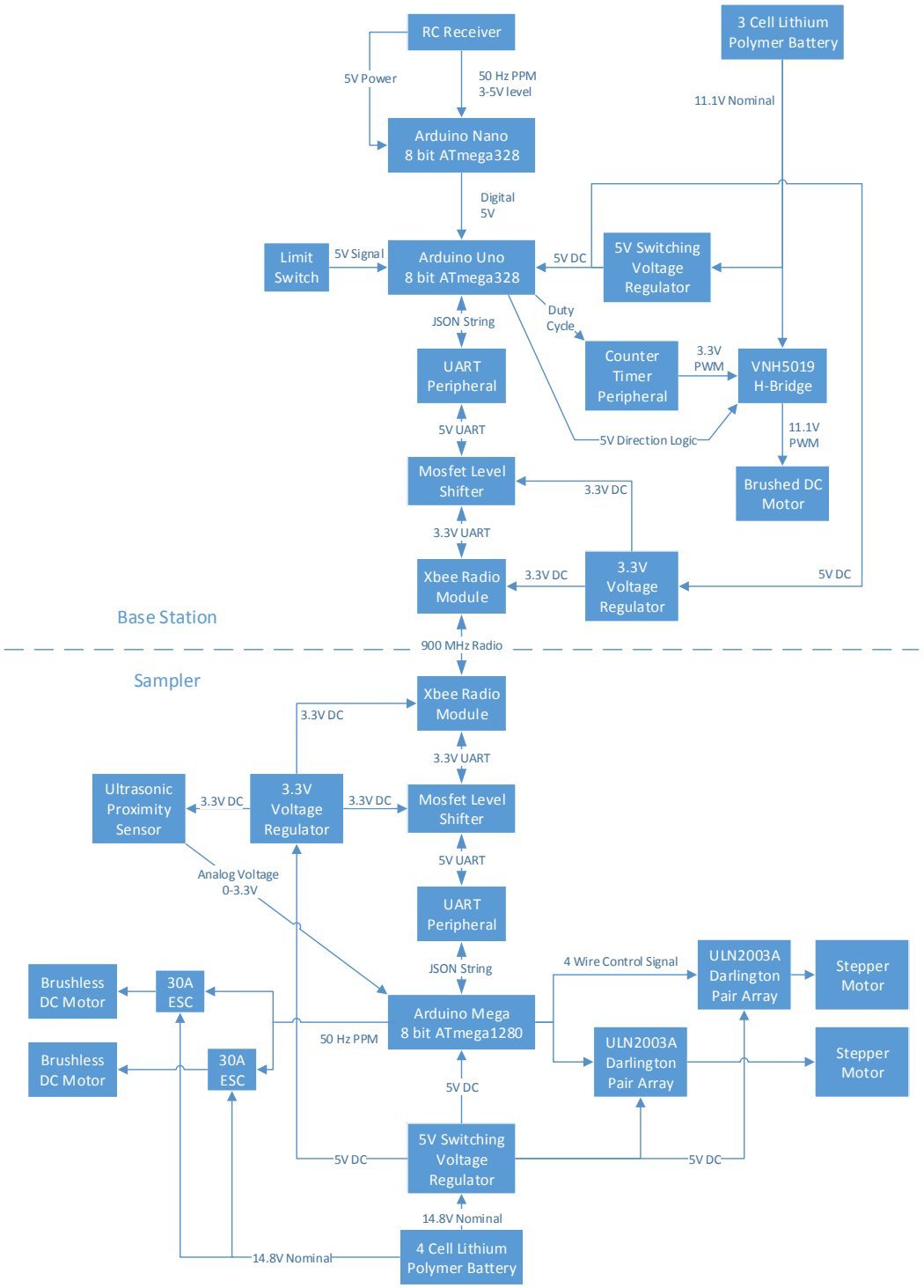
The sampling device requires two ducted fan units to generate suction to capture bugs. Ducted fans are common and cheap because of their use in the hobby RC aircraft industry.

They typically use brushless motors and are compatible with a myriad of driver circuits called "ESCs" that are brushless speed control circuits. They typically use back-EMF sensing to perform closed loop speed control in the absence of hall effect sensors that would sense the position of the magnets in the motor. Because of the mass production for the hobby industry, these items can be purchased for a price far cheaper than I could ever build them myself. The disadvantage is the lack of documentation on these parts. Because most RC hobbyists are looking for "plug and play" solutions, integrating these parts into a design or determining whether they have enough power for a certain application can be difficult. Because of their low cost, I decided that the easiest solution in this case was to purchase the most basic ducted fan I could find in an appropriate size and evaluate its performance experimentally. My initial purchase was a generic 4500 KV plastic ducted fan with a diameter of 64mm that lacked an identifying part number or any documentation. The first test I ran on this component was to place it in the main body of the sampler, place objects of varying density from the lab beneath it, and run the motor at varying power levels. To our surprise, even at a modest power level it was obvious from visual inspection and qualitative observation that the suction generated by this ducted fan was sufficient and possibly even excessive for our needs. Because I was able to verify that this component is easily sufficient without any time consuming analysis, I turned our time and attention to analyze and design more vulnerable points in our system.

1.5 Electronics Design

In this section, I will discuss the prototyped electronics that drive the mechanical design and run the firmware. I will also show how the prototyped electronics I used in testing can translate to a far lighter and more integrated solution using custom printed circuit boards. Figure 1.5 is a block diagram of the functional prototyped electrical system that will be discussed first.

Figure 1.5: *Electronics Block Diagram*



1.5.1 First Prototype for Functional Demonstration

The circuitry used initially to demonstrate basic functionality was mostly composed of development boards stitched together by headers and jumper wires. While this may not be the most durable or compact solution, it is very effective for demonstrating the efficacy of a design concept. The main electrical components of the system are two microcontroller development boards, two XBee radio modules, eight darlington pairs, two brushless speed controllers, one ultrasound proximity sensor, one H-Bridge, and any voltage regulators or level shifting circuitry to power and interface these components with each other.

1.5.1.1 Basic Component Selection

When selecting components for the first prototype, development speed was imperative. Despite the disappointing performance characteristics of the ATmega328 compared to ARM processors of comparable price, I chose to use the "Redboard" arduino compatible ATmega328 development board from Sparkfun Electronics. The main reasoning for the choice of this specific product was the availability of plug-and-play arduino compatible shields that would allow us to use XBee radios to get a wireless microcontroller communication solution up and running in a matter of hours. I used an Arduino Mega development board equipped with an ATmega1280 processor for the sampler because it has more digital input/output (DIO) pins. This was a consideration because of the additional IO pins that would be required to drive stepper motors. The Arduino Mega is equally compatible with the arduino shields used on the base redboard, so there were no integration concerns arising as a result of changing processors in this case. I used XBee Series 1 radio modules because I only required point-to-point communication, and the Sparkfun XBee shields I purchased were equipped with onboard 3.3V regulators and mosfet level shifters to allow the 3.3V radios to integrate easily with the 5V Arduino boards. For the proximity sensor, I decided to use the XL series of MaxSonar sensors from Maxbotix. These sensors offer 1 cm resolution between 0 and 765 cm, and output the latest sensor readings in the form of pulse width, analog, and serial communication. The presence of three different interfaces made this sensor ideal for prototyping

because of increased flexibility. For driving the stepper motors, the seller of the 28BYJ-48 provided small driver circuits that used the ULN2003A IC containing seven Darlington pairs with common cathode diodes to prevent damage from back-EMF when driving inductive loads. At first I used the provided boards to drive the stepper motors, but upon inspection of the boards I concluded that the electrical connections would be less prone to failure if I etched a custom arduino uno shield with two ULN2003A ICs and dedicated labeled IO ports to ensure external components were connected correctly. Driving the brushless motors is the system was best accomplished by external driver circuits that use back-emf sensing to achieve sensorless speed control of brushless motors. I selected the HobbyWing Skywalker 60A ESC for this purpose, because it was inexpensive and rated for far more current than I ever expected our brushless fans to require. This ESC was specifically designed to run on a 4s lithium polymer battery, so battery selection for the sampler was extremely easy. The amount of current draw during full power operation of these fans is extremely large, so I decided to use 100A rated busbars to create power and ground buses. This drastically reduced the amount of soldering required and resolved some odd behavior I observed in initial component testing. Because our brushed motor on the base was designed to run at 12V, I selected a 3s 11.1V nominal lithium polymer battery to power the base. Because of these relatively high voltages (compared to our processor operating voltage) I added a pair of 5V switching regulators from Pololu and used them to bypass the linear regulator on the arduino development boards. This prevented excessive heat buildup and extended battery life because the typical efficiency of the switching regulators was 85%-95% depending on current draw and operating conditions, as compared to around 45% efficiency of the on-board linear regulators. The final component is the brushed motor driver, which required sa bit more design because there was no prepackaged solution to use for quick prototyping.

1.5.1.2 Detailed Motor Driver Selection

The RKI-1420 datasheet states that the nominal operating voltage for the motor is 12V, and the maximum current draw is 8 A. The most convenient method for controlling a

motor is to use a pulse width modulated signal (PWM) to open and close the gates of an H-bridge circuit. A PWM signal is similar to a square wave, except that the high state and the low state can differ in length. The ratio of time at the high state to the total time (high state plus low state) is called the duty cycle. PWM signals roughly approximate an analog voltage of $V \cong (\text{duty cycle}) * V_{\text{MAX}}$, this approximation is most effective when the PWM frequency is much faster than the natural frequency of the system being controlled. This approach is easily implemented using the counter-timer peripheral supplied on microcontrollers and doesn't require a negative power supply rail like a comparable analog amplifier circuit would, because the transistors in a PWM driven H-bridge circuit can easily reverse the output voltage terminal connections. Although an H-bridge consists of four transistors, most major manufacturers offer H-bridges as fully integrated circuits (IC) for convenience. In particular, the VNH5019 H-bridge from ST Microelectronics has the capability to handle input voltages up to 24 V and continuous currents of 12 A. It can operate at frequencies up to 20 kHz and also features integrated current sensing. These capabilities indicate that this IC should be capable of driving the RKI-1420 at its maximum current and at an appropriate PWM frequency without dangerous heat buildup or significant noise in the motor voltage. For prototyping purposes, I chose to purchase the "VNH5019 Carrier Board" from Pololu because it is a convenient platform for testing the motor driver without needing surface mount soldering or a custom breakout board.

1.5.1.3 First Prototype Results

While the construction of the first round of circuitry may appear crude when compared to commercial solutions, it ultimately performed its task of demonstrating a functional prototype very well. In simple benchtop tests the system was able to perform all desired tasks and demonstrated the ability to collect test objects placed beneath it, including dead insects of varying sizes. Although mostly successful, the prototype was prone to occasional failures. Testing with an oscilloscope showed that many of the connections, especially the ones made using jumper wires, were prone to noise and signal interruption due to unreliable electrical

contacts. In the following section, I will present a far more reliable, robust, and compact solution based on the same technologies used in the first prototype.

1.5.2 Integrated and Optimized Electrical Prototype

Integrating a multifaceted design like this onto a single custom printed circuit board has many advantages. Firstly, printed circuit boards are generally lighter because they avoid the weight of cables and wires used for connections in non-PCB setups. Because they reduce the length of wires and contain ground planes under each trace, well designed PCBs significantly reduce noise from electromagnetic interference, parasitic resistance, and parasitic inductance when compared to the alternative implementation of a circuit. The electrical connections are also far more reliable, and the designer has freedom to decide the size and shape of the finished product to fit with other mechanical components of the design.

1.5.2.1 Revised Sampler Design

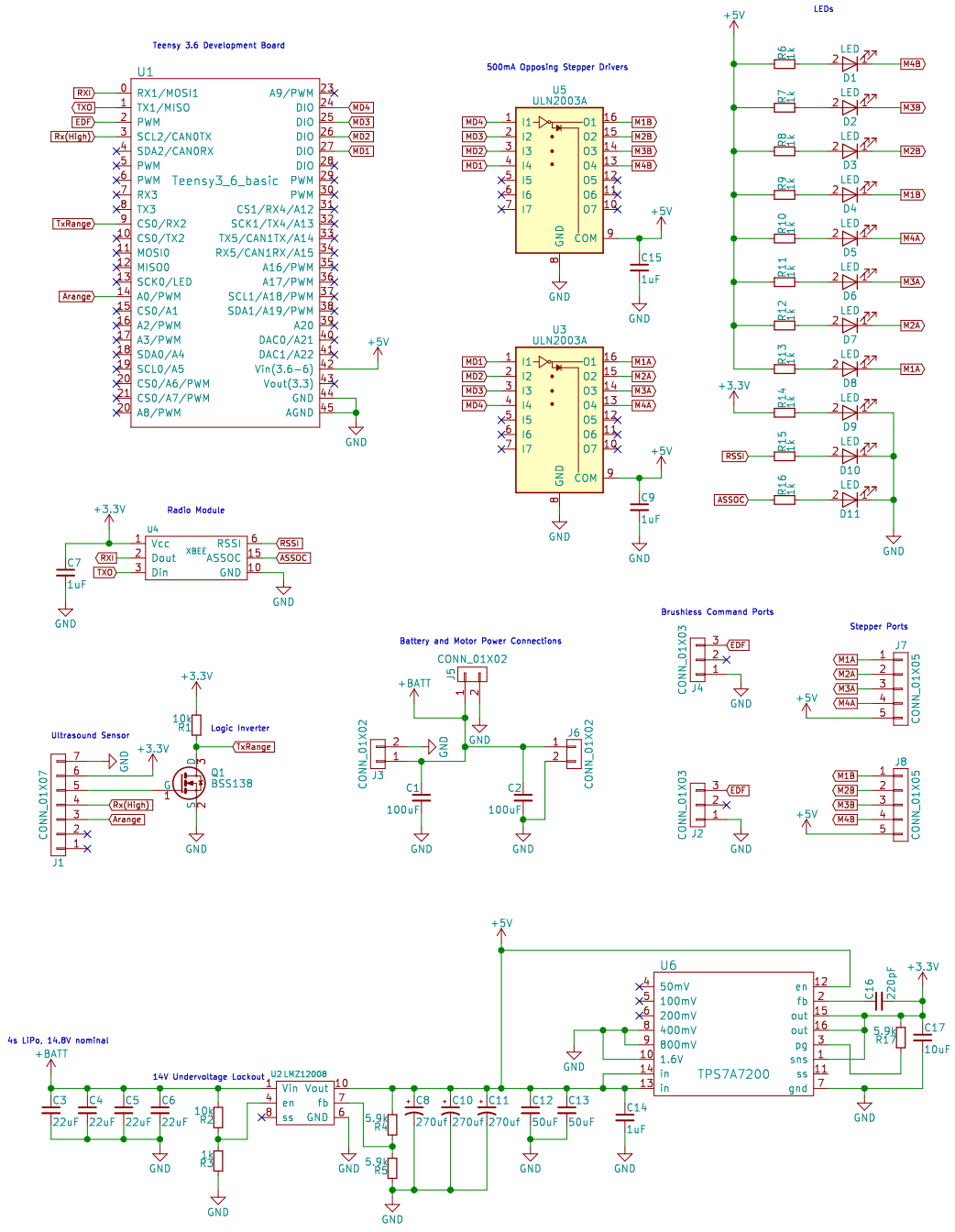
One major difference between the previous iteration and the integrated circuit board is the voltage level. Instead of using a 5V arduino, I decided to use a PJRC Teensy 3.6 development board. The Teensy is much smaller, lighter, and is based on a much more powerful ARM Cortex-M4 processor clocked at 180 MHz. The operating voltage for the Teensy board is 3.3V, which means that no level shifting will be required to interface with the radio module. However, the stepper motors still need to be run at 5V to achieve the necessary torque. To regulate the 14.8V battery efficiently down to 5V, I chose to use a Texas Instruments LMZ12008 integrated switching module. The efficiency of the fully integrated module is around 88%, smaller than a typical system with an external inductor. Despite this reduction in efficiency, the module simply offers far superior board space utilization, EMI shielding, and is far easier to solder into place than most alternative configurations. The switcher uses two external voltage dividers to set the output voltage to 5V and the input undervoltage disable condition at 14 V. For stability, the output needs three 270 microFarad capacitors with very small equivalent series resistance. This can only be achieved

using special solid polymer aluminum capacitors as noted in the product datasheet. To further regulate the 5V into 3.3V for most of the components, I added a Texas Instruments TPS7A7200 Low Dropout Regulator. This device is unique in that it doesn't require an external resistor network to set its output voltage, and instead has a series of internal resistors that can be connected to ground to set the output voltage. This has the benefit of saving board space that would otherwise be used for the resistor feedback divider.

The serial output of the ultrasound sensor is an inverted TTL serial, so I used a mosfet with a pull up resistor to invert the signal and let it be read by an ordinary UART. I also used the surface mount version of the ULN2003A to build motor drivers schematically identical to the ones provided with the stepper motors. The XBee radio requires very little support circuitry, I only added a simple bypass capacitor to the power line and connected LEDs with current limiting resistors to its status outputs. I designed the power ports so that XT60 connectors could be soldered directly to the board, allowing LiPo batteries and brushless drivers to be directly connected to the power and ground planes, eliminating the need for an external busbar. Finally, I added indicator LEDs for power and the stepper motors, and all relevant output ports, and the integrated design was complete for the sampler. The schematic of the integrated board for the sampler is shown in figure 1.6.

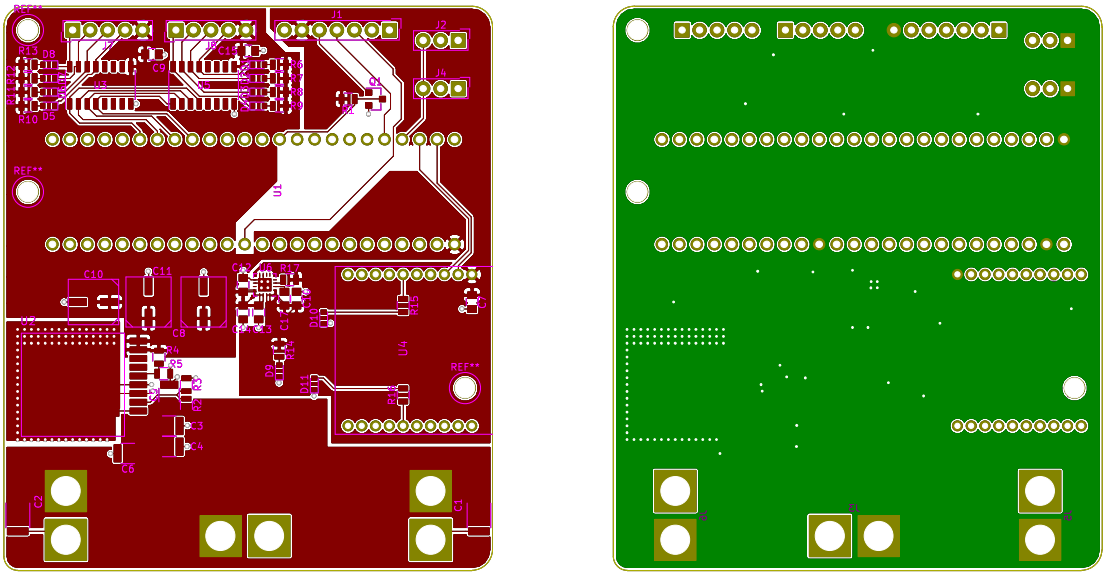
Based on the schematic shown in figure 1.6, I designed a two layer printed circuit board to house all the components and minimize space. In PCB design it is important to consider the length of each connection made, as long connections introduce additional parasitic properties to a trace. The bottom layer of the board is a mostly continuous ground plane to prevent the top layer traces from producing magnetic fields. This is generally effective because the ground plane is a near-ideal conductor and therefore any electric fields passing through it are canceled out by induced currents in the plane. In this case, when large currents may flow between the battery connector and the brushless motor outputs, it is important to ensure that enough copper is present to carry the currents without introducing losses from parasitic resistance. These high current areas should also be kept physically as

Figure 1.6: *Sampler PCB Schematic*



far away from any sensitive signals as possible to prevent interference. With these ideas in mind, I produced the board layout shown in figure 1.7. A photo of the fabricated circuit board is shown in figure 1.8.

Figure 1.7: *Sampler PCB Layout*

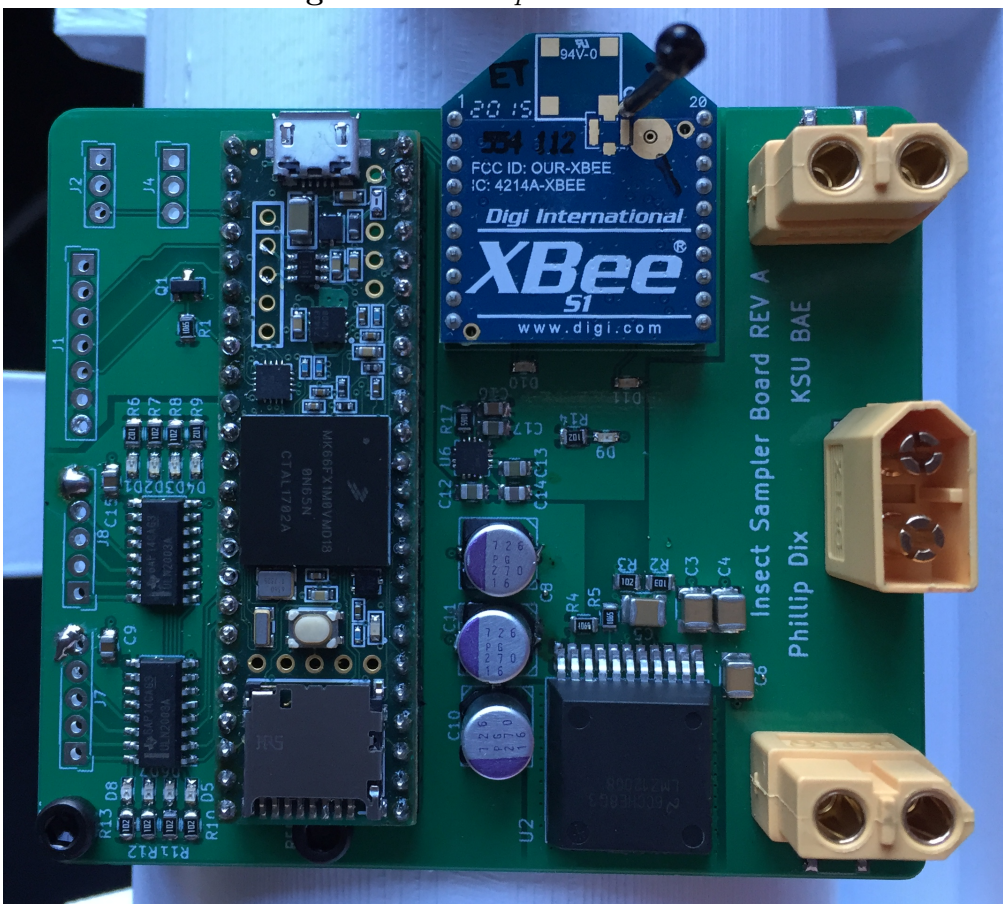


1.5.2.2 Revised Base Design

In the case of the PCB design for the base, problems with component availability and temporal constraints ultimately made the construction of the board design impossible. I am including the design here regardless, as I believe it will be helpful to researchers working on improving this project in the future.

I made a number of changes to the base design in the second revision. The voltage regulators and new processor are identical to the changes described for the revised sampler design, with the exception that the base uses a Teensy 3.2 instead of a Teensy 3.6 for spacing purposes. The one additional change made to the base design is the H-bridge choice.

Figure 1.8: *Sampler PCB Photo*



The VNH5019 is perfectly effective for driving our brushed motor, but its current rating is significantly higher than the stall current of our motor, and in that sense it is overspecified for its task. By using the similar but smaller VNH5180 IC, I was able to reduce the size and component costs of the circuit. A schematic of my proposed design is shown in figure 1.9, and a proposed board layout is shown in figure 1.10.

Figure 1.9: *Base Circuit Schematic*

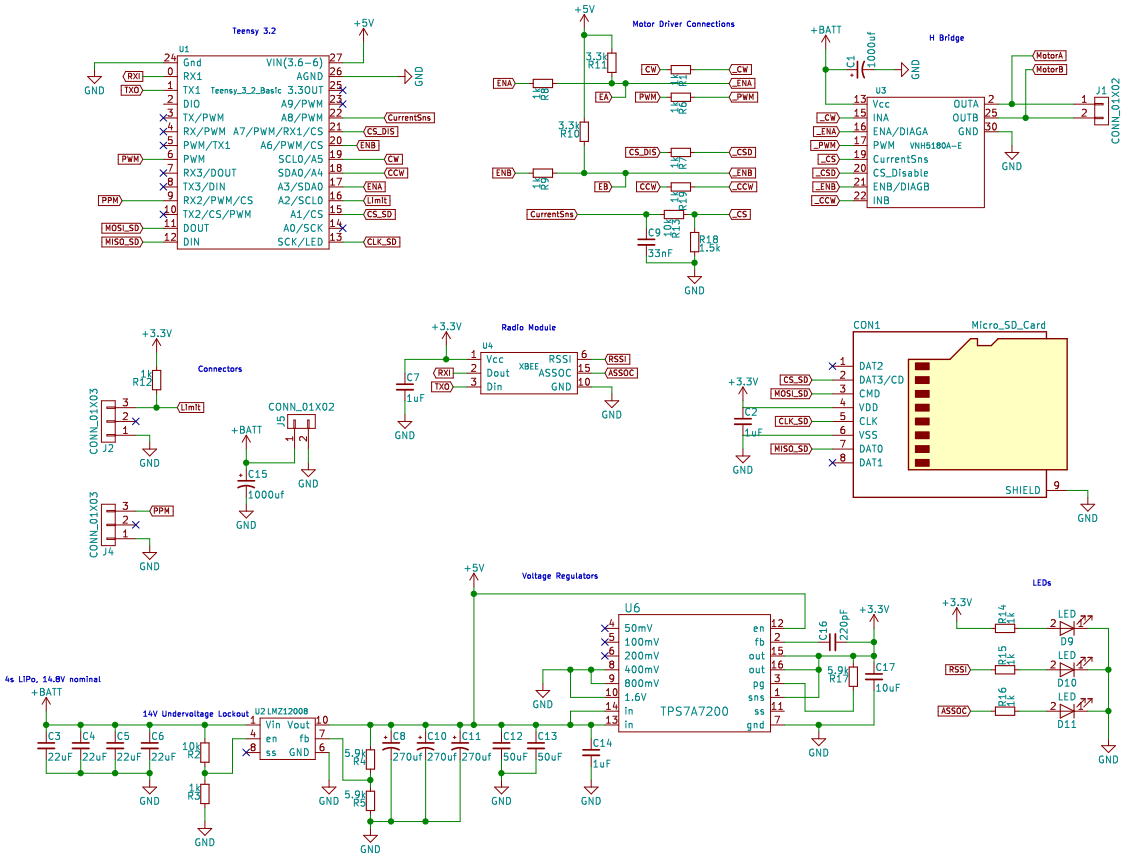
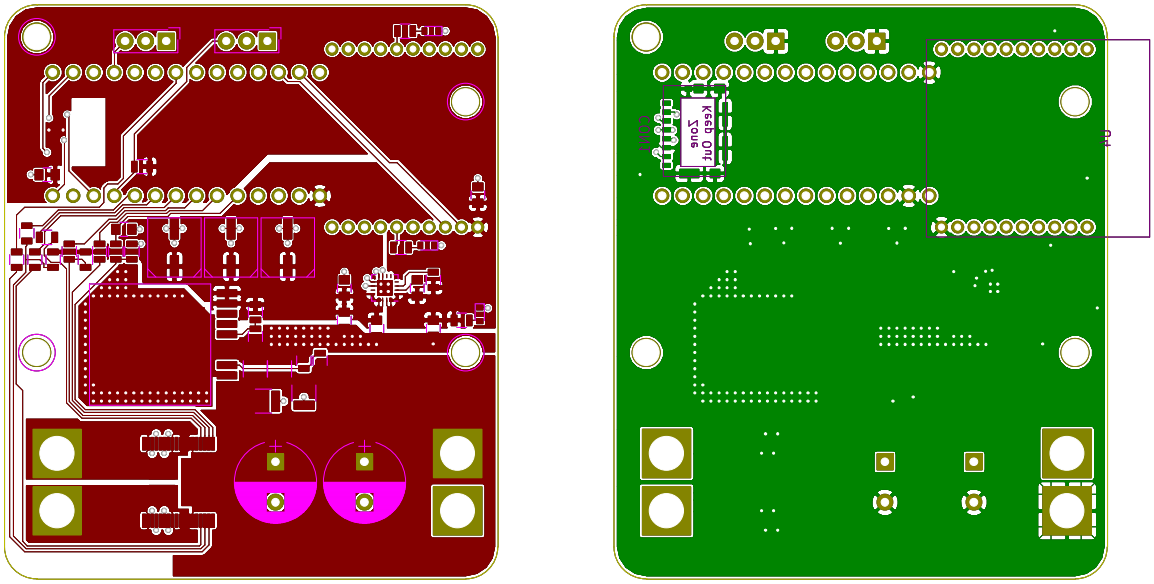


Figure 1.10: *Base PCB Layout*



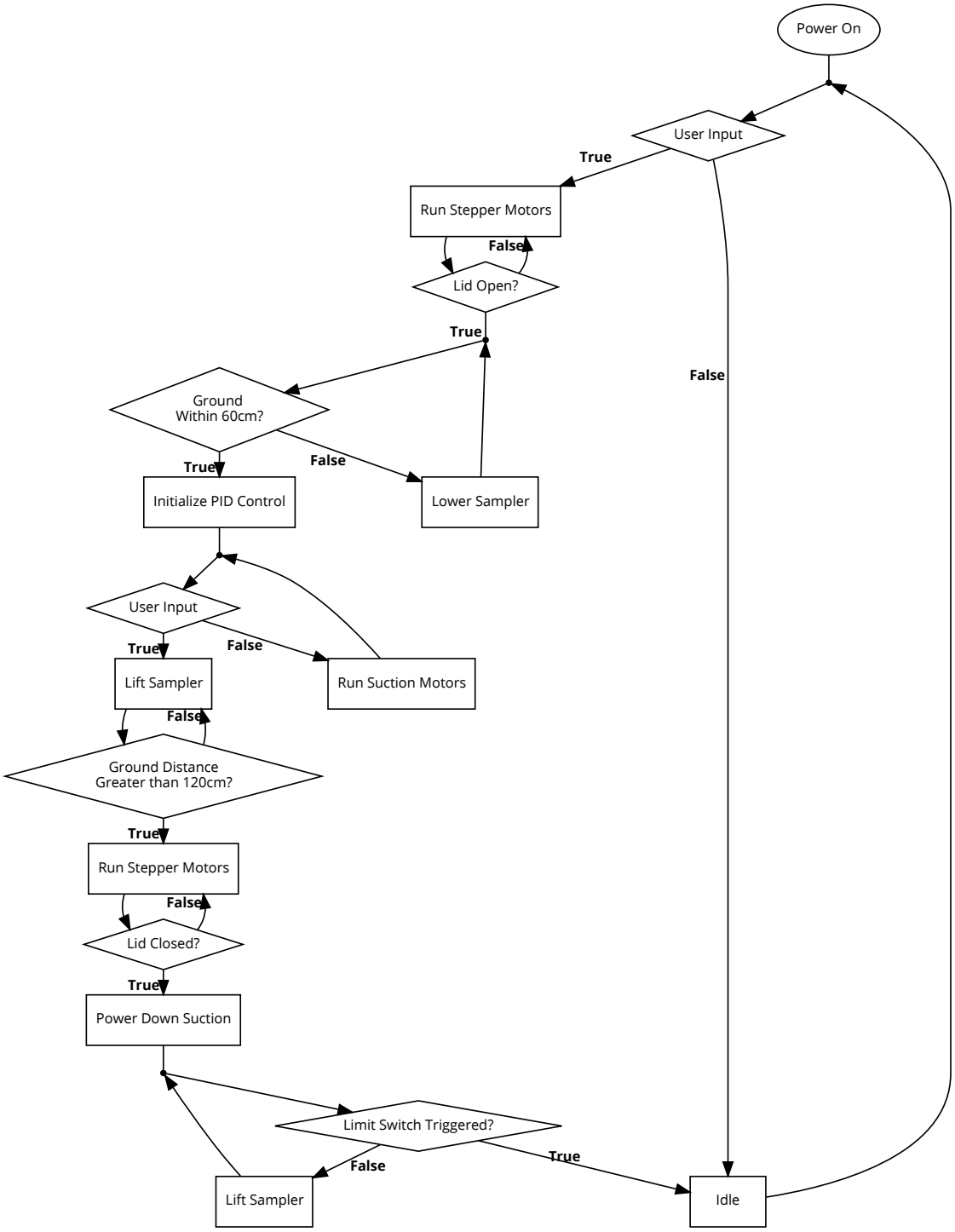
1.6 Firmware Design

The firmware portion of this design is probably the most important, and definitely the most error prone of the three. The firmware controls both the electrical and mechanical components of the design, and is responsible for ensuring that they work together to complete the complex sampling task. Figure 1.11 is a summary flowchart showing the series of successive tasks and logic that the firmware on this device must perform.

1.6.1 Development Platform

In the interest of prototyping quickly, I decided to use the popular Arduino development environment. The primary advantage of Arduino is the abundance of reliable community developed libraries that are typically open source. Arduino uses a slightly modified version of the C++ programming language and a series of libraries to abstract interaction with processor registers. However, in most cases it will compile C code and allow the user to bypass the abstraction layers it uses if modifications need to be made at the processor

Figure 1.11: *Firmware Architecture: Operational Overview*



level. In the past, using Arduino for development restricted users to a small subset of 8 bit microcontrollers. Recently developed products like the Teensy line of development boards from PJRC are breaking that mold and allowing code compiled in Arduino to run on 32 bit ARM processors. My initial prototype used an Arduino Uno development board equipped with an 8 bit AVR microcontroller. After inital testing was successful, I switched to a Teensy 3.6 development board equipped with a 32 bit ARM Cortex-M4 microcontroller for superior performance and peripheral availability.

1.6.2 Architecture

At the most basic level, the firmware on this device consists of two complimentary state machines linked by a JavaScript Object Notation (JSON) based communication protocol for the exchange of commands and data. Two state machines are necessary because the system contains two processors with two distinct sets of inputs and outputs. Each state machine has eight possible states: paused, opening, lowering, sucking, retracting, closing, raising, and terminated. The system is designed so that the base station acts as a master, sending commands to the sampler and awaiting predefined responses indicating that its commands have been carried out. This is intended to ease the debugging process by allowing a tester to easily emulate one of the nodes via a serial terminal. It also simplifies the interface behavior between the two parts of the system, allowing a designer to make changes to one node and cause minimal impact on the operation of the other.

1.6.3 Base Station

I designed the firmware for the base such that all relevant information about the state of the physical system could be stored in a single global variable. This serves to make the firmware less complicated and therefore less error prone, but also far more readable. I declared an enumerated type to track the logical operating states of the machine, and then I included that enumerated type as the first entry of a struct that also contains the physical parameters needed to determine when to change from one state to another. Listing 1.1 is

a code snippet showing the declaration and initialization of both new types and the global instance used to drive the state machine logic.

Listing 1.1: *Type definitions for tracking states*

```
enum state_t {LOWERING, OPENING, SUCKING, RETRACTING, CLOSING, RAISING, PAUSED
    , TERMINATED};

typedef struct {
  state_t mode;
  int height;
  bool lidOpen;
  bool fanOn;
} status_t;

status_t Status = {PAUSED, 0, false, false};
```

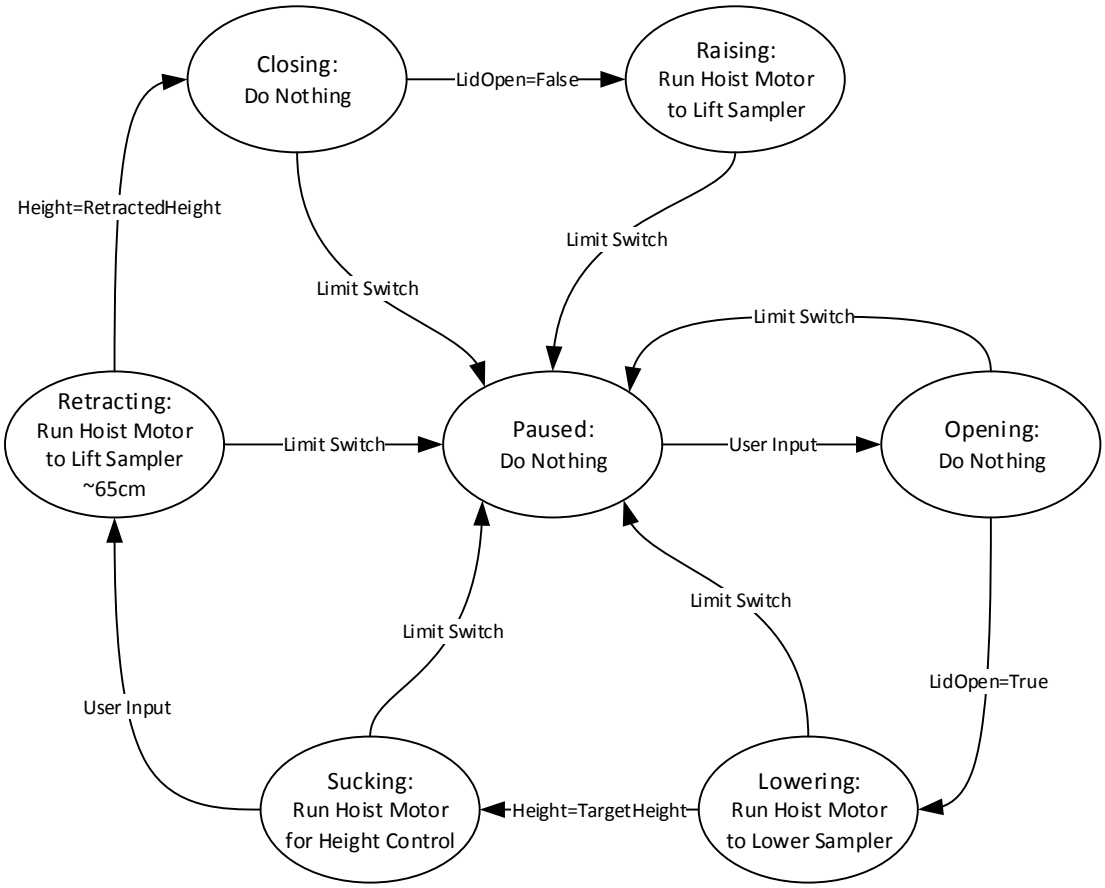
During each loop iteration, the base waits for a response from the sampler, and upon receiving any message (including an invalid one) it executes one iteration of its main loop, outputs a message, and then continues awaiting a response. If a high input from the limit switch on the spool is detected, this indicates that the sampler is in contact with the base, and the base immediately pauses operation to avoid damage. The actions performed in each case, and the conditions to cause a case switch are shown in figure 1.12.

In addition to performing its state machine logic, the base station has two additional tasks it must perform during its loop. First it must monitor the RC receiver for user input, and then it must also filter the height data received from the sampler.

1.6.3.1 User Input

The user input to our system comes in the form of a pulse from an RC receiver. The signal frequency is at 50 Hz, and consists of a pulse with a duration of 1-2 ms long. The Arduino development environment contains a function PulseIn() that measures the length of an incoming pulse on a specific pin. However, this function is known for being expensive in

Figure 1.12: *Base Station State Machine*



terms of code execution timing and often introduces limitations to a project when used. To alleviate this, I considered several possible solutions including an analog filter and amplifier to convert the pulses to analog voltages, or writing my own interrupt handlers to measure the pulse length with non-blocking code. In the end, the most economical solution was to add a second processor on an Arduino Nano board, and program it to constantly poll input pulses and output a digital high when the pulse is longer than 1.5 ms or a digital low when the pulse is shorter than 1.5 ms. In the second iteration, this component was eliminated because the new 32 bit processor was able to easily handle the PulseIn() function without delays. In both cases, the firmware checks the user input during each loop and alters the state machine case in the event of a change.

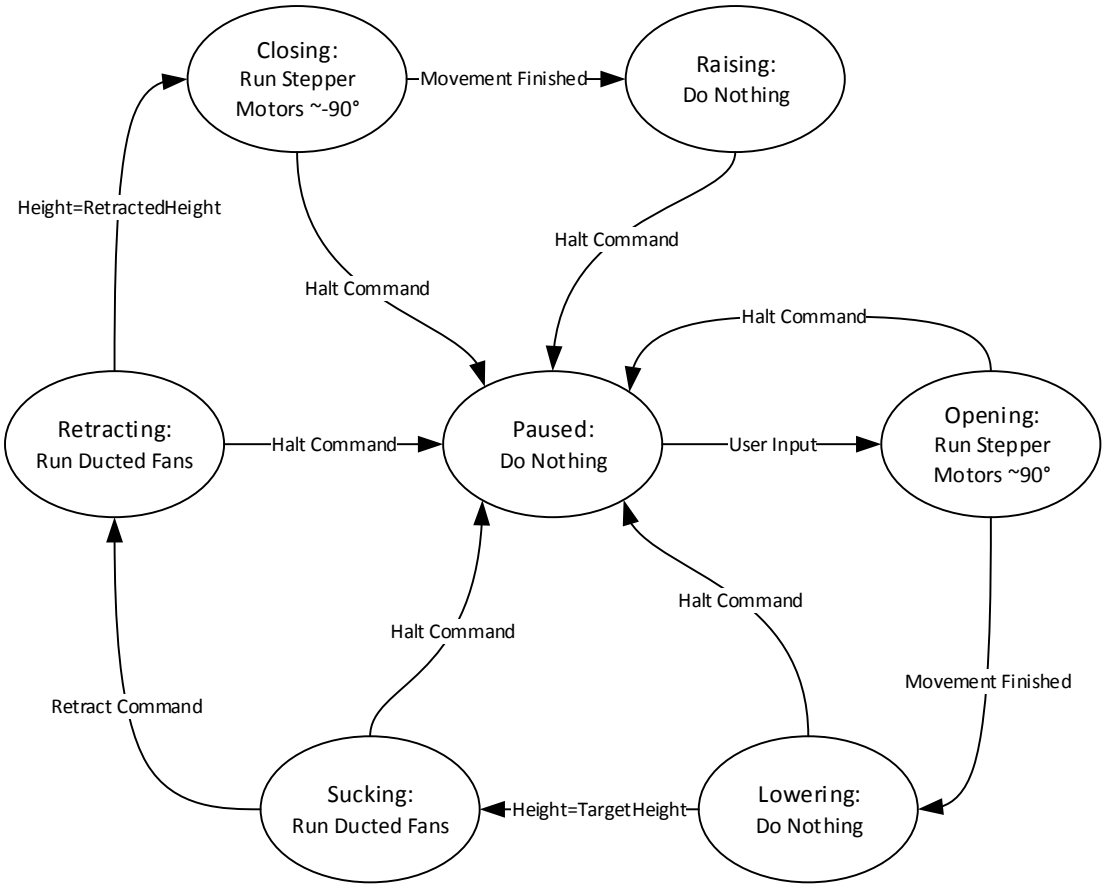
1.6.3.2 Data Processing

Because the information from the height sensor can be noisy, the base station uses a discrete approximation of a low pass filter to ensure stability of the height control loop in the event of large spikes in the sensor reading. The limitations of ultrasound sensing means that the update rate of the incoming data is only 10 Hz, which means that the Nyquist Frequency for the system is only 5 Hz. In this case, I found that any filter slower than 5 Hz caused unwanted delays in the system response time, so I drove the cutoff frequency up to 10 Hz. This obviously invalidates any continuous time approximations of the impact of the filter, and instead it simply becomes a less computationally intensive alternative to a moving average, but with a similar effect. While this filter certainly is not an ideal solution, it does exhibit the desired behavior of limiting the effect of noise in the sensor signal while not significantly limiting the controller response time. Future design iterations of this device should investigate ways to increase the sampling rate of the height sensors or implement a more sophisticated state estimation algorithm to improve this element of the design.

1.6.4 Sampler

The sampler also contains the code shown in listing 1.1, because the states and variables of the system must be common between the sampler and the base. The primary differences between the sampler and the base are the actions that are performed during each state, the triggers that cause movement between the states, and the additional tasks that each must perform regardless of their state. Figure 1.13 shows the actions taken by the sampler in each state, and its causes for moving between the states.

Figure 1.13: *Sampler State Machine*



1.6.4.1 Brushless Driver Calibration

The brushless speed controllers used to drive the fans operate based on the same type of RC pulse used for the user input to the base. In this case, our microcontroller must produce a 50 Hz wave of pulses between 1 ms and 2 ms. Unfortunately, the standard of 1 and 2 ms pulses is only loosely defined, and each device has its own input errors to deal with. To cope with this, most speed controllers have a programming and calibration sequence that can be used to set the minimum and maximum pulse lengths it will read. In the case of the Hobbywing Skywalker 60A controllers I used, the calibration must be performed via a series of pulse outputs from the movement of the control stick and throttle of a hobby remote control device. In my case, I was able to use the microcontroller to simulate the appropriate pulse output to automatically perform the programming and calibration procedure to match our pulse outputs each time the sampler is powered on.

1.6.5 Communication Protocol

Because of the complexity of the data that needs to be exchanged, and the relatively slow transmission rates required, I chose to use a human-readable ASCII character based protocol for transmission. This allows for easy error checking by the microcontrollers and debugging by any person watching the data stream. Specifically, I decided to encode all data about the state and inputs of the system into JavaScript Object Notation (JSON) because it is a simple, lightweight, and platform independent way to organize data. Most programming languages have free libraries available to parse and generate JSON strings, and Arduino is no exception. This makes it easier to interpret or edit the code I've written for the JSON data exchange, and it also provides a simple and effective way for other devices to interface with my code in the event that this system is expanded in future research. Specifically, KSU BAE uses National Instruments LabVIEW in a majority of research projects, which has inbuilt JSON support.

1.6.5.1 JSON

The "JSON Data Interchange Syntax" is defined in standard ECMA-404 Second Edition[13], adopted in December of 2017. A JSON string consists of some combination of "objects" or "arrays" which may be nested. An object in JSON consists of a series of key-value pairs and must be enclosed by curly brackets. The key must be a string, and it is separated from the value by a colon. The value may be a number, boolean, string, array, or object. Values can also be null, but this is rare and not used in my implementation. Multiple key-value pairs are separated by commas within an object. An array in JSON consists of a series of values in a specific order, separated by commas and surrounded by square brackets. The values in an array can take any of the forms allowed for the values in an object.

1.6.5.2 Implementation

Both the base and the sampler use the same bits of code in their main loop to await a string, process it, and then construct and output a new string reflecting any changes. Two functions, void outputState(status_t current_status) and void updateState(status_t* current_status, char* inputStr), are called in this bit of code. They have the same declaration, but are implemented differently on the sampler and the base to reflect each node's unique responsibility to update certain fields of the exchanged message. Listing 1.2 shows the code snippet that is common between the two nodes, in which each processor awaits a complete string message and calls the two data exchange functions at the appropriate times.

Listing 1.2: *Data Exchange Code: Common*

```
{ //add a scope here,we don't need any of the temporary variables later
 int i = 0;
 char inStr[100];
 while (1) {//wait for a new string
   if (Serial1.available()) {
     char tempstr = Serial1.read();
     if (tempstr == '\n') {
       inStr[i] = '\0';
       break; //string received
     }
     inStr[i] = tempstr;
     i++;
   }
 }
 updateState(&Status, inStr);
}

...State Machine Logic...

outputState(Status);
```

The updateState function has the task of parsing the data out of the input string and writing it to the relevant status_t struct for access by the rest of the code. In the event that local inputs would indicate a different state than the one received, the function alters the appropriate information to match the local inputs before updating the struct. This ensures that the logic in the main loop acts on the most recent information available both locally and from the other node. In the case of the sampler, this means that the lidOpen value should be updated according to the position of the stepper motors, and the height value should be updated according to the readings from the analog to digital converter. Note that

the sensor's onboard digital to analog converter has the same resolution as the analog to digital converter on the ATmega328, meaning that the value can be read directly with no need for processing. In the case of the base, no local values need to overwrite the incoming data values because the base doesn't read any sensors. Listing 1.3 shows the implementation of this function on the sampler, while Listing 1.4 shows the implementation on the base.

Listing 1.3: *Data Exchange Code: Sampler updateState Implementation*

```
void updateState(status_t* current_status, char* inputStr) {
  StaticJsonBuffer<400> jsonBuffer;
  JsonObject& root = jsonBuffer.parseObject(inputStr);
  if (root.success()) {
    int dummy = root["mode"].as<int>();  //pull mode as integer
    current_status->mode = (state_t)dummy;  //cast integer mode as state_t
    current_status->height = analogRead(0);
    if (stepper1.currentPosition() == 0) {
      current_status->lidOpen = false;
    }
    else if (stepper1.currentPosition() == (REV * 2.0 / 8.0)) {
      current_status->lidOpen = true;
    }
    else {
      current_status->lidOpen = root["lidOpen"]; //preserve value
    }
    current_status->fanOn = root["fanOn"]; //preserve value
    outputState(Status);
  }
  if (!root.success()) {
  Serial1.print('\n');
  }
}

```

Listing 1.4: *Data Exchange Code: Base updateState Implementation*

```
1  void updateState(status_t* current_status, char* inputStr) {
2    StaticJsonBuffer<400> jsonBuffer;
3    JsonObject& root = jsonBuffer.parseObject(inputStr);
4    if (root.success()) {
5      current_status->height = root["height"];
6      current_status->lidOpen = root["lidOpen"];
7      current_status->fanOn = root["fanOn"];
8      outputState(Status);
9    }
10   if (!root.success()) {
11     Serial1.print('\n');
12   }
13 }
14
```

The output state function implementation is incidentally the same between the two nodes, because it is currently impossible for any operation to take place between the two function calls that would change the state. The output function is essentially the opposite of the update function, it takes the information about the current state in the form of a struct and transforms it into a JSON string. This string is then delimited with a newline and printed to the appropriate serial port. Listing 1.5 shows the common implementation of this function.

Listing 1.5: *Data Exchange Code: Output State Implementation*

```
void outputState(status_t current_status) {
  char temporary_string[100] = "";
  StaticJsonBuffer<400> jsonBuffer;
  JsonObject& root = jsonBuffer.createObject();
  root["mode"] = (int)Status.mode;
  root["height"] = Status.height;
  root["lidOpen"] = Status.lidOpen;
  root["fanOn"] = Status.fanOn;
  root.printTo(temporary_string);
  for (int i = 0; i < strlen(temporary_string); i++)
  {
    if (temporary_string[i] == ('}'))
    {
      temporary_string[i + 1] = '\n';
      temporary_string[i + 2] = '\0';
      break;
    }
  }
  Serial1.print(temporary_string);
}

```

1.7 Testing and Performance

1.7.1 Indoor Test Methods

I constructed a small indoor testing apparatus that suspends the attachment approximately one meter above the ground for indoor functionality testing. This testing rig allowed me to test and make improvements much more quickly than if I had to drive out to the testing site and fly the UAS each time I wanted to evaluate the system.

The KSU entomology department provided me with a sweep sample of insects that had been captured by sweeping a net across the canopy in a patch of damaged cropland. The sample contained a variety of insects of different weights and sizes. I scattered the sample on a tray, placed it below the sampling system, and turned it on. Once the sampling system had completed its routine, I removed the sample jar and visually inspected the insects had been captured by the suction and compared them to the insects that remained on the tray.

1.7.2 Results

My tests initially revealed some minor errors in our design. These were primarily software bugs that caused the system to behave in an unexpected manner under certain specific circumstances. Once these problems were revealed in testing I was able to rework the firmware so that the system functioned as intended. Once these issues were resolved, the system successfully captured 100% of the insects in the sweep sample regardless of size, weight, and species. Repeated execution of the procedure revealed no inconsistencies or remaining design flaws, and the battery life of the system during normal operation was demonstrated to be greater than the typical flight time of 20-30 minutes. A photo of the captured insects is shown in Figure 1.14.

1.7.3 Outdoor Test Methods

I performed our outdoor testing at the Kansas State University Agronomy North Farm. My first set of tests revealed that the UAS remote control and communications system operated on the same 2.4 GHz band as our XBee radios, so I simply exchanged the XBee radios for 900 MHz models with the same footprint and pinout. This fixed the communications problem. A photo of the apparatus in flight can be seen in figure 1.15. The figure shows the apparatus being tested shortly before I made improvements to the cable management and lengths.

Figure 1.14: *Bugs Collected in Lab Tests*

This system works well on the ground while the UAS is powered on, but when the UAS is in the air the base station fails to send the appropriate commands to the sampler. A series of tests to rule out possible causes eliminated all possibilities except interference from the additional EMI created by the extra current drawn to the brushless motors on the UAS while lifting a full load.

Figure 1.15: *Flight Test*

1.8 Conclusions and Recommendations

In this chapter, I have demonstrated a full design for a prototype device capable of performing sampling operations and collecting insects via suction in indoor bench tests. I have also presented an analysis and diagnosis of the barriers preventing the system from operating in flight. Future work should focus on reducing the impact of environmental EMI from the UAS. One possible solution is to create an EM shield to protect the circuitry in the device from EMI. An alternative solution is to reduce the weight of the system and thereby reduce the load on the UAS motors, this will reduce the current draw and therefore reduce the EMI generated by them. The solution most likely to be successful will incorporate both of these strategies to minimize the system's sensitivity to EMI.

Chapter 2

Integrated IMU and GNSS System for Improved Image Geolocation

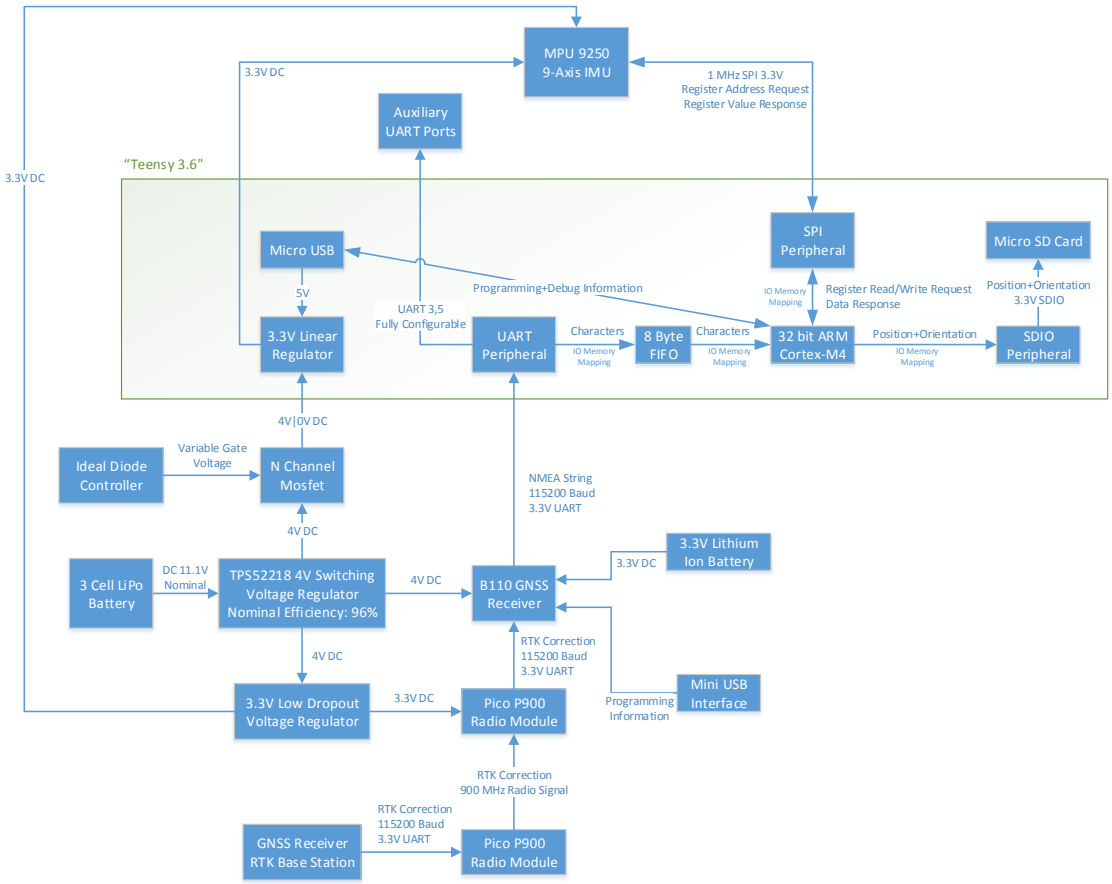
2.1 Introduction

As part of a larger initiative to improve the process of collecting aerial imagery to assess crop health, we are investigating ways to quickly and efficiently geolocate the images that we collect during multirotor flights over cropland. Onboard GNSS systems have proven to be too imprecise for these operations, and the aircraft weight constraints prevent the use of gimbals for image orientation. To correct these insufficiencies, we resolved to develop a lightweight and platform independent device that can be attached to a multirotor aircraft to record precise position, velocity, and orientation information for use in geolocating images that do not have sufficient landmarks to be identified by image stitching software. This chapter will cover the design and integration process as it pertains to the electronics and firmware of the first functional prototype of this device.

2.2 System Level Operation

The proposed solution is centered around 4 core components: a B110 GNSS Receiver, a Pico P900 Radio Module, a Teensy 3.6 Microcontroller Development Board, and an MPU-9250 Inertial Measurement Unit (IMU). The solution must also incorporate whatever additional circuitry is required to support the operation of those four components, including voltage regulators and IO interfaces. The overarching operational goal is to receive GNSS correction packets using the Pico P900, transmit them to the B110, receive NMEA strings from the B110 into the Teensy microcontroller, where the data is concatenated with an orientation quaternion obtained by performing a sensor fusion algorithm on the sensor readings from the IMU obtained by the Teensy over SPI, and output the complete data string to the Teensy's onboard microSD card. A detailed block diagram showing the system level operations of the prototype is shown in figure 2.1.

Figure 2.1: *Electronics Block Diagram*



2.3 Components Operation and Design

2.3.1 Voltage Regulators

The purpose of the voltage regulators on this device are to supply the varying input voltages required by each component by efficiently reducing the voltage from either a 11.1V nominal battery or a single cell 3.7V nominal battery. All of the circuitry on this board can be powered by two input voltages, one in the range from 3.6V-4.5V and another at 3.3V. This system includes two voltage regulators to supply these needs, an 8A 4V switching regulator and a 3A 3.3V low dropout linear regulator. If the board is being powered by a single cell battery, the switching regulator can be bypassed entirely, as the battery voltage is already within the desired range.

2.3.1.1 Operation Theory

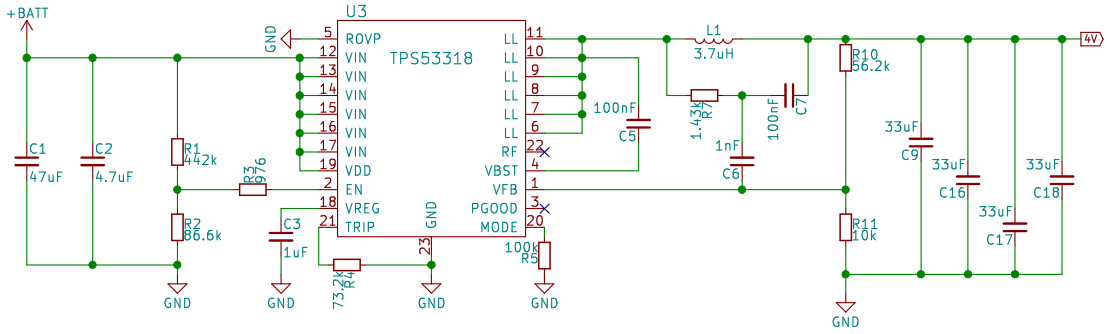
The switching regulator works by switching an integrated mosfet on and off at 500 kHz, with the output current flowing into the load through an inductor. Because current through an inductor cannot change suddenly, when the switch turns off the energy from the collapsing magnetic field in the inductor draws current from ground through a diode, allowing the circuit to drive the load even when the switch is closed. This effect inherently creates a ripple at 500 kHz that must be managed by a large bank of low ESR ceramic capacitors at the output. The linear regulator takes this 4V output as an input, and uses a comparatively simple analog circuit to reduce the input voltage to 3.3V while the current remains almost the same from input to output. This causes the linear regulator to be significantly less efficient than the switcher, but it outputs a much cleaner voltage and requires significantly less output capacitance.

2.3.1.2 Component Selection and Design

The primary design concerns when designing the switching voltage regulator were board space, efficiency, and weight. Because all of the power drawn by the board passes through

this circuit when powered by the main battery on the multirotor, power efficiency could have an effect on the total flight time. Because of concerns with payload weight and size, I looked for a solution with few external components and simple implementation. I decided to use the Texas Instruments TPS53318 because it is capable of up to 91% efficiency and comes in a small 5mm by 6mm QFN package. The MOSFETs for power switching are integrated into this IC, reducing the external component count. It also has large voltage and current ranges of up to 8A and 1.5V to 22V, making it flexible if the design should change later on in the prototyping process. Figure 2.2 is a schematic of the connections to the TPS53318 IC and all the external support components that form the switching voltage regulator.

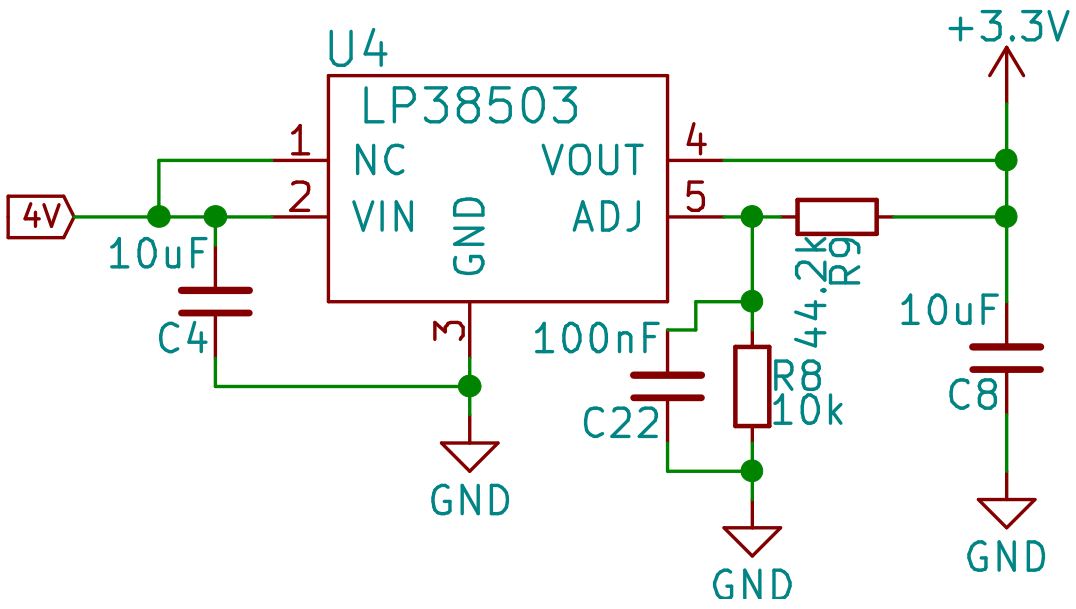
Figure 2.2: *Switching Voltage Regulator*



The primary design concerns when designing the linear regulator were dropout voltage and current rating. In a linear regulator, the dropout voltage describes the minimum difference between the input voltage and output voltage that can exist while still allowing the regulator to maintain the desired output voltage. In our circuit, the switching regulator or fully charged battery outputs about 4V and we need a stable source of 3.3V from a regulator that takes this 4V supply as an input. This means that the linear regulator must have a bare minimum dropout voltage of 0.7V. Preferably the dropout voltage would be lower than this because battery voltages can drop below their nominal values while they are being discharges, so a lower dropout voltage will increase the time that our circuit can be safely powered by a single charge of a one cell battery. The other concern is current capability. Most voltage regulators I found with low dropout voltages had relatively low current ratings. Our low dropout regulator needs to power an IMU and a radio module. The main issue is

that the radio module's current needs can fluctuate based on its configuration. A customer service representative from Microhard advised me that the radio could draw as much as 2A depending on a variety of factors, so I started looking for regulators with suitable dropout voltages and current capabilities greater than 2.5A to give some headroom in case of incidental jumps in power demand. The LP38503 Low Dropout Regulator from Texas Instruments meets these two requirements with a maximum dropout voltage of 550mV (420mV typical) and a current rating of 3A. Figure 2.3 shows our circuit design based around the LP38503 with the necessary feedback voltage divider to set the output voltage and capacitors to ensure stability and reduce noise.

Figure 2.3: *Low Dropout Voltage Regulator*



2.3.2 GNSS

The GNSS receiver in this system is a B110 receiver from Topcon. The subject of GNSS is exceptionally complex, so I will restrict this paper to a cursory overview. The basic principle relies on a large number of satellites orbiting the earth in predictable orbits with extremely precise clocks. They use radio signals to transmit their time and identification number. If a

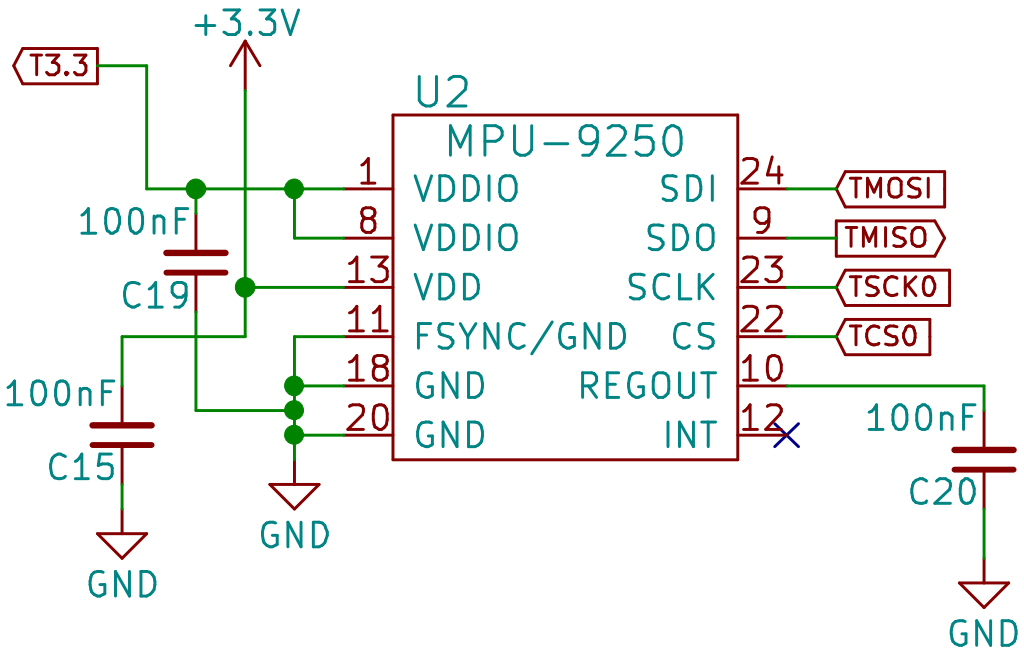
receiver can see three or more satellites, it can solve a clock correction algorithm to determine its own local time precisely and then compute the distance to each satellite based on the clock time differences. From this information, it can use a process called trilateration to estimate its own position. There are several problems with this approach that introduce error, chief among them being ionospheric interference in the signal. Sophisticated devices like the B110 have a series of strategies to mitigate these errors, some requiring another stationary GNSS receiver on a base station to communicate with the B110 via a radio connection. We have configured the B110 to receive its error correction information from the radio over a simple UART connection to a radio module, and output an NMEA string over another UART to our Teensy development board.

2.3.3 Inertial Measurement Unit (IMU)

Inertial measurement units such as the popular MPU-9250 use MEMS (MicroElectroMechanical Systems) sensors to implement three axis accelerometer, gyroscope, and magnetometer sensing inside the IC chip. This is achieved by measuring the electrical properties (such as capacitance) of tiny internal mechanical elements that are affected by motion and ambient magnetic fields. This process requires internal analog to digital converters, embedded signal processing, and other circuitry elements to support operating requirements such as charge pumps, buffers, and voltage regulators. The MPU-9250 is an extremely robust example of this technology that includes advanced features such as internal programmable sensor ranges, signal processing, and extremely fast 1MHz SPI communication for reading and writing its registers for both configuration and data retreival. It is also extremely small, in a 3mm by 3mm QFN package. The MPU-9250 requires only a three external capacitors and communicates with our Teensy board using a SPI bus at the maximum supported speed of 1MHz. The MPU-9250 allows its IO voltage level to be different from its operating voltage, provided the prior is the lower of the two. I took advantage of this feature when designing the board, and I was able to keep the board to two layers and avoid cutting the ground plane under the SPI traces by sourcing VDDIO from the Teensy 3.6 onboard regulator and VDD from

the LDO discussed in section 2.3.1. Figure 2.4 shows the schematic of the MPU-9250 as it is used in my design.

Figure 2.4: *IMU*



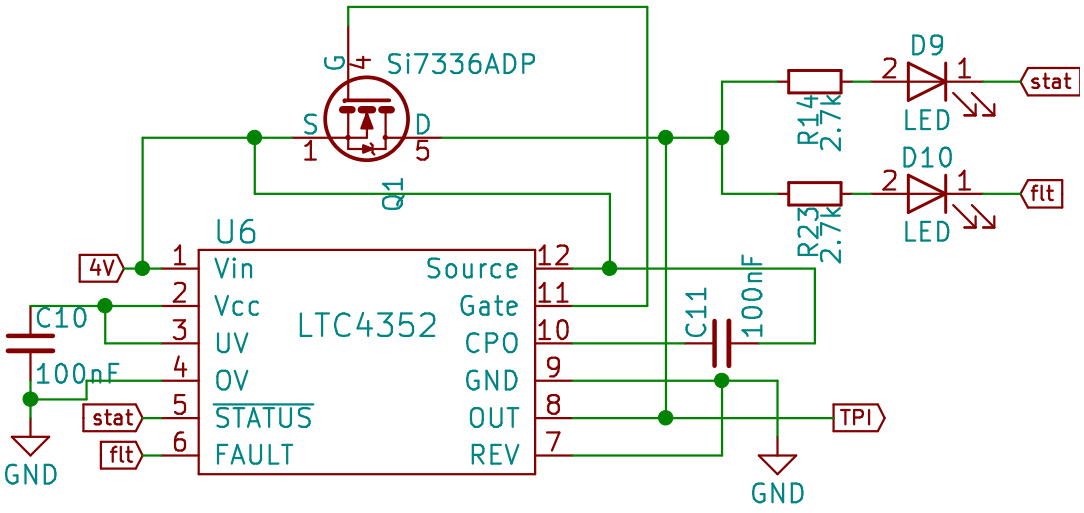
2.3.4 Teensy 3.6

The Teensy 3.6 is a development board that houses an ARM Cortex-M4 microcontroller. It is set up by its manufacturer with a loader application that allows the M4 processor to be configured and programmed using the Arduino development environment. In the case of this project, it performs all input and output operations and processes data from the IMU and GNSS receiver into a useful format. This includes running the open source Madgwick sensor fusion algorithm at a 1KHz timestep to filter the IMU information into useful orientation estimates.

2.3.5 Ideal Diode

The Teensy 3.6 can be powered either by the 4V regulator on the board, or through a micro USB cable. To allow both power sources to be safely active simultaneously, I used an ideal diode controller with a low R_{DSon} mosfet to create a circuit that very closely mimics an ideal diode by sensing voltages on either side of the mosfet switch and opening or closing it accordingly. When the USB power is unconnected, current can flow through the mosfet from the 4V switcher into the Teensy V_{in} connection. However, when the USB power is connected, the mosfet is closed and the 5V USB power is completely isolated from the 4V output of the switcher. The schematic for this section of the circuit is shown in figure 2.5.

Figure 2.5: *Ideal Diode Circuit*

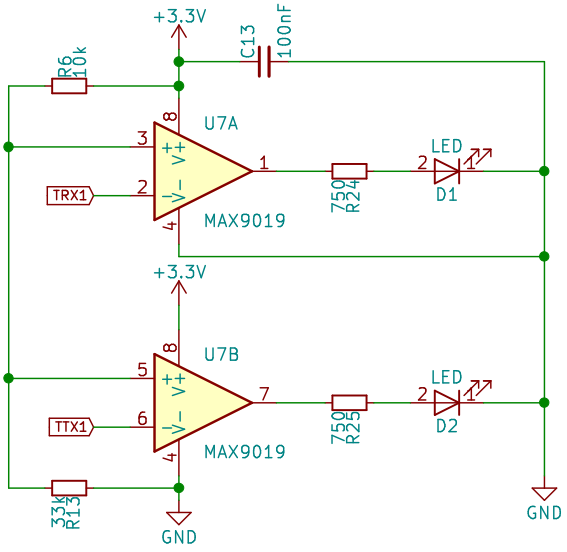


2.3.6 Indicator LEDs

The B110 outputs operate at LVTTL, which essentially means that they have very little output current capability. For this reason, I decided to drive my UART indicator LEDS with a pair of comparators with well defined input resistance characteristics. I selected the MAX9019 dual comparator because it contains two comparators in a single IC and it retains its accuracy even when the input voltages extend marginally beyond the supply voltages (sometimes called "rails"). This is important because the B110 has its own onboard

regulators, and I cannot guarantee that my nominal 3.3V supply will not be marginally below the B110's logic level voltage. If this were to be the case, most comparators would be operating in a range of undefined behavior, because the inverting input voltage would be larger than the rail voltage. I used a resistor divider to create a 1.1V reference voltage on the noninverting inputs of the comparators and connected Tx and Rx from the B110 to the inverting inputs of the comparators. At the output, I placed a current limiting resistor and LED. This circuit results in the LED lighting up when the UART line is low and data is being transmitted. The schematic for this section of the circuit is shown in figure 2.6.

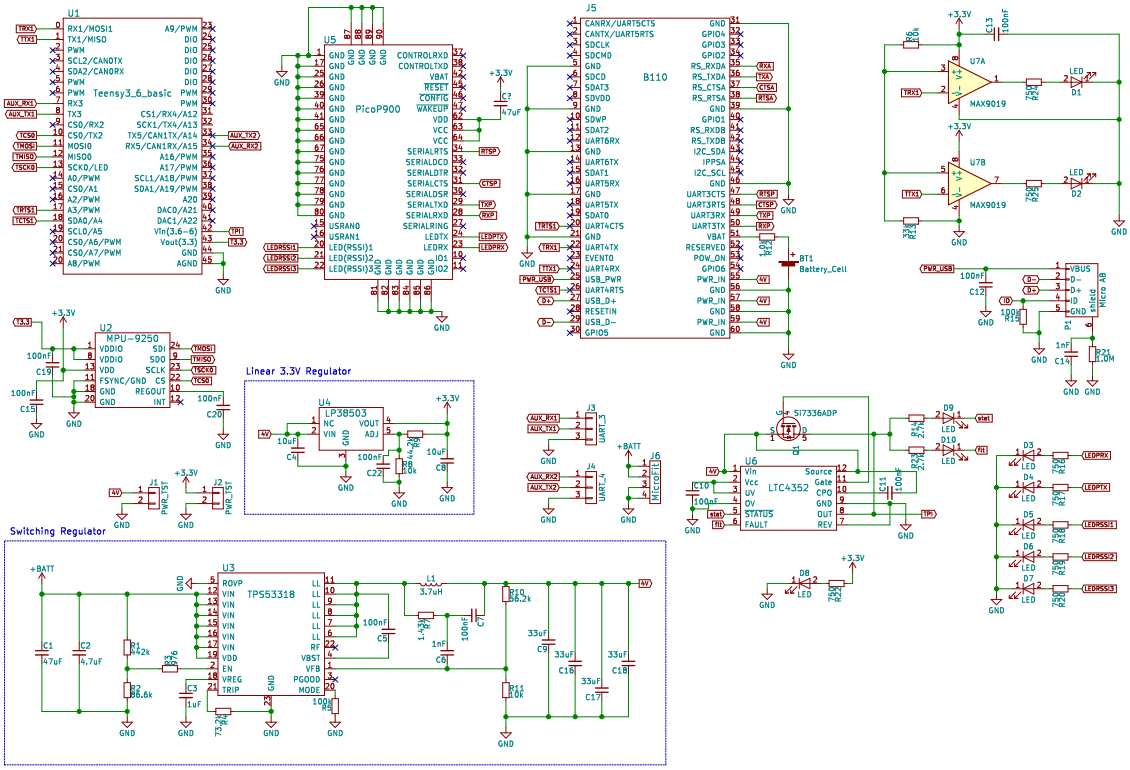
Figure 2.6: *UART Indicator LED Driver*



2.3.7 Full Schematic

With inclusion of all components listed in this section, the full design schematic showing all electrical connections is shown in figure 2.7. All schematic designs shown in this document were created using a free and open source electronics design suite called KiCad. Specifically, this schematic was created in a subprogram called Eeschema.

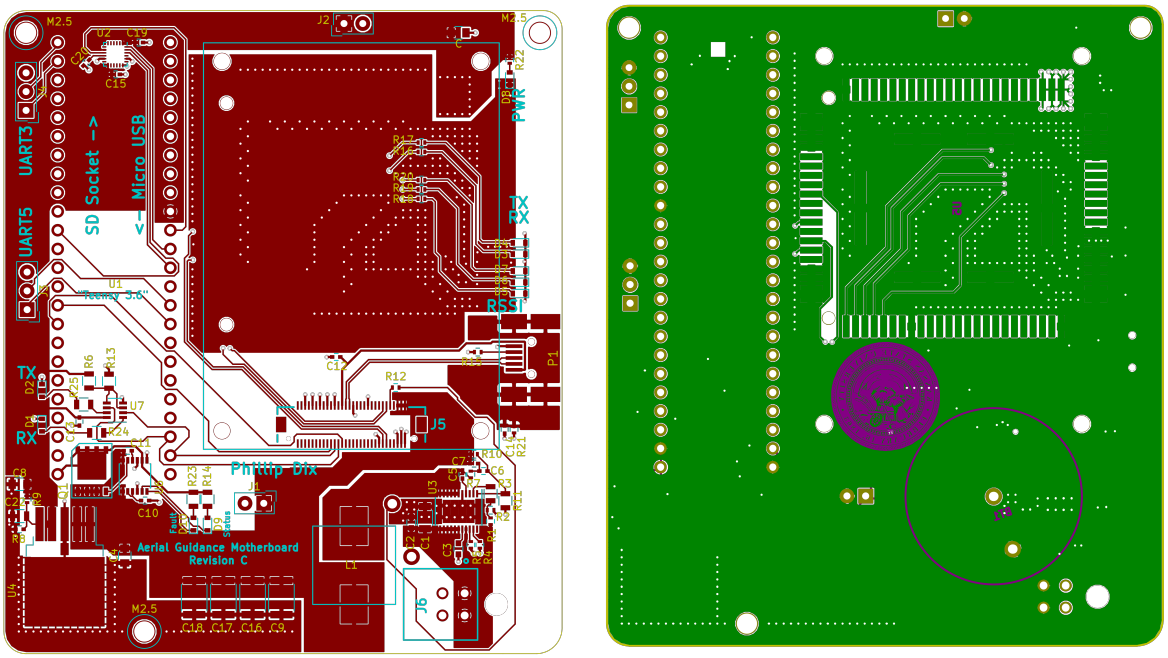
Figure 2.7: *Full Design Schematic*



2.4 Physical Design and Board Layout

The final product of this work takes the form of a printed circuit board that serves as a motherboard for all components shown in figure 2.7. I performed the circuit board layout in KiCad, specifically a subprogram called PcbNew. Careful component placement allowed all the connections to be made on a two layer board for superior cost and manufacturing lead times while preserving the ground plane under all signal traces. I gave careful consideration to power traces and planes to minimize the inductance of any copper shapes carrying significant current. For the USB connection traces, I used a feature of KiCad called "differential pair routing" to ensure that the D+ and D- traces have exactly the same length, and I placed the relevant components in such a way as to minimize the length of those traces. This endeavor produced the printed circuit board shown in figure 2.8.

Figure 2.8: *PCB Layout*



2.5 Firmware

During this stage of the project, our first priority was to show a functional prototype within our time constraints, and so our firmware development was focused on developing a minimum viable solution very quickly. Development speed is important in this case because other researchers at KSU need a device meeting the minimum specifications to perform evaluations on the accuracy and effectiveness of the B110 GNSS receiver for their research applications.

2.5.1 Platform

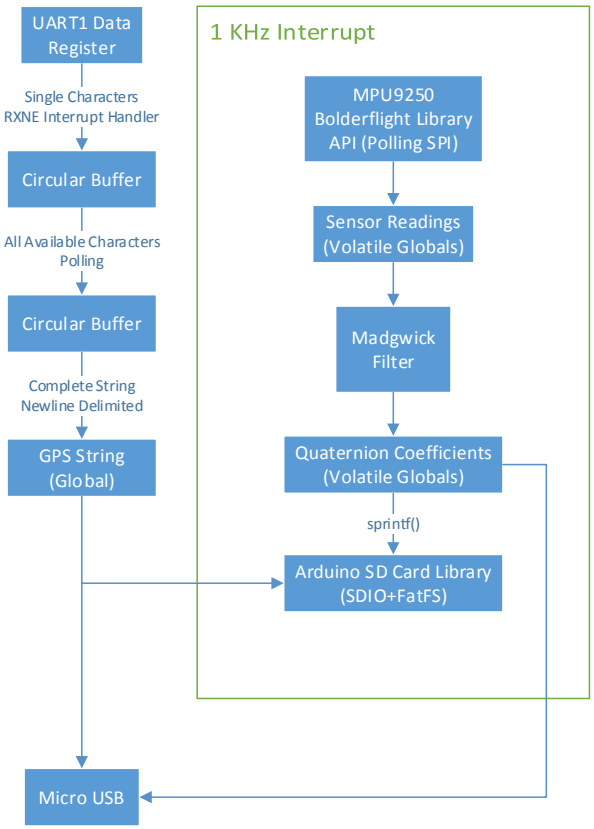
We decided to develop the first prototype firmware using the Arduino environment to program a PJRC Teensy 3.6 development board. As discussed in chapter 1, the Arduino environment has the benefit of a large user base and hardware abstraction, at the cost of precision and performance. We saw huge value in two particular open source libraries for the Teensy 3.6 in Arduino. From a communication perspective, the two most difficult items on the board for the processor to interact with are the MPU-9250 and the micro SD card slot included on the Teensy 3.6. These both require SPI communication, and reading any data from the MPU-9250 requires sending a one byte request message containing the address of the register containing the desired value. Obviously this is a cumbersome task to program, as it requires mapping of the MPU-9250 registers and interpretation of the returned binary values. The software package that allows Arduino to interface with the Teensy board (called "Teensyduino") includes a library with a simple class to read and write files on the SD Card. There is also a library available under the MIT license from Bolder Flight Systems that handles all interactions with the MPU-9250 using a simple API. It is these two libraries that primarily led us to choose the Arduino platform over a more traditional microcontroller.

2.5.2 Architecture

The firmware for this device has two main sections: a 1 KHz interrupt handler that performs most of the calculations, and a series of serial interrupt driven buffers that process incoming

strings. A third more trivial element of the firmware is the main loop, that outputs debugging information to the micro USB port. Figure 2.9 is a block diagram showing the order of the tasks performed by the firmware and the interactions between the interrupt handler and the rest of the code.

Figure 2.9: *Firmware Block Diagram*



2.5.3 Input and Output

2.5.3.1 UART Serial

In Arduino, all serial communications are automatically handled by a serial buffer object. These circular buffers are driven by interrupt handlers bound to the serial port's register status and are generally opaque to programmers using the platform. Arduino is typically

used to program microcontrollers with relatively small amounts of memory, so these buffer objects have very small capacities. NMEA strings on the other hand, can be somewhat large, leading to concerns that the serial buffers might not be large enough to manipulate the B110 output. Because the Teensy 3.6 has about 1 MB of memory, there is little concern for the memory space consumed by these buffers. I examined the supporting files used by the Arduino IDE and found the location in a file named serial1.c where the input and output buffers are declared for UART1. I changed their size from 40 (TX) and 64 (RX) to 4096 to avoid any chance of overflow and subsequent data loss.

2.5.3.2 SPI

The Bolder Flight library for reading the MPU-9250 uses blocking code to read and process the data from the IMU registers. This means that the processor must wait for the read operation to be completed before performing any other calculations. Changing this operation to run either via processor interrupts or direct memory access (DMA) should be a subject of focus in future work on this project.

2.5.4 String Processing

In my main loop, I continually read all available characters out of the modified serial object into my own circular buffer implementation build using C structs. My implementation of the circular buffer checks for delimiting characters, and is able to easily detect when a complete message is available in the buffer. This approach allows the 1 KHz interrupt to easily detect when a new string is available and operate on it with minimal processing requirements. All strings from the B110 are delimited with newlines, so the buffer is set up to detect newlines and use them as the basis for when strings begin and end.

2.5.5 Sensor Fusion

The inherent error introduced by each sensor in our IMU is a significant problem when attempting to estimate the orientation of an aircraft. Accelerometers can effectively use

gravitational acceleration to detect the downward direction in an inertial reference frame, but any acceleration due to speed changes or turns will cause temporary errors in the reading. Similarly, gyroscopes detect short term changes in angular rate very well, but their reading drifts over time because the mathematical procedures used integrate the error they produce. Magnetometers provide a good reference direction for magnetic north, but are susceptible to local distortion in magnetic fields caused by certain nearby objects with electromagnetic properties. In this application, I've used an optimized C implementation of the Madgwick Filter developed by Sebastian Madgwick at the University of Bristol[14]. The algorithm uses a series of mathematical methods to approximate the gyroscope error using magnetometer and accelerometer readings, and ultimately filter all the data together to achieve very accurate and stable estimates of orientation. The performance of a Madgwick filter rivals that of traditional Kalman Filter based approaches but with significantly less computational intensity or memory usage.

2.6 Results and Conclusions

Upon testing the circuit board for functionality, we discovered the need for additional output capacitance on the linear regulator, and additional decoupling capacitors near the P900 Vin pads. We added an additional 47 uF capacitor to each of these locations, and it stabilized the circuit. Further testing outdoors revealed that when the B110 reads more than about 8-9 satellites with the GR5 base station switched on, the 4V power supply is prone to sudden brown outs. These appear to be caused by sudden current demand that the switching regulator is unable to compensate for. The solution to this problem was simple, as we discovered that from a flight logistics standpoint it is much more convenient to operate this board powered by its own single cell battery. We simply scrapped the 4V regulator because it was adding extra weight and causing unwanted behavior for no measurable benefit. When powered through an independent single cell battery, the system performs all desired functions and has demonstrated the ability to achieve an RTK correction solution and properly log the desired orientation and position information. A more detailed evaluation of the performance

parameters of this system is reserved for future work.

Bibliography

[1] Thomas Henneberry. Insect pest management. *Encyclopedia of Pest Management*, pages 1–3, 2003.

[2] E-C Oerke. Crop losses to pests. *The Journal of Agricultural Science*, 144(1):31–43, 2006.

[3] JK Kreuger and N Brink. Losses of pesticides from agriculture. *Pesticides: food and environmental implications. International Atomic Energy Agency, Vienna, Austria*, pages 101–112, 1988.

[4] Manuel Arias-Estévez, Eugenio López-Periago, Elena Martínez-Carballo, Jesús Simal-Gándara, Juan-Carlos Mejuto, and Luis García-Río. The mobility and degradation of pesticides in soils and the pollution of groundwater resources. *Agriculture, Ecosystems & Environment*, 123(4):247–260, 2008.

[5] Jay Ram Lamichhane, Silke Dachbrodt-Saaydeh, Per Kudsk, and Antoine Messéan. Toward a reduced reliance on conventional pesticides in european agriculture. *Plant Disease*, 100(1):10–24, 2016.

[6] Peter WJ Baxter and Hugh P Possingham. Optimizing search strategies for invasive pests: learn before you leap. *Journal of Applied Ecology*, 48(1):86–95, 2011.

[7] Lars WJ Anderson. Californias reaction to caulerpa taxifolia: a model for invasive species rapid response. *Biological Invasions*, 7(6):1003–1016, 2005.

[8] Jan Rudolf Karl Lehmann, Felix Nieberding, Torsten Prinz, and Christian Knoth. Analysis of unmanned aerial system-based cir images in forestrya new perspective to monitor pest infestation levels. *Forests*, 6(3):594–612, 2015.

[9] Chunhua Zhang and John M Kovacs. The application of small unmanned aerial systems for precision agriculture: a review. *Precision agriculture*, 13(6):693–712, 2012.

[10] DJI. Spreading wings s1000 user manual v1.10, 2014. Retrieved from www.dji.com.

[11] Eastman. Technical data sheet eastman amphora 3d polymer am3300, 2015. Retrieved from colorfabb.com.

[12] RoboKits India. High torque dc geared motor specifications, 2014. Retrieved from robokits.co.in.

[13] ECMA International. Standard ecma-404 2nd edition, the json data interchange syntax, 2017.

[14] Sebastian Madgwick. An efficient orientation filter for inertial and inertial/magnetic sensor arrays. *Report x-io and University of Bristol (UK)*, 25, 2010.