UNIVERSITY OF CENTRAL FLORIDA

College of Engineering and Computer Science

Knight's Intelligent Reconnaissance Copter - KIRC

EEL 4914 – Senior Design I
Preliminary Design Review
Fall 2013
Group #14

Nathaniel Cain
James Donegan
James Gregory
Wade Henderson

Table of Contents

1. <u>Introduction and Executive Summary</u>

In this document, the design, analysis, and testing results are detailed for the Knight's Intelligent Reconnaissance Copter (KIRC) project. Information about the team, references, and the project sponsors are also included within this document in the appendices section.

KIRC is a UCF Senior Design project unofficially sponsored by the National Aeronautics and Space Administration (NASA) that includes two quadcopters that will image an area autonomously for earth science missions. This project is intended to test the Delay Tolerant Networking (DTN) protocol developed by another university that is being studied by NASA. DTN is a virtual networking protocol that excels in areas where communication networks can tend to have long delays or disruptions. The goal is to test the protocol in an active field setting in order to gain more experience with it as well as provide two quadcopters for later missions. Currently, the quadcopters developed in this project will be used during some upcoming NASA missions such as the Pavilion Lake Research Project, in Canada.

The common goal of this project is to provide two quadcopters that can be reconfigurable on a mission-by-mission basis, and be autonomous to complete any task required. As shown in figure 1 below. The two quadcopters as well as the ground station computer will form a three corner mesh network. With this arrangement, the goal is to illustrate DTN's ability to dynamically route messages and bundles of information to any member of the triangle either directly or indirectly.

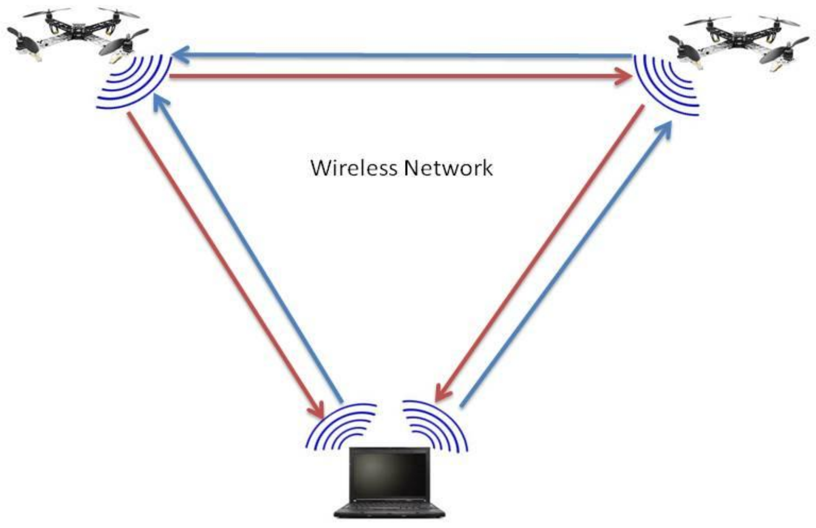


Figure 1: System Overview Diagram

2. Project Description

2.1. Motivation & Sponsorship

KIRC started as an idea during the summer of 2013 as a potential student project for a Co-op student working for the Center Planning and Development Office at NASA's Kennedy Space Center. This project evolved into a potential senior design project as it grew more and more. While this project is being funded by NASA, it still remains an unofficial NASA project. The ideas behind the project were created to achieve multiple NASA goals, as well as provide future.

The main motivation behind this project is to test an experimental virtual communication protocol, DTN2, developed by University of California, Berkeley. This Delay Tolerant Networking, DTN2, architecture is built for environments where communication is prone to delays and disruptions. During deep space missions, DTN2 is ideal for the long distances and the intermittent links between planets. For this reason, NASA is interested in implementing this software on much of its upcoming space missions, which is why they are motivated to learn more about it. Another main motivation, parallel to the first, behind this project is to provide two autonomous quadcopters for imagery during the Pavilion Lake Research Project in the summer of 2014. NASA decided to combine the two goals into one project, hence satisfying both goals in a single project. These quadcopters will patch together an image of an area cooperatively while transmitting telemetry over a wireless network using the DTN2 protocol. Control of these quadcopters will be done either manually or by using a computer to send commands to the autonomous system. The operating systems of both the navigation portion of the quadcopter and the ground station laptop will be Linux distributions.

The team was provided unofficial sponsorship from NASA and an unofficial budget of about $1000 for certain parts. With the unofficial budget provided, the team has been tasked with the construction of 2 quadcopters with full autonomous capability, a wireless DTN2 network, and the ability to send images to the ground for stitching. Over the course of senior design, the group will research, design, build, and test a system that will perform the tasks set forth by NASA. This is a difficult project at best, and requires application in the areas of control, power, software development, operating systems, and computer networking.

One of the important details in the project is that the team needs to make two custom built quadcopters capable of autonomous flight. The reason behind making a custom built copter versus buying a commercial off the shelf (COTS) quadcopter is that the custom control will make the guidance portion of the project easier. With COTS items, many vendors will not make the software open source to the public. This complicates the design of the guidance control algorithm significantly, and also makes the software less predictable. Another reason is that quality COTS quadcopters tend to be quite expensive. With all this in mind, NASA decided that it was in its best interest to design a custom quadcopters for this project.

While the Pavilion Lake Research Project is the initial application of this project for NASA, it will also be planned to be used in future projects as well. For this reason, the project system should be designed to be reconfigurable based on open ended mission needs. A long term goal of the project is to provide a modular payload that can be a platform for the core services of this project, but be able to fly other UAVs. This goal will not be accomplished during the course of Senior Design, but will be a future application of the project.

Overall, this project is a fairly open ended one. It has several goals, which NASA wants achieve, all within a single project. The rest of this paper will contain the analysis and discuss the challenges associated with achieving NASA's goals. Having discussed NASA's sponsorship and motivations behind this project, the next sections will discuss the team's goals followed by the requirements.

2.2. Goals & Objectives

NASA's primary goal in this project is to have two autonomous quadcopters that can image an area based on commands entered over a wireless network with DTN2 as the network architecture. Based on these goals, we devised sub-goals that will help us achieve these main goals. In this section, our goals and objectives are outlined and how we plan to meet these criteria. At the end of the section, a summary of these goals and objectives are provided in an outline form.

The primary goals and objectives of the project include the following: being lightweight, durable, adequate flight time, stable flight, ease of manual flight control, consistent autonomous flight, must use DTN2 application, two final quadcopters, and flexible software that can be easily based on mission requirements. These core objectives make up the description of how the quadcopters should perform as a polished product, and they are hard goals. This means that above all else, these are the goals that the team must meet more than the others. At this point, it also may be useful to define what autonomous flight means in the context of this project. For this project, autonomous flight means that the finished quadcopters can fly to a user defined coordinate without need for a human pilot. It also means that the quadcopter can make changes to the flight plan, return home, or relay information without need for human intervention.

The quadcopters will have several features that can be entered over the network via a command terminal. There will be commands that tell the copter to fly-to a location, or take a snapshot. Some of the larger commands will be to input four coordinates and image the bounded area inside those coordinates, or to cooperative image an area. There will also be safety commands such as return home, hover, or land quadcopter. The home location should be set every flight. All the commands will be bound into a program that will be stored on a ground computer. The parsed commands will then be sent to the quadcopters where they will be interpreted as instructions.

The imagery portion of the project will be done using a simple bread-crumb trail GPS coordinate system. After the quadcopters' computers boot up, a user can input commands. One of the prime features of the quadcopter is to autonomously image an area based on a four coordinate boundary. In order to do this, the plan is to have each quadcopter choose half of the boundary. From this, each quadcopter will make a flight path that will include locations to stop, hover, take a picture, and move on. The locations where the quadcopters should decide to stop at will depend on the horizon of the camera and the altitude of the quadcopter. Each of these locations will be calculated real-time based on these parameters.

The KIRC Project is split into three prototyping phases. Ultimately, the team is planning to produce two completely autonomous reconnaissance quadcopters. The first phase is to create a manually controlled prototype that will allow us to refine our flight stability algorithms before moving to an autonomous flight system. The second phase is to convert the quadcopter into KIRC, a completely autonomous reconnaissance copter. Phase three is to replicate KIRC, naming it SPoK (Secondary Partner of KIRC). At the end of this project, the goal is to have both KIRC and SPoK working cooperatively to image an area.

During phase 1, the plan to achieve stable manual flight. This will be done using a microcontroller as a means of digitally controlling the stability of the quadcopter. The microcontroller will also be using a Real Time Operating System (RTOS) in order to achieve a deterministic processing pattern. In order to make the quadcopter dynamically stable, the plan is to use a discrete-time proportional-integral-derivative (PID) loop to control its roll, pitch, and yaw. The inertial feedback of attitude data will be achieved using a 10 degree of freedom (10DoF) IMU. Once this part of the project is done, a printed circuit board (PCB) will be designed with all these previous parts integrated on board.

During phase 2 of the project, the goal is to make the quadcopter autonomous. This will be done through the use of a second computer with an embedded Linux distribution. This computer will take GPS coordinates from the stability computer and compare them to a desired position. Based on this error, a heading and distance to the target location will be established in real time, while using the magnetometer to steer the quadcopter. Other tasks of the embedded Linux processor will be to handle all communications between the quadcopter and the ground, as well as retrieving images from a mounted camera device and relaying them. With this phase complete, the first copter, KIRC, will be complete.

During phase 3 of the project, the primary objective is to replicate KIRC, naming it SPoK. There will also be considerable programming of the autonomous features of both quadcopters during this phase. The goal by the end of the project is to have the quadcopters working cooperatively towards a common task. During this phase, the team will also be working on making a usable software set for the ground station that will facilitate all user commands to the quadcopters. A software program that stitches images together will be used to stitch together all the images from a given flight. Images will have to be coordinate stamped in order to provide enough information to stitch a composite image together.

Some of the objectives of this project were learning objectives. The microcontroller must be an ARM processor, and there must be a Real Time Operating System on it.

In summary, the primary goals that the team would like to achieve related to performance of the quadcopters have been defined. The team also has smaller goals and objectives during this project, which are mainly driven by certain pieces of hardware or software that they would like to implement for learning experience. Success in this project will be determined by the ability for the team to meet the goals and objectives of this project by the spring 2014 Senior Design II deadline. A summary of the goals and objectives is listed below.

Goals & Objectives Summary:

- Lightweight and durable
- Ability to be manually controlled easily
- Dynamically stable in flight
- Flexibility of the software to be reconfigured on a mission-by-mission basis
- Consistent, accurate, and stable autonomous flight
- Two quadcopters that are identical to each other
- The user will also be able to address each copter individually for instructions to fly-to, image, or hover-at a coordinate
- Copters must individually and collectively be able to image a coordinate or area based on user input of four GPS coordinates representing a boundary to map
- Fail-safe commands to send to the quadcopter to land on the ground, or return home
- Ability to set home location easily
- The project should demonstrate the main features of DTN: data hopping over a mesh network, store and forward, and bundle handling.
- Use an imaging camera that is capable of reasonable clarity and is light and mobile enough to be mounted on a quadcopter
- Use an embedded Linux processor (Off the Shelf, not integrated as part of our PCB)
- Use an ARM processor for the flight stability computer
- Use DTN2 software on the Navigation and Ground computers for our Network Layer
- Use an image stitching software that can stitch together a composite image from multiple coordinate stamped images
- Use a Real Time Operating System (RTOS) on our ARM processor for deterministic behavior
- Implement digital PID loops to provide compensation to motors for control
- Develop or use an existing algorithm for filtering the noise from the IMU sensors
- Use PWM as the control signal going from the ARM processor to an Electronic Speed Controller (ESC) for each motor

2.3. Requirements & Specifications

Now that the major and minor goals and objectives have been defined, the requirements and specifications can also be defined. These will include particular performance criteria, specific hardware, and software schemes that we wish to use. A summary of all the parts, software, and performance requirements will be given at the end of this section.

Performance will be defined by the flight time per charge, durability of the system, maximum flight altitude, rate of climb, cruising speed, and wind requirements. We would like our quadcopters to stay in the air for 10-15 minute intervals before having to recharge. NASA did not provide any specific performance requirement for our project, but we have some requirements that we would like to meet.

During flight, the quadcopters must be able to withstand dropping on the ground from a small height of about 3 to 4 feet. While being able to withstand a large drop would be desirable, it is not realistic within the design constraints of the project. The quadcopter's climb rate and cruising speed should be adequate enough that the navigation of a large area can be done without too many charging waits. The quadcopter should also be able to fly in light winds of 0-5MPH and be stable enough for autonomous navigation. The maximum flight altitude should be around 150 feet from ground level in order to provide adequate imaging range. All these requirements make up the performance characteristics for the quadcopters in our project. While these characteristics are important, some of these requirements are more important than others. The main hard requirements are the flight time, flight altitude, and wind requirements. All the other performance requirements are less critical as long as they appear to be adequate.

For this project, the team has certain requirements on the parts used as a result of the NASA part of the project (for a list of reasons for each part see Research and Preliminary Design in the next chapter). The reason behind using some of these parts can be a requirement from NASA, or the team's requirement in order to gain more experience for example. NASA has the requirement that the team uses an embedded Linux platform as the navigation/imagery computer. This requirement is for two reasons, one being that the DTN2 software (required software, see next paragraph) only runs on Linux, and the other being that they want that part of the hardware to be interchangeable. Other hardware requirements include the use of a digital camera on board the quadcopter that interfaces to the embedded Linux computer. This piece of hardware will be used for the imagery portion of the project, and it will be connected to the embedded Linux platform directly. One of the hardware requirements that comes from an educational standpoint is the requirement to use a separate ARM processor for the flight stability and sensor processing. The team wishes to learn more about how to program on an ARM processor in order to gain more educational experience. The team also wishes to integrate the ARM processor, sensors, and support hardware onto a single printed circuit board (PCB).

Hard requirements for the software have also been defined. The team has certain software that they are required to use, or that they wish to use. NASA's software requirement is that the team uses DTN2 as the virtual network layer of our digital network. This software will also be installed on the quadcopter's navigation computer as well as the ground station's laptop computer. NASA also requires that we find or develop a software the stitches coordinate stamped images into a large composite image for the imagery part of the project. As part of the team requirements, a Real Time Operating System (RTOS) must be used on the ARM processor. A digital control flight stability algorithm based on a discrete PID design will be implemented, and a filtering scheme to reduce the sensor noise.

In conclusion, there are many requirements from both NASA and the team. Some of the requirements are hard requirements, while some are lighter ones that are less critical. The requirements are basically split up into three categories: performance, hardware, and software requirements. Success, again, is determined by the ability to meet these requirements and specifications, as well as the goal and objectives. A summary of requirements and specifications is given below:

Requirements & Specifications Summary:

- 10 to 15 minutes of flight time per charge
- Stable flight in winds of up to 5MPH
- Maximum altitude of about 150 feet from the ground
- Survive drops from 3-4 feet
- Integrate the ARM processor, sensors, and supporting hardware onto a single printed circuit board (PCB)
- Must use DTN software

3. Research & Preliminary Design

3.1. Similar Projects & Products

There are many different ways to implement a quadcopter design, and tough decisions on system design must be made. For the KIRC project, the team had to come up with a system and figure out how far down the design had to go. At the top level, the team could have bought a commercial quadcopter and called it their own, but that wouldn't satisfy the UCF engineering requirements. At the lowest level, the team could also build much of the sensors and computer hardware themselves, but that would be an unreasonable and unnecessary approach. So, in order to satisfy the requirements set forth by UCF and still remain a viable project, the team decided to define on what level was the design to be done and what type of system to implement. The project design level and system layout can be determined by looking at others have done and drawing ideas from their projects. To get good feel for what other ideas people have implemented on their quadcopter designs, a formal market research must be done. In this section, similar products and projects are evaluated, and a formal project design level is established.

One approach to the project would be to buy a commercial of the shelf (COTS) quadcopter from the market and adapt it to the requirements of the project. Indeed, there are a lot of quadcopters available on the market from various sources such as Parallax, DJI, and Parrot. Each of these companies produce quadcopters for commercial purchase, but buying quadcopters from them would not work well in this project. First of all, when buying a commercial quad-copter will make some parts of the project very difficult. Most commercial enterprises do not make their code or design documents freely available to the public which makes customization very difficult to do. Another reason not to go with a commercial quadcopter is that they tend to be quite expensive. Most quality quadcopters that are able to perform under the requirements this project are usually at least $500 or more. With the budget being around $1000 for everything, buying two commercial quadcopters would decimate the budget and not leave any room for the other parts for imagery, autonomous flight, and the PCB. Unfortunately, these commercial quadcopters don't really leave much in terms of technical ideas to draw from either. With much of the technical details of the quadcopters not available to the public, it makes gathering information from these types of quadcopters impossible. The only thing that is listed that helps is the performance characteristics. Using this information, the team was able to make realistic design constraints on the performance of the quadcopter under certain conditions.

In the website for previous electrical and computer engineering senior design projects at the University of Central Florida, there were several teams of students who made UAVs. One in particular had a single rotor and was encased in a wire cage which made for a very interesting design. Automatic Flying Security Drone, or AFSD for short, was intended to provide a UAV to security forces to 'patrol' an area autonomously. They had some novel designs that intrigued our senior design team. In order to maintain attitude control and steer the device, they used small rudders under the rotor to deflect the air and create a small controllable force. With the cage design holding in all the electronics and keeping the motors out of reach, it made for a very safe design. The project also proves to be very innovative with its rudder design. While this project did have a few good points to draw from, unfortunately it wouldn't fare well in a higher altitude application. At higher altitudes, wind becomes a large issue, and it could create a potential problem where the cage design would tumble in the wind and the rudder for the motor couldn't compensate enough. This could cause a potential crash of the copter system and may result in the total loss of the system. Another drawback is the copter's limited cruising speed. The rudder design, while novel, doesn't provide the force necessary to push the copter to higher speeds. For this project, the cruising speed must be enough such that large areas can be covered with a limited battery life. What can be drawn from this project is their use of control algorithms for flight navigation and specific parts they used. Their use of state machines for their navigation control routine is a very viable option for the KIRC project. Using a GPS as a navigation feedback sensor, they were able to fly the quadcopter between two points while using simple geometry to determine a trajectory. When taking off, flying, or landing there are states for each that are going to be integrated into the state machine. These are ideas that may be implemented in the final KIRC design.

Another design of interest also came from a UCF electrical and computer engineering senior design project. Autonomous Quadrotors Utilizing Attack Logic Under Navigational Guidance, or A.Q.U.A.L.U.N.G. for short, was another quadcopter project that utilized the use of autonomous flight. The original idea behind their project was to create a game whereby competitive laser tag players could play a laser tag game against multiple autonomous quadcopter opponents. There are many ideas from this project that can be implemented on the KIRC project. Their use of motors, batteries, frames, and computers will be potentially useful to draw ideas from. Their autonomous flight algorithms also make for a good place to start on KIRC's own autonomous flight algorithms. Their use of computer vision to provide obstacle avoidance, however, might prove too complex for this project though. A.Q.U.A.L.U.N.G. does provide a good resource of information on quadcopter design that the team will keep in mind for design purposes.

While buying a quadcopter would prove expensive and wouldn't satisfy the requirements of the project, making everything from scratch would also be unnecessary. So, for the KIRC project, it has been decided that the design will be done with the assumption that certain technologies are available: Microcontrollers, IMU sensors, Motors, Speed controllers, Frames, Propellers, Operating Systems, and all associated support hardware and software.

In summary, all the projects assessed in this section have portions that may be implemented in KIRC. In the next sections, decisions and driving factors on KIRC parts and designs are given. These decisions are made are based on the research given in this section and other research into cost and viability.

3.2. Relevant Technology & Strategic Components

In this section, the general technologies that are likely to be used in the KIRC project are given based on the research effort made by the team. Extensive research was done to find common parts, processes, and software used in other UAV projects, specifically ones that had to do with quadcopters. These technologies will be integrated into the core the KIRC's design. The goal behind using common technologies in this project is to provide a system based on proven technologies that are known to work properly and be predictable.

One of the most important items in the KIRC project is the use of microcontrollers for the flight computers. The microcontrollers will be critical to keeping the quadcopters stable in flight. In researching quadcopter designs, it was found that the majority of them used some form of microcontroller to control stability. Finding the right microcontroller is difficult, however, with so many to choose from. Usually, the microcontrollers most commonly used had the ability to do floating point arithmetic, support for interrupt driven software, and lots of peripherals such as UART, I2C, SPI, PWM, and A/D conversion. In order to satisfy UCF requirements, the team must create a PCB with an integrated microcontroller. This puts another requirement on the microcontroller, meaning that it should be available on a Launchpad for prototyping as well as a standalone chip for the final PCB prototype. This also means that the microcontroller must have support for JTAG in circuit debugging.

Sometimes, but not always, projects will implement a real time operating system (RTOS) as part of the software on the microcontroller. The advantage of this approach is that is prevents scheduling conflicts with the software, and also ensures that the processing of the data occurs in real-time. Unfortunately, RTOS software tends to be very large and requires enough memory, and processor speed in order to run. The advantages, however, come in handy when implementing a digital control system.

Many of the quadcopter systems researched used a digital control system implemented on a microcontroller for stabilizing the quadcopter. Quadcopters are naturally unstable in flight and need an active attitude control system to maintain stability at all times. In researching flight control algorithms, many control topologies came up. Many people tend to use Proportional Integral Derivative (PID) loops to dynamically minimize the error from the roll pitch and yaw of their quadcopters. By stabilizing the roll, pitch, and yaw of the quadcopter, it becomes easier to control the quadcopter in flight and it also makes autonomous navigation possible. Even though there are many ways to implement a digital control system, one thing remains constant for all of them. Digital control systems only work under the assumption that the feedback and controller output occur at discrete time intervals. For this reason, the RTOS comes in handy is because it creates a predictable time interval where control processing can occur. Using these windows of processing, all of the discrete math associated with the digital control system can be accurately done in the proper time.

Feedback in digital control of UAVs is usually done using inertial measurement units (IMU). This sensor group contains an accelerometer, rate gyroscope, magnetometer, and altimeter, and usually uses I2C as a serial protocol over a common line. The accelerometer and gyroscope are usually used to find an attitude state vector, while the magnetometer is used to find absolute heading and the altimeter finds the altitude. Most IMU sensors tend to be noisy and biased when reading them directly, so frequently people use digital filters to 'filter' out the noise from the sensors. The filters most commonly used to get rid of the noise from these sources are adaptive filters. Adaptive filters actively change based on optimization algorithms to most optimally get rid of noise. There are two common filters of this type most commonly used in practice: The Kalman Filter and the Least Squares Filter. Each of these filters have advantages and disadvantages that come into consideration when implementing them. First, the Kalman Filter, is a very accurate filter that provides excellent noise reduction for any application. The downside of this algorithm is that it is very computationally intensive and can bog down a computer system rather quickly when implementing it in an embedded application. The other, the Least Squares Filter, uses a computationally efficient algorithm that does a moderate job of getting rid of noise. While this filter is not as accurate as the Kalman filter, it is more ideal for computationally limited applications.

The input of the digital control system needs to come from somewhere. Usually while holding still in flight, the digital controller needs to just keep the error minimized while the input is zero. In reality, in order to move the quadcopter, there must be an input from a controller. For phase 1 of the project, the controller is planned to be a user with a RC controller device. The most commonly used type of RC controller for UAV applications is a hand held, joystick operated, controller device. These devices usually have settings for trim, offset, and mode. When looking for a RC controller, it is important to note the number of channels available to the user because this affects how many different signals the controller can send to the quadcopter at a time. Each of the channels represents an input to the quadcopter control system that can change roll, pitch, yaw, throttle, etc. Another thing to keep in mind when buying the RC controller is the price. Unfortunately, RC controllers can tend to be quite pricey. The plan is to only buy one RC controller, as it is not planned to fly both quadcopters manually at the same time.

Pulse Width Modulation (PWM) is common method for motor control in UAV projects. PWM signals can be easily generated using a microcontroller, and many microcontrollers include built-in hardware for it. This control signal is ideal for motor control because it provides a modulated signal that contains load voltage percentage information. This means that the PWM signal can change the output of a motor based on the duty cycle of its square wave. Electric motors, however, don't usually take PWM directly as an input, so there has to be an intermediate circuit that can translate the PWM signal into a DC voltage for the motors. Electronic Speed Controllers, or ESCs, are circuits that can take a PWM input and send proper voltage and current to the motors to provide predictable motor output. ESCs need to be chosen based on their current rating for motor support. This means that if a motor is drawing 20 Amps, the ESC must be able to source 20 Amps without overheating, distortion, or causing an electrical failure.

Finding the right electric motors for the project is also an important task. The most commonly used type of electric motor in quadcopter projects is the brushless motor. In the process of looking up brushless motors, there are several specifications that come up. The Kv rating, the current rating, the power rating, the thrust, the weight, and the size are all things that are frequently listed among tech specs. For this project, motors have to be found that fit the frame of the quadcopter and be able to lift the whole system while not drawing too much current. The motors are definitely the most responsible for the current draw of the system, which directly affects the battery life and thus the flight time. Because the motors have such a direct effect on the flight time of the quadcopters, care must be taken in the decision of the motors for the project.

The most crucial part of the KIRC project is the ability for the quadcopter to be autonomous. For that section, the team will draw ideas from both the AQUALUNG and the AFSD project. The flight path and control loops will be calculated in real-time, while obstacle avoidance will not be taken into account. NASA required the team to use a Raspberry Pi ™ Model B, an embedded Linux platform, to test the DTN2 application as well as handle the image retrieval and telemetry relay. The Raspberry Pi can also be used to set flight paths and communicate (via UART) with the microcontroller being used for stability. Though the Raspberry Pi ™ has more processing power than necessary for the project, it is necessary for the testing that NASA has in mind for the DTN2 protocol. The plan is to utilize the Raspberry Pi's ™ extensive processing power to handle the autonomous portion of the project.

In order to find the absolute position of the quadcopters, the IMU is not enough. The absolute position of the quadcopter is necessary for the autonomous navigation. To find absolute position is relatively easy with using a global positioning system (GPS). There are inexpensive light weight GPS units available on the market that can interface to a microcontroller using a UART serial port. These units have a standard for communication called NMEA, which is named after the National Marine Electronics Association. This standard is almost always used in commercial GPS units, which makes the code for parsing NMEA strings quite common. The software created for this project will have to include some NMEA parsing code in order to extract important information from the GPS.

Power management in quadcopter systems is critical to keeping flight time adequate enough. Many quadcopters, and indeed the ones evaluated in the previous section, use lithium polymer (Li-Po) batteries for their primary power source. The batteries for the KIRC project will have to be as lightweight as possible, still be able to hold enough charge, and provide enough energy to power all the devices and motors. This means that the batteries will need to have a high energy density and low weight overhead. Lithium polymer batteries provide this performance in a small size and weight package. Compared to traditional alkaline batteries, Li-Po batteries have a longer battery life and larger power capabilities while not being too much larger or heavier (source). Li-Po batteries usually come in configurations where multiple Li-Po cells have been linked in series to provide more voltage. Each cell for a Li-Po provides about 3.7V nominally when charged (Source), and quadcopters usually require 15V sources to power the motors. So, for this project, a 4 cell Li-Po battery would be ideal. Another thing to keep in mind when purchasing the battery is the battery charger that comes with it. When the energy stored in the battery becomes depleted, the battery must be charged before using it again. So, after the right batteries is found for the project, a battery chargers to go with it must also be found.

In summary, all these technologies are ones that are likely going to be used in the KIRC project. The goal was to use common technologies in the project so that everything was guaranteed to work properly and predictably. In the next section, specific parts are given based on the general technologies given in this section.

3.3. Specific Components and Decision Factors

In this section, the specific components and software used in this project are given. The driving decision factors for each product is also given along with comparison of selected other products of interest.

The first component chosen for this project was the microcontroller due to the fact that it has the most impact on the software portion of the project. Based on the relevant technologies shown in the previous section, some requirements were made on what peripherals the microcontroller must have and performance criteria it must meet. A summary of the requirements is given below in a bulleted list:

- Must have a Floating Point Unit (FPU) to support control algorithm calculations
- Must have multiple UART ports for serial communication with the GPS and the Raspberry Pi
- Must have I2C ports for reading IMU
- Must have multiple PWM channels for motor control output
- Must have A/D converter
- Must have Real Time Operating System (RTOS) support
- 32 Bit Processor with fast enough clock rate to perform all control functions reasonably
- Must have a Launchpad/PCB available for prototyping
- Must have individual microcontroller available for own PCB development

With these factors in mind, research was made into different microcontrollers from different vendors. There were four options that stood out from a controller perspective that closely fit these requirements. In table 1 on the next page, the four microcontrollers are compared for the decision factors for this project.

Name	Vendor	Processor	RTOS Support	Availability	Price
UNO32	Digilent	PIC32MX320F128	No	Launchpad, Stadalone	$26.95
Piccolo Launchpad	TI	F28027F	Yes	Launchpad, Stadalone	$17.00
Stellaris Launchpad	TI	EK-LMF120XL	Yes	Launchpad only	$13.49
Tiva C Launchpad	TI	TM4C123GH6PMI	Yes	Launchpad, Stadalone	$12.99

Table 1: Comparison of Select Microcontrollers

The final decision was to go with the Tiva C ™ Launchpad from Texas Instruments for the prototyping part of the project and then obtain the chip and embed it in its own PCB with all the peripherals already integrated. While this all the microcontrollers would have worked for the project, the Tiva ™ C was decided based on the price and performance. The Tiva ™ C is a very capable microcontroller with a 32 bit, 80MHz, ARM Cortex M4 Processor. It has plenty of flash ROM to hold the RTOS, with 256KB. It also has enough RAM to support any amount of data or variable necessary for the control algorithms, with 32KB available. The microcontroller also has support for multiple UART, I2C, SPI, PWM, A/D, and GPIO ports in hardware. One very attractive feature of this microcontroller is its available RTOS from Texas Instruments. The TI RTOS can be specifically optimized for this microcontroller. The microcontroller is also low power, drawing only 45mA in nominal mode according to the datasheet.

After choosing a microcontroller, the next step was to finalize on an IMU. The IMU is critical to the flight stability control, and without it, the quadcopter would be flying blind. The reason for obtaining this part early was to start reading from it, developing filtering and control algorithms, and getting an early start on the software development. Choosing an IMU, however, can be very difficult. They can be very expensive, with prices being up to the thousands of dollars. On the other hand, going with an inexpensive unit can result in the compromise of quality, accuracy, and precision. With all these compromises evident, it is useful to provide a list of requirements specific to this project. Below is the list of requirement that the IMU must meet in order to satisfy the requirements of the project:

- Must be less than $100 (preferably less than $50)
- Must be I2C compatible
- Must have the following sensors: Accelerometer, Gyroscope, and Magnetometer
- Must work on 3.3V power and low current
- Must fit on through-hole mounting shield of size less than the microcontroller
- All on board sensors must be available individually from at least one vendor so that they can be incorporated into the PCB design

Using these requirements, the team searched for candidate IMUs for use in the project. There were several available from Ebay or even Amazon for a good price, but unfortunately, getting approval from NASA for these items proved difficult. There was also one from Sparkfun ™ that fit perfectly, but had an expensive price of $99.95 each. While this price does reach the limit of the budget for the part, it was an acceptable source to buy parts from. So, the 9 Degrees of Freedom – Sensor Stick from Sparkfun ™ was the part of choice for the team.

The 9DoF sensor stick from Sparkfun ™ has three sensors on board: the ADXL345 accelerometer from Analog Devices, the HMC5883L magnetometer from Honeywell, and the ITC-3200 gyroscope from InvenSense. These sensors are all available for individual purchase from their respective companies. All the sensors have support for I2C slave communication, and have a breakout header available for prototyping. The stick runs on a 3.3V source, which the microcontroller can source. The stick itself is also very small, taking up only 1.37” by 0.42” of space. Sparkfun provides a fair amount of documentation of this part for interfacing and troubleshooting.

The GPS is the most important part of the autonomous system. With this part, the quadcopters will know absolute position and be able to correct navigation. This part, just like the IMU, can tend to be very expensive. The GPS can come in varying sizes as well. For this project, a smaller GPS must be used that still has the signal strength to receive signal from even in the midst of the EMI caused by the motors. Below is a list of requirements for the GPS unit:

- Must have signal strength enough to overcome motor EMI
- Must be small enough to fit
- Must meet sensitivity requirements
- Must acquire fixed location within a short time
- Must be low cost

A big requirement is both the speed to connect with usable satellites and the number of satellites desired to use. Previously the standard was 16-channel, but for this project a 50-channel is better for precision. 50-channel is more precise due to the fact that the amount of channels correlates to the amount of satellites that can be used at one time. Though there will not be 50 satellites designated to global positioning during the design of this project, it will be able to use as many as are available. The connection to a larger amount of satellites correlates to a faster speed at measuring changes in location. When choosing a Global Positioning System, especially with a small target as a quad-copter and a need for precision, sensitivity is crucial to the outcome of the system. In order to maintain route and avoid missing images in the mapping process the sensitivity needs to be less than -160dBm during tracking and navigation.

Table 2 shown on the next page contains a list of the GPS units considered. One of the concerns for the GPS is the start up time. This is known as the TTFF or time to first fix. The drain on the power source being the limiting factor on the flight time of this project brought the requirement that the TTFF be less than 30 seconds. With a start up procedure that can enable a low power part with long start up time as this one before powering more high power components will assist in the longevity of flight. This part will also need to have low powered, but with these components' power is usually not a real issue.

When it comes to the budget of this project, being cost efficient is important. The budget set before this project is one of the major challenges due to the desire to replicate. For this reason all of the components used need to be efficiently chosen to be cheap and up to our specification needs.

Name	Vendor	Power	Number channels	TTFF (seconds)	Sensitivity (dBm)	Price ($)
GS407	S.P.K. Electronics Co.	3.3V@75mA	50	29	-160	$89.95
GP635T	ADH Technology Co.	5V@56mA	50	27	-161	$39.95
D2523T	ADH Technology Co.	3.3V@74mA	50	29	-160	$104.00

Table 2: Comparison of Select GPS Units

The GP-635T produced by ADH Technology Co. Ltd. is not only slim and easily implementable, it is also cost effective at $39.99 each, half the price of other competitors. The GPS receiver will work in coordination with the IMU in the process of flight, while the IMU will be completely responsible for flight equilibrium. This GPS receiver is accurate to -161dBm with a UART default baud rate of 38400 bps, and the implementation of digital I/O. Environmentally, this unit can maintain operational status between -40 and +85 degrees Celsius and less than 500 Hz of vibration, less than the amount expected under the worst of conditions. The accuracy and speed of communication with this receiver is crucial due to the nature of this project. Each image and, by extension, movement must be precise in order to get a good mapping of the area and accurate autonomous navigation throughout the routine. The power needs are relatively small, but we sacrificed a small bit of battery life for the precision of such a component with 4.75 to 5.25 power supply needs with max of 56 mA current, which in comparison to the rest of the system will be close to nothing. As packaging goes this component only sits at 8x35x6.55 mm, this assists in keeping the payload of the quad-copter compact. At the price and precision of this component and its ease of implementation made it a clear choice for KIRC.

Electronically, the frame is of little importance, but it is a crucial piece part in the quad-copter when it comes to packaging and flight abilities. When in actual flight the frame must withstand the environment, and affects speed, control, and maneuverability based on rotor separation. A frame that can be bought premade can come in a variety of dimensions, materials, and styles. Below is a list of requirements for the frame:

- Must be lightweight
- Must be agile in motion
- Must be rugged enough for most environments
- Must not hinder system capabilities

For this project it was between three different types of frames. Due to the fact that finding a frame made of a lightweight material is not an issue, part decision came down to agility, precision and system integration.

The first option that was considered was a frame done by a previous UCF senior design group known as the "Automatic Flying Security Drone". In this project a cage in the shape of an orb was implemented. Though this design is creative, aesthetically pleasing, and well designed for the previous group's requirements, it didn't possess the agility and mobility that was expected for the missions KIRC will be involved in. In addition, this design would have issues in dealing with the environment at high altitudes due to the fact the aircraft would be blown off course in certain conditions.

The second option for a frame was a hex-copter frame. With this design we would have all of the agility, speed and control needed for completion of any mission set before the aircraft. This frame design was tossed, however, when system integration was considered. Using a frame that would require not only four motors, which are the main pull on the power source to begin with, but six motors made this frame unrealistic with requirements set forth for flight time.

After these considerations it was decided that a lightweight quad-copter frame would be the best for system integration into the Kirc project. This frame would give agility, and mobility enough to counteract any environmental issues seen, and would have as little pull on the battery as necessary for mission purposes. The frame has reasonable size dimensions (450mm x 450mm x 55 mm) with motor mount bolt holes preset for ease of implementation, and lightweight (270g., 1.04 lbs.).

Motors carry a lot of weight when deciding the aeronautical capabilities of this aircraft. An analysis of flight and flight time capabilities with each of the motors available is the best way to narrow down the options. After the motor is chosen, then the ESC is decided by implementation abilities with selected motor.

- Must have thrust capabilities to hover payload at less than 50% thrust capacity
- Must be powered by 15 V or less
- Must be low priced, less than $20
- Must adhere to the above requirements and maintain a flight time greater than 12 minutes with a 5 Amp/hour battery

After narrowing the search based on the price ceiling there are still many options readily available. The implementation of flight time calculator was implemented based on weight and thrust capacity of the motors. In order to do this, add the mass of each piece part including a tolerance for lightweight parts to be implemented on the PCB later. Then take the total thrust of the four motors (assuming they are all four the same type of motor) and divide the total mass by this maximum thrust capability. Take this quotient is multiplied by the maximum total current drawn by the entire system. Finally, divide the life of the battery by this product, giving the total static flight time aiming for a minimum of 12 minutes. This analysis may be seen in the two equations and table 3 shown on the next page for the chosen motor.

$$\%Thrust = \frac{m}{4 * Max\ Thrust}$$

$$FlightTime = \frac{\mathrm{Q_{bat}}}{\mathrm{\%Thrust} * \mathrm{I_{sys}}}$$

m = mass of the entire system, in grams
$Max\ Thrust$ = max thrust for each motor, in grams
Q_{bat} = lifespan of the battery in Ampere Hours
I_{sys}= current draw of motors and electrical circuits

Part	Description	Part #	Mass (g)	Qty	Current (A)	Thrust (g)	Capacity (Ah)
Frame	Fiber Glass Quadcopter	Q450	280	1	0	0	0
Propeller Motors	Prop Drive Motor	NTM2830S-900	66	4	10.6	750	0
ESC	Afro 20Amp ESC	919200131-0	23	4	0	0	0
Battery	Turnigy 4s Lipo Battery	N5000.4S.45	552	1	0	0	5
Propeller	8x4.5 Propellers	17000055	15	4	0	0	0
Avionics	Controls, Processors, Sensors	Tiva/Raspi	200	1	1	0	0
	TOTALS	N/A	1448		43.4	3000	5

%Throttle to keep level	0.482666667
Current Drain (A)	21.46506667
Flight Time (Minutes)	13.97619698

Table 3: Flight Time Calculation Table

This calculation was done for all motors considered. Three NTM Prop Drive motors were considered above all others, an 800kV/300W, a 900kV/270W, and a 1000kV/235W. The analysis outcomes are shown in table 4 on the next page.

	800KV/300W Motor	900KV/270W Motor	1000KV/235W Motor
%Throttle to Keep Static in Air	0.343809524	0.482666667	0.373532551
Current Drain (A)	25.4792381	21.46506667	23.26254002
Flight Time (Minutes)	11.77429242	13.97619698	12.89627013
Price	$15.29	$14.99	$15.99

Table 4: Comparison of Select Motors

The 900kV/200W motor is the most cost efficient and allows for the longest flight time. For this project, with the equipment so chosen, the NTM Prop Drive Series 28-30S 900kv draws enough current and supplies enough thrust to give the quad-copter the agility and mobility that is desired, as well as optimizing the flight time, and cost.

Electronic speed controllers, ESCs, do the job of taking I2C PWM signal from the microcontroller and filtering the current to the motors in order to scale the thrust output of each motor.

- Must be easily integrated with the chosen motors
- Must be capable of drawing at least 15 amps
- Must implement I2C standard for communication

For this component only two Afro ESCs are considered, due to the fact that they are low cost and I2C standard. These are both 20 amp ESCs one being slim and the other a box package shape. Upon further search into the slim ESC it is found that the product is actually 30 amp capable, only being toned down by the 20 gauge wire being used. Do to this fact, along with the higher price and more difficult package to implement, the Afro ESC 20Amp Multi-rotor Motor Speed Controller was chosen as the project moves forward.

When it comes to a flight system, it is only usable as long as the power source last. For our project being powered by a battery, the lifespan of this battery is the limiting factor on flight time. The best batteries as far as lifespan that are within the budget of this project are lithium polymer, LiPo for short. Requirements for the battery are as follows:

- Must be rechargeable
- Must have a rating of at least 4500 mAh
- Must be rated for at least 200 W
- Must be a 4 cell

In the decision of battery, the power vs. flight time table and calculations was implemented as when choosing the motor. When deciding whether to have a stronger battery or multiple batteries, the decision came down to price vs. addition to flight time. In addition this battery needs to be rechargeable, so that it can be put in the package and not have the necessity of being removed and replaced with each use. If a disposable battery is chosen, the budget of the project becomes time dependent, as the aircraft is used repetitively the cost of batteries adds up quickly.

The calculation table used in the analysis for the motor selection is also implemented in this situation. The power source is the last thing chosen due to the fact that its requirements are dependent on the needs of the system. In this situation there are a couple of options when it comes to powering the system as a whole. The implementation of two batteries could be used or having a single battery with higher output.

The initial thought is to keep the budget as low as possible without diminishing the ability of the quad-copter's ability to complete its missions. In order to accomplish this a range of rechargeable 4 cell LiPo batteries were considered, centering around the range of 4500 mAh to 6000 mAh. The analysis of operation ability can be seen in Table 5 below.

	4500mAh	5000mAh	6000mAh
Flight Time (Minutes)	12.57857729	13.97619698	16.77143638
Weight (g)	513	552	623
Cost	$57.15	$65.03	$64.75
Operating Temperatures (C)	45-90	45-90	25-50

Table 5: Comparison of Battery Size vs. Energy Capacity

Table 6 on the next page can be used in combination with table 5 above to consider additional aspects in design. The 6000 mAh batter lacks the range of operating temperatures that KIRC needs to be capable of performing in. Additionally, the weight of the battery does not have provide a large enough increase in flight time to encourage the loss in operating environments. The 4500 mAh battery was in the right range of temperature operation, but in terms of getting an additional minute and a half of flight time for the cost of $7.88 only during replacement of the battery made the 5000 mAh battery the best power source. This extra time will be useful during the longevity of some parts when cold starting.

When looking at the description in comparison with the rest of the system parts, it can be seen that the weight of the battery is much more than that of the other parts. For this reason, when two batteries are implemented an increase in flight time can be seen, but it isn't a large enough increase in order to be worth the increase to 66% of the motor capabilities just to maintain static flight. A six minute increase in flight time can be seen, however this increase in weight makes the drive on the motors too large, thereby taking away from its mobility and aerial agility. In addition the battery is one of the most costly components in the budget. For all of these reasons it is decided that a single 4 cell, 5000 mAh, rechargeable battery is best suited for this project.

Part	Description	Part #	Mass (g)	Qty	Current (A)	Thrust (g)	Capacity (Ah)
Frame	Fiber Glass Quadcopter Frame	Q450	280	1	0	0	0
Propeller Motors	900KV/270W Motor	NTM2830S-900	66	4	10.6	750	0
ESC	Afro 20Amp ESC	919200131-0	23	4	0	0	0
Battery	Turnigy 4s Lipo Battery	N5000.4S.45	552	2	0	0	10
Propeller	8x4.5 Propellers	17000055	15	4	0	0	0
Avionics	Controls, Processors, Sensors	Tiva/Raspi	200	1	1	0	0
	TOTALS	N/A	2000		43.4	3000	10

%Throttle to Keep Level	0.666666667
Current Drain (A)	29.26666667
Flight Time (Minutes)	20.50113895

Table 6: Flight Time Calculation Spreadsheet Example

4. <u>Design Summary</u>

4.1. Hardware Operation

Through the research and design phase of the project, the team has decided to divide the project into three phases with distinct hardware structures, as mentioned before. The hardware configuration of phase one with a simple manual flight configuration using RC controlled quadcopter will obviously be much simpler when compared to phases two and three which will add a separate navigational computer to achieve a fully autonomous prototype which takes and stores pictures. Although the hardware of phase one will be somewhat redesigned, the fortunate fact of the matter remains that each phase will build on one another. The team's integration of the microcontroller and the inertial measurement unit on the printed circuit board in phase two will use similar connections designed and implemented in phase one.

The hardware operation will essentially consist of a description of the specific hardware used, the connections between functioning hardware, and a general explanation of how each component will communicate with one another. As the design of the quadcopter project has evolved, the team has gone through a number of different configurations, comparing and contrasting different ways to accomplish the same objectives. A large obstacle to overcome has been deciding exactly what tasks the Raspberry Pi navigational computer will be in charge of and what tasks the microcontroller's avionics computer will be in charge of. This is important in the hardware design because the specific roles of both computer systems designate which components will be connected to either one of these computers or both of these computers. Although there has been some debate as to where the exterior sensors should be connected, the power distribution system of the hardware design has been pretty consistent throughout the entire design. The hardware operation concerning the power distribution will be the first topic discussed.

A physical description of each component will aid in visualizing the system. The battery chosen is 14.8 volt discharge four cell lithium polymer nano-tech in the shape of a rectangular prism with three cables coming out of the top edge. One of the three protruding cables is a JST-XH connector which is used to recharge the battery after depletion. This recharge connector will go through an intermediate power adapter which is then plugged into a standard 110 to 120 volt wall outlet. The team has decided to purchase extra batteries in order to have spare extra batteries charged at any given time. This strategy will prevent the team from having to wait on charging batteries thus increasing the efficiency of the testing phase. Additionally, extra batteries can increase the potential area mapped out during future reconnaissance missions, granted the team must program the quadcopters to return to the ground station to manually switch out the batteries before resuming their paths. The other two protruding wires coming out of the battery are black and red in color and correspond to the positive and negative terminals of the battery. In the circuitry, the team will designate the black wire to be the “ground” wire and the red wire to be the high voltage line. The red and the black cables have 4mm bullet connectors at each end. The battery will be directly connected to a voltage regulator circuit and all of the electronic speed controllers. The voltage regulator circuit will include a 5 volt rail and a 3.3 volt rail to power the electronic components on board while the electronic speed controllers will provide power to the motors.

In order to distribute power to each of the four motor controllers, the team has decided to purchase a splitter cable which simply splits the battery’s single red wire into four red wires and the battery’s single black wire into 4 black wires. Each of the electronic speed controllers will take an input of one red wire and one black wire from splitter or breakout cable. The breakout cable’s connectors are 4mm on the two wire side and 3.5 mm on the eight wire side. This is an extremely convenient part due to the dimensions of the electronic speed controller’s input cables. The breakout cable fits perfectly to the 4mm bullet connectors from the battery as well as the 3.5mm bullet connectors to the electronic speed controller. Each of the four electronic speed controllers have three input cables and three output cables. Two of the ESC’s input cables, mentioned previously, are red and black in color and correspond to positive and negative inputs from the battery. The single remaining input is a servo connector that will be attached to the microcontroller. The servo connector does not supply power to the electronic speed controller but instead provides a pulse width modulation signal. This pulse width modulation signal essentially tells the electronic speed controller the speed at which it should rotate the shaft of the motor. Three wires colored black, red, and yellow serve as the output of the electronic speed controller. Someone who does not have experience in robotics may be tempted to think that the red and black wires are ground and high voltage lines, similar to that of the DC input coming from the battery to the electronic speed controller. In actuality, the motors the team has chosen are three phase brushless motors.

Three phase brushless motors take in an AC current input. This fact is important in understanding why the electronic speed controller is so vital in the team's quadcopter design. The electronic speed controller takes a DC current input and a PWM signal and conversely creates AC current output. Specifically, each of the three output wires from the ESC to each motor carry an alternating trapezoidal wave at different phases compared to each other at any given time. One may be inclined to believe the electronic speed controller controls the speed of the motor by varying voltage or current, but surprisingly it is controlled by the timing of these three phases with respect to one another. In contrast to the motors, the other electrical components on board are not powered by an AC signal from the ESCs but instead are powered by a DC voltage coming from the voltage regulator circuit. The first portion of the power system established concerns powering the four motors. Because the combination of the four total motors will pull a relatively large amount of current compared to the other electronics on board, the motors' subsystem will be the main cause of battery depletion.

The next portion of the power system is to implement a voltage divider circuit. This will be accomplished by tapping into the middle of the breakout cable. The team will simply unravel the shrink-wrap casing at the point where the red and the black wires split from one to four chords. The team will carefully solder in one extra wire connected to a female banana jack for the black wire and another extra wire connected to a female banana jack for the red wire effectively transforming the two-to-eight breakout cable into a two-to-ten breakout cable. To protect the point tapped into the breakout cable, the team will proceed to cover the exposed wires with electrical tape. The two added female banana jacks will connect to the input of the voltage regulator circuit. The team could have simply soldered wires to the breakout cable and used said wires as the input to the voltage divider; however, the team has decided to include a banana jack design in order to increase compatibility when moving to successive phases. In phase two, the voltage divider circuit will be on the PCB. The team will use the same banana jack method to connect the PCB to the breakout cable. This will be accomplished by creating a custom cable which has a male banana jack on one side and a jack to the PCB on the other side.

Initially in phase one's design, the voltage divider circuit will be implemented using a small breadboard. The breadboard will be mounted with screws to the top plate of the quadcopter frame. The electrical components included on the breadboard will be two voltage regulators as well as other passive and active components. The team has decided to attach capacitors to the input and the output of the 5 volt and the 3.3 volt regulators in order to smooth out ripple and decrease noise. The team has also decided to include diodes from the output to the input of both voltage regulators. The diodes will protect each regulator in case the input power is cut off. When the input power cuts off, the large capacitors at the output will discharge their voltage. If the voltage is significantly higher at the output node of the regulator compared to the input node, the regulator might get damaged or destroyed. In the case of this scenario, the diode will let the current flow from the output to the input thus bypassing the regulator, preventing harm to the part. The team has decided to designate the output of each regulator to create a 5 volt rail and a 3.3 volt rail in addition to a ground rail. The Tiva C Launchpad, the Raspberry Pi, and the GPS unit all take a supply voltage of 5 volts. To power these three devices, the team will connect wires from the ground rail of the breadboard to the ground pins of each component and connect wires from the 5 volt breadboard rail to the 5 volt pins of each component. Similarly, the team will power the IMU, which takes a supply voltage of 3.3 volts, by connecting a wire from the breadboard ground rail to the ground pin of the IMU and another wire from the breadboard 3.3 volt rail to the 3.3 volt pin of the IMU. Had the team tried to directly connect all 14.8 volts of the battery to the devices, they would be destroyed hence the need for a voltage regulator circuit. The other remaining components will be powered by either the microcontroller or the Raspberry Pi. The 802.11g wireless card, the SD card, and the high resolution camera are each made specifically for the Raspberry Pi and attach directly to its board. Because the connectors have built in power lines to these elements, there is no need to design a power distribution subsystem to these components. There is in fact one more circuit that is included on the breadboard which is related to the power distribution system though not vital to the systems functionality.

The team has decided to add a so called “battery life” feature to each quadcopter. In order to measure the amount of battery life in the power source, the team will utilize one of the analog to digital converter pins on the microcontroller. The idea is as follows: The microcontroller will periodically read in a DC analog voltage from the battery. The microcontroller will take this analog input and convert it to a digital signal. The team will write a program that uses this digital information to calculate a number that represents the amount of time in minutes left on the battery. From there the team will send the battery life data to the navigational computer which will send this information wirelessly to the ground station where it will be displayed on a graphical user interface of a laptop. The battery life feature is simply one of several features displayed on the ground station laptop.

One may wonder what is the missing circuit included on the breadboard. Because the analog to digital converter pin on the microcontroller cannot exceed a specified voltage, the team had to devise a solution to decrease the voltage from the battery to stay below the microcontroller’s voltage limitations. The missing circuit is simply a voltage divider circuit using two resistors and an operational amplifier. For this circuit, the input voltage, Vi, will be taken from the battery and the output voltage, Vout, will be wired to the positive terminal of the unity gain buffer. Obviously the negative terminal of the unity gain buffer is wired to the output of the op-amp which is also wired to the analog to digital converter pin of the microcontroller. The first resistor, R1, will be between nodes Vi and Vo, and the second resistor, R2, will be between node Vo and ground. The resistor values chosen are a 100kΩ resistor for R1 and a 400kΩ resistor for R2. The equation below will give insight as to the rationale behind the resistor values chosen.

$$Vo = Vi\left(\frac{R1}{R1+R2}\right) = Vi\left(\frac{100k\Omega}{100k\Omega+400k\Omega}\right) = Vi\left(\frac{1}{5}\right)$$

When the battery is at maximum power, the input voltage, Vi, is equal to 14.8 volts. The voltage divider circuit, with the given resistor values, will force the output voltage, Vo, to equal one fifth of 14.8 volts or 2.96 volts which is below the voltage threshold of the A/D converter pin on the microcontroller. As the battery depletes, the output voltage will, for the most part, decrease proportionally with respect to the input voltage, Vi. The team will take this into account when writing the battery life program, corresponding max voltage to max battery life. The team will go through extensive analysis to test how long the battery lasts. In this testing phase, the team will record the voltage at Vo while exhausting the battery until depletion. From this data, the team will generate a graph with the output voltage, Vo, on the dependent vertical axis and the amount of time elapsed on the independent horizontal axis. This graph will be manipulated to have battery time left on the horizontal axis. To sum up, the microcontroller will read in a voltage, send a number corresponding the amount of time left on the battery to the Raspberry Pi, and the navigational computer will wirelessly transmit this data to the display on the ground station laptop for each quadcopter. Keep in mind the breadboard will not exist after moving to phase two because the voltage regulator circuit will be built into the PCB.

With a sound understanding of the power distribution system, it seems logical to move on to the next segment of the hardware design, the on board interconnections to and from each computer system. The hardware operation is heavily dependent on the communication method to and from each component. There are several different types of communication methods with specific connectors depending on the device of interest. There will not only be wireless transmission from the RC controller to the receiver on board, but there will also be wireless transmission from the Raspberry Pi to the laptop on the ground station. These two communication methods will however be using different radio frequencies in order to prevent interference. The microcontroller, IMU, and GPS will all be separately attached to the frame in phase 1 with cables running from the microcontroller to the IMU and from the microcontroller to the GPS. In phases two and three, after designing and printing out the PCB, all of these components will be on one chip and the electrical components will be connected through etched conductive paths. This method of connecting components is ideal due to the proximity of the components, allowing for near instant communication between devices. For the team's quadcopter project specifically, there will be a very high frequency of communication between the IMU and the microcontroller. This is necessary to stabilize the quadcopter in the air. Having the hardware of the IMU and microcontroller on the same chip, mere millimeters apart, permits the fastest possible communication.

The first on board communication relationship discussed may be the most important. The avionics computer's main job is to control flight and stabilization of the quadcopter. It accomplishes this task by taking in information from an electronic device called an inertial measurement unit or an IMU. The IMU includes an accelerometer, an altimeter, a gyroscope, and a magnetometer and sends signals to the input on the microcontroller. The microcontroller processes these signals to determine the velocity and orientation of the quadcopter. The IMU interacts with the microcontroller through a communcication method called I2C. This method requires two lines between the microcontroller and the IMU. One of the two lines is referred to as the master slave line, and the other line will be the clock. With information from the IMU, the microcontroller will send a pulse width modulated square wave signal to each of the 4 motor controllers. As mentioned above, each of the 4 motor controllers takes their PWM signal and convert it to an AC voltage which goes to each of the 4 motors on the quadcopter. Each of the 4 motor controllers have the capability of sending varying amounts of speed to manipulate each corresponding motor to first stabilize the quadcopter in flight then move the quadcopter in any direction in space.

One way to visualize this stabilization process is to picture one of the team's two quadcopters in midflight, hovering perfectly in one location in space. At this moment, the accelerometer in the IMU will let the avionics computer know that the quadcopter's velocity is zero and the gyroscope in the IMU will let the avionics computer know that the quadcopter is perfectly level. In this stable condition, the avionics computer is communicating to all 4 motor controllers to individually send approximately the same speed to each of the 4 motors to exactly counteract the downward force of gravity, keeps the quadcopter at a constant altitude. Keep in mind that the team's avionics computer is running through velocity and position calculations several times per second in order to correct any deviations to perfectly level as soon as possible. All of the sudden, a gust of wind blows the team's quadcopter out of its perfectly stable condition. For a split second after the gust occurs, two of the motors are slightly higher in altitude compared to the other two motors, and the team's quadcopter is now slightly tilted at an angle. The microcontroller, which is reading in information from the IMU several times per second, quickly realizes a slight tilt from the gyroscope's sensor as well as a slight change in velocity from the accelerometer's sensor. The avionics computer's microcontroller then quickly does calculations and sends appropriate amounts of speed to the 4 motor controllers to correct this tilt. In this specific example, the microcontroller could level out the quadcopter by providing slightly less speed to the two motors residing at a higher altitude, slightly more speed to the two motors residing at a lower altitude, or a combination in order to make the gyroscope's sensor read perfectly level once again thus stabilizing to a perfectly level quadcopter.

Just as the avionics computer sends signals to each of the 4 motor controllers to stabilize the quadcopter in the air, the avionics computer can manipulate the signals to the motor controllers in order to move the quadcopter around in the air. An RC controller will send data to a receiver connected to the avionics computer. When a team member toggles a single joystick or both joysticks on the RC controller, the avionics computer takes this information from the receiver and sends signals to the motor controllers depending on how the joysticks are oriented. The RC controller has two joysticks. The team decided to use a common convention for commercial quadcopter joysticks. The left joystick on the controller is to control thrust and rotation. Thrust will be the forward to backward motion of the joystick, and rotation will be the left to right motion of the joystick. The right joystick will control the pitch and the roll of the copter. The pitch and the roll can be thought of as tilted deviations from a level plane. A useful analogy would be to think of a person standing upright. A hypothetical perfectly leveled wood board balanced on this individual's head makes up the so called level plane. The action of leaning forward, looking more toward the floor, or leaning backward, forcing the individual to look upward toward the sky, would correspond to a change in pitch given the level frame of reference. If this individual were to stand up straight again and either lean left or right, he would change the so called roll of the level plane. In this analogy changing the so called yaw could be accomplished not by leaning forward, backward, right or left but instead by turning this person's head either left or right. This causes no tilt but changes the orientation of the plane to face a different direction. To sum up, the team's left joystick controls the thrust and the yaw, and the team's right joystick controls the pitch and the roll of the team's quadcopter.

The team's first goal is to have a fully manual RC controlled quadcopter functioning perfectly then the team will move on to program the autonomous flight mode. The RC controller is not necessary after the team successfully programs both quadcopters to move autonomously, but the team will keep an RC controlled override feature just in case the team wishes to leave autonomous mode. This feature is accomplished with a switch on the RC controller to turn manual mode on or off. The team has decided to include the manual override feature to protect the quadcopter from flying away or crash landing during the testing phase. Ironing out the kinks in autonomous flight may be one of the team's most time consuming objective. Now that a thorough understanding of the hardware of the avionics computer has been established, a discussion of the intricacies regarding the navigational computer's hardware will provide insight into how the two interact.

The Raspberry Pi ™ serving as the navigational computer will have several connections to and from its peripherals. The Raspberry Pi ™ will be connected to the avionics computer, to a camera module, to the wireless transmission adapter, and to an SD card. Although the quadcopter project will not utilize all of the external ports, the chosen Model B Raspberry Pi ™ is a powerful machine with two USB ports, RCA video port, HDMI out, a 3.5mm audio jack, a 10/100 Ethernet port, 8 GPIO pins, a UART, i2c bus, and SPI bus with two chip selects. The SD card, the wireless adapter, and the camera module don't require any hardware modification whatsoever; however, it should be mentioned that there will be a great amount of programming to incorporate all of these external devices. The SD card will connect directly to the underside of the Raspberry Pi ™ similar to plugging an SD card into a digital camera or any other electronic device. This SD card will serve as extra data space and utilized when saving high definition images from the camera on board each quadcopter. Ideally, the Raspberry Pi would send the high definition pictures wirelessly to the ground station midflight. This feature would be useful for the scenario in which, for whatever reason, the quadcopter gets lost during a reconnaissance mission and cannot return home. The camera images taken during this flight would not be lost. Unfortunately, the team suspects there will not be enough bandwidth to support this feature. Instead, the Raspberry Pi ™ navigational computer will receive pictures taken by the camera and store them on the SD card attached directly to the SD port of the Raspberry Pi. The camera chosen is conveniently made specifically for the Raspberry Pi. The camera module comes from the manufacturer with a 15-pin flexible connector attached to the board of the camera module. The flex connector will attach directly to the MIPI Camera Serial Interface 2 (CSI-2) interface connector without the need for purchasing any extra parts. The flexible connector simply connects to the Raspberry Pi board by pulling on two tabs on the top of the connector and snapping into place. Although the connections to the SD card and the camera module are relatively simple, the connection from the Raspberry Pi to the microcontroller proved to be more complex.

In phase one of the quadcopter design, the navigational computer on the Raspberry Pi ™ will use a UART to communicate with the avionics computer on the Tiva C ™

Launchpad. There will be two lines for the UART communication which are known as a Clock line and a transmit/receive line. The team will tie together input/output UART pins from either computer to accomplish this task. Unlike standard USB cables, HDMI cables, or an Ethernet cables, the team must put effort when tapping into the GPIO pins of the raspberry pi and the Tiva C microcontroller. The team has decided to connect the UART pins of the microcontroller and the Raspberry Pi using single port female to female jumper wires. This method is ideal in that we will only be using a small number of pins on either GPIO ports on both boards. The team has decided against using a full ribbon cable for this very reason. The only remaining component left in the hardware operation is the wireless USB adapter conneted to the Raspberry Pi.

The wireless USB adapter will connect directly to one of the two USB ports on the Raspberry Pi ™. The ground station laptop will have its own wireless adapter which will communicate with the one onboard each quadcopter. The team has decided to use the wireless transmission to the ground station as somewhat of a status indicator for both quadcopters simultaneously. The ground station will receive the following information from the Raspberry Pi: GPS coordinates of both quadcopters' current location, both of their specific altitudes, and an estimation of the amount of battery life left on board each quadcopter. Although the avionics computer will serve as the direct communicator to the global positioning sensor on board, the avionics computer will send the information from the GPS to the Raspberry Pi and relay these coordinates to the ground station via the wireless network. One may wonder why the GPS sensor is not directly connected to the Raspberry Pi. An explanation to be discussed in the next section 4.2., software operation, lies in the way the autonomous flight is conducted.

4.2. Software Operation

The software operation has gone through several changes as the design phase has progressed. As mentioned in the previous section 4.1., hardware operation, the team has gone through some debate as to exactly what tasks are to be handled by the Raspberry Pi navigational computer, the microcontroller avionics computer, and the ground station laptop. This section will present an overview of what the team has decided upon for the software to be programmed on each of these three computer systems.

To begin, the team will first focus on the software to program the manual flight of the quadcopter. The team will accomplish manual flight using an RC controller on land which sends signals to a receiver connected to the avionics flight computer. The team will program the flight computer to respond to movement of the two joysticks on the RC controller. In this context, it is to be understood the word respond entails sending signals to each of the 4 motors through the ESCs. Manipulating the left joystick upward should correspond to an increase in throttle by increasing the speed of all four propellers. Manipulating the left joystick from left to right will correspond to a change in yaw. In order to understand how this will be accomplished, one must understand that two propellers spin clockwise and two propellers spin counterclockwise. Propellers spinning in the different directions are directly adjacent to one another and propellers spinning in the same direction are on the same axis on opposite ends of the quadcopter. A change in yaw will be accomplished by increasing or decreasing the speed of two propellers spinning in the same direction. The complexities of programming these functions become apparent as more details are exposed; however these complexities seem trivial compared to the programming of the autonomous mode. Varying the right joystick up and down will change the pitch of the quadcopter, tilting the "front" of the quadcopter downward and upward, respectively. The word front is placed in quotations because there are two ways of orienting the front of the quadcopter as shown in figure 2 on the next page.

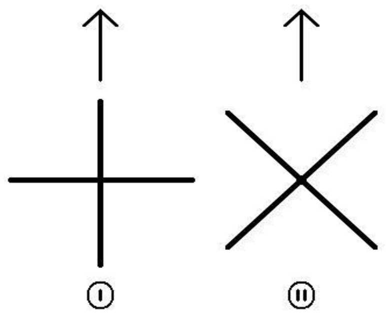


Figure 2: Quadcopter Movement Orientation

In a typical helicopter or a plane, the front of the vehicle is obvious, but due to the symmetrical nature of quadcopters, the front is more ambiguous. If the team designates the front to be on the axis of the wing indicated by I in figure 2, only two motors' speeds are manipulated in changing the pitch. An example is helpful in explaining how the pitch is changed in the first orientation. Imagine the pilot presses the right joystick on the RC controller forward to tilt the nose of the quadcopter downward. In the first orientation, the two "side" motors perpendicular to the front axis are unchanged and the "front" motor's speed is decreased with respect to the "back" motor. In orientation II in figure 1, in which the front is designated between two wings, two front motors' speeds are changed with respect to the two back motors' speeds.

As one may imagine, pressing the right joystick on the RC controller from right to left also depends on how the front of the quadcopter is oriented. In orientation I, the one of the two side motor's speed is manipulated with respect to the opposing side motor change the roll of the quadcopter. In orientation II, either the two left motors or the two right motors' speeds are changed with respect to each other to roll left or right. The team has decided to go with orientation II because orientation II has the potential for greater torque during turns. This larger torque is possible because the team can program the quadcopter to utilize all four motors during turns in orientation II. In orientation I the maximum amount of motors utilized in turning is only two. These concepts will be taken into consideration in programming the manual flight mode.

To program the autonomous flight mode, the team's avionics computer will use a Tiva™ C series ARM® Cortex™ microcontroller running an RTOS or a real time operating system. An RTOS is necessary for the information from the IMU to be transmitted rather quickly in order to quickly stabilize the quadcopter during flight. The avionics computer has a high frequency of checking altitude, orientation, and velocity to prevent the quadcopter from traveling off course or crashing. By contrast, the information coming from the GPS will not need to be checked as quickly because it's coordinates in space are not nearly as important. The team will program so called priority to the more important tasks in the avionics computer. The navigational computer and the avionics computer of each quadcopter share an interesting relationship to achieve autonomous flight. The avionics computer is inadvertently used to control the motors in flight, and the navigational computer is used to keep track of where the quadcopter is located at any given point in time as well as where the quadcopter is to travel. The avionics computer will take in data from the GPS sensor and send this information directly to the navigational computer. When traveling, the navigational computer will constantly compare the GPS coordinates of where it is currently located in space as well as the location it is to travel to. The navigational computer will subtract the location of its current coordinate from the desired coordinate and generate a direction in which the quadcopter is to travel. The avionics computer will receive this directional vector and compute some computations to extrapolate a specific thrust, yaw, pitch, and roll. These instructions will be used to send to the electronic speed controllers just as the RC controller does in order to move the quadcopter in space. The end result is reaching a desired destination for which the quadcopter is to hover and snap a photograph.

The navigational computer will also be in charge of the paths of each quadcopter. The team will specify an altitude as well as the latitude and longitude of 4 distinct points on the earth in the shape of a rectangle. The rectangle created by connecting these 4 points will serve as the area in which the quadcopters will survey. The team will first perfect the reconnaissance with a simple rectangular geometry. Once the quadcopters can accurately create the map of this rectangular shape, the team will test other shapes. Eventually, the team strives to program the navigational computer to take any 3 or more input coordinates and map out a variety of geometric maps. In the rectangular map, the paths of both quadcopters will be somewhat simple, sweeping either from left to right or from top to bottom. The quadcopters will stop and hover to take pictures at locations which are in line parallel to the outside lines of the rectangle. The paths and the locations at which the quadcopters will take pictures become more complex to program at more unfamiliar geometries. The navigational computer will determine distance between photos taken based on the altitude specified. A higher altitude will permit more space between consecutive pictures due to a larger area covered per photograph.

The quadcopter will ideally maintain a specified altitude, and fly horizontally from one location to the next. It would be convenient if the quadcopters did not have to stop and hover to take pictures. Unfortunately the quadcopter cannot take pictures while traveling along its path without stopping since the camera is mounted directly on the bottom plane of the quadcopter. The team considered an alternate design in which the camera dangled directly below the quadcopter. In this alternate design, the force of gravity on the camera would force the camera's vision to look directly downward at all times. After much contemplation, the team decided against this design reasoning one major goal of the project aimed to have resilience toward disruption. A dangling camera would be subject of unsuspected outside forces such as wind or surprise collisions.

The last, and arguably most complicated, programming lies in generating two quadcopters working in tandem. This task is daunting due to the implications of both quadcopters communicating with each other to avoid mid-space collisions, as well as coordinating the two copters' paths to optimize surveillance paths. In summary, the software algorithms throughout the duration of the project progressively get more and more complex providing an difficult challenge for the team to tackle.

5. Detailed System Design and Analysis

5.1. Detailed Block Diagrams

In designing the quadcopter project, the team decided to begin with general concepts and move into more detailed system design. In this section, the team will discuss a few block diagrams. Starting from basic block diagrams and moving on to more specific block diagrams will paint a picture of exactly what subsystems are included in order to make the team's end goal of dual quadcopters, working in tandem, a reality. Each block diagram will provide a visual representation of how each subsystem in the team's assembly will interact. The team has created three basic block diagrams titled individual system block diagram, hardware block diagram, and software block diagram. After a thorough understanding of the basic hardware block diagrams, this section will delve into two detailed system block diagrams called prototype block diagram and PCB block diagram and lastly discuss the software block diagram.

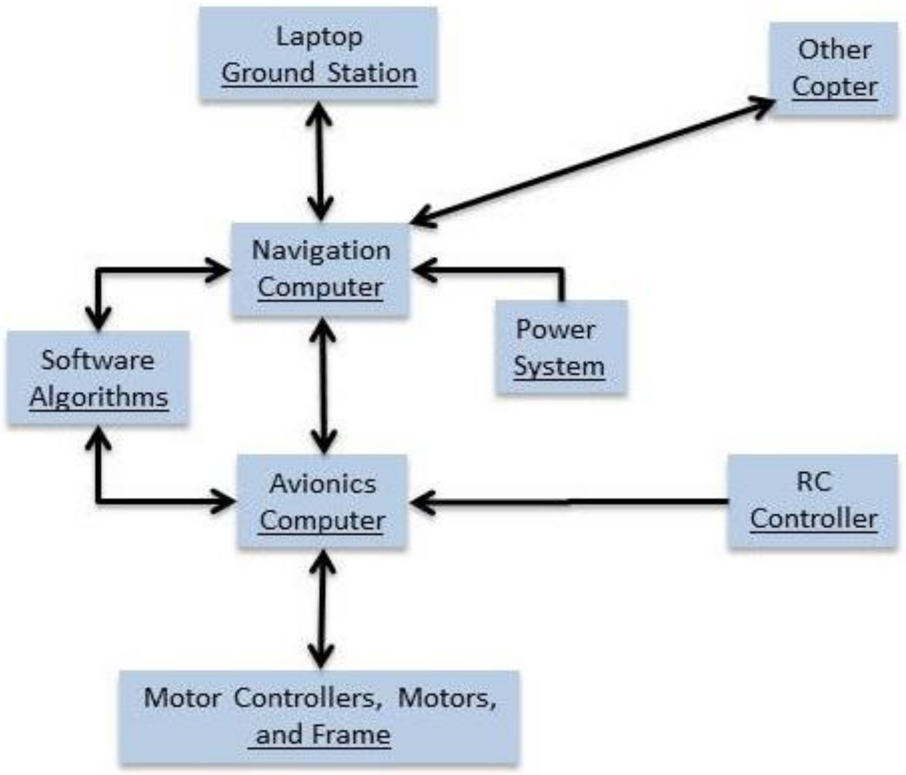


Figure 3: Individual System Block Diagram

The first block diagram is shown above in figure 3. The first and the second block diagrams discussed are primarily used to obtain a general idea of the constituents of the quadcopter project while the following two block diagrams are expose the fine details and contain boxes for each electrical component and the connections between them.

Specifically the prototype block diagram corresponds to the first phase in our quadcopter design, and the PCB block diagram corresponds to the second and third phases in our quadcopter design integrating the microcontroller, GPS, and components in the IMU onto a printed circuit board. The last block diagram, the software block diagram, will provide insight into the programs which link the hardware components to each other.

The team's first block diagram, shown above in Figure 3, is titled "Individual System Block Diagram" and shows how each of the systems will interact. The individual system included in this block diagram include: a ground station controlled by a laptop, a navigation computer, software algorithms, a power distribution system (one for each quadcopter), a computer in each quadcopter controlling the avionics, a manual RC controller, motor controllers, motors, frame, and a separate quadcopter. The black lines with arrows on one or both sides, pointing from one box to another, give specific direction showing how each individual subsystem interacts with one another. When the black lines have arrows pointing in both directions, to and from each box, the individual systems will interact in a way that exchanges information. With two sided arrows, the subsystems take turns sending data back and forth. When a black arrow points in only one direction, as one may guess the transfer of information is in one direction, and no information is sent back.

The only communication of the other copter shown in the diagram is with the navigational computer of the initial copter. It is worthwhile to note that there exists another navigational computer within the box called "other copter" which also has its own avionics software, power system, RC controller, motor controllers, motors, and frame not shown on this block diagram. One can think of the individual system block diagram as the block diagram for the initial copter. A hypothetical block diagram for the other copter, if it had been created, would look identical to this individual system block diagram with one distinction being the text "other copter" would be changed to "initial copter". The rationale behind not including all of the components of the other copter in our block diagram of concern is to minimize complication. The block diagram's function is to give a simple representation that can be understood upon first glance. It is to be implied that the hardware and avionics software of the other copter is very similar to that of the first copter. Both of the quadcopters will send information to and from each other's navigational computer, and both of the quadcopters will send information to and from the ground station laptop. The individual system block diagram visually embodies the information presented in the previous section 4, design summary, where one can identify the connection of the avionics computer to the navigational computer directly through the hardware as well as through the software algorithms. The individual system block diagram provides a solid foundation which can be expanded upon with the hardware block diagram.

The hardware block diagram shown below in Figure 4 is similar to the individual system block diagram in that the team has chosen to use black arrows to show the relationship between elements in the diagram. One main distinction is that in the hardware block diagram, unlike the individual system block diagram, there is a specification of the communication method used to transport data from box to box. By looking at the diagram, one may notice the indications "USB," "SPI," "UART," "PWM," "I2C," and

"Wireless" above or to the right of each black arrow. All of these abbreviations are different signals when communicating with each electronic device. The team has carefully chosen to use each communication method for reasons such as speed and compatibility. The IMU will send the data from its sensors to the flight computer using I2C, and the GPS unit will send data from its sensors using UART. I2C is an abbreviation for inter integrated circuit and is a type of bus called a multi-master bus. UART is an abbreviation for Universal asynchronous receiver/transmitter and is used in this case for serial transmission or delivery. PWM is an abbreviation for pulse width modulation and will be the signal sent from the avionics computer to the motor controllers. Pulse width modulation is an ideal method of communication because it is relatively simple to program to an output pin of the microcontroller. The PWM signal from the microcontroller is a digital square wave with a constant frequency. The signal can either be on or off. The amount of time the signal is on divided by the amount of time for each period will be a fraction called the duty cycle. The duty cycle can vary from 0 percent to 100 percent and will correspond to the percentage of full speed a propeller will spin.

The RC controller will send information wirelessly to an on board receiver, an electric device that will aid in manually controlling the quadcopter. A significant amount of software programming and implementation in the microcontroller of the flight computer is required to interpret input signals from the wireless receiver.

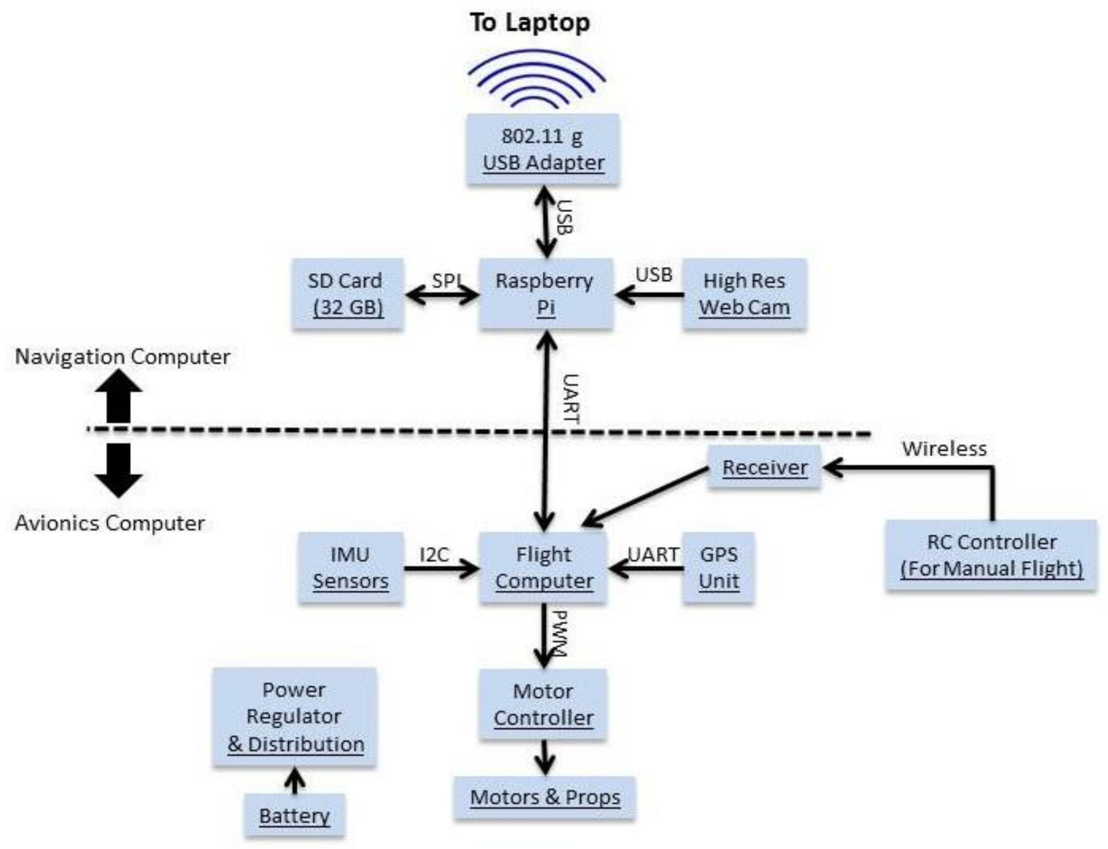


Figure 4: Hardware Block Diagram

The hardware block diagram shown in Figure 4 is a visual representation of the information discussed in the previous section titled 4.1., hardware operation. By looking at the diagram, one can see the navigational computer connected to the flight computer, the camera module, the SD card, and the 802.11g wireless adapter. The navigational computer can be thought of as a separate mechanism, not necessarily needed for stability and manual flight. This is visually represented by a horizontal dotted line through the center of the hardware block diagram. In this diagram, it is implied that this power distribution system gives power to every electronic device on board in order to eliminate the need of drawing lines from the battery to the motor controllers and the voltage regulator circuit and from the voltage regulator circuit to the Raspberry Pi, the flight computer, the IMU, and the GPS unit. The concepts visually portrayed in the hardware block diagram are to be expanded upon in the following block diagram called prototype block diagram.

The prototype block diagram shown in figure 5 on the next page shows all of the connections to and from each of the electrical components in the first prototype of the team's phase one design. This block diagram is extremely detailed and relatively difficult to read. The team decided to use the individual system block diagram and the hardware block diagram as a prelude to this more meticulous block diagram with arrows pointing each and every direction and even overlapping one another. The lines from the battery to the 4 ESCs and the voltage regulator circuit will be composed of two wires with a voltage difference equal to that of the battery. The lines from each ESC to Motors one through four are composed on three wires due to the nature of the input power of each brushless outrunner motor. The lines from the Tiva C ™ launchpad to each of the four ESCs are known as the radio connections and have three lines. The typical nomenclature for these three lines is conventionally the positive, the negative, and the signal lines. In many radio controlled applications, the radio connections go directly into what is called a servo which communicates with an RC controller. The team's quadcopter project is significantly more sophisticated than an average hobbyist's design. The team will connect the radio connections directly to the microcontroller. This design allows for the quadcopter to achieve autonomous flight, controlled by the Raspberry Pi ™. The arrow from the Tiva C ™ launchpad to the sensor stick will be two lines. The two lines, required by the communication method called I2C are the clock line and the master/slave line. The arrow between the Raspberry Pi and the Tiva C Launchpad as well as the arrow between the Tiva C and the GPS Receiver will also two lines but will use UART as the communication method.

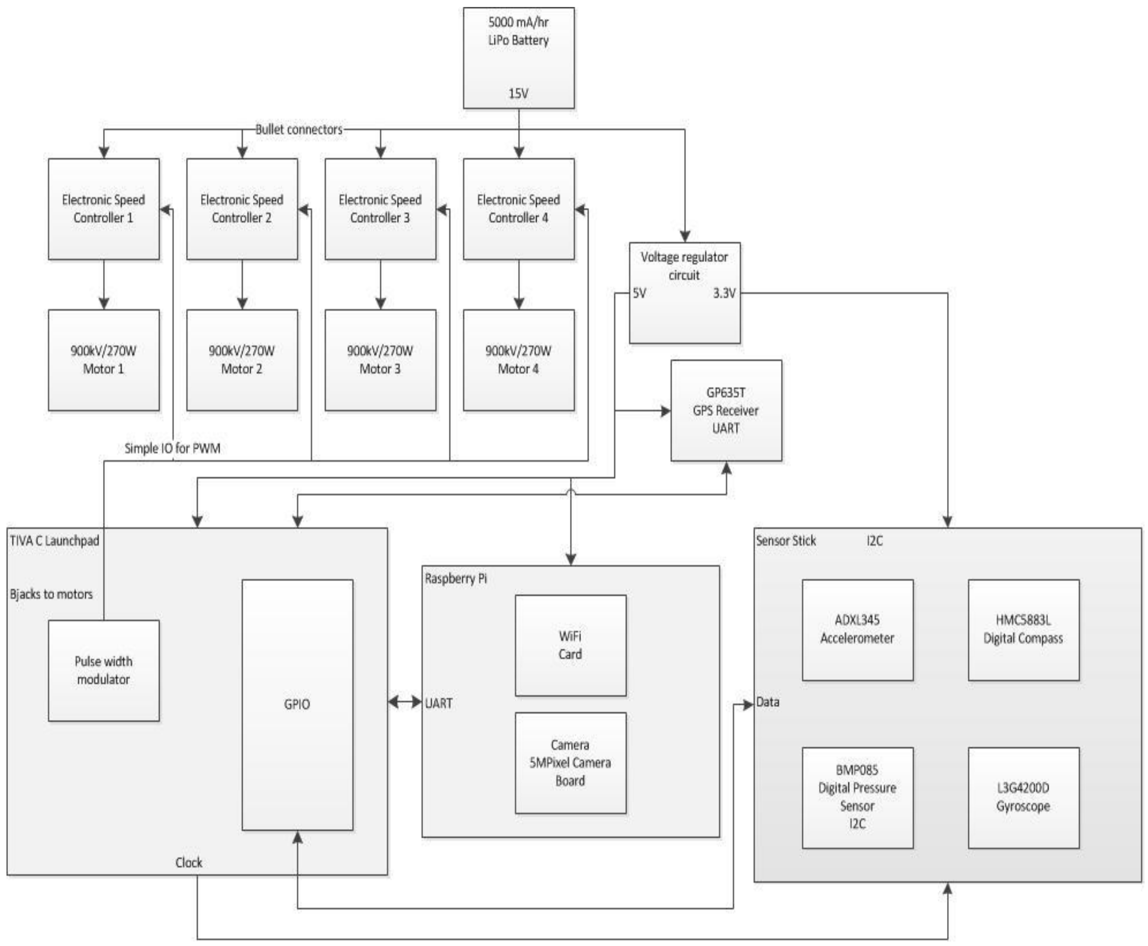


Figure 5: Prototype Block Diagram

The next block diagram shown in figure 6 on the next page titled PCB block diagram is very similar to the prototype block diagram with the exception of a few key distinctions. As one can see, the PCB box in the diagram contains all of the components from the sensor stick in figure 5, namely the accelerometer, the digital compass also known as a magnetometer, the gyroscope, and the digital pressure sensor which will be used as an altimeter. The PCB box also contains the GPS receiver and the voltage regulator circuit. The use of a printed circuit board to embody all of the external components will make the appearance of phase two and three's quadcopters more streamlined as well as significantly improve the performance of the system due to the proximity of the elements incorporated. Comparing the prototype block diagram and the PCB block diagram illustrate the advantages of the ladder design. The remaining block diagram to be discussed regards the software implemented to achieve fully functioning quadcopters.

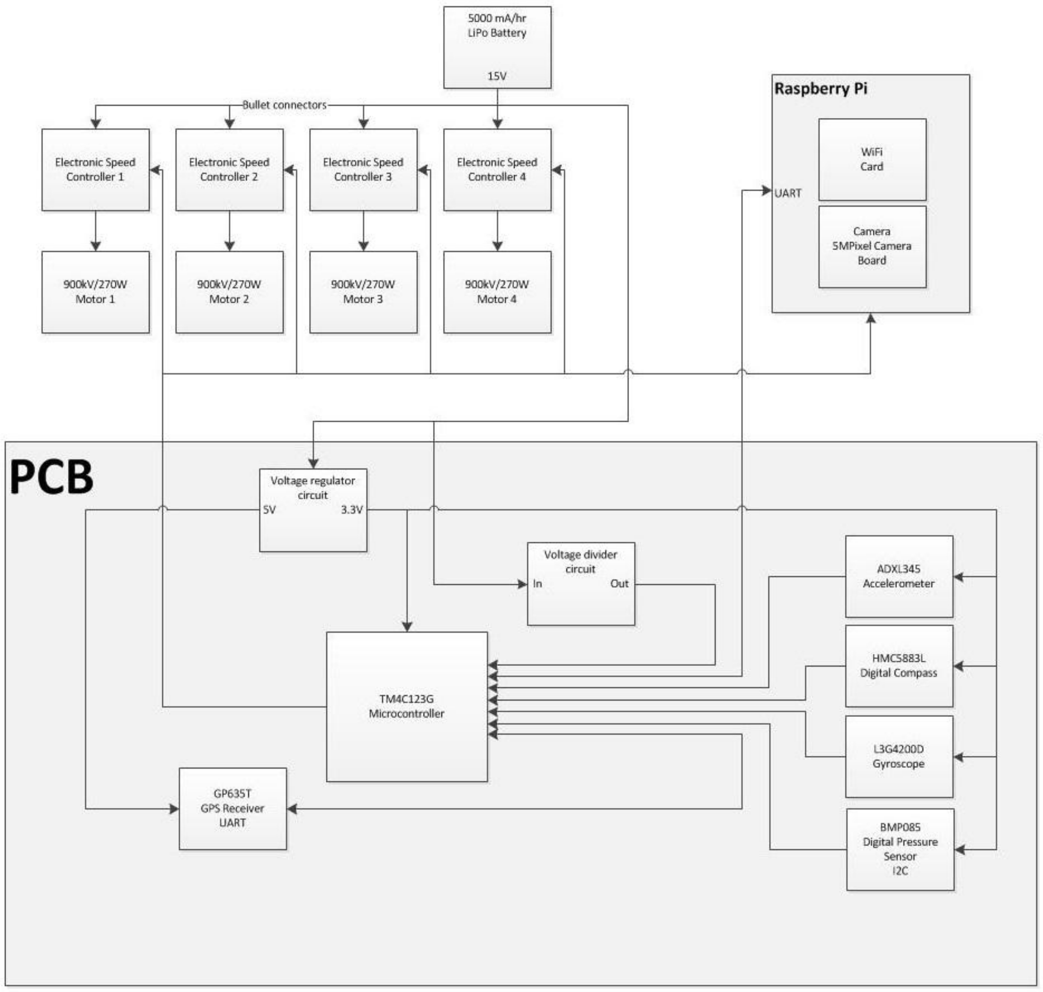


Figure 6: PCB Block Diagram

In the software block diagram shown in Figure 7 on the next page, there are three different computer systems, each of which with three components. Notice that every block on the diagram has a black arrow to and from the subsystem in which it is communicating with. The laptop is equipped with a Linux Script operating system or OS called Ubuntu 12.04. This particular OS is useful in the amount of open source programs available to the public which can be found online. As previously mentioned, one of the team's end goals is to stitch together images from the on board web cam in order to create an aerial map of a specified area. The team will take advantage of previously existing programs to help us accomplish this task. Two other software components tied to the Laptop are the disruption tolerant networking 2 (DTN2) daemon program and Telemetry Script, both of which directly communicate with the Ubuntu 12.04 OS on the laptop.

The Raspberry Pi ™ will use a different mode of operation called Raspbian. Raspbian is an operating system optimized for the Raspberry Pi hardware. Using Linux script language for communication the Raspberry Pi will have 3 software components, namely DTN2 Daemon software, Telemetry Relay software, and software for the USB Camera.

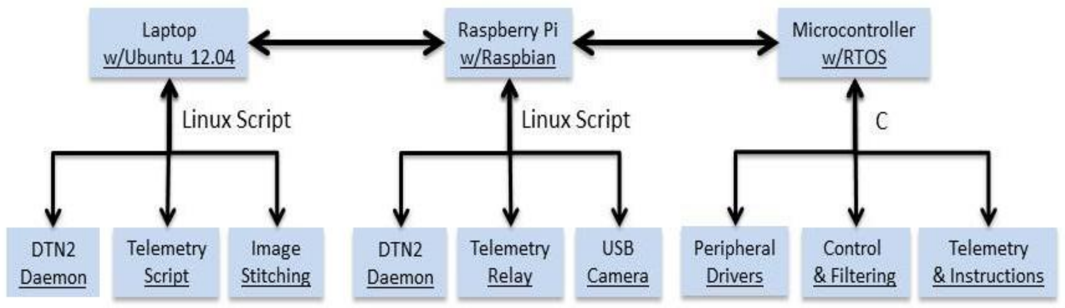


Figure 7: Software Block Diagram

The microcontroller will implement a real time operating software or RTOS. Its software will be programmed in the C programming language. Its software components include software for its peripheral drivers, control & filtering software, and software for telemetry & instructions. Programming the microcontroller during phase two will prove to be a bit trickier than in phase one. In phase one, the team will be using a Tiva C ™ launchpad with its peripherals connected to the sensors. If one looks at the Tiva C ™ launchpad, that individual will notice two square black processors one in the very center of the board and the other centered near an edge on one side. The first processor in the very center will be the processor programmed and will serve as the brain of the flight computer. The second processor on the launchpad will be used to program the first processor. Phase two's custom PCB will not include the second processor, and in this design, the team will program the processor with a program called JTAG. This JTAG programming process will require a separate computer connected to external pins created on the PCB.

5.2. Physical Design & Layout

The physical design and layout of each quadcopter will consist of integrating the mechanical components such as the frame, the propellers, and the motors with the electrical components. This section will discuss the geometry of the parts as well as how they are to be physically pieced together. The descriptions will be supplemented with pictures of models created using the Solidworks ™ modeling program. A brief narrative of the modeling process provides an initial platform to develop the physical design and layout. Each of the components in the design is modeled individually on Solidworks ™. The team has decided to put forth great determination in creating a model that is completely to scale. By modeling each component individually and combining all of the elements into an assembly, a visual representation of the prototype will assure that all of the parts chosen will fit together in a reasonable manner. If there are any sizing issues in the model, for example if the battery is too large for a chosen frame, the team can take proper actions accordingly by either choosing a battery of different dimensions or choosing a different frame with a larger clearance. Another option would be to rearrange the components in the assembly to fit a desired part. After assembling all of the components and piecing them together in a functional Solidworks ™ model, the team has made a decision to keep all of the components chosen in the design phase. Figure 8 below shows a diametric view of the assembled Solidworks ™ model. The orientations of the components in the assembly have been chosen by the group in an effort to maximize performance.

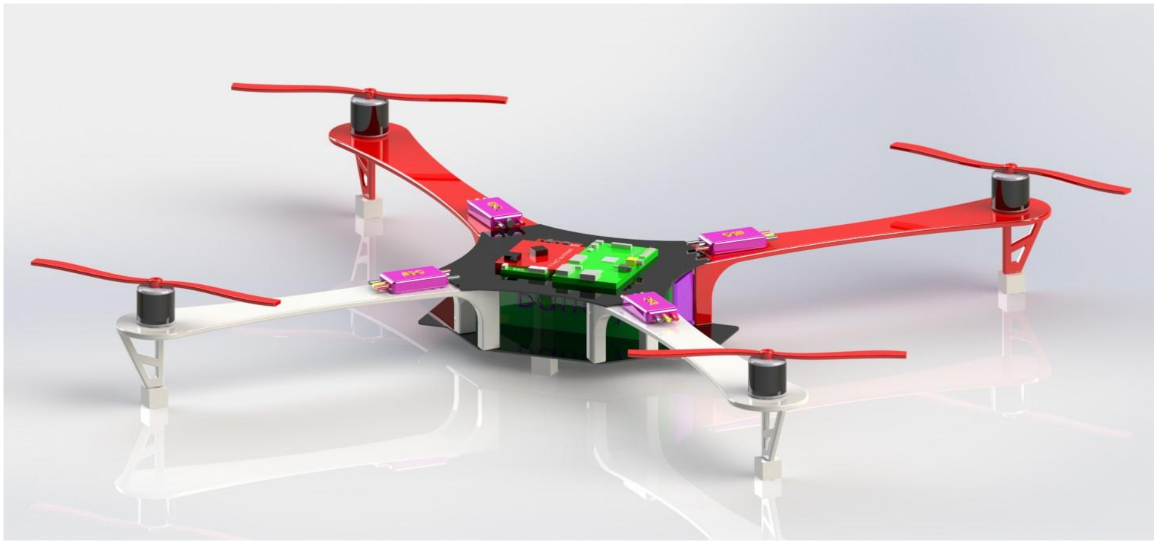


Figure 8: Solidworks Model of Full Setup

The meticulous details of the assembly are difficult to see in the zoomed out picture seen in Figure 8 above therefore more pictures are provided to reveal the assembly in further detail. Figure 9 below on the next page shows one of the wings and the parts attached to said wing. Each of the 4 wings are nearly symmetric with the exception of the color. Two adjacent wings are colored red, and the remaining two adjacent wings are colored white. The purpose of this color difference makes the orientation of the quadcopter more obvious and is especially advantageous when the quadcopter if flying at high altitudes. As one can clearly see in Figure 9, each wing has a motor and propeller pair mounted at the center of the circular section toward the outer end of the top face of the wing. Also on the top face of the wing, toward the inner end, resides a purple electronic speed controller. One may notice three small cylinders colored black, yellow, and red facing radially outward toward the motor. These three cylinders represent the starting points of the three wires feeding power to the motor. There are also three cylinders facing radially inward on the right hand side of the ESC in this figure. The two cylinders colored red and black, located on the top and the bottom in this figure, correspond to the wires going to the battery. The middle cylinder with a larger diameter is a model of the capacitor on the ESC. Although the three wires facing radially outward appear to be cut off after a short distance, it should be made clear that these wires in fact extend the entire length of the wing and connect to the three wires coming from the motor which are not shown.

Similarly, the black and the red wires facing radially inward will not be cut off, but in practice the wires will run to the positive and negative terminals of the battery. It is common practice in mechanical engineering to leave out the wires when modeling. The team has decided it impractical to model wires going to and from every element. This task would prove especially difficult for wires going from each ESC to the battery in which each asymmetrically swoop from the top plane of the frame to one of the top edges of the battery.

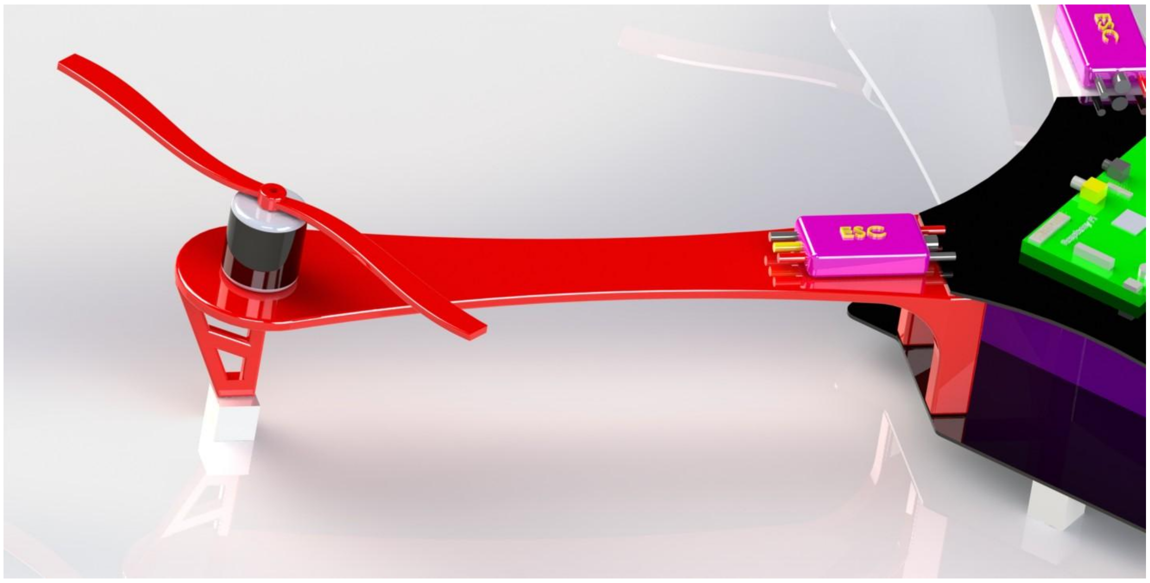


Figure 9: Solidworks Wing Model

The battery is difficult to see in the first two figures, but its position becomes more obvious after viewing the quadcopter from a different angle. Besides the four wings, the frame consists of two parallel plates, a top plate and a bottom plate. These plates will make up the center portion of the quadcopter where the avionics computer, sensors, and the navigational computer will be mounted. The frame comes with 24 identical screws. Each wing will use 4 screws to mount to the top plate and two screws to mount to the bottom plate. By inspecting figure 10 shown on the next page, one can see the front face of the battery. In the figure, said face is the green with purple letters in between the vertical supports of the wings. The battery is to be mounted to the bottom plate of the frame. The group justified positioning the battery on the bottom plane in order to maintain a lower center of gravity. This heavy base design, as opposed to a more top heavy design, has a much lower probability of flipping over during flight or during emergency landings. As one can see from the figure, there is little separation from the top of the battery to the bottom of the top frame, nonetheless, any separation will aid in keeping heat dissipating off the battery away from the to the electrical components on the top frame. The underside of the bottom plate and a white box connected to each of wing supports.

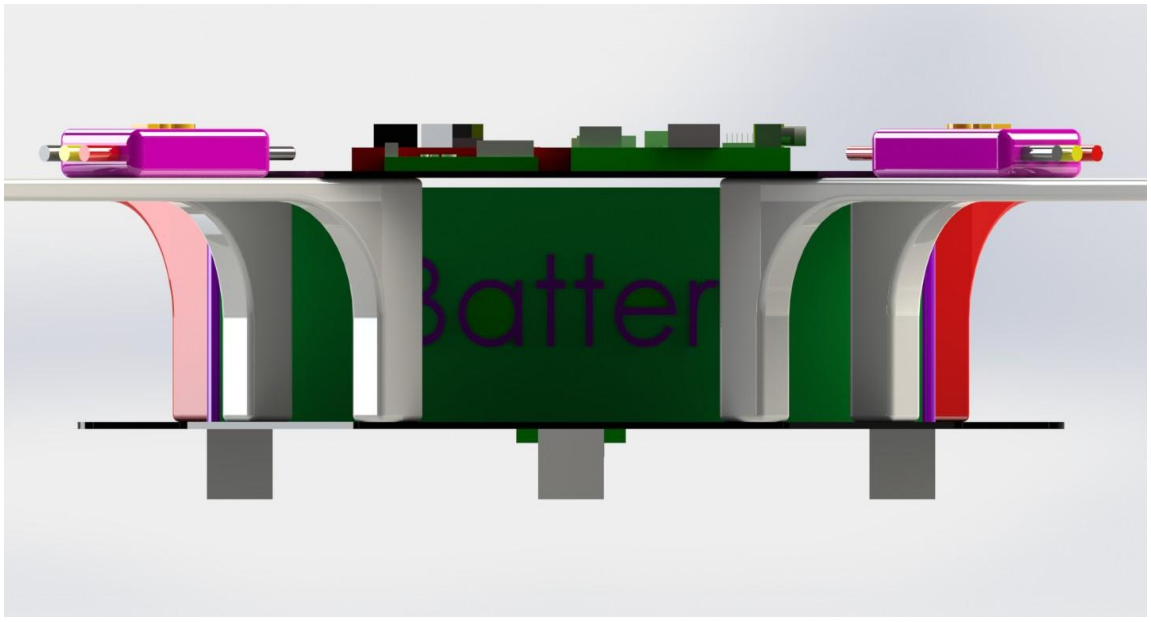


Figure 10: Solidworks Front View

Figure 11 shown below is a zoomed in top view of the prototype version of the quadcopter. It is worthwhile to mention the entire Solidworks ™ model is for the phase one portion of the team's project. Remember, phase one corresponds to our design before implementing the PCB. For this reason, one can see the IMU and the GPS separate from the microcontroller. Again, it is obvious that the connectors between components are not shown on the model. In actuality, there are a plethora of wires from the microcontroller to the sensors, ESCs, and Raspberry Pi ™.

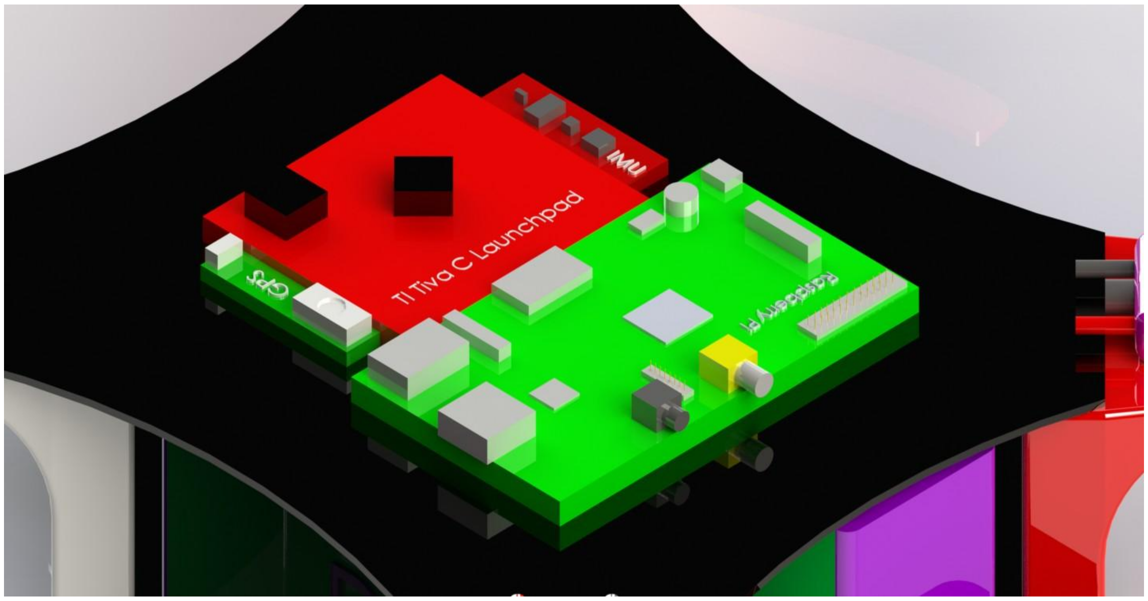


Figure 11: Solidworks Top Panel

Also not included in figure 11 is a ribbon cable from one rectangular prism on the Raspberry Pi ™ stretching downward around both the top and bottom plate of the frame to the camera mounted to the underside of the bottom frame. This camera will be pointing directly downward and will serve as the camera used to take aerial photographs during the surveillance missions. Figure 12, shown below, is an underside view of the quadcopter. The green rectangular prism represents the board on which the camera is mounted and the black cylinder is the camera itself. The four white boxes mounted to the underside of the bottom plate of the frame are made of Styrofoam. The team realized since the bottom plate of the quadcopter is almost level with the ground the camera's lens was actually protruding further than the legs on each wing. The addition of Styrofoam buffers to the bottom plate of the frame as well as on each support leg keep the bottom of the quadcopter frame elevated off the ground enough to protect the camera from getting damaged by the ground.

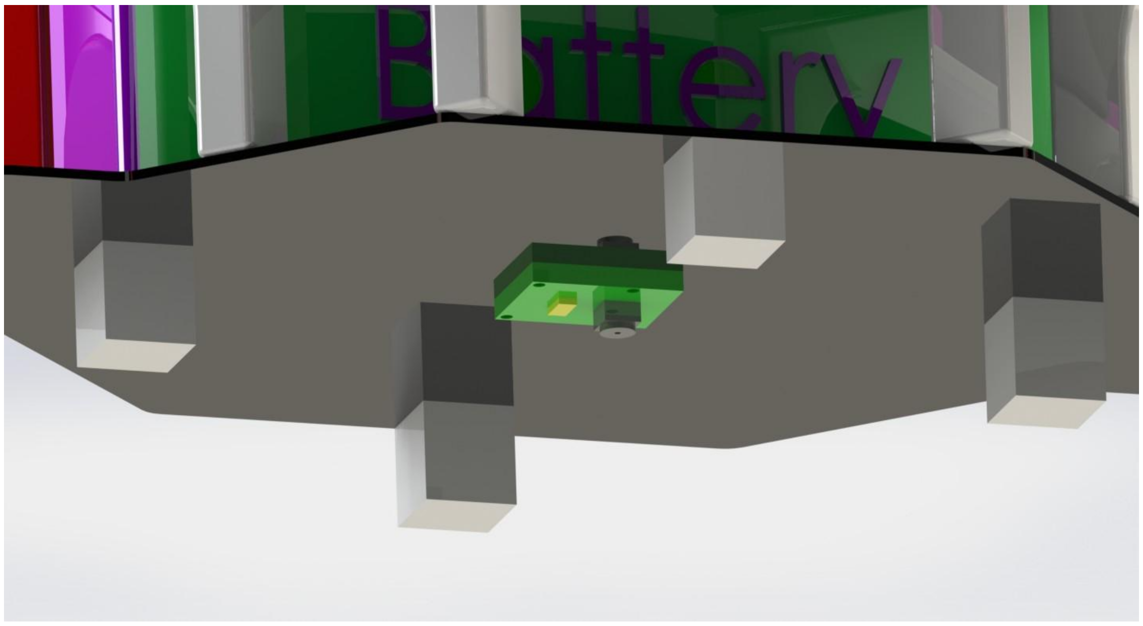


Figure 12: Solidworks Camera Model

To get a general idea of the dimensions of our quadcopter, figure 13 below shows a side view of the frame from wing to wing. The width of the quadcopter is 450 millimeters in length or approximately a foot and 5.7 inches. The height including the buffers, the frame, the motors, and the propellers is 90 millimeters or a little over 3.5 inches. This is a relatively large quadcopter. The large quadcopter design is beneficial for stability aiding in remaining steady when the camera takes pictures at high altitudes.

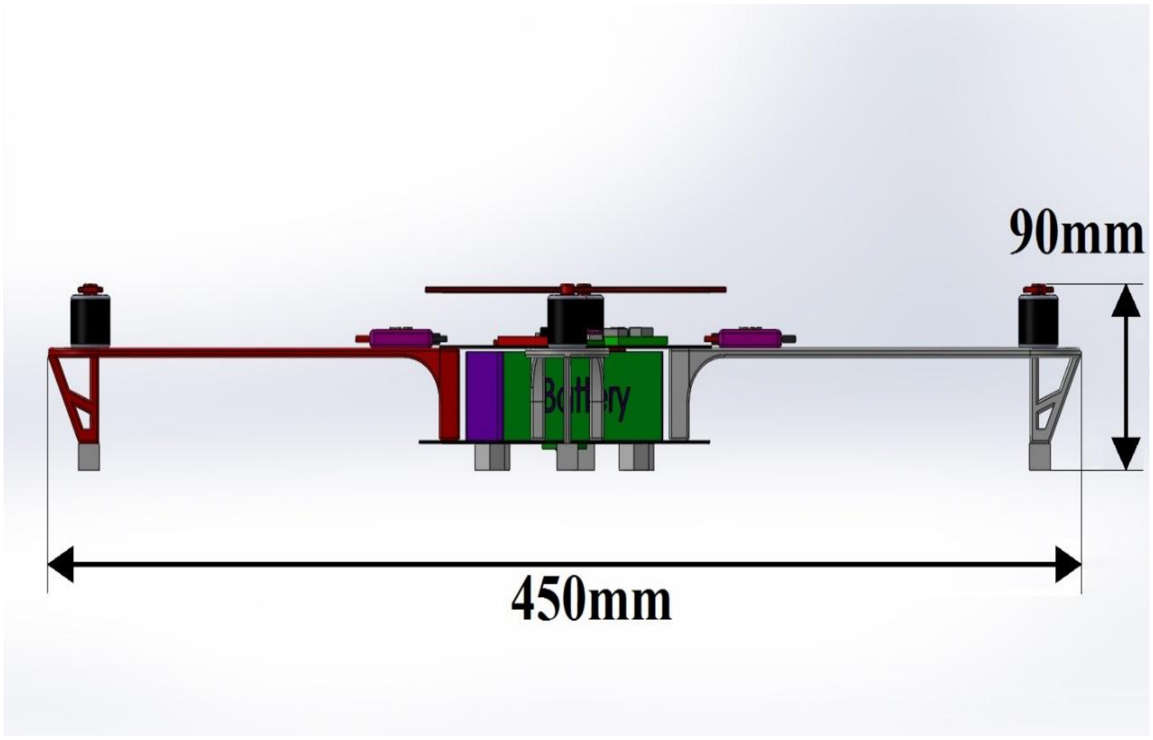


Figure 13: Quadcopter Dimensions

5.3. Microcontroller: RTOS and Peripheral Drivers

In order for a microcontroller to autonomously control anything, it needs to have peripherals. These will be sensors that provide the microcontroller with information about its state and surroundings. The processor will then have to make decisions based on the information received by its sensors. Since this project involves autonomous flight, the team will have to write code in such a way that the quadcopter is fully independent in any situation. Since procedural programming is capable of entering into a state in which it can cause the program to freeze or crash, the microcontroller will be using a real time operating system provided at no charge from Texas Instruments called TI-RTOS.

When the code for the microcontroller is compiled with the Code Composer Studio™ Integrated Development Environment, only the components of the RTOS that were used will be complied with the application to save memory on the microcontroller. Figure 14 shown below, reprinted with permission from Texas Instruments, provides some insight on how the code for this project is going to work with the TI-RTOS. Everything that needs to be used for the quadcopter application will be declared in main( ). This does not actually execute until the BIOS called and everything is declared before that call. The drivers that will be used for this application such as I2C and UART will be declared in the main function as an array. This allows the CCS IDE to only compile the drivers that are needed. When the application wants to use a driver it simply makes an API call to the RTOS which then queues the task to be executed. When task is ready to be executed, the RTOS sends it to the TivaWare component that converts the task into the assembly language appropriate for the exact Tiva™ C microcontroller.

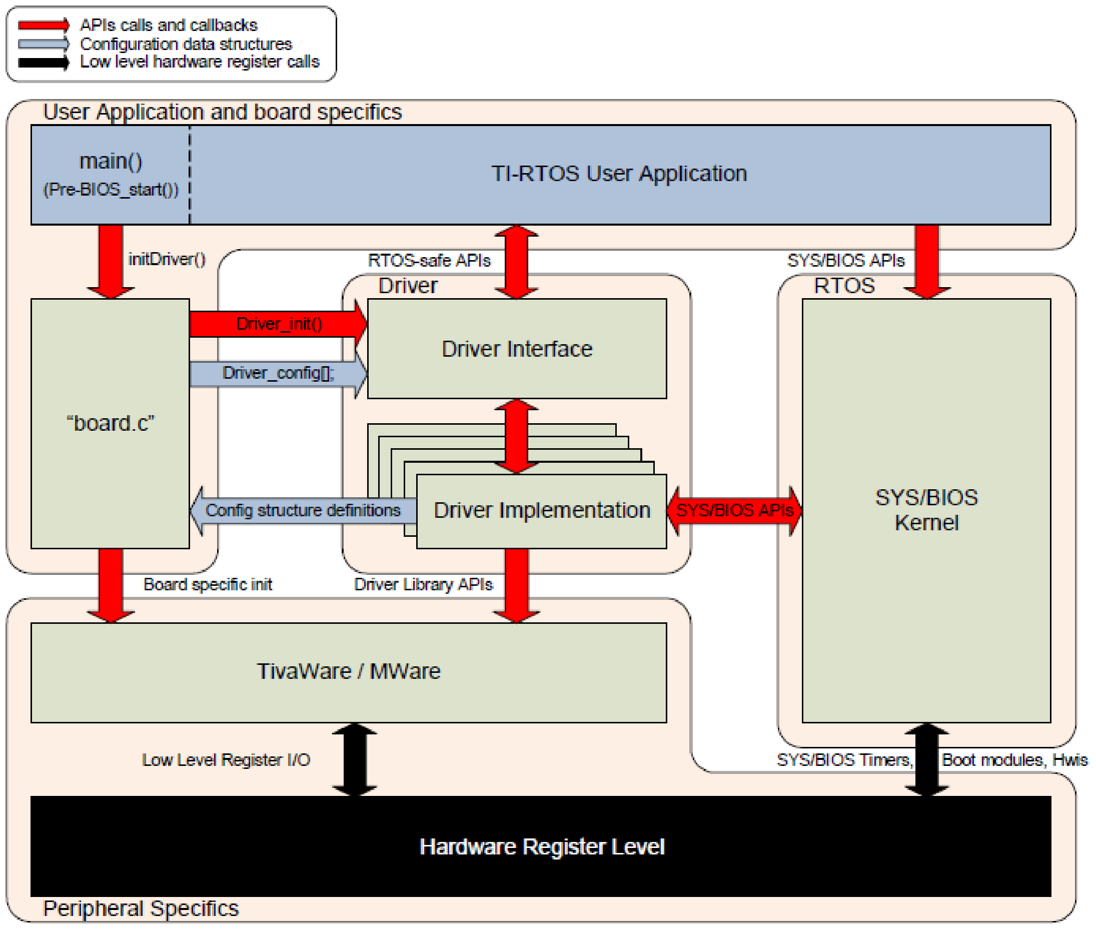


Figure 14: TI-RTOS Overview Diagram, Courtesy of Texas Instruments

The real time operating system will be running on the Tiva™ C series ARM® Cortex™ microcontroller at all times. The operating system has to be in real-time or the quadcopter will not function properly. The microcontroller is controlling everything related to flight and stabilization. It will have several peripherals such as an accelerometer, magnetometer, GPS, altimeter and gyroscope. The RTOS will be checking each of these peripherals at scheduled intervals. Each sensor has a minimum time frame for the data to be sampled. The reason the project needs an RTOS is so it can read all of the sensors in an efficient manor without tying up the processor and waiting for a response. The most important features of the RTOS are scheduling, priorities, and preemption.

All of the sensors have to run on a predefined schedule. Each time a sensor needs to be read, a task will be sent to the operating system to access the appropriate driver. The three axis accelerometer and gyroscope are the most important components of the sensors in order for the quadcopter to stabilize itself. The accelerometer can detect movement on any axis less than a single degree. This will have to have a high frequency of reads performed by the RTOS which will be done multiple times a second. The gyroscope will also have to be read at a high frequency since it can provide feedback on slight movements similar to the accelerometer. Having only these two sensors will enable the quadcopter to fly level. The magnetometer is not as important for stabilization. This is just used for orientation in reference to the planet. This will be used for controlling the direction the quadcopter should move in. The microcontroller will not be scheduled to run magnetometer tasks as much as the accelerometer and gyroscope will be. The altimeter is used for determining the altitude that the quadcopter is at. This is only important for stabilization on the z-axis and navigation.

The global positioning sensor is set by default to take a measurement once per second. It could be increased to five times per second, but is not necessary since the quadcopter does not move very fast and the system does not need to know its exact position that frequently. This is not used for stabilization; it is used for the quadcopter to move to a given location. Dozens of accelerometer reads would have been scheduled before a single GPS reading. The voltage sensor for the battery will be scheduled to run at the lowest frequency since the remaining battery life cannot be changed quickly. The team is expecting to have at least ten or more minutes of flight time so the battery would only have to be checked once every 20 seconds. Handling tasks on a schedule prevents the processor from having to waste clock cycles continuously checking sensors that will either have nearly the same information as the last time they were read form or the information is not helpful at that time. This is one of the reasons why the team is using a real time operating system.

All of the tasks will have a priority assigned to them on the RTOS. Each sensor read task will have a priority based on the importance of the sensor to stabilization. Going off-course, not being at the correct altitude, or having a communication error with the Raspberry Pi ™ is nowhere near as important as stabilization. Having an unsuccessful mission is much better than having the quadcopter crash due to a relatively unimportant task preventing an important sensor read task such as the accelerometer from being read. In most cases if a quadcopter crashes it’s due to stability, assuming it’s in a controlled environment. In general it would seem intuitively appropriate to assign priority levels that match the frequency at which the task is scheduled since important sensors need to be read from more frequently. This is something that has to be experimented with to get right. The danger here is having a task starve. For example, if the accelerometer was scheduled to be read from 20 times per second and each read takes 50 milliseconds, then the processor would be infinitely busy and when the task of reading the GPS is sent the processor, the operating system will suspend it since it has a lower priority than the accelerometer task. The quadcopter would not be able to use the GPS in this case. On the other hand if the GPS had a higher priority than the accelerometer and it came into the processor but took more than 500 milliseconds to perform then the accelerometer would not have been read in a long time and the quadcopter could become unstable and crash. In the first example, if there were only 10 reads per second then the situation would not occur. In the second example, the GPS would have to be quicker or scheduled differently. Figure 15 shows the peripheral's priorities in order with the top of the pyramid having the highest priority.

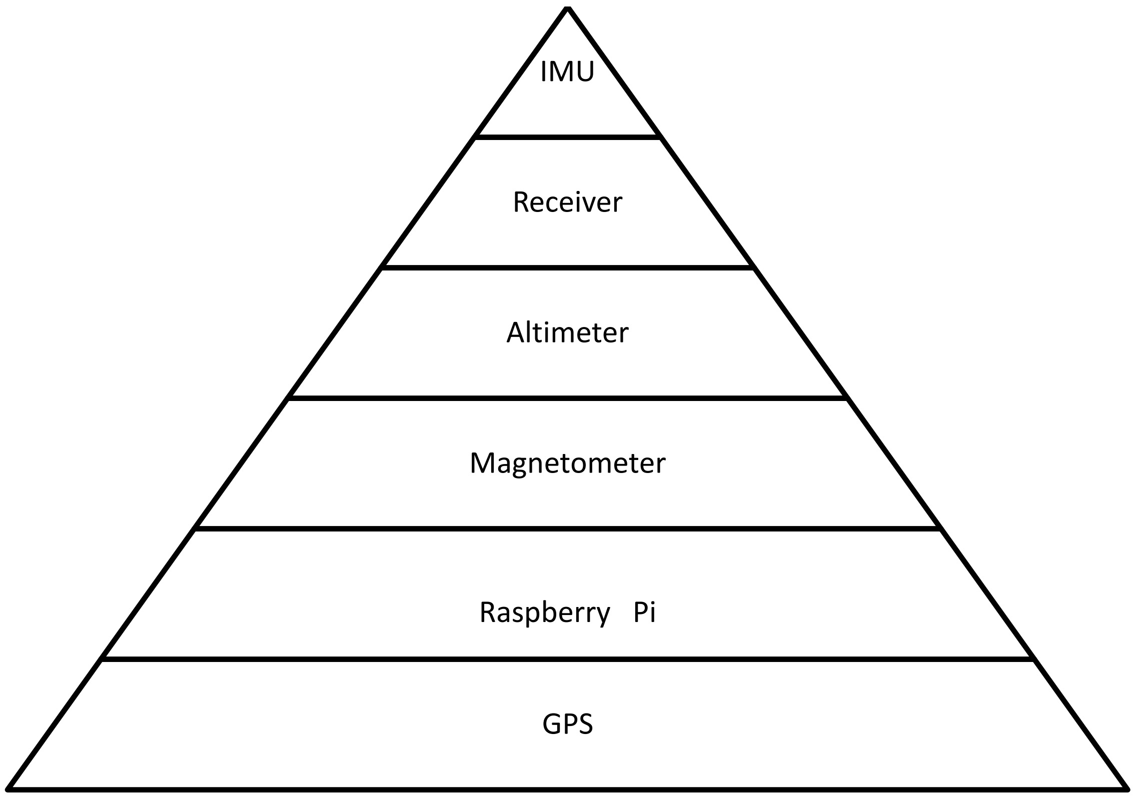


Figure 15: Peripheral Priorities

Preemption allows the RTOS to remove a process while it is being processed. This is useful when a process gets stuck, a peripheral stops responding, or when something more important needs to be processed immediately. A process can get stuck for many reasons. For example, if the code being executed enters an infinite loop, the processor would be held up processing the infinite loop and no other processes could run such as the gyroscope sensor, and this could cause the quadcopter to crash. With preemption the RTOS allows the process to execute for a certain amount of time. If the process is not complete when the time is up, it will remove the process and allow the next scheduled task to execute. This would prevent one task from stopping all of the other tasks that need to be executed on time. Communication protocols such as I2C will be used to communicate to peripherals. With I2C there is a START bit and a STOP bit. If the peripheral device malfunctions during a read such as not providing a STOP bit then it could hold up the process. Since the RTOS has preemption, it will be able to remove the process communicating with I2C and allow the next process to begin even if the peripheral device does not respond. Safety features could be added here when this happens to let the quadcopter descend gradually if it was not a critical sensor that failed. If it was a critical sensor such as the accelerometer, the microcontroller can try to restart the communication.

Preemption is also useful to have in the RTOS for this project to stop a process if a time limit was exceeded or an error occurs in the process. This would not happen very often but it could be critical. For example, if the quadcopter is flying and gets hit by a bird or runs into an object just before it is about to communicate to the Raspberry Pi ™, the microcontroller would have to stabilized itself as soon as possible in order to return back to a stable condition. If the communication procedure with the Raspberry Pi ™ normally takes one to two seconds, that might be too long for the quadcopter to wait in order to regain flight control. When a situation like this occurs, the operating system will cancel all nonessential tasks, preempt the current process, and focus on stabilization before trying to communication with the Raspberry Pi again. In order for this to happen, code will have to be written in the application layer for the essential drivers to take over based on the state the quadcopter is in.

The peripheral drivers are going to be written with the help of the examples provided with the Tiva™ C microcontroller such as the I2C and UART examples. Each device will have its own protocol for communication. Multiple devices will use I2C but each device will have its own address and may require different commands. This is why a separate driver will be written for each device. The sensors used for this project are the ADXL345 Accelerometer, ITG-3200 gyroscope, HMC5883L magnetometer, BMP085 altimeter, and GP-635T GPS. The receiver will provide feedback from the controller/transmitter. The accelerometer, gyroscope, and magnetometer are all on the same sensor stick. The power saving modes that some of these sensors have will not need to be used since they will be active the whole time during flight and their power consumption is negligible compared to the motors and the Raspberry Pi ™. All of the sensors that are using I2C will be communicating at a rate of 400 kHz.

The microcontroller will communicate to the accelerometer via I2C at address 0x1D. The accelerometer has four ranges: ±2g, ±4g, ±8g, and ±16g. Since the accelerations that the quadcopter will experience will all be small, the ±2g range will be appropriate. This is done by setting the OFSX, OFSY, and OFSZ registers to 0x7F because the scale factor is 15.6 mg/LSB in two's complement form. When the whole system boots up, the microcontroller will initiate the self-test on the accelerometer. This is done by setting the data rate to 800 Hz (400 kHz transfer rate) which requires the rate bits D3 through D0 in the register BW_RATE to be 0x0D. The accelerometer needs to be in normal power mode by clearing the LOW_POWER bit. Bit D3 of the DATA_FORMAT register needs to be set to 1 so the accelerometer will output at full resolution and Bits D1 and D0 need to be set to 0 in the DATA_FORMAT register so the range is in ±2g. Finally Bit D7 needs to be set to 1 in the DATA_FORMAT register to enable the self-test. When the self-test has finished Bit D7 will be set back to zero. The Accelerometer Block Diagram is shown below in Figure 16.

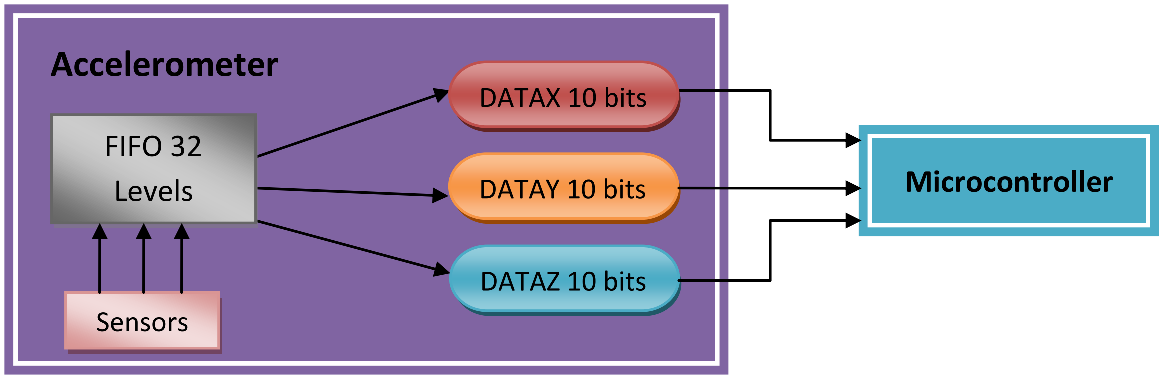


Figure 16: Reading from Accelerometer

The information that the accelerometer reads will be temporarily stored in a memory management buffer called the FIFO. The FIFO will be set in stream mode by setting the FIFO_CTL register bits D7 to 1 and D6 to 0. This fills the latest 32 measurements and overwrites old information in the FIFO as new information comes in. The reason why this mode was chosen is because the old data is obsolete as soon as new data is taken. It does not benefit the quadcopter to know what the accelerometer history was because it is essentially streaming the new data into the microcontroller since the events will all happen in real time. To read data from the accelerometer, the microcontroller will access registers DATAX, DATAY, and DATAZ for the roll, pitch, and yaw types of motion respectively, from the accelerometer. This is shown in Figure 16 above. Once the registers are read, the FIFO will automatically fill the registers with new data. The microcontroller must wait at least 5 micro seconds before reading again to ensure the new data has been fully transferred to the registers.

The gyroscope's I2C address is 0x68 if the logic level on pin 9 is low or 0x69 if it is high. The salve address can be read from register zero in the gyroscope chip. The three-axis MEMS gyroscope has built in analog-to-digital converters for each axis that are 16-bits each. The maximum internal sampling rate is 8kHz. To sample at this rate, the DLPF_CFG (digital low pass filter configuration) portion of Register 22 has to be set to 0x00 and the FS_SEL (full scale selection) has to be set to 0x03. Now the SMPLRT_DIV (sample rate divider) Register 22 has to be set. The formula below shows how to calculate the value needed between 0 and 255.

$$Sample\ rate = \frac{Internal\ gyro\ sample\ rate}{SMPLRT\_DIV + 1}$$

Since a maximum sampling rate is desired, setting register 21 to 0 will result in a sample rate of 8kHz. The table in Figure 17 shows the data output registers. These half-registers are all in 2's complement format since the reading can be positive or negative. Counterclockwise movement is positive in respect to each axis. The microcontroller will use all of them at the same interval.

8-bit Registers	**Description**
#29 GYRO_XOUT_H	Upper half of the roll (x axis)
#30 GYRO_XOUT_L	Lower half of the roll (x axis)
#31 GYRO_YOUT_H	Upper half of the pitch (y axis)
#32 GYRO_YOUT_L	Lower half of the pitch (y axis)
#33 GYRO_ZOUT_H	Upper half of the yaw (z axis)
#34 GYRO_ZOUT_L	Lower half of the yaw (z axis)

Figure 17: Gyroscope Data Registers

The magnetometer's I2C slave address is 0x1E. The microcontroller must send 0x3D to read from a register or 0x3C to write to a register. The register pointer is automatically incremented by 1 after a register has been read. This reduces the amount of communication the microcontroller has to do with the magnetometer since the microcontroller is always going to read all of the data registers in order. The magnetometer samples data in Gauss which is the unit of measurement for measuring how strong a magnetic field is. The magnetometer will only be used in continuous-measurement mode so new data will always overwrite the old data as measurements are continuously made just like the accelerometer. This is done by setting the Mode Register (MR) bits MD1 and MD0 to 0. The first thing that will be done with the magnetometer when the quadcopter boots up is to initiate the self-test. This is accomplished by setting bits MS1 and MS0 to 1 on Configuration Register A when testing positive bias and changing MS0 to 0 when testing the negative bias. The complete algorithm for the self test is covered in Section 7. The data registers shown in Figure 18 for the magnetometer are similar to the gyroscope because they are separated by two 8-bit registers, high and low. The difference is the order that roll, pitch, and yaw are in.

8-bit Registers	**Description**
#3 MSB	Upper half of the roll (x axis)
#4 LSB	Lower half of the roll (x axis)
#5 MSB	Upper half of the yaw (z axis)
#6 LSB	Lower half of the yaw (z axis)
#7 MSB	Upper half of the pitch (y axis)
#8 LSB	Lower half of the pitch (y axis)

Figure 18: Magnetometer Data Registers

For flight Configuration Register A, bits CRA5 to CRA6 will both be set to zero so none of the samples will be averaged per measurement output. Bits CRA4 to CRA2 will be b110 for a maximum data output rate of 75 Hz. The final two bits are both set to zero for normal measurement. Putting this all together the CRA register will be set to 0x18. Configuration Register B controls the device gain. This will be set to 0xA0 since only CRB7 to CRB5 set the gain which is 5 followed by zeros. The output values range from 0xF800 to 0x07FF. This will be used in the default ±1.3 Ga range since the earth's magnetic field does not exceed 1 Gauss. This is done by setting bits CRB7 and CRB6 to 0, and CRB5 to 1.

The digital altimeter is not on the sensor stick but is also accessed via I2C. The microcontroller will use address 0xEF to read from it and 0xEE to write to it. The mode used for this project will be the ultra high resolution mode by reading register address 0xF4 which requires the over sampling setting to be 3. This collects 8 samples, converts them in a maximum of 25.5 ms, and has the lowest noise. Pressure is measured in steps of 1 Pa (equivalent to 0.01 hPa). The pressure data is 19 bits long. In order to calculate what the actual pressure is from the raw sensor data, the microcontroller needs to read all of the calibration data from the E2PROM. The pressure is then calculated by using the algorithm shown below in figure 19. This is important to include because the device would not be usable without this flow diagram. This will have to be converted to C for the microcontroller to run it. The missions will use absolute altitude. The quadcopter will take a sample of the pressure when it is idle at the ground station. This will be the reference altitude for the entire mission. After the pressure is calculated through the algorithm, it can be converted to meters with the formula below. The variable p_0 represents sea level which is 1013.25 hPa. The pressure calculated is p.

$$Altitude\ (meters)\ =\ 44330 * (1 - \left(\frac{p}{p_0}\right)^{\frac{1}{5.255}})$$

Figure 19 is a summary of the actual figure in the BMP085 altimeter datasheet which is located in Appendix D, Figure 1. The steps of the algorithm will be copied into the code for the quadcopter microcontroller. The main purpose of each step is the same for both figures. The important thing to point out is that the raw data is not something that can be directly used. It has to be converted trough a series of calculations. The altitude relies on the pressure data, which relies on the temperature data, which is all calculated with the calibration data.

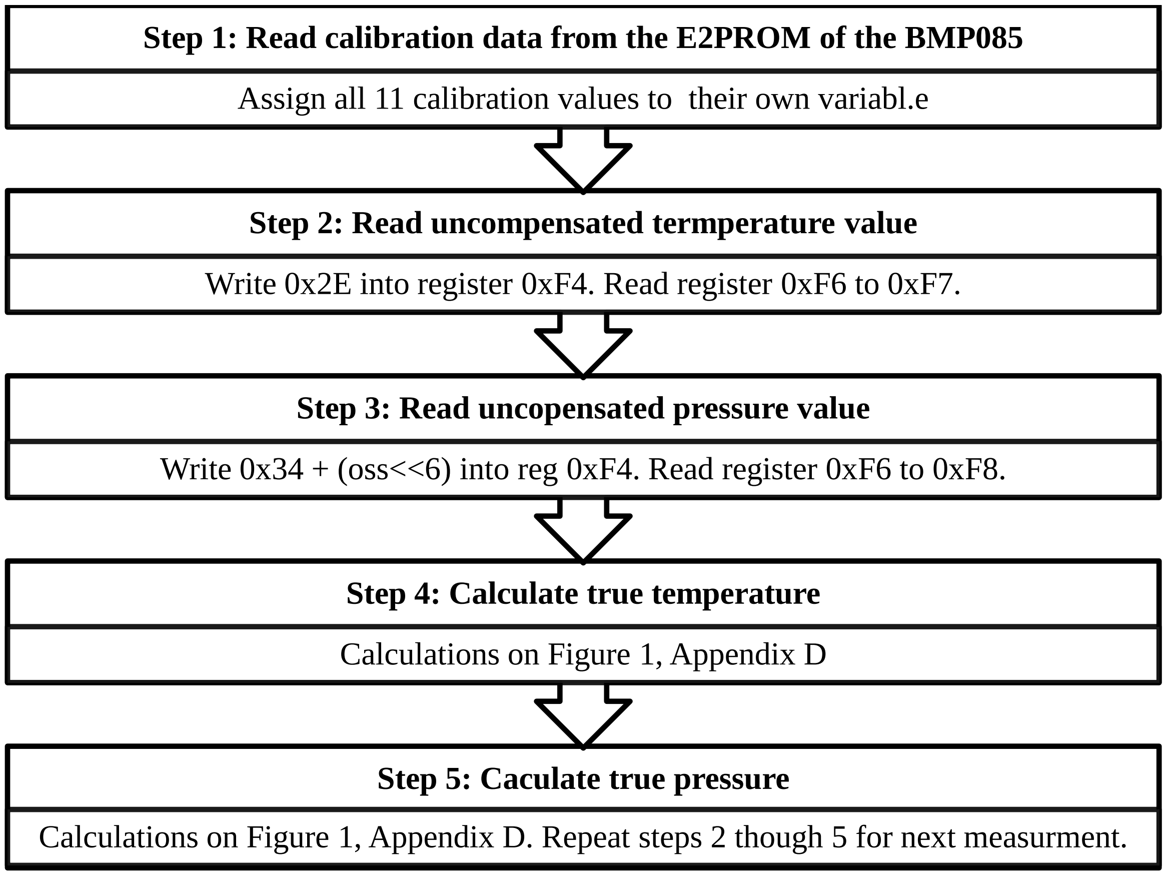

Step 1: Read calibration data from the E2PROM of the BMP085
Assign all 11 calibration values to their own variabl.e
Step 2: Read uncompensated termperature value
Write 0x2E into register 0xF4. Read register 0xF6 to 0xF7.
Step 3: Read uncopensated pressure value
Write 0x34 + (oss<<6) into reg 0xF4. Read register 0xF6 to 0xF8.
Step 4: Calculate true temperature
Calculations on Figure 1, Appendix D
Step 5: Caculate true pressure
Calculations on Figure 1, Appendix D. Repeat steps 2 though 5 for next measurment.

Figure 19: Temperature and Pressure Algorithm

The GPS sensor will communicate via UART to the microcontroller at 9600 bps 8 data bits, 1 stop bit, and no parity. The cold start time for the GPS is 27 seconds so the quadcopter will have to stay on the ground at least that long before the GPS is ready for flight. It provides data at a rate of 1 Hz. There are seven types of National Marine Electronics Association (NMEA) sentences described in Figure 20 on the next page. Each sentence will be output in the format: $GPXXX, X,X,X,X...[checksum]. The table provides a definition for each number that would be output separated by commas for each sentence type.

Sentence	**Description**
GPGGA	Global positioning: UTC time, latitude, north or south, longitude, east or west, position indicator (0: invalid, 1: GPS SPS mode, 2: differential GPS, 6: dead reckoning mode), number of satellites used, horizontal dilution of precision, mean sea level altitude, unit, geoidal separation, unit, age of differential GPS data in seconds, differential reference station ID, checksum.
GPGLL	Geographic positioning: latitude, north or south, longitude, east or west, UTC time, data validity (A valid, V invalid), and mode indicator (A autonomous, D: differential GPS, E: dead reckoning), checksum.
GPGSA	Dilution of precision (DOP) and active satellites: mode 1 (M: manual, A: automatic), mode 2 (1: no fix, 2: 2-dimensional mode, 3: 3-dimensional mode), up to 12 channels displaying the number of satellites used, position DOP, horizontal DOP, vertical DOP, checksum.
GPGSV	Satellites in view: number of messages, message number, satellites in view, ID number, elevation, azimuth, signal to noise ratio, [repeat, starting at ID number...], checksum.
GPRMC	Recommended minimum specific transmit data: UTC time, data validity (A: valid, V: invalid), latitude, north or south, longitude, east or west, ground speed, course speed, date, magnetic variation, mode (A: autonomous, M: manual, D: differential GPS, S: simulation, E: dead reckoning, N: data invalid), checksum.
GPVTG	Course over ground and ground speed: course over ground degrees, reference (true), course over ground degrees, reference (magnetic), speed over ground, unit in knots, speed over ground, unit in km/hr, mode (A: autonomous, D: differential GPS, E: dead reckoning), checksum.
GPTXT	Message from u-blox. This will be ignored.

Figure 20: NMEA GPS Output Sentences

It is important to know all seven sentences that the GPS will output since the GPS will provide all of them to the microcontroller and it will have to forward all of them to the Raspberry Pi ™ for it to parse and make sense out of the data. Some of the data is redundant, but will be updated in a non-uniform fashion. For example, the GPS may output GPRMC first then GPGGA twice in a row and then five GPGSV's immediately afterwards. The frequency at which certain sentences are provided will be found experimentally. The GPGLL could be the used the most for the quadcopters navigation since it provides the latitude and longitude needed, and is shorter than GPGGA so it may occur more often. GPGGA is like GPGLL but includes much more detailed information. This could provide more insight on the accuracy of the data. All of this data will be sent to the ground station. The ground station could perform more analysis on the data if necessary.

The receiver does not have a standard communication protocol like UART or I2C. It has pins used for controlling individual servo motors. Since there is no datasheet for the receiver, each pin will be tested individually with an oscilloscope or voltage meter. All of the joy sticks and their positions will be tested to find the correlation between movements and voltage changes on the pins. Once all of the movements are figured out, these pins will be mapped to GPIO pins on the microcontroller. The driver for this device will have to be written to control the motors of the quadcopters through pulse-width-modulation (PWM) on the electronic speed controllers (ESCs).

All of the peripheral drivers should be made into functions that work with the RTOS and can easily be called upon anywhere in the application code for requesting data or sending commands. Figure 21 is a quick summary of all the peripheral drivers.

Peripheral	**Interface**	**Address**	**Self-test**
Accelerometer	I2C	0x1D	Yes
Gyroscope	I2C	0x68 / 0x69	No
Magnetometer	I2C	0x1E	Yes
Altimeter	I2C	0xEF / 0xEE	No
GPS	UART	N/A	No
Raspberry Pi	UART	N/A	No
Receiver	Custom	N/A	No

Figure 21: Peripheral Driver Summary

With the RTOS running and all of the peripheral drivers written, the software foundation is complete. Writing code for flight should be strait foreword since the peripherals are integrated with the operating system. Function and API calls will be the primary interaction with the higher level flight stability and control code. For example, the process of getting the altitude from the altimeter involves interaction, delays, and calculations. When this is called in the application code, it will be a simple function call like: "float height = getAltitude( );".

In summary, the flight computer will have an RTOS to keep all the tasks running coherently. The flight computer will have to read all the sensors, filter the data, relay information to the Raspberry Pi ™, calculate control compensation for all axes, and output information to the motors. All these tasks will be difficult to keep running in parallel without overlap, so clever coding techniques and special care must be taking during flight computer software development.

5.4. Control & Sensor Fusion Algorithms

In this section, the design and analysis of the flight control and stability system along with the details of the sensor filtering algorithms is given. In flight, quadcopters are naturally unstable air vehicles that need to be dynamically stabilized in order to work. This conundrum can be fixed by using the proper compensation control algorithm to minimize the error on the roll, pitch, and yaw in order to produce predictable motion. When using a compensation algorithm, however, some form of digital feedback is necessary in order to have an estimate of the orientation. Unfortunately, all digital sensors have inherent noise in the system which is unfortunately unavoidable. This noise, however, can be reduced by using the proper filtering algorithm. The control and filtering algorithms used in this project are analyzed in this section.

The quadcopter must have an established orientation in order to base all control assumptions on. Figure 22 (II) shows the assumed orientation of the quadcopter with respect to the direction of travel. Also, note that opposing rotors of the quadcopter will be spinning in the opposite direction. This is to counter the motor torque that can cause the quadcopter to spin out of control. This method does not completely get rid of the rotation, but it does reduce it quite a bit. The yaw stabilization control will have to handle the rest.

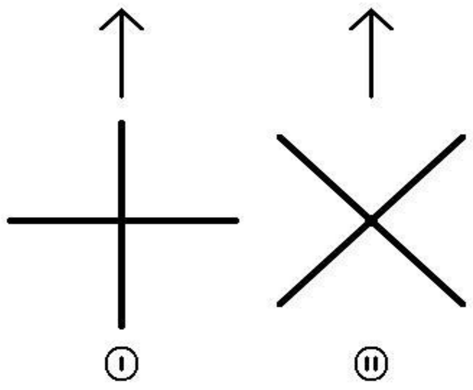


Figure 22: Quadcopter Movement Orientation

In three dimensions, the quadcopters attitude is a vector of its three Euler angles: Roll, Pitch, and Yaw. These angles will be denoted by the Greek letters: $\phi, \theta, and\ \psi$ respectively. Figure 23 on the next page shows the orientation of the quadcopter in three dimensional space. These conventions will be the basis of the calculations for the control system.

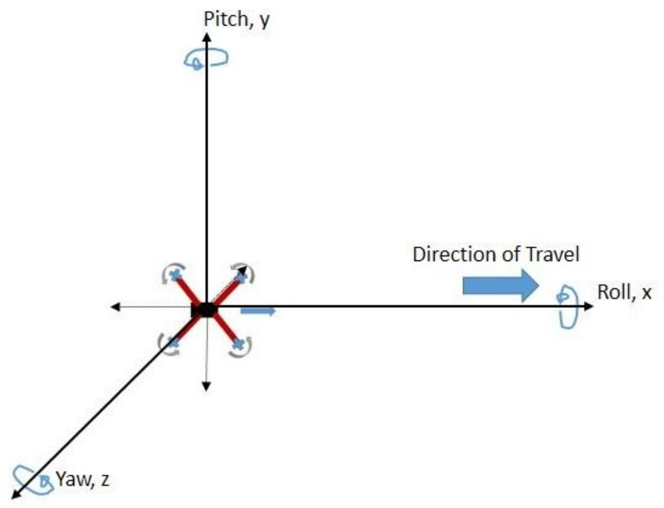


Figure 23: Orientation of the Quadcopter in 3-space

In order to move the quadcopter, a differential force must be applied on either side using the motors. This differential force creates a coupling moment that causes the quadcopter to spin along an axis. For example, referencing figure 22, if all the motors on the left produced more torque than the ones of the right, than the quadcopter would begin to roll right. Conversely, if the motors on the top half were producing more torque than the bottom half, the quadcopter would begin to pitch up. Changing the attitude of a quadcopter is critical to its motion, as the angle at which the quadcopter is oriented will produce a force vector pointing out. This principle, illustrated in figure 24, is what creates motion for the quadcopter along a specific axis.

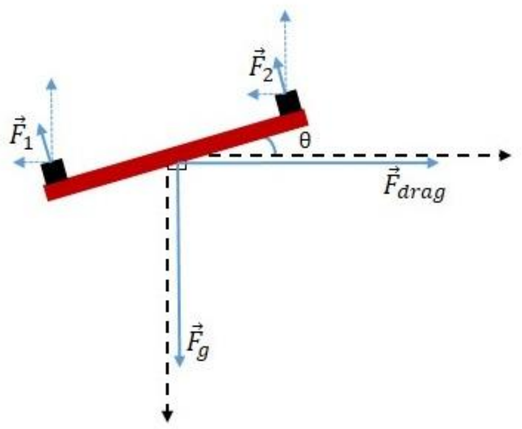


Figure 24: Quadcopter Free Body Diagram in 2-space

The roll and pitch are relatively easy to control on a quadcopter, but the yaw isn't. The yaw in a quadcopter system is caused by the torques from the spinning motors. As mentioned before, the opposing motors on the quadcopters will have opposite spins in order to cancel out the torques. While will never naturally equal zero, it can be dynamically corrected by a feedback loop. In order to make a controlled yaw spin in any direction, the speed of each motor can be adjusted in each of the motors in order to make a controlled torque deviation. This deviation will start the yawing the quadcopter in a controlled manner.

With the roll, pitch, and yaw conventions established, the control system can now be explained. The stability of a quadcopter is determined by its ability to precisely control its attitude along any axis. This prevents the quadcopter from spinning out of control and crashing. In order to control these parameters, a feedback loop will be put in place to control them. The feedback in the system will be done using the gyroscope and the accelerometer which read the angular rate and linear accelerations along three axes. Figure 25 shows the feedback control system for attitude stability.

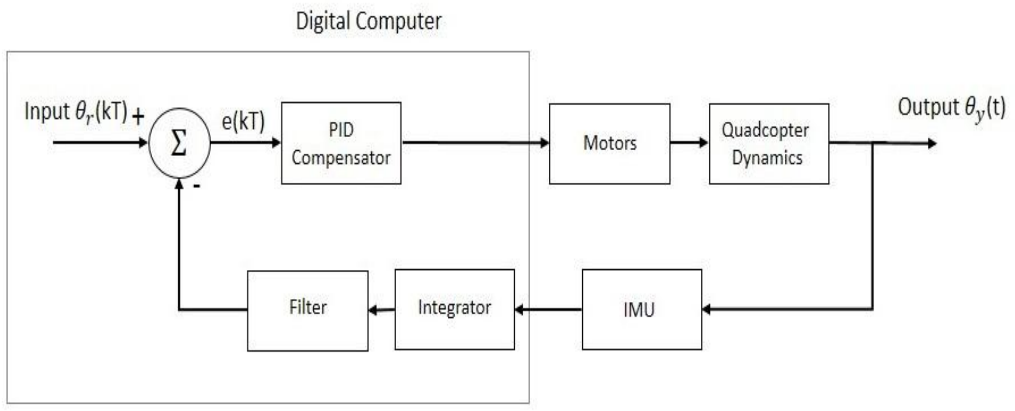


Figure 25: Attitude Feedback Controller

When the vehicle starts up, it will have to find its orientation with respect to earth. This orientation will be the initial state vector as well as the defined origin of the roll, pitch, and yaw. This means that when the roll, pitch, and yaw are all zero, the quadcopter is level with respect to the earth. So, in order to make the quadcopter stable in flight, the feedback controller will have to force the output towards the input. When keeping level, as mentioned before, the input will be zero.

Another thing to note is that the roll, pitch, and yaw each will have their own separate feedback control loop. Since each is a separate parameter, each will need its own loop. There will also be a feedback loop for the altitude of the quadcopter. This will not be done for the phase 1 prototype, but will be done when the focus shifts to the autonomous navigation.

The feedback controllers will be digitally implemented on the Tiva C ™ flight computers for this project. There are several advantages and disadvantages of doing it this way versus analog. The several advantages include robustness, adaptability, environmental stability, and cost. The disadvantage comes in the areas of delay and need for real-time processing. One assumption of most digital control algorithms is that the sample frequency is always a fixed rate and never changes. This is why the processing of the data on the flight computer must be done in real-time, which is the driving factor behind using the RTOS.

The digital sensors that will provide the feedback for the control system also need some filtering done before they can be used. Unfortunately, there is always noise within sensor system, and the must be filtered to improve accuracy and precision. In order to bring the noise down, there are a few adaptive filters that perform well. As mentioned in section three, the Kalman and the Least Squares filters are commonly used filters to get rid of digital noise. The first, the Kalman filter, is very accurate and greatly reduces noise but is computationally heavy and can hold up the processor for lengthy periods of time. The other, the Least Squares filter, doesn't perform as well as the Kalman, but it can run more efficiently on a computer with limited processing power.

For the needs of this project, it was decided that the Least Squares filter would be a better fit for the filtering algorithm. The Least Squares filter is an adaptive filter which means that it adjusts its transfer function automatically to adapt to the noise. The filter essentially works by providing the filter an estimate of what a value should be and using previous filter weights to 'adapt' the filter and minimize the error. The filter essentially learns how to most efficiently reduce the noise over time.

According to the book 'Statistical Digital Signal Processing and Modeling' by Monson H. Hayes, a least squares filter can be implemented recursively using the following method (Hayes, 784). The block diagram of the recursive least squares filter is shown in figure 26 below.

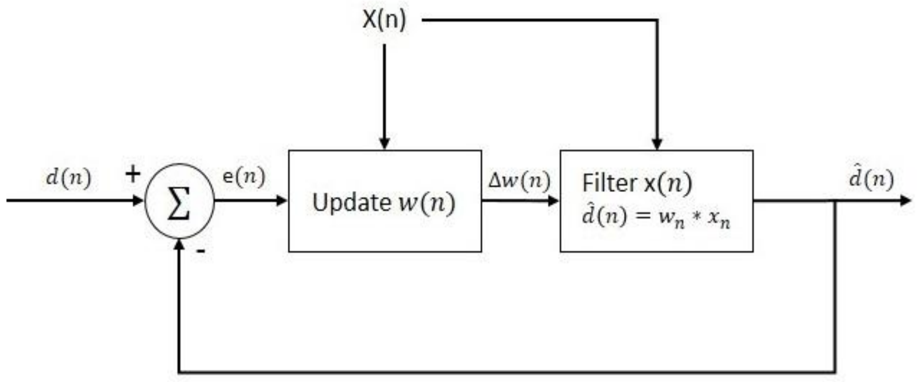


Figure 26: Recursive Least Squares Filter Diagram

Let $d(n)$ be a signal of interest, and let $x(n)$ be the signal transmitted over a noisy channel. The signal $d(n)$ can be extrapolated by a filter $w(n)$ into an estimate called $\hat{d}(n)$. This can be put into the equation form, shown below, where the noisy signal and the filter are convolved to get the estimate. The parameter 'p' represents the order of the filter.

$$\hat{d}(n) = \sum_{k=0}^{p} w(k)x(n-k) = \boldsymbol{w_n^T x_n}$$

The error in the estimate is used along with the noisy signal in order to update $w(n)$. The algorithm uses least squares methods to extrapolate a reasonable state estimate. The equations and the block diagram of the least squares filter update algorithm are given below in figure 27. The parameter 'λ' is the 'forgetting factor' of the filter, and 'δ' is the is the value for initializing P(0). The matrix 'I' is the identity matrix. This process repeats, hence the term recursive.

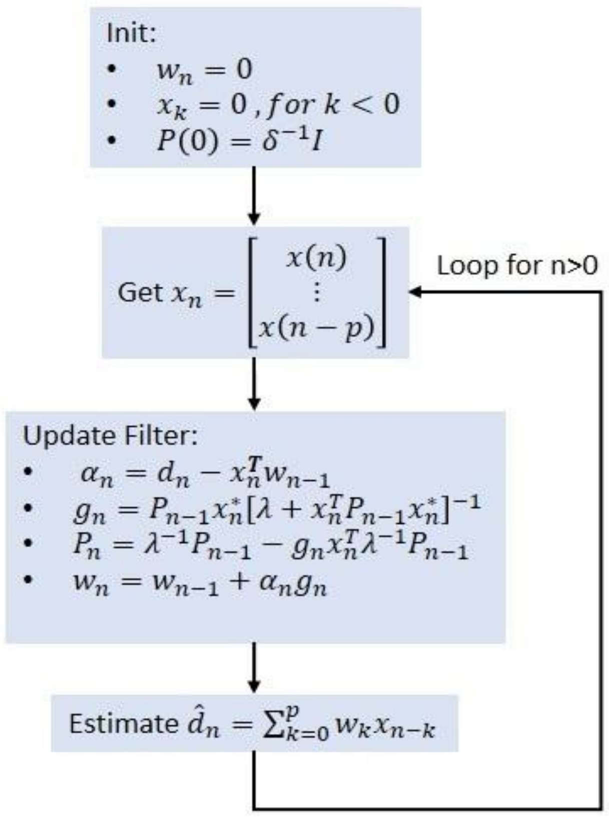

Init:
$w_n = 0$
$x_k = 0, for\ k < 0$
$P(0) = \delta^{-1} I$
Get $x_n = \begin{bmatrix} x(n) \\ \vdots \\ x(n-p) \end{bmatrix}$
Loop for n>0
Update Filter:
$\alpha_n = d_n - x_n^T w_{n-1}$
$g_n = P_{n-1} x_n^* [\lambda + x_n^T P_{n-1} x_n^*]^{-1}$
$P_n = \lambda^{-1} P_{n-1} - g_n x_n^T \lambda^{-1} P_{n-1}$
$w_n = w_{n-1} + \alpha_n g_n$
Estimate $\hat{d}_n = \sum_{k=0}^{p} w_k x_{n-k}$

Figure 27: Recursive Least Squares Algorithm

One thing to note when using any adaptive filter is that the each state must be predicted before it can be filtered. This allows for the filter to correct the state as it changes dynamically and with a control input. This can be put in equation form, shown below, where $A(n)$ is the change caused by the dynamics of the system, and $C(n)$ is the control input.

$$d(n) = \hat{d}(n) + A(n) + C(n)$$

The sensors being used as part of the IMU system include the altimeter, accelerometer, gyroscope, and the magnetometer. Each of these sensors measures specific quantities that are important to the feedback control system. The altimeter will measure the altitude of the quadcopter using the pressure and temperature of the surroundings. The accelerometer will measure the forces along all the axes using a piezoelectric resistance sensor. The gyroscope works in a similar way, but it measures angular rate over all the axes. The magnetometer will be measuring the heading of the quadcopter by using the earth’s magnetic flux as a compass.

The gyroscope being used for this project is a rate gyroscope, which means that is measures angular rate. For the control system, however, the roll, pitch, and yaw angles are what is being used to stabilize the quadcopter. This means that the rates coming from the gyroscope must be integrated with respect to time in order to get the attitude. Unfortunately, it isn't as easy as just integrating the angular rates because it isn't the same thing as the angular velocity. The angular velocity must be normalized before it can be used to find the attitude.

Given the raw sensor data, the angular rates are given by the equation shown below:

$$Angular\ Rates = \vec{w} = \begin{pmatrix} p \\ q \\ r \end{pmatrix}$$

These can be normalized and integrated using the following kinematic equation shown below:

$$\vec{\theta} = \begin{pmatrix} \phi_{k+1} \\ \theta_{k+1} \\ \psi_{k+1} \end{pmatrix} = \begin{pmatrix} \phi_k \\ \theta_k \\ \psi_k \end{pmatrix} + \begin{pmatrix} 1 & sin\phi_k tan\theta_k & cos\phi_k tan\theta_k \\ 0 & \cos\phi_k & -sin\phi_k \\ 0 & sin\phi_k sec\theta_k & cos\phi_k sec\theta_k \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix} \Delta T$$

Using these equation, the attitude can be determined that will be used in the control equations. Note that at all times, the last value of the attitude must be known. So, three memory locations must be preserved for this.

After these values are solved for, and the input is filtered, the flight computer can start performing the control algorithms. The input of the control system of this project can come from two sources, either the autonomous guidance system, or the RC controller. Either the input can throttle the motors up, tilt the quadcopter in any direction, or spin the quadcopter. Using these methods, as mentioned before, is the basis of how quadcopters can fly and maneuver.

The RC controller has a receiver on the quadcopter that has multiple channels that each represents an input to the system. Each of these channels can be used to control a flight parameter of the system. Using this as the input, the flight computer can subtract it from the feedback and obtain an error function that it can use as the compensator input. This can be expressed in equation form, shown below ,with $r(kT)$ as the input, $y(kT)$ as the feedback, and $e(kT)$as the error.

$$e(kT) = r(kT) - y(kT)$$

This is the input to the compensator, which is the heart of the control system. The compensator design is one that can be very tricky, and has many approaches. The compensator has to take the error input, magnify it in some way, and create an output that the plant can understand. The plant, in this case, is the quadcopter and the motors that are controlling the motion. The compensation can be dynamic or it can be static, depending on the needs of the project. For this project, however, the compensation will have to be dynamic so that it can have a fast and accurate enough response time, and damp any unwanted oscillations.

The compensator of choice for this project is the PID controller, otherwise known as the Proportional Integral Derivative controller. This controller gets its name from exactly what it does to the input. That is, this controller multiplies the error input by a proportional value, integrates it, derives it, and sums them up. The equation below shows an equation for the output.

$$output = u(t) = K_P e(t) + K_I \int_0^t e(t)dt + K_D \frac{de(t)}{dt}$$

Each term in the equation brings some form of value to the compensator. For instance, both the integral and the proportional terms tend to drive the error to zero, but increases instability in terms of oscillations. The derivative term damps oscillations and makes the system more stable, but slows the response and creates steady state error. The integral term by itself also gets rid of any steady state error.

With each term contributing as a whole to PID controller, the brunt of the design is to determine the gains: K_p, K_I, and K_d terms of the controller. There are several methods to determine these terms, and all of them are right, but each has advantages and disadvantages. The team narrowed the search to just two methods, and eventually decided on one of them.

The first method, finding the plant dynamics and using classical control theory to find the values, can be very lengthy and messy, but it guarantees the most efficient control algorithm that works properly. Finding the plant dynamics, however, would involve having to solve all the kinematic and kinetic equations of the system for all the parameters of interest, and turning them into s-domain equations. The plant models can't neglect gravity and air resistance in the form of drag either. These factors play a big role in the dynamics of the system.

The other method, the Ziegler-Nichols method, is a well known method of finding the three parameters based on trial and error. This method, while much easier than the first, still requires patience and time. The trial and error process goes through several iterations before finally solving for all the parameters. The end result of this method isn't guaranteed to be as efficient as the other method either. This method also requires the use of a test setup where the system can be isolated to one axis at a time and be restrained from crashing. Figure 28 shows the process that the Ziegler-Nichols method uses to iteratively solve for each of the PID parameters (Ziegler et al, 759)

	Kp	**Ki**	**Kd**
1st step	Ku	0	0
2nd step	0.6Ku	2Kp/Tu	Tu/8Kp

Figure 28: Ziegler-Nichols Method Procedure for PID Tuning

Note that the first step in the Ziegler-Nichols method was to set the proportional and integral gains equal to zero. The purpose of this step is to find the ultimate proportional gain at which the output of the system oscillates at a constant amplitude. At this gain, the period of oscillation of the system is used along with the value of the gain itself to calculate the values of the PID parameters. This process has to be done for every feedback loop for everything that is being controlled by the flight computer.

The PID controller needs to be converted to a discrete-time equation before it can be used on the flight computer. The process of doing that is relatively simple. First step is to assume that the discrete derivative of any arbitrary function used in the system. This is defined in the equation shown below, where T is the sample rate (Franklin et al, 66)

$$\dot{x}(k) = \frac{x(k) - x(k-1)}{T}$$

Using this equation, the PID control equation can be turned into the following discrete equation shown below which can be implemented digitally:

$$u(k) = e(k-1) + K_P e(k) + K_I T e(k) + K_d \frac{e(k) - e(k-1)}{T}$$

This discrete equation can easily be implemented in software on the Tiva C ™ flight computer. The algorithm for realizing this transfer function is given below in Figure 29. The recursive PID function will be called for each control parameter, and will repeat every time a new sample comes in. There will be a PID function in the code memory that accepts parameter structures, and each parameter structure will have at the most recent value, the gains of the PID, and the sampling time.

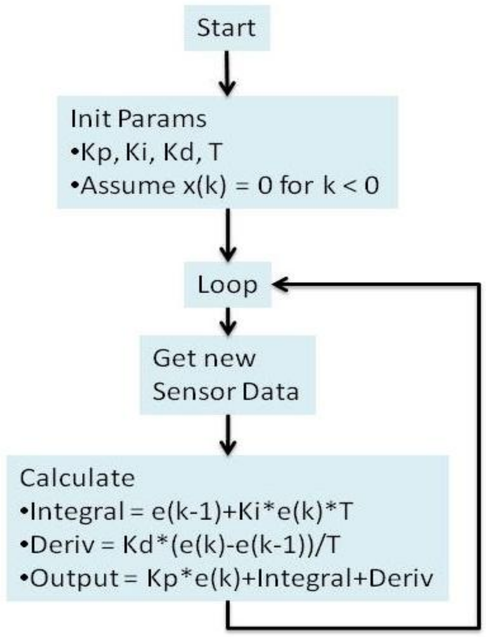

Start
Init Params
•Kp, Ki, Kd, T
•Assume x(k) = 0 for k < 0
Loop
Get new Sensor Data
Calculate
•Integral = e(k-1)+Ki*e(k)*T
•Deriv = Kd*(e(k)-e(k-1))/T
•Output = Kp*e(k)+Integral+Deriv

Figure 29: PID Control Algorithm

After the PID algorithm is done for each control input, the compensator output has to be sent to the motors. Unfortunately, the most microcontrollers, including the one used in this project, don't have a port for interfacing directly to a motor. Most microcontrollers do, however, have Pulse Width Modulation (PWM) which can be sent to an intermediate circuit which will translate the signal into a current that the DC motors can use. The intermediate circuit, called the electronic speed controller (ESC), sources a current to the motors that is proportional to the duty cycle of the PWM signal. The PWM principle is shown below in Figure 30.

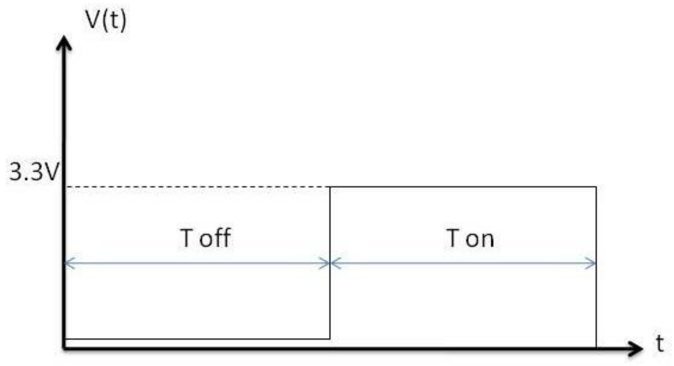


Figure 30: PWM Signal

The basic assumption is that the total time is always constant, and never changes. This means that the time on and time off the can vary, but the addition of the two times must always stay the same. This also means that the frequency is also held constant while the duty cycle changes. ESCs can take this signal and use some transistors and passive filters to turn the signal into a proportional current for a motor.

The AFRO 20A ESCs from Hobbyking.com will be the ESCs for the quadcopters in this project. These ESCs can source up to 20 amps, which is more way more than the motors being used in the project. Each motor needs its own ESC, so each quadcopter must have four ESCs. The PWM frequency of this particular ESC is 1KHz. This means that the PWM signal coming out of the flight computer must have the same PWM frequency.

During autonomous navigation, the control system will be taking input from itself essentially. This means that there will have to be pre-programmed flight maneuvers, as well as altitude feedback control. This will become taxing on the flight computer, as many calculations will have to be done in a very short amount of time. The guidance state machine, which will be explained in the next section, will handle which state the quadcopter is in so that the control system can be adjusted accordingly

In summary, there will be a feedback control loop for every axis and every parameter of attitude and movement. The attitude feedback controller should be able to perform well when the input is the RC controller or the autonomous navigation. The sensors on board the quadcopter will all have to filtered to get good data. The Recursive Least Squares filter will be used for all the high noise sensors. The compensator of the feedback controller will have to output PWM signals to the motors so that they can be controlled digitally.

5.5. Autonomous Navigation & Navigation Computer Software

In this section, the plan for implementing the control system will be shown along with the navigation computer software. For each quadcopter, there will be a navigation computer which will be the Raspberry Pi ™ embedded Linux platforms. The Raspberry Pi ™ runs Raspbian, an ARM optimized variant of the Debian Linux distribution. Since the Raspberry Pi ™ runs a Linux operating system, it is capable of running multiple processes simultaneously, which is ideal for computationally heavy tasks. These computers have more than enough memory, RAM, and processing power to do all the tasks required for the project. The Raspberry Pi's will be responsible for autonomous navigation, network communication, and imagery. The flight computer, aka the Tiva C ™, will also be assisting in the autonomous navigation portions of the project.

Since the Raspberry Pi ™ runs Linux as an operating system, it is capable of multitasking. The multithreading features, and the powerful processor on board the Raspberry Pi ™ will be very important to the outcome of the project. These features are especially important because the navigation computer will be handling the autonomous functions, imagery, and the digital network all at the same time. Each task will have a priority and will be running in parallel. The highest priority job is to navigate the quadcopter. Under that, the computer will have to keep taking images from the camera and storing them on the SD card. The lowest priority job will be to send data over the network using DTN2. All of these processes will make up the software of the navigation control system. Below, in Figure 31, is the navigation computer software overview diagram. This shows how the operating system fits into the software for the navigation computer. In fact, the applications developed for the navigation computer will run on top of the operating system, while the operating system handles all priorities, resources, and storage.

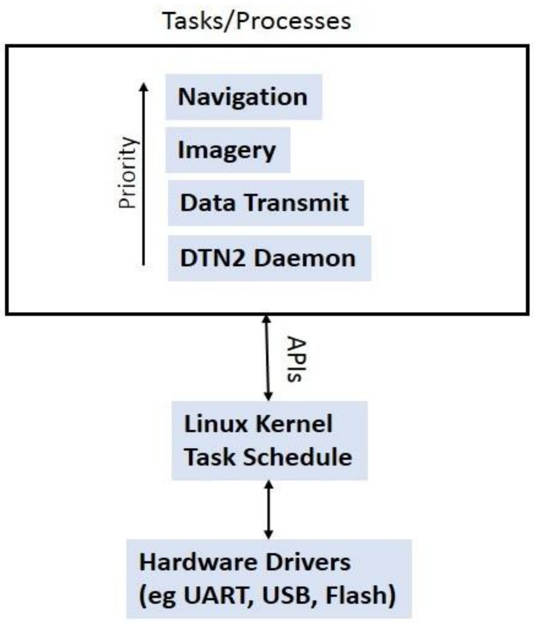


Figure 31: Navigation Computer Software Overview

The main reason behind using the Raspberry Pi ™ is NASA's requirement to use the DTN2 software. The DTN2 application has only been developed to work on Linux so far. This means that the DTN2 application would not run on the Tiva C ™ without major modifications. So, it was decided that the Raspberry Pi ™ would be used for this function. Another goal behind using the Raspberry Pi ™ is to make the Navigation computer interchangeable. When the Navigation computer receives information from the flight computer, the goal is to make the dynamics abstracted enough that the Raspberry Pi ™ and supporting hardware will be able to be ported to any UAV application. It should be able to make navigational decisions regardless of the vehicle it runs on.

DTN2 is the network layer of the communication network being used in the project. While running DTN2, users can use APIs or user interface commands to send messages, files, or even stream data. Essentially, DTN2 is a virtual router that can be implemented on any computer. This powerful application can prevent data from being lost from bad transmission or delays. More about the DTN2 application will be explained in the next section. The daemon for DTN2 will be running in the background while the computer handles the other navigation and imagery functions.

In Figure 32, the startup process for the Navigation computer is shown. It is critical to make sure that all the applications being used are opened and assigned priorities. The peripherals for the project also need to be opened as well, so that they can be used by the Navigation applications. After a quick system check, the software then moves into a system script that opens and handles the command parser, Navigation, imagery, and telemetry applications. The Navigation computer will have the task of communicating with the flight computer as well as the ground station.

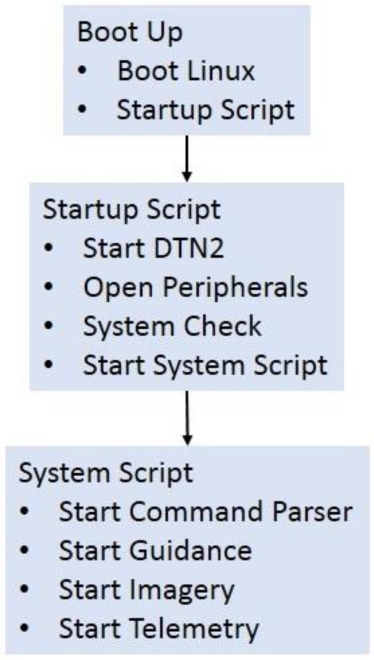


Figure 32: Navigation Computer Startup Process

Most of the software developed on the Navigation computer will be in the Linux native bash script. The reason behind doing it that way is to make interfacing with the peripherals and the DTN2 application much easier. The state machine, however, will have some parts that will be done using the C programming language. C is still far more useful than the bash script when it comes to the computation and logic for the Navigation system. The scripts developed will call on C programs to do computation or logic for certain tasks.

The Navigation computer will be communicating with the flight computer over a serial UART link at 115200 Baud. The Navigation and flight computer will be sending data back and forth to each other on things like state, battery life, GPS coordinates, and altitude. Both computers will have the UART connection interrupt driven, as the communication between the two needs to be able to be done at any time. The strings between the two will have a specific format to be determined at a later date. This will make parsing the data between the two relatively easy to do and should not require too much processing time.

The parser for the Navigation system communication from the ground station will be sent as commands for the system to be interpreted. These commands will be translated into instructions that the flight computer can understand in bite size pieces. Some of the commands will be single step, while others may be multiple steps. Commands such as hover, land, change altitude, return home, or fly to a location will be sent almost directly to the flight computer without much alteration. The single step commands are mostly setup as safety instructions that can change the state of the flight computer state machine in the case of an emergency. The image an area command will be done using multiple steps. This will be done by making a bread-crumb trail of coordinates to 'fly-to' that act as way points for the quadcopters. When flying both quadcopters, they may also communicate position and other information to each other while in flight so that each may optimize path dynamically. One of the commands that will not be translated for the flight computer is the take a picture commands. This command has no need to be routed through the flight computer, as the camera system is connected directly to the Raspberry Pi ™.

While the Navigation computer is parsing the user commands, the system will also be processing the data from the camera and the flight computer. This data will be stored on the SD card as well as transmitted in real-time during flight. The imagery, however, will likely not be sent during flight, and will likely be post processed after flight. The user on the ground station will have access to GPS location, altitude, battery life, velocity, and state. These will be processed in real-time during flight on the ground station and be packaged in a graphical user interface (GUI).

The flight path determination of the Navigation computer is perhaps the most difficult of the Navigation computer software. The quadcopter will have to halt and hover during flight in order to take stable pictures. So, based on the altitude of the quadcopter, and the horizon of the camera, a reasonable assumption can be made as to how far apart the stops must be for imaging. The determination of these stops will be shown in the next section. In this section, however, it is important to note that stops must be made during autonomous flight navigation to allow for imaging. With this in mind, the flight path of the quadcopter will be determined based on what mode it is in. In the fly to location mode, the quadcopter will simply be instructed to take a straight path to its destination. This path will not include any stops for imaging, nor will it provide collision detection. It is important to note that since the quadcopters in the KIRC project DO NOT have object avoidance software, operators MUST NOT draw a path that has objects in the way. This, however, can be avoided, as the cruising altitude can be set above most objects during flight. In the image an area mode, the Navigation computer will set several pit stops along the path where it will stop and take in image. In order to scan a whole area, the Navigation computer will begin to scan the area in a serpentine pattern. Figure 33 on the next page shows how the flight path might look for a single quadcopter in image area mode.

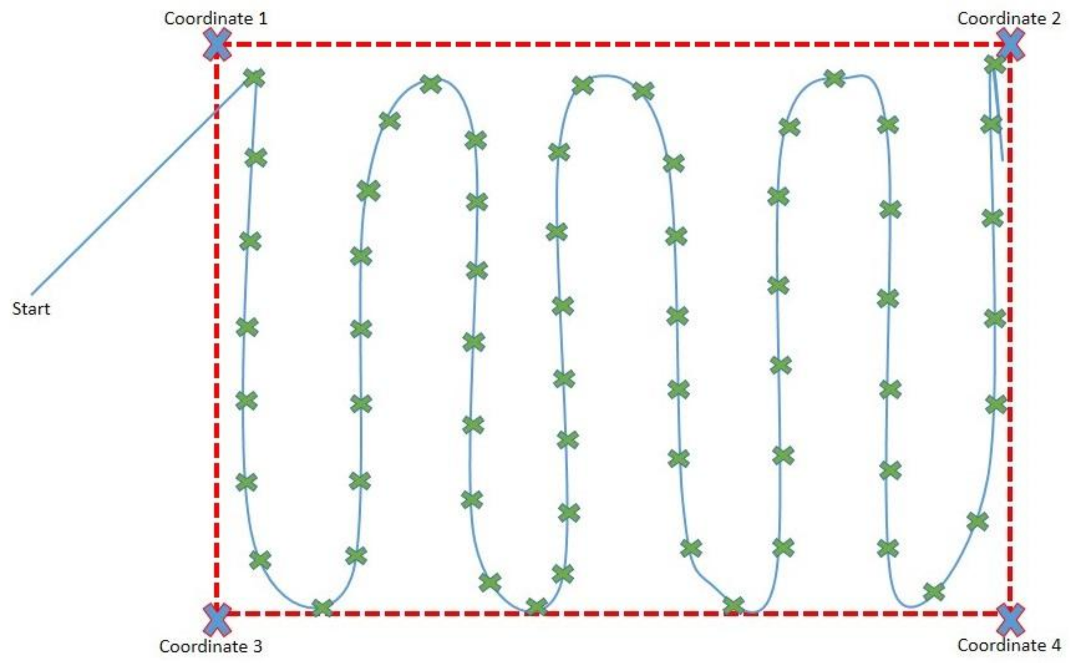


Figure 33: Example of Area Imaging Flight Path

When both quadcopters are being used, both will have the task of imaging an area. Each quadcopter will have half of the area to image. Cooperatively, they will have to decide how to take the area and divide it up. One of the quadcopters will have to be the master one that decides for both, as this will resolve any conflict where both quadcopters try to fly in the same area. During flight, the quadcopters will relay information to the ground about their respective states. Through the use of DTN, the quadcopters can also route through each other in the case where the ground station is out of range.

The Navigation computer essentially acts as an abstraction barrier for the flight computer that will be handling the brunt of the work. The flight computer, after the stability algorithms have been established, will be reprogrammed to include software support for reading GPS, sending telemetry strings to the Navigation computer, and processing a guidance state machine. The state machine, shown below in Figure 34, is the workhorse of the Navigation system. The navigation computer can command the flight computer to change states for the system at any time. The GPS strings will be sent from the flight computer directly to the Raspberry Pi ™. The Raspberry Pi ™ will be doing all the processing of the GPS strings.

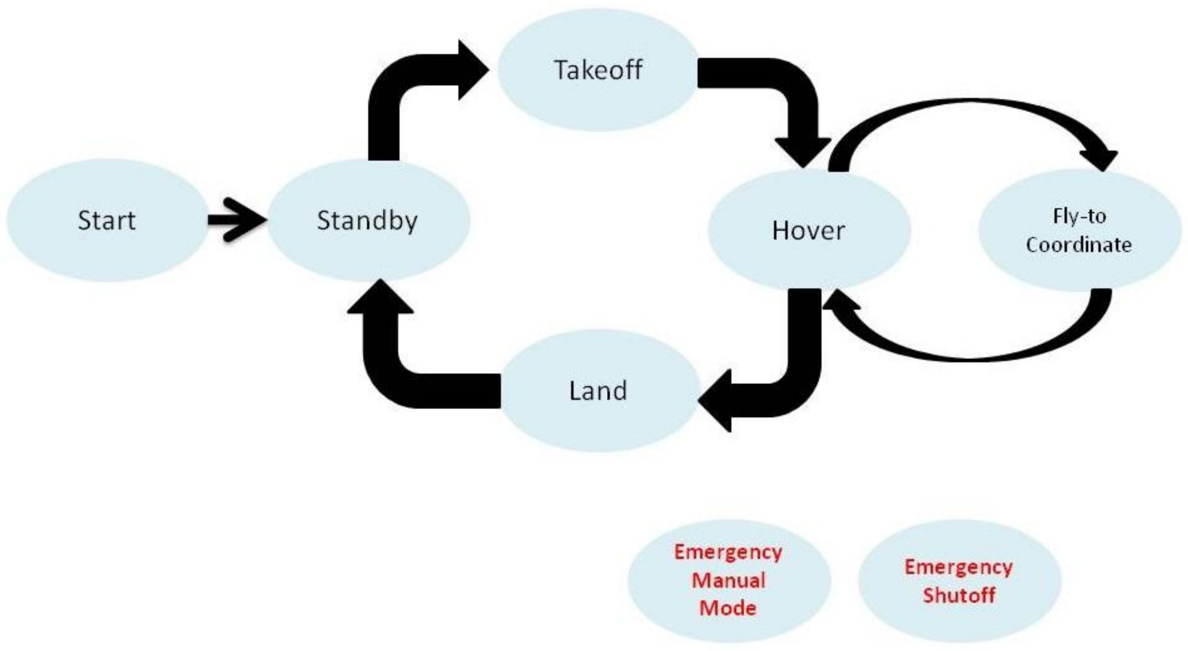


Figure 34: Navigation State Machine

When the flight computer first boots up, the quadcopter will be put into a standby mode while waiting for instructions from either a manual controller or the Navigation computer. In standby mode, the quadcopter rests on the ground with the motors throttled off. This mode acts as a default safety mode when the quadcopter is idling. After this mode, the quadcopter is set into takeoff mode, where the flight computer shifts its primary computing power towards getting to a set altitude while maintaining stability. The taking off and landing states are the most difficult states because the quadcopter is experiencing instability. Once the quadcopter has reached a desired altitude, the quadcopter will shift its state to hovering. In this state, the quadcopter solely tries to maintain altitude and attitude in order to stay level. The hover state is the default state during flight, as it is the most stable when off the ground. The next state can either be to fly-to a coordinate, or to land the quadcopter. When flying to coordinates, the default to return back to hover when the destination is reached. After the quadcopter has completed its mission, it will shift to a landing state, where it will use an on-board ultrasonic range-finder in order to land the quadcopter on the ground. The quadcopter will thereby enter the standby state and safe itself so that the battery can be removed and the flight data can be retrieved.

One of the most important features of the quadcopter guidance state machine is the emergency states that can be reached no matter what state it is in. The quadcopters can either have a complete shutoff or enter manual mode when in flight as an emergency precaution in the case where a flight isn't going as planned. The shutoff mode, planned only for absolute emergencies, will cut off signal to the motors and the quadcopter will drop to the ground from whatever height. The problem with this state is that it will likely destroy the quadcopter if dropped from considerable distances, which is why this mode is preferred not to be used. It is, however, necessary in the case that the quadcopter starts taking a turn towards the wrong direction and starts going out of control. The other state, manual mode, automatically shifts control of the quadcopter to a user with a RC controller. This mode is the preferred way of dealing with emergencies over completely cutting the motors, which is why there should always be a skilled pilot available to stop the quadcopter from crashing just in case.

In all the states of the quadcopter's guidance state machine, the flight computer will still be constantly running the PID loop and stabilizing the quadcopter. The only exceptions to this, of course, are the startup and standby states where the quadcopters won't be flying. The other states will be running the loops and constantly be processing data while achieving the tasks of each state.

The most important part of the guidance system, the ability to find a path from one location to another, is critical to whether or not the quadcopter will be able to fly autonomously. The quadcopter will have to be able to take in a GPS coordinate and compare it to its current location, find a vector from its current location to the new one, and fly itself to that point (illustrated in Figure 35). The difference of distance between the two points will create a vector that will help guide the quadcopter to the proper location.

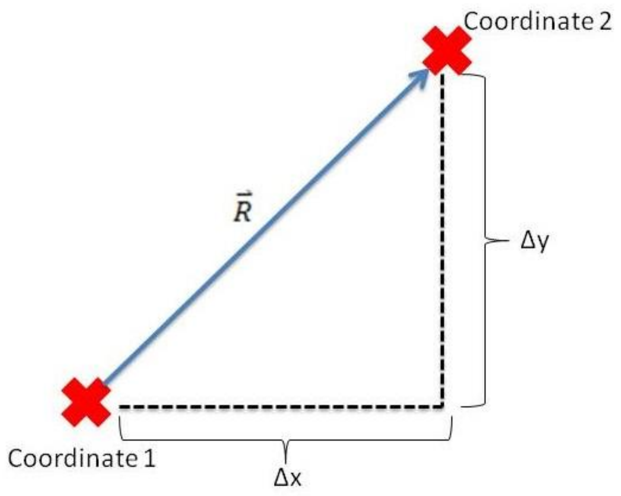


Figure 35: GPS Coordinate to Navigation Vector Transformation

When using GPS, the relative position between two objects can be found by subtracting the distances between the two points. If the GPS coordinates are already translated into UTM (Universal Transverse Mercator) coordinates, the two coordinates can be directly subtracted to find a vector between the two points in meters. This method, however, only works for relatively short distances, as it does not take into effect the roundness of the earth into consideration. Assuming short distances, it is possible to use this 'flat earth' model in order to simplify calculations considerably.

Each GPS coordinate comes with two parts, the north and the east coordinates. These two coordinates can each be stored as a floating point variable in RAM. To find the distance north that the quadcopter must travel in order to arrive at the destination, it must subtract the north coordinate of the current position from the north coordinate of the desired position as shown in the equation below.

$$\Delta north\ (in\ meters) = \Delta y = P_{2\ north} - P_{1\ north}$$

Similarly, the distance east can be found by subtracting the east coordinate of the current position from the east coordinate of the desired position as shown in the equation below.

$$\Delta east\ (in\ meters) = \Delta x = P_{2\ east} - P_{1\ east}$$

Using these two distances, a vector can be formed that will guide the quadcopter to the proper location as shown in the equations below.

$$\vec{\mathrm{R}}\ (in\ meters) = (\Delta x)\hat{x} + (\Delta y)\hat{y} = R\hat{u}$$

$$R = \sqrt{(\Delta x)^2 + (\Delta y)^2}$$

$$\hat{u} = \frac{(\Delta x)\hat{x} + (\Delta y)\hat{y}}{\sqrt{(\Delta x)^2 + (\Delta y)^2}}$$

With this, the vector can be used to find a heading and distance to travel which will be the basis of the control system that will be used to navigate the quadcopters between points. The distance between the two points is the error that will be minimized, as the quadcopter moves towards the destination, this is shown in Figure 36. The quadcopters will use the magnetometer to find the heading and make sure that the quadcopter is moving in the proper direction. The speed at which the quadcopter moves between the two points will be capped and that value will be determined at a later time. The control action to get to each point determined by the attitude stability control portion discussed a few sections earlier. While flying between two points, the quadcopter will also be using feedback to maintain altitude as well in order to prevent the quadcopter from falling.

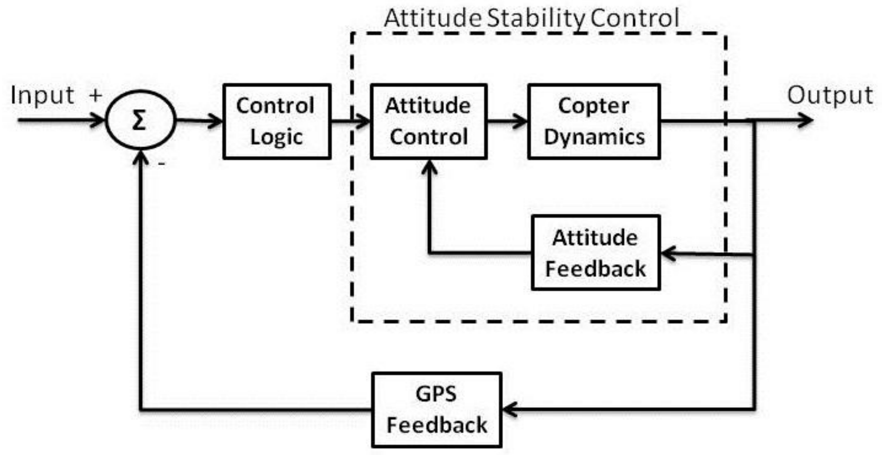


Figure 36: Navigation Feedback Diagram

Most GPS units have a low bandwidth, and a very low resolution for most commercial application. With feedback coming in at only at a few hertz, and the data being off by up to several meters, tolerances must be made in order to make it work. Since the current position of an object as determined by a commercial GPS unit can be off by several meters, it is assumed that the final destination is a range around a specific point rather than the specific point itself. This means that there is a bubble range where the quadcopter can say it has reached its destination. In order to counter the low bandwidth of the GPS, the maximum cruising velocity will be capped. Capping the velocity will help prevent overshoot and inaccurate headings. As the quadcopter approaches the target, the quadcopter will also begin to slow down. This will be accomplished by turning the quadcopter in the opposite direction of travel in order to counter the forward momentum. This will slow down the quadcopter.

Figure 37 below shows the control logic for the point-to-point Navigation system. This simplified logic diagram shows the continual looping to find the error vector and fly to that point. The diagram does not depict, however, the complex speeding up and slowing down portions of the logic.

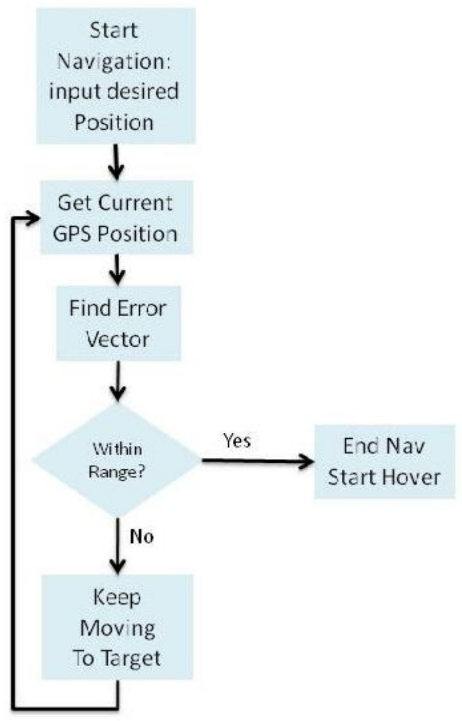


Figure 37: Control Logic for Navigation System

The guidance and navigation portions of the project can be very tricky. The plan is to use both the Tiva C ™ and the Raspberry Pi ™ to do these tasks respectively. The Tiva C ™ will have the guidance state machine that helps guide the quadcopter to specific points while the Raspberry Pi ™ will have the task of navigating the quadcopter through all the coordinate ‘way points’ and collect the data in the process.

5.6. Imagery & Network Architecture

In this section, the plan for implementing the imagery system and the network architecture are given. Although each of these topics is slightly different, they are both going to be implemented on the same system. The imagery system and the DTN2 application will both be running on the guidance computer (the Raspberry Pi™). Both will be low priority applications that will run when the processor has time for them. The DTN portion of this section will include the overview of how DTN2 works, how it’s planned to be implemented, and how it is called in a program.

One of the major bonuses of the Raspberry Pi ™ is the never-ending availability of extra hardware add-ons. The Raspberry Pi ™ has a camera system available to it that plugs straight into one of its ports and works relatively well. The five mega-pixel camera module was specifically designed to work with the Raspberry Pi ™. The Raspberry Pi ™ can even address it using the operating system file system of attached devices. When the camera is mounted, images can be retrieved and stored on the SD card of the computer. Overall, the goal was to keep the imagery system as simple and cost effective as possible while still achieving the major goals of the KIRC project.

When setup properly, the camera will be able to addressed in the /dev directory on the Raspbian operating system. The plan is to use this feature to take images remotely with the Raspberry Pi ™ during flight. The cameras will be facing down and attached to the bottom of each quadcopter in flight. The mount will have to be done carefully as not to pose the risk of damaging the camera in flight. Figure 38 shows how the mount might look while in flight. The mount will be put under the battery which is underneath the frame of the quadcopter.

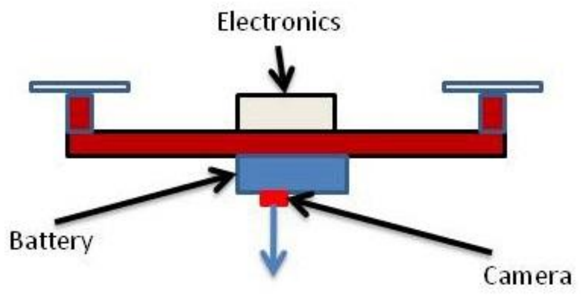


Figure 38: Quadcopter Imagery Mount Plan

During flight, the guidance computer software will retrieve the images from the camera, store them on the hard drive, and possible send a sample set to the ground. Images will be saved in JPEG form so that they may be easily be viewed by any computer. Image stitching will be done on the ground computer, by post processing the images. Each of the images will be GPS coordinate stamped for processing later.

The ground station and the two quadcopters, KIRC and SPoK, will be connected on a digital wireless communication network. The plan is to use IEEE standard 802.11g wireless in an ad-hoc configuration to form a mesh network. The DTN2 application will virtually act as the network layer of the system. Using this application will give the computers the ability to dynamically route links and store-forward bundles of data between any nodes. DTN, or delay tolerant networking, is made to excel in situations where links are intermittent or opportunistic. The DTN2 protocol only works on the Linux operating system for now, so this was the major reason behind using Raspberry Pis ™ for the project.

DTN2 is a networking protocol that was developed recently and has been steadily growing popularity in the space industry. It is an open source software available from dtnrg.com and is written in C++, and is still under heavy development. DTN2 is essentially an overlay virtual networking protocol that operates on the network layer as a virtual router. The protocol holds support for convergence layer adapters such as TCP and UDP, allowing it to be used with higher level applications. It also comes with a daemon for use on the command prompt and several APIs written in C++ for use in custom programs. For this project, the plan is to use the daemon in the command prompt, because interfacing with the APIs might prove to be too difficult for the team to manage. The overview of the DTN2 software from an academic paper is shown in Figure 38.

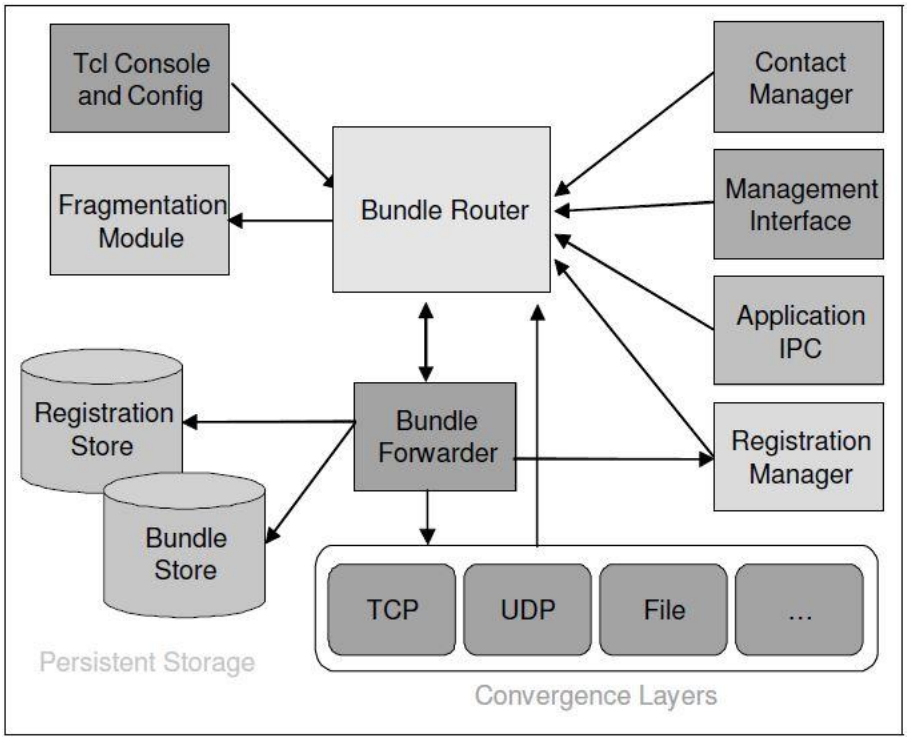


Figure 39: DTN2 overview diagram

DTN2 handles data in ‘bundles’, where each bundle can vary in size from a few bytes to even gigabytes. Each bundle to be sent is stored and forwarded to a destination, where the destination will acknowledge receiving the bundle. This exchange reaction is one of the features of DTN that makes it so durable in rough networking environments. When the sending node does not receive an acknowledgement from the receiving node, it keeps resending the bundles until an acknowledgment is received or the operation times out. The time-out period for DTN2 is set for 1 hour, but it is adjustable depending on the situation. For this reason also, TCP must be used as the convergence layer, as it has support for the send/ack structure of DTN. While DTN2 can be used with UDP, its feature set is limited when using this protocol. Figure 40 below shows a block diagram of the send/ack logic.

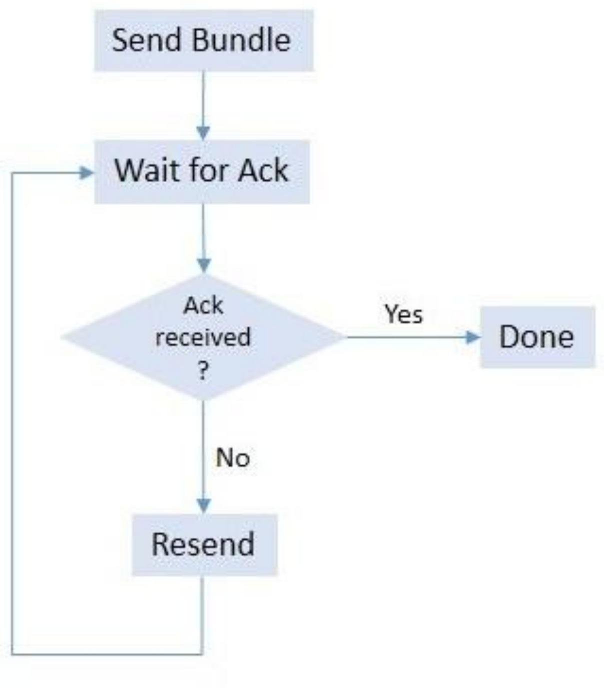


Figure 40: DTN Sending and Receiving Exchange Diagram

Another feature of DTN2 is its ability to dynamically route based on routing table and opportunistic links. When configuring DTN2, a routing table must be established that has all the IP addresses, domain names, and convergence layer information of every node in the network. After all this information is given, DTN2 must be configured to work in a dynamic routing mode. Once setup properly, DTN2 has the ability to route through each node dynamically to create a path to the destination when a direct route is not there. This is especially ideal for situations when links between two nodes are cutoff or have intermittent connections.

While DTN2 is being used in this project, it isn't the only DTN software available. JPL has also developed a low overhead version of DTN called ION, which is short for Interplanetary Overlay Network. This name reflects NASA's future vision for DTN as a deep space telecommunications networking protocol. ION is written almost entirely in C, and much like DTN2 is still very much in the development phase. The reason behind using DTN2 instead of ION is because DTN2 is more developed and has more features that are useful to this project.

Before using DTN2, it must be installed on the host computer running Linux. For the Raspberry Pis™, the process of installing DTN2 can take quite a long time, but it can be shortened by cross compiling the software on a faster machine. The software is available, as mentioned, from dtnrg.com. DTN2 comes with two separate packages that must be installed in order for the program to work. One is the DTN2 application itself, and the other is a support application called Oasys. Oasys contains the support library for DTN2 that has the utility functions that the program needs. While installing DTN2, there are several package dependencies, so it is recommended to install GCC 3.3 or newer, TCL

8.5, BerkeleyDB 4.8, and Xerces 2.6+ before installation. The install may also require some other packages depending on the machine it is running on. After configuring and compiling Oasys and DTN2, with Oasys being the first, the program is now ready to run.

Before running DTN2, there is a configuration file that must be edited. In the configuration file, the user must list his IP address, domain name, and connection information. The user must also do this for any computer on the network that will be using DTN2 also. This process creates a routing table for DTN2 that can be used to route bundles to the proper destinations. If any changes are made to the configuration file after DTN2 startup, the application must be restarted in order to incorporate the new changes.

DTN2 comes with a command prompt daemon that can be used on the Linux console. The daemon can also be run in the background while other programs can have time to run. The daemon can be opened by typing in 'dtnd' in to the Linux console with the appropriate arguments. The full list of appropriate arguments can be listed by typing in 'dntd –help'. Before running the daemon, new bundle database must be initialized. This can be done by using the command 'dntd --init-db'. It is recommended to clear and remove any old databases before initializing a new one though. After initializing the database, the daemon can now be run. For this project, the daemon must be running in the background, so the DTN2 application is started up by typing in the command 'dtnd –d –o dtn.log' into the console. The dtn.log portion creates a log file for the daemon as it is running as a way of checking if there was an error. This process must be done anytime the DTN2 application is restarted.

Once DTN2 is running, there are several commands and applications that will be available to the user on the command prompt. One of the most basic commands is the 'dtnping' command. This command requires a destination argument, and when used it pings the destination repeatedly while waiting for responses. This command is very useful during debugging and helps check for working links within a network. Another command that is available is the 'dtnsend' command. This command sends messages, files, or timestamps to another node within a network. The only downside to this command is that the other computer must be using the 'dtnrecv' command before receiving any bundles. The 'dtnrecv' command is a blocking function that holds up the command prompt until the user decides to quit. There are two application commands, however, that are more suited to this project than the normal sending and receiving commands. The commands 'dtncp' and 'dtncpd' are command console functions for sending and receiving respectively. 'dtncpd' is the listening function that can be set to run in the background while other tasks are performed. After receiving a file, it is put in a local directory. 'dtncp' is the complimentary function to the receiving one. It sends files of any type to any of the nodes on the DTN2 network. These two functions will be used extensively in this project as the standard communication functions.

From start to finish, there are several steps that must be done to configure DTN before it is ready to use. Figure 41 shows the summary of the initialization process of the DTN network for each computer. This process will be streamlined by packaging it into a script for the computer to run that will be called on startup. After running the script, users should be able to start executing other programs such as the guidance system, while the daemon will handle all the communications in the background.

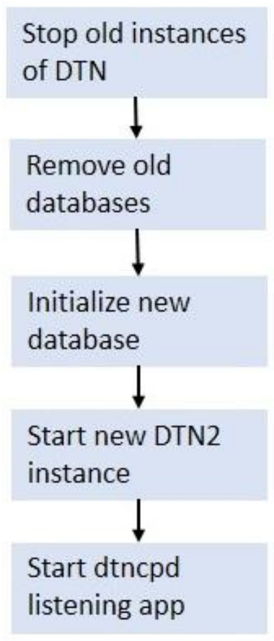


Figure 41: DTN2 Startup Script Overview

The on the guidance computers for the quadcopters as well as the ground station laptop, there will be scripts that will handle all the functions of the DTN software as well as the other tasks that need to be performed. The plan is to have a script on each of the guidance computers that periodically check if a file has been received, or if one needs to be sent and perform the appropriate action thereafter. The ground station will have a similar structure, but it will be actively reading the information from both quadcopters and displaying the information live in a terminal as well as a GUI.

In summary, the imagery systems as well as the DTN2 network architecture overview is given. Each of these systems will be incorporated into the software of the guidance computers as well as the ground station. The plan is to use the network as a means to relay telemetry and control the quadcopter remotely over a command prompt. The imagery is going to be post processed by a third party application and is explained in a later section.

5.7. Electric Design & Schematic

In the initial schematic design for this project it was decided to begin with getting from the power supply to the test equipment including the Tiva C ™ launchpad, IMU sensor stick, altimeter, GPS Sensor, and Raspberry Pi model B. The easiest way to maintain drop voltage to an amount for powering microcircuits is through the use of voltage regulators. The voltage regulator circuit is shown in Figure 42 below.

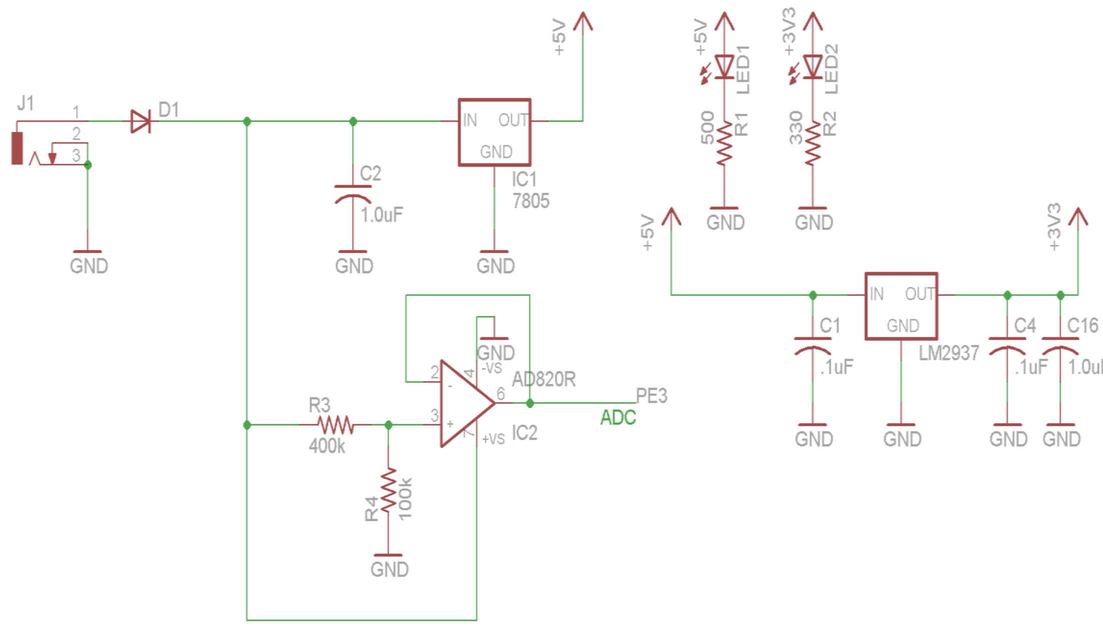


Figure 42: Voltage Regulators and Battery Monitoring

The 15 volt power comes in through the J1 power plug, this comes directly from the battery. In order to protect the battery a safety diode is implemented. This is followed by a split into two circuits, a voltage regulator to drop the voltage down to 5 volts and a voltage divider into a unity gain operational amplifier.

The desire is to have the voltage going in to the non-inverting terminal to be less than 3.3 volts. The operational amplifier is implemented in order to cut out the current and send this low current, voltage signal to an available analog to digital converter on the microcontroller and measure the slope in output seen by the battery as the system is in flight. This monitoring is done in order to force a landing of the quad-copter in the event of dangerously low power.

On the other line coming from the safety diode goes into a 5 volt regulator. Just before this voltage regulator is a capacitor placed to maintain circuit in the case of a voltage drop and filter low frequency noise. The 5 volts then flows into a +5V net that goes to connectors for powering other system components, as well as another voltage regulation circuit and verification light emitting diode, LED. In prototyping LEDs are important for signifying what stage of operation the system is in, which is why one is implemented in order to verify proper powering of the system.

This 5 volt regulator circuit is followed by a 3.3 volt regulator circuit. Due to the fact that most of the components on the board are powered by 3.3 volts, if this portion of the system doesn't work properly then none of the system will be functioning properly. For this reason the 3.3 volt regulator circuit is also given a power on LED. In order to keep the other circuitry and this one from being altered by low to medium frequency noise, capacitors are implemented on both sides of the 3.3 volt regulator.

As schematic design moves forward the best way of preparing for implementation is the use of any resources available. A big resource was for some of this design is the suggested circuit design developed for the sensor stick used in testing as well as those design suggestion in the datasheet of many of the components. In addition it makes the system design more reasonable if design starts with simplicity and moves forward to more complex network design from there. The IMU sensor circuitry is shown below in Figure 43. The circuits themselves were derived from some similar product from Sparkfun.

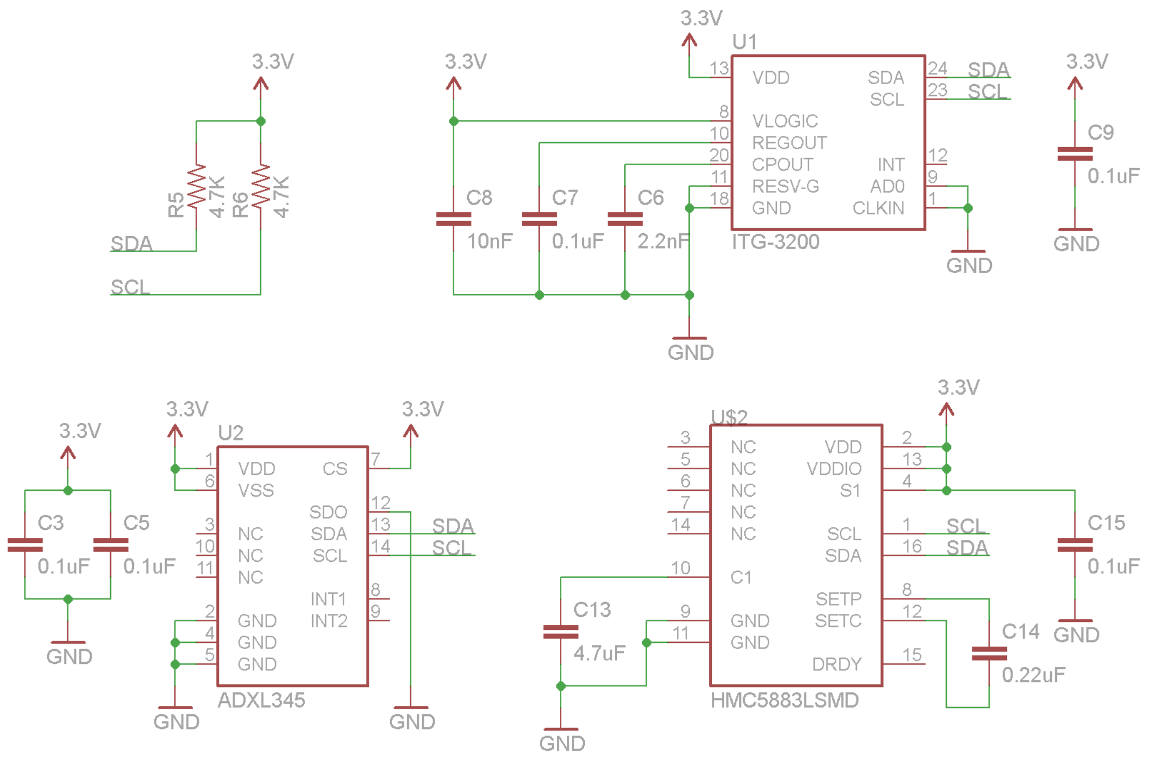


Figure 43: IMU Sensors

The next design to be implemented is the configuration of the senor stick, short the altimeter. This circuit will be communicating through the use of I2C, see section 6.3, and with this communication configuration the data and clock lines leading to the microcontroller need pull-up resistors for proper performance. In addition, when dealing with sensors the output and power need to be stable. For this reason the VDD and any other powering signals are filtered for medium and high frequency noise, in addition to those mentioned in the voltage regulator circuit previously. In this system all of the sensors have different functions and for that reason have different pin-out signals and configurations. In this system all signals are digital and for this reason all grounds are designated by the same node. For many of these sensors there is a built in standby mode that is not going to be taken advantage of in this project due to the fact that data will be continuously pulled from all of them and the desire is for all sensors to be constantly taking in measurements.

The accelerometer, ADXL345 as seen in lower left of Figure 42, will have both VSS and VDD powered by the 3.3 volt signal. This configuration powers the device and puts it into standby mode, waiting only for the command to enter measurement mode. By connecting the CS signal to the VDD the sensor is put into I2C mode as desired, but then the seven bit I2C address must be determined. This is accomplished by connecting SDO to ground giving this sensor the address of 0x53 hex preventing it from conflicting with the other sensors.

The magnetometer, HMC5883L as seen in the lower right of Figure 42 has a much different configuration than that of the accelerometer. It has the occurrence of 5 no contact pins that are most likely used in programming the sensor or in manufacturer calibration as well as a data ready or interrupt pin that will not be connected do to the fact interrupts are not necessary for our systems performance. As suggested in the datasheet for this sensor, when using a single power supply for the sensor the VDD,VDDIO, and S1 signal should be tied together, connected to the 3.3 volt power signal, and connected through a 0.1 micro farad to ground. The C1 value is set to 4.7 micro farads and connection between SETP and SETC is set to 0.22 micro farads in order to handle peak current pulses and low voltage drops seen between the control and the set/reset strap driver in the sensor. SETP being the set/reset strap positive signal to noise ratio capacitor, SETC the driver signal to noise ratio connector and C1 being the reservoir capacitor

The gyroscope, ITG3200 as seen in the upper right of Figure 43, also has a suggested configuration by the datasheet. The 3.3 volt power supply of the system will power the system by connecting to VDD as well as VLOGIC for simplicity and due to the fact that an external clock isn’t desired, and interrupts are unnecessary CLKIN is tied to ground and INT is no contact. Pins 2-7, 14-17, 19, 21, and 22 are no contact as specified within the datasheet as well as RESV-G signal is a reserved signal needed to be connected to ground. As suggest the regulator filter capacitor is set to 0.1 micro farads, the charge pump capacitor to 0.0022 micro farads, and a noise filter capacitance on the VLOGIC signal set to .01 micro farads.

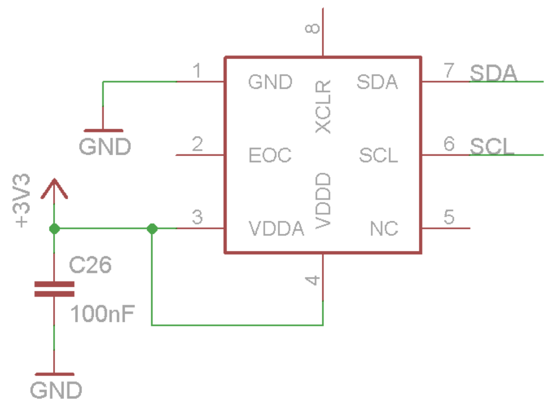


Figure 44: Altimeter Circuit

The last sensor to be implemented is an altimeter, BMP085 in Figure 44, which is not associated with the sensor stick, but will be used in this system in order to maintain height readings for imaging definition. This connector has the same I2C communication as the other three sensors and therefore will have the same pull up resistors for along the SCL and SDA lines. This sensor will be powered by the 3.3 voltage line and connected to both VDD and VDDA with the addition of a 0.1 micro farad capacitor to ground. As with the other sensors there is an NC contact that will be no contact along with XCLR which is a master clear signal, no contact allowing conversation, and EOC, which is an end of conversion signal or interrupt, that is going to be no contact as done with the other sensors.

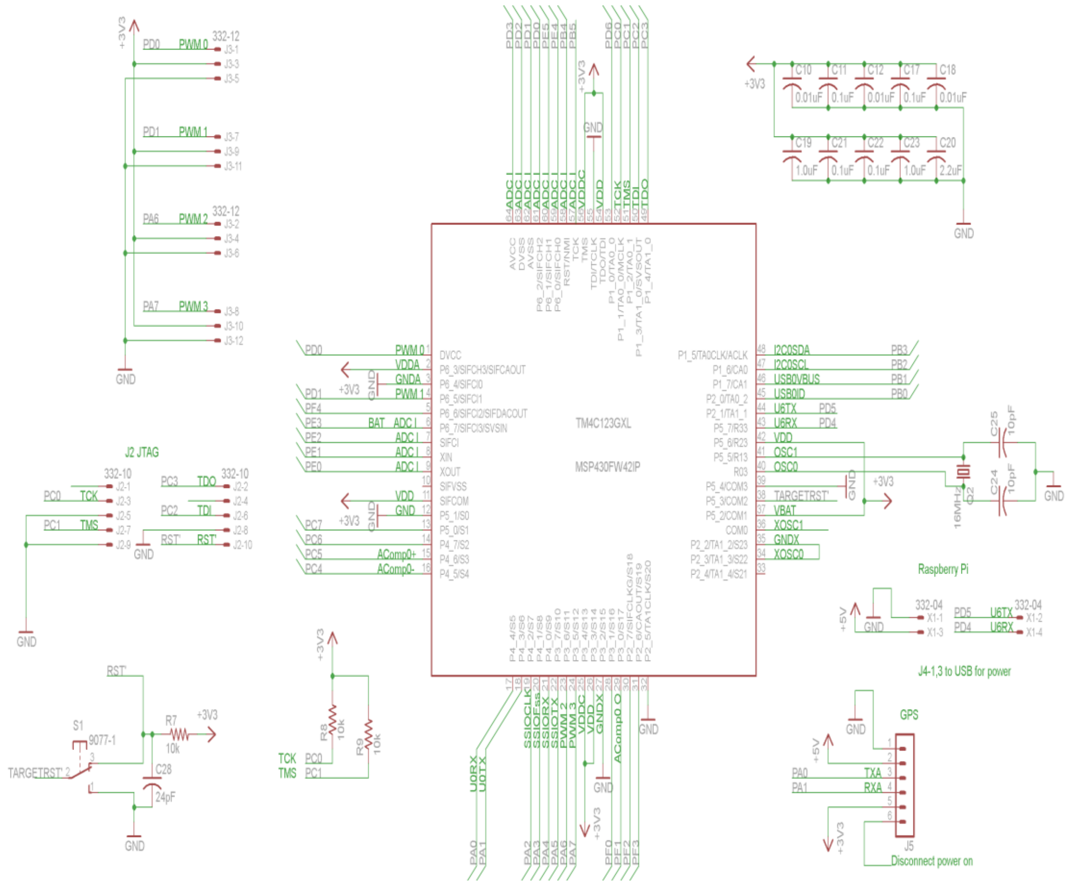


Figure 45: Microcontroller Interconnect

The real difficulty in circuit design came when doing the schematic for the interconnection of the TM4C123GXL microcontroller seen in the center of Figure 45. The first challenge when working with this component is the fact that it is so new that the version of schematic design software available for use doesn't have it available in the library. It just so happened that the very popular MSP430FW42IP is available with the same 64-pin dimensions, capable of being used for setting of the footprint. For this reason, in the image the gray internal names of the signals should be ignored when looking at the purpose of each signal pin.

When doing the design for this circuit there is a lot of resources that can be taken advantage of. The most useful of these resources is the Tiva C ™ launchpad that will be implemented during testing of the system. When moving from testing with the implementation of a manufactured board to a designed circuit without the excess capability given on the launchpad it is wise to keep the system as close as possible so as not to cause back tracking in development. However, this program will not be implementing to microcontrollers as done on the launchpad, simply because the system should only have enough processing power to perform its mission causing understanding of the datasheet.

Even with the single microcontroller there are many pins that will be no contact. Due to the fact that the entire system is digital, there will only be one analog-to-digital converter, ADC, used for battery monitoring. For this reason pins 7-9 and 57-64 will not be used. PE3 is the net chosen to watch battery activity. A second factor when dealing with an entirely digital system is that there is a single ground plane allowing for all grounds to be connected together, this includes GNDA (Pin2), GND (Pin12, 27, 39, 55), and GNDX (Pin35). This allows for all of the powering of the system to be powered by a single voltage source. The TM4C123GH6PM is a 3.3 volt powered microcircuit that is connected at VDDA (Pin 2), VDD (Pin11, 26, 42, 56), VDDC (Pin25), and VBAT (Pin37).

Along with most of the ADCs there are more signals and abilities that are not going to be taken advantage. Due to the fact that the microcontroller will be at work for the entire flight time it is unnecessary to prepare for this quad-copter to go into hibernation mode. For this reason the hibernation function involved with Pin32-37 will be voided. HIB' will be no contact because there is no need to show it is in hibernation when it will never happen. Similarly the hibernation oscillation module input and output (XOSC0 and XOSC1 respectively), which would usually contain a lower frequency crystal than the system uses under normal operation conditions, will be no contact at XOSC1 and XOSC0 will be tied to the hibernation oscillator ground, GNDX. VBAT signal would normally be a lower powered input due to the lightening of processing while hibernating will now be attached to 3.3 volts as the other power signals are and GNDX will be connected to the same ground plane as all other GNDs in the system. WAKE' is an active low signal that signifies for the microcontroller to come out of hibernation mode. In order to keep the system from ever falling into hibernation mode the WAKE' signal will be connected to the ground plane.

The microcontroller doesn't have an internal timing mechanism. For this reason an external oscillator crystal must be implemented to maintain synchronous timing. The suggested oscillation frequency of the crystal is the 16 megahertz. For this reason a 20 picofarad crystal oscillator is connected to OSC0, the main oscillator crystal input and OSC1, the main oscillator crystal output P40-41 respectively. This is seen in center right of Figure 44.

When powering the microcircuit it is important to filter as much noise as possible on the power source line. For this reason there is a chain of capacitors, seen in Figure 46 that range from 1.0 micro farad to 0.01 micro farads. The larger of the capacitances will filter the low frequency noise and protect the source from a slow drop in voltage. The lowest capacitances will assist in maintaining a consistent power while experiencing high frequency additive white noise.

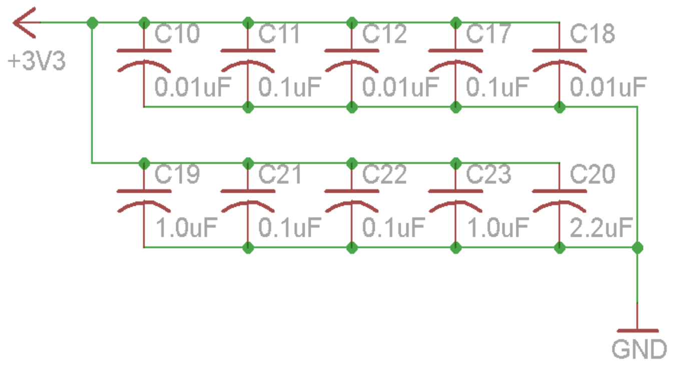


Figure 46: Noise Filtering

Another small circuit deemed necessary for the system is a system can reset the system by two ways, both manually and via JTAG programmer. For the manual switch circuit seen in Figure 47 TARGETRST' is connected to Pin38 of the microcontroller and resets it when the signal goes low. The manual push button is spring loaded in order to consistently keep the signal charged with a capacitor to keep the noise from driving the signal low randomly. The 3.3 volt is connected to a 10 kΩ resistor. This is done in order to keep constant feedback on the JTAG RST' pin.

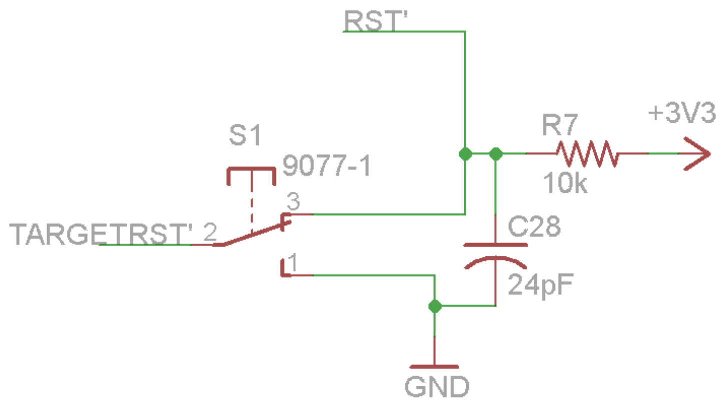


Figure 47: Reset System

The JTAG connector uses ten pins to program the microcontroller, seen on the center left of Figure 44. The JTAG test clock (TCK) at J2-3 and the JTAG test mode select (TMS) at J2-7 inputs signals are each connected to the 3.3V net though pull-up resistors, and connect to PC0 and PC1 (Pin52, 51) respectively on the microcontroller. The reference grounds for this connector are located at J2-5,9,8. The test data in signal at J2-6 and the test data out at J2-2 are shorted to PC2 and PC3 (Pin50, 49) respectively on the microcontroller.

The microcontroller receives all the data sent by the GPS through UART as seen in figure 48. This connector is not only used for receiving and transmitting data, but also for powering the sensor itself. The reference is linked to the same ground plan in the circuit at J5-1. The GPS's main power is sent as a 5 volt signal via J5-2 and a 3.3 volt signal through J5-5 as a backup power source. The transmit signal TXA coming from the GPS

J5-3 is connected to the PA0 net and connects to the U0RX receiving port Pin17 on the microcontroller. Similarly the microcontroller transmits from Pin 18 the U0TX along the PA1 net to the RXA pin J5-4. J5-6 is a power control signal that can be used to turn off the GPS if desired. Due to the fact that all sensors will be sending and transmitting data throughout the entire flight time this pin is either driven high or disconnected in order to power on the sensor.

The other UART connection is going to connect to the Raspberry Pi this will be a much simpler connection due to the signals needed to be sent are so few, seen in Figure 48. It will need to be powered by this PWB via the output of the 5 volt regulator output through X1-3 and the ground through X1-1. The microcontroller is very useful in that it has an opportunity to transmit through a universal serial bus or a UART signal transmission. In this case it is better to send simple receive and transmit signals via PD5 and PD4 net, U6TX and U6RX respectively.

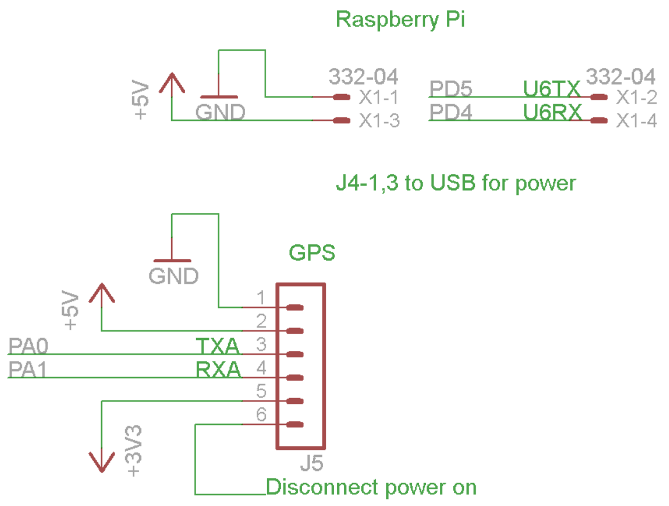


Figure 48: UART Connections

The last signals that are sent from the microcontroller off of the PCB are pulse width modulated signals that are to be sent to the ESCs. Due to the fact the ESCs are pulling power through to the motors directly from the battery, the power signals are only sent as a reference for this connection. There are four ESCs being implemented as there are four sets of three pin output pins. One signal is designated for 3.3 volts, a second for a ground reference and the third signal is the modulated signal, see top left of Figure 44. The microcontroller has two motion control module with the ability to send 7 PWM signals each. It was decided that two PWM signals will be sent from one module, M0PWM0 and M0PWM1, and two from the second module, M1PWM2 and M1PWM3. These signals correlate to nets PD0, PD1, PA6, and PA7 or pins 1, 4, 23, and 24 on the microcontroller, respectively.

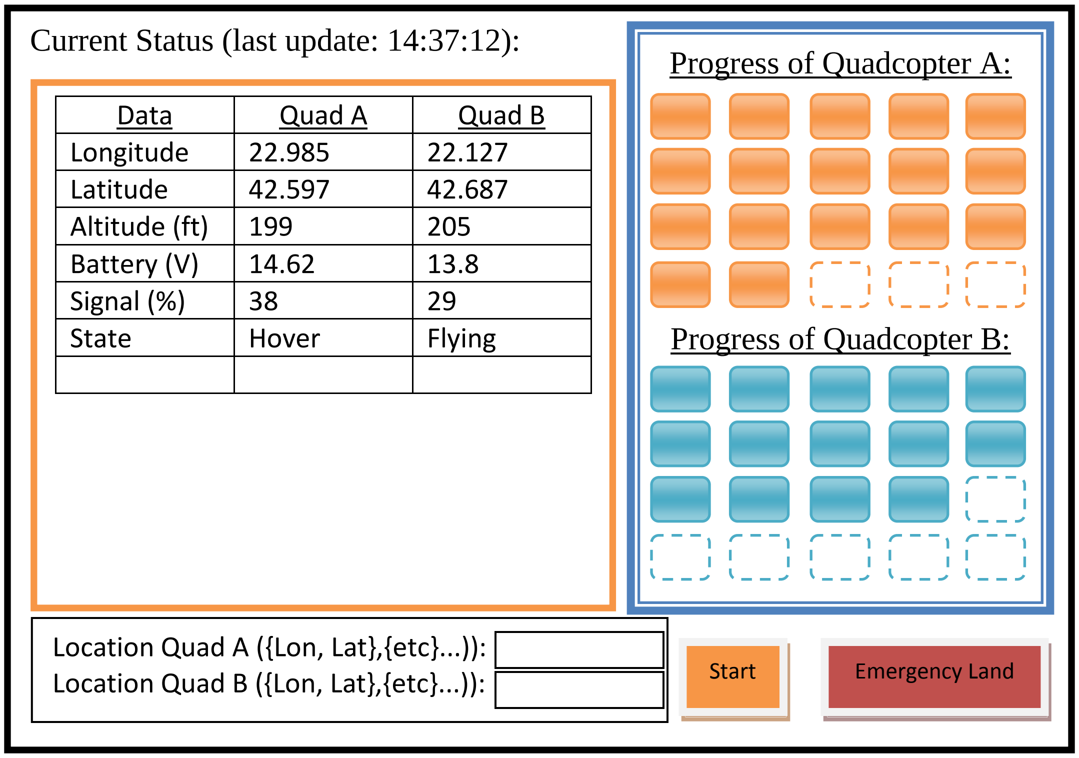
Due to this being the entire electrical design section it is necessary to explain a portion of the cable design that will need to be done for implementation. The first issue is seen as the voltage comes from the battery jacks and goes into a splitter. The splitter goes from an two to eight, however the ESCs call for 2 each, so in order to have two for the PCB power an additional splice will be added in with the proper head for plugging into the PCB. There will need to be a basic pin to pin ribbon cable for the GPS. The Raspberry Pi will void its warranty if powered by the GPIO, so the ribbon cable will be altered to have a micro USB for powering the board and a two port for transmission through GPIO.

At this stage in design the schematic will take into account all of the circuitry necessary for a proper working system while stripping unnecessary components and circuitry. After testing and development is completed, the implementation of this altered circuit will pose little issue. This, however, has been the mindset of all engineers as projects have moved into testing and development. For this reason this schematic will be open for change until design is finalized and completed.

5.8. Ground Station Overview

The main purpose of this project is to have a map generated from an aerial view. The ground station is the final step to creating this map. As KIRC and SPOC fly within the specified bounds of the designated area to observe, they will both take pictures facing downward in reference to the quadcopter and store images on the SD card in the Raspberry Pi ™. The ground station will be a laptop with a 802.11 card to communicate to the Raspberry Pi ™ with both quadcopters on the ground and during flight. This is also where the quadcopter will lift off and land. During the flight, each quadcopter will send information back to the ground station about its status. The laptop will have a custom interface used to track everything that the quadcopters are doing. Once the mission is complete, the SD cards will be removed from each Raspberry Pi ™ and inserted into the laptop which will then process the picture data into a single image. This will be done using open source software.

Missions always start at the ground station with the laptop. Once the quadcopters are booted up they will connect via Wi-Fi with DTN running on top to the laptop. Next they will fly to the specified coordinates that the user typed in the laptop. During the mission each quadcopter will send its status back to the laptop every five seconds. This will include its current GPS coordinate, altitude, battery level, state the quadcopter is in, and the mission progress. An example of what this interface will look like is shown in Figure 49. The state that the quadcopter is in is the state machine used for the Tiva C ™ microcontroller. This is also the interface that will be used to start the mission. The user enters the four coordinates that each quadcopter has to map. This interface allows the team to track how long each portion of the mission will take. This is useful for estimating what size areas at certain distances can be mapped. The mission progress is based on how much of the area has been captured with the camera. Each copter only considers its own progress so the total mission progress would be the combination of the two on the laptop. The program on the laptop will have an abort function that will tell the quadcopters to immediately abort the mission and return home. This is necessary in case the user needs to leave due to an emergency or if it suddenly starts to rain.

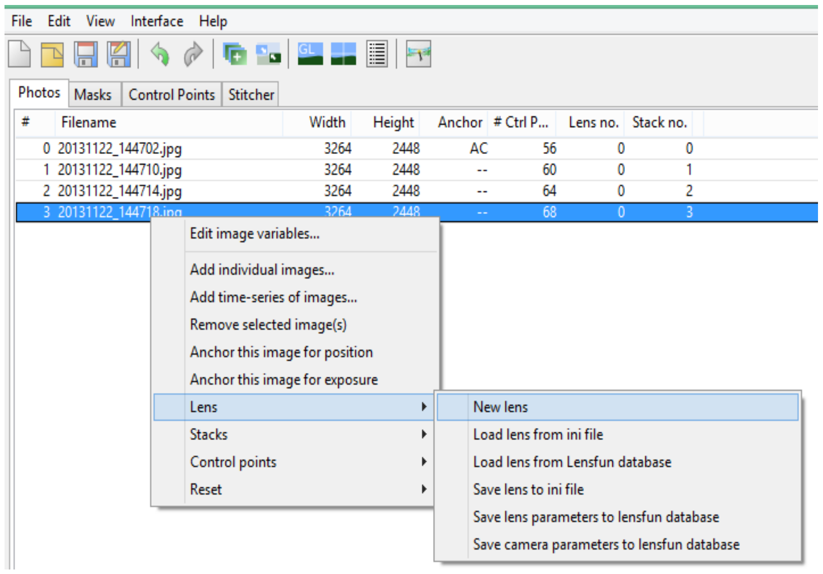

Current Status (last update: 14:37:12):

Data	Quad A	Quad B
Longitude	22.985	22.127
Latitude	42.597	42.687
Altitude (ft)	199	205
Battery (V)	14.62	13.8
Signal (%)	38	29
State	Hover	Flying

Progress of Quadcopter A:
Progress of Quadcopter B:
Location Quad A ({Lon, Lat},{etc}...)):
Location Quad B ({Lon, Lat},{etc}...)):
Start
Emergency Land

Figure 49: Laptop Interface Example

When the quadcopters have landed after successfully completing their mission, the images will be removed from the SD card and placed onto the laptop's hard drive. The pictures will be stitched together using a free software called Hugin. This will create a map with the pictures in a mosaic form since the quadcopters are translating in reference to the ground. Most image stitching software is made for panoramic photography so it is going to require each image to be set as its own lens in order for Hugin to properly stitch the images in a 2-dimentional fashion. It requires the user to enter advance or expert mode. From there the user has to right click each image and select new lens. This is shown in Figure 50. This can be time consuming since each picture as to individually be clicked on and set as a new lens. The projection will be set to rectilinear and positioning will be done in mosaic mode. Each image has to be optimized separately since it was taken independently from the other images. The final image will be stitched together and saved as a JPEG image file type.



#	Filename	Width	Height	Anchor	# Ctrl P...	Lens no.	Stack no.
0	20131122_144702.jpg	3264	2448	AC	56	0	0
1	20131122_144710.jpg	3264	2448	--	60	0	1
2	20131122_144714.jpg	3264	2448	--	64	0	2
3	20131122_144718.jpg	3264	2448	--	68	0	3

Figure 50: New Lens Setting

Figure 51 shows six individual pictures taken of a fence in a backyard. Each picture was taken at a different location on the ground. The angles are also different which will not intentionally be the case with the quadcopter but it was done in this example to show that Hugin can adjust the pictures accordingly, regardless of the different angles. Also this object is much closer than the objects that will be taken during the mission. The overlap of these pictures is noticeable if you look for distinctive objects. Hugin looks for distinctive patterns in the picture that match.

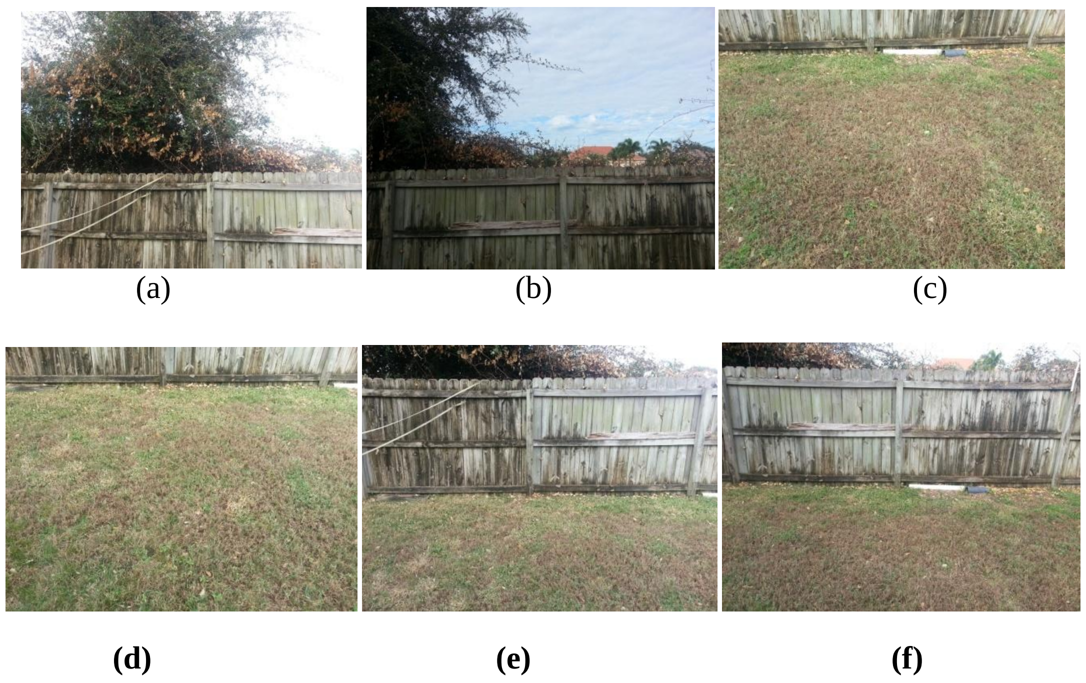


Figure 51: Individual Pictures (Stitching Software Example)

It is essential to have the pictures overlap. Figure 51 shows what image stitching would ideally be on the left. This would be the most efficient way to cover the largest area possible. Theoretically this would be possible if the pictures were all taken in order and there is no under or overlap. Since this is not possible with the quadcopter at 100 plus feet, the images will have to be overlapped. The stitching software needs to make control points which represent the points on picture one that correspond to the same points on picture two. The overlapping of the pictures can vary significantly based on focus, angle, and lighting. Since the program uses an algorithm to try and find matching points, this means the more overlap that exists between two pictures, the more points can be found. For this project the team is going to have around a 50% overlap on every picture in every direction like shown on the right side of Figure 52. This will reduce the amount of theoretical area that the quadcopters can map, but the quality will be much better than pictures with gaps and misalignments. Since the software is just using overlapping as the only method to determine where each picture belongs, it is not necessary to keep track of the order that the pictures placed or taken. The downside of this is the larger the area is, the more pictures will need to be taken. This is something that not only affects the total flight time but the stitching time as well. Each picture that is added will exponentially increase the amount of time for the program to place each picture. This is because Hugin has to compare each picture with every other picture to look for control points.

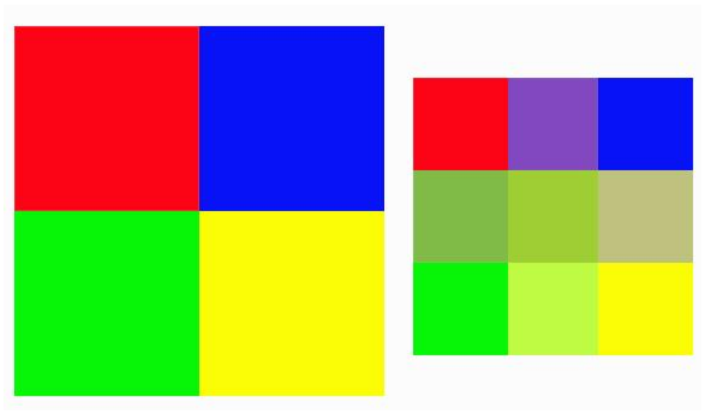
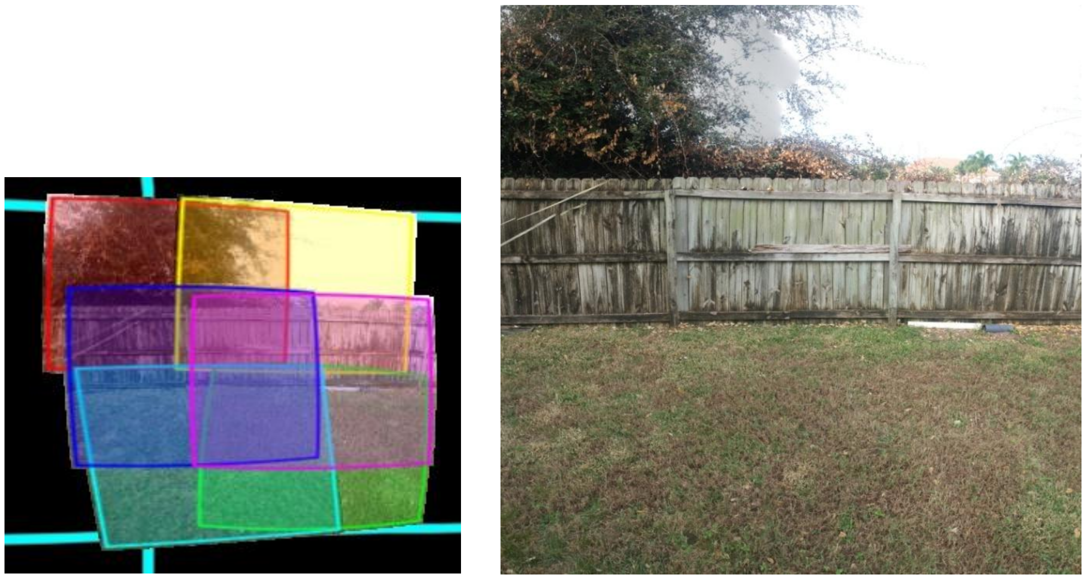


Figure 52: Example of Capturing an Area

Figure 53(a) is an image preview in Hugin that shows all of the individual pictures together with the overlaps. Each picture is approximately overlapping the neighboring pictures by 50% just like the example in Figure 53(b) shows the final render of all the pictures stitched together. It is not entirely perfect since there are so many overlaps, the program has to pick which picture takes priority in each overlap that matches the surrounding overlaps the best. Overall it produces a decent uniform picture.



(a) Placement (b) Final

Figure 53: Pictures Stitched Together

The ground station is where the mission starts and ends. It is a convenient way to start a mission since no wires are needed to hook up. The intuitive interface allows the team to track the entire progress of the mission while it is happening. When the mission is over, the laptop puts all of the images together on the spot so everything needed to do a mission (or even multiple missions) is present. The user also has the option to stitch all of the images together later and just get the photographs while out on the field. The controller can optionally be located at the ground station during testing. It only comes with one receiver so only one quadcopter can be overridden and manually controlled during the mission. This is not essential to the functionality of the ground station or project.

6. Prototype Construction & Coding

The following sections will proceed to explain part acquisition, PCB design and assembly, part integration, and a brief coding overview. Part acquisition includes any distributors and resources that will be used in order to gain access to needed passive and active components. In part integration the off board interfacing is discussed, for PCB communication see section 5.7. PCB design and assembly will describe the layout, size of board and optional manufacturers to fabricate the board. Lastly, the code overview describes the operating systems and compilers to be used as well as a description of the software to be tested.

6.1. Parts Acquisition

When keeping a project to a low budget, not only is part selection an important factor, the distributor becomes a resource to compare. This project is sponsored completely by the National Aeronautics and Space Administration, NASA, and for that reason this organization will be supplying all funds for part acquisition with few exceptions. An advantage of being involved with such a prestigious organization is the ability to have many passive and active components on hand for use, as well as the ability for board development, if necessary, is a capability handled internally by NASA.

The first part chosen is the microcontroller, a TM4C123GH6PM, produced by Texas Instruments. Texas Instruments Incorporated, TI, is a useful resource for many aspects of the design phase of prototyping a project. They provide resources including Launchpad and expansion options, a large technical support team, and large level of production.

One of the benefits that will be taken advantage of is the opportunity of having a Launchpad available for immediate implementation of testing and development. This resource is valuable in that it provides an excess of ability that will not be needed in the final prototype, but is of great use in the time between design and PCB fabrication. This ability to continue forward in development instead of having down time while components are being placed and soldered will allow for a faster schedule with more time in the event of unforeseen issues arising.

Secondly, TI is known for maintaining a very capable engineering staff filling the technical support department. In the event of misunderstanding or confusion in the use of a product TI develops, this department is available for extended office hours with a more in depth knowledge of TI's products than can be simply read from a user manual. In addition to the assistance in the implementation of products, these engineers are also an available resource for discussing troubleshooting options that may have been overlooked.

Due to the fact that Texas Instruments is such a largely used company they consistently produce a large amount of each popular product. In the case of the TM4C123GH6PM microcontroller, the large quantity of supply allows for lower cost to the consumer for both the microcontroller and the Tiva C Launchpad. This allows not only for low cost reproduction of circuit design, but also a hasty part replacement in the event of one being damaged.

In the case of many of the mechanical components being implemented in this design, Hobby King is one of the best resources available. Hobby King has a specialized inventory of parts solely for the development of multi-rotor copters. This offers the opportunity to compare prospective parts quickly and efficiently in terms of price, and specification. In addition there are also so additional suggestions for easily compatible parts that they also provide. The frame, motors, electronic speed controllers, propellers, lithium polymer battery, and controller will all be procured from Hobby King.

One of the many benefits of Hobby King is the interface of the website. The ease of navigation and the quick representation of important data with the option of additional information quickly narrow the list of capable parts. With the benefit of the initial search supplying not only an image, name, and slight description, it also renders a price. This allows for a budget, which is of importance to this design, to be maintained without a waste of time searching into overpriced piece parts.

Due to the fact that this design includes a lot of opportunity for mechanical design components it is a great resource to have the opportunity to use a supply such as Hobby King that makes these specifications easy and straightforward for understanding. Upon selection of a part the buyer is immediately met with multiple images on different angles of the part, a brief description and an organized, summary table of part specifications including, but not limited to, dimensions, and weight. When choosing the motors for this design, there was no necessity of undergoing a study of dynamics in order to decide on propeller sizes; Hobby King provides current pull associated with different propeller lengths. In addition, due to the fact that this is equipment specifically used for development of aeronautical systems, the specifications on each part are isolated to only the data needed for implementation in such a system.

Sparkfun is another useful resource in the acquisition of electronic components and testing cards. Both the four sensors to be used (gyroscope, altimeter, digital compass, and accelerometer) and the testing sensor stick were found on Sparkfun. When it comes to design Sparkfun has many beneficial additions that simply ordering from some large part distributor will not provide.

When inquiring into different parts, Sparkfun has the additional benefit of having a brief bit of background and critical choosing point. When looking into different altimeters a description of the interfacing abilities were given. In order to avoid wasting time on a component that only uses USB communication instead of the desired I2C, making me read through pages of datasheet while verifying other specification needs, Sparkfun immediately stated that this altimeter interfaces with I2C as well as USB.

The sensor stick that will be implemented for initial testing and board development is to be acquired from Sparkfun. In addition to providing the datasheets for all of the sensors that are implemented on the breakout board, Sparkfun also provides a full schematic of the card. As the project moves forward with the design and development of the PCB, having a basis for schematic design for most of the sensors will fast track the process.

If supplying the schematic weren't enough, Sparkfun also supplies an Eagle file, Eagle being popular PCB design software, of the breakout board itself. By supplying this file to the customer sparkfun allows for an ease integration of parts. If such a file were not provided and the parts used are not provided in the Eagle library, i.e. the sensor chips, the designer is forced to go through a painstaking process of adding the part to the library.

Another useful distributor of components such as the Launchpad is Amazon.com. Not only does this website have a large quantity of components readily available, they also have a quick shipping time. It can take as little as overnight to 3-5 business days. This 3-5 business days shipping time is free of shipping and handling charges for students. This is extremely useful as this project moves forward into testing and implementation.

During testing and troubleshooting it is common that a couple of small components or piece parts could be damaged here and there. With this in mind, ordering and shipping small and cheap parts can become expensive relative to the budget when the cost of shipping is roughly the same cost as the component.

Another benefit in the use of Amazon.com is their return policy. When using components that are sensitive to electrostatic discharge packaging and handling are damaging when not done properly. Before purchasing a product on Amazon the return policy of the seller is readily available, saving the buyer from misrepresentation by a seller.

At the University of Central Florida, senior design projects are done completely by the students, but the university has readily available piece parts on hand for students to do testing with. They provide many piece parts in order to prevent the students from needing to purchase small quantities while readily available in bulk to the university. This will be very useful when beginning implementation and testing of the design because the students will be able to use lab equipment for development before final design of the PCB is confirmed and fabricated. Mr. David Douglas is readily available for students to present their needs that he has in his stockroom at all times.

6.2. PCB Design and Assembly

The printed circuit board, PCB, is the last stage in the development of design. This is because the circuit must be proven to work properly before packaged for final prototype. In order to avoid wasting funding and ruining the budget the circuit needs to be tested in as many small parts as possible. For this project beginning with using the launchpad, GPS, IMU sensor stick, Raspberry Pi ™, Li-Po battery, and small regulation circuit. Stepping away from the manufactured ICs, save the Raspberry Pi ™, and toward the completed design on a breadboard as the last step before sending the design for fabrication. In Figure 54 on the next page, is the PCB design up until Senior Design 1. Note, however, that this design is likely to change over time.

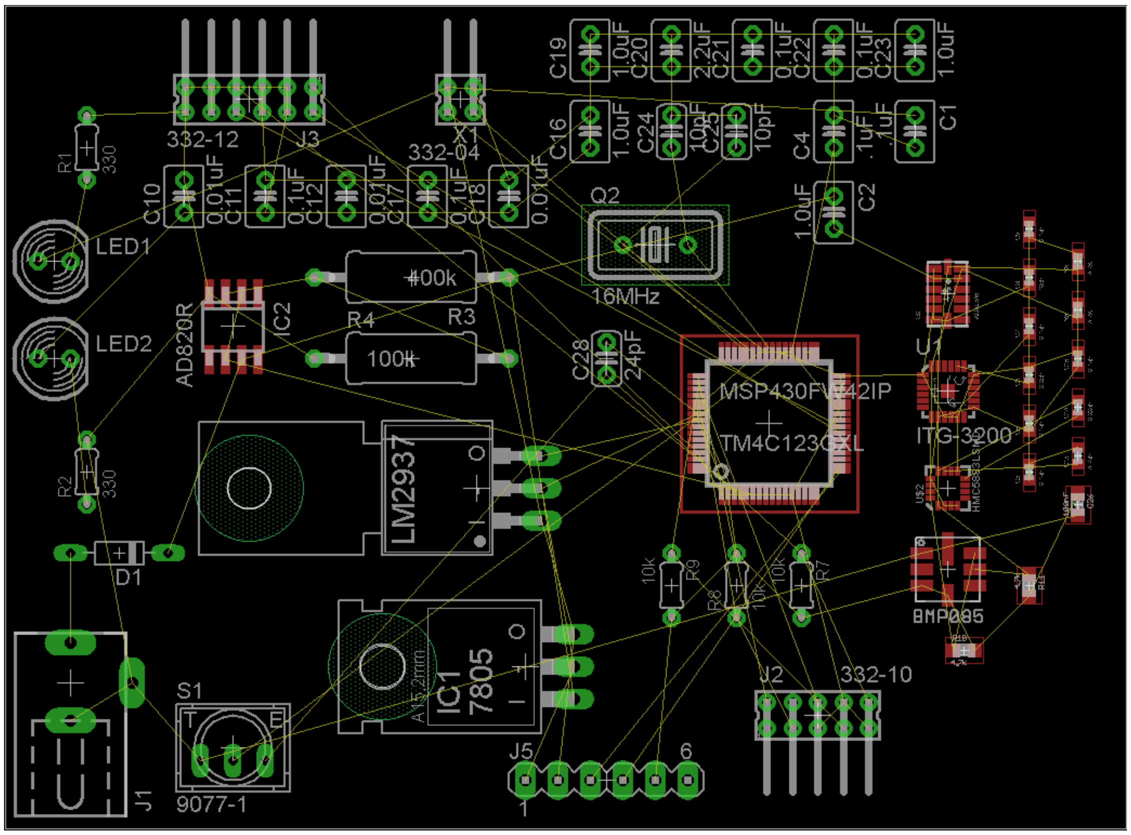


Figure 54: Printed Circuit Board

On this PCB design, as with most, all parts are organized in sections and oriented for an optimal use of space. The organization strategy is based on purpose, location of implementation, and amount of power. This organizational strategy assists in keeping the trace lines more logical and will require less layers for the PCB assembly.

First the in the organization was the location of the nets connected as compared to the component use. There is no sense in running fifteen volts around circuitry that isn't powered by it in order to cram as many components as possible onto the smallest board possible. For this reason as the components move farther from the power connector, in the bottom left as seen in Table 7, the components are powered by lower voltages and manage less current. For this reason the voltage regulator circuit and the battery monitoring components, operational amplifier and higher power resistors, take the lower left corner board and near as possible to their terminating locations. The 5 volt regulator that is only implemented in UART connections is organized close near to the connectors the signals will be transmitted from. That same logic places the 3.3 volt regulator close to center of the board due to the fact most components are powered by its output

The net oriented organization was taken into consideration when placing the passive components. Due to the size of these components being so small and so popular it can be seen how the crossing of so many of these lines will cause unnecessary need for additional layers. For this reason the IMU sensors were set with those capacitors used for noise compensation and pull-up resistors used. The IMU sensors were also kept together due to the fact that by being implemented on the same sensor stick these sensors do not affect the performance of one another and they will all be using the same communication lines. The TM4C123GXL microcontroller is placed as close as possible to the center of the board with the orientation chosen to minimize cross lines and those traveling under the controller to reach the necessary pin, i.e. the clock oscillator, IMU sensors and connectors.

As for the purpose coordinating to the location, this mostly refers to connectors, with a few sets of other components. When implementing this PCB in the final, full system the connectors should be as close as possible to the location of termination to prevent voltage drop. It should also be clear that signals crossing over the board for any reason can have effects on sensor data as well as desired outcomes in circuitry. The PWM signals are going out to the ESCs therefore need to be on the outskirts of the board and the UART connectors for the Raspberry Pi, hence location of J2 and X1. However the GPS is on the opposite side of the board as the Pi, hence the location of J5.

When it comes to the fabrication of the board itself there are a couple of options presented. The most logical option is to have NASA develop the PCB internally. The second option is to outsource the board to a low price fabrication company known as Advanced Circuits. Both of these are viable options that will have the ability to produce a product up to expectations that will work as designed, in addition to the fact that both have great incentives to issue them this business.

Advanced Circuits is and has been a popular PCB fabrication center for UCF senior design students for many years. This is mostly due to the fact that they implement a student program that offers a lowered price to students. In many of these senior design projects a large amount of layers in the PCB are not necessary, including the design for the PCB to be implemented on this quad-copter and for this reason Advanced Circuits offers a four layered board for 66$ and a two layered board for $33. With space on the racks of the frame being small in size, a four layered board will be used.

Another incentive for the use of Advanced Circuits for fabrication is the haste with which they complete the board. For a 2 layer board, as basic as possible, there is a maximum of 9 day turnaround time and for the 4 layer it is slightly greater than two weeks, depending on the quantity of work placed before them. In a situation where there will be test equipment available for continuation of testing while the PCB is being made and populated, the lead time isn’t a great issue, however any unnecessary time that isn’t taken advantage of to verify performance of the PCB is a hindrance on the program.

Doing a project for a company like NASA allows for many benefits because of the strength of resources they can provide, both personnel and material. Due to the fact that NASA is mostly sponsoring this program the price isn’t really a large issue. When taking into account the cost of fabrication for this board will be materials and the amount of time that it takes to fabricate, hindering the development of other internal projects. For this reason timing is an obstacle.

In order to avoid hindering other program development the PCB must be ready for fabrication early and on a moment’s notice. If it so happens that the schedule allows for a break long enough for fabrication it would be critical to get the board developed before anything can alter it. This presents another issue, however. If this plan of action is implemented and the PCB design has yet to be full tested with quality mission success it could turn out to be a waste of time to make the PCB. With this in mind there is also the option that the developed design will have no issues, or at most a couple vias can be implemented as a fix.

In addition to the waiting on availability of the fabrication center, there is an additional lead time on the build. Due to the fact that NASA commonly does very complicated boards compared to those involved in this program the average turnaround is five to six weeks. This is not considering that with this project being a student design NASA will be more accommodating to deadlines.

After comparison of the two options for PCB fabrication centers, as the project moves forward Advanced Circuits will be chosen manufacturer. This is in the case that NASA is unavailable to develop the PCB when prototyping has reached that stage. If NASA has the availability to produce the board it will be unlikely that they will fund Advanced Circuits doing the work.

6.3. Parts Integration

Through the use of this microcontroller gives an opportunity to control as many components as we could possibly need for system control and performance. In any design the part integration is a big decision based on communication speeds and more often capability. The Tiva TM4C123G-H6PM microcontroller is capable of implementing Universal Asynchronous Receiver/Transmitter (UART), Synchronous Serial Communication (SSI), Inter-Integrated Circuit (I2C), and Joint Test Action Group (JTAG) communication with the pins specified in Table 7 below.

GPIO Pin	Default State	GPIOAFSEL Bit	GPIOPCTL PMCx Bit Field
PA[1:0]	UART0	0	0x1
PA[5:2]	SSI0	0	0x1
PB[3:2]	I2C0	0	0x1
PC[3:0]	JTAG/SWD	1	0x3

Table 7: Microcontroller Communication GPIO

JTAG, a common name for the IEEE 1149.1 standard test access port and boundary-scan architecture, will be the source of communication for programming the microcontroller. JTAG is a fairly common choice for programming and communicating with specialized integrated circuits due to the fact that it was original implemented in boundary scanning for testing and troubleshooting PCBs. With the Tiva microcontroller and the Launchpad that is being used for initial testing already JTAG integrated, it will assist in an ease of integration as the project moves forward.

The GPS will be communicating with the Tiva microcontroller through the implementation of UART serial transmission. Due to the speed and desire for precision in the data received from the GPS unit this is a useful tool. With the GPS being a unit that is to be purchased from a third party and not running off the same oscillators as the microcontroller the ability to configure transmission speed will be useful. Also, the use of a real time operating system (RTOS), which is based on a series of priorities, will make the ability to only allow data passage via a ready to receive bit prevent the loss of useful or skewed data.

UART is will also be implemented in communication between the Tiva Microcontroller and the will used in the communication with the Raspberry Pi model B. UART is a form of serial communication to be implemented by way of the RS-232 standard commonly implemented by computers and integrated circuits to peripherals devices. The UART is a very useful tool due to the fact that data formats and transmission speeds are configurable. The Raspberry Pi ™ contains a powerful processor and will be in control of designating to the microcontroller the location of the quad-copter's next way point via a UART serial port.

The Raspberry Pi ™ will additionally be communicating with the ground station that is used for the alignment and meshing of the grid images sent from the Quad-copter. Due to the fact that the mapping is not of great importance while in flight and is only represented as a whole, the Raspberry Pi ™ will be communicating with the assistance of a WiFi card. While in range all images will be sent back to the ground station for image processing, and if it should happen that the aerial vehicles falls out of this range the Raspberry Pi ™ will store the image internally until a strong enough connection that can ensure no data loss occurs. The implementation of Wi-Fi is the best option for this means of communication, not only because it is easily implemented on the Raspberry Pi ™, but also because it has the ability to quickly send large files, allow for higher quality imaging.

The Raspberry Pi ™ will be taking these images using a specialized 5 megapixel camera that is made for implementation with the Raspberry Pi ™. Communication and data transfer between the camera PCB and Raspberry Pi ™ is implemented by way of a 15 way ribbon cable. This communication is done by way of camera serial interface, CSI-3, which allows for the higher resolution imaging.

I2C communication, commonly referred to as I-squared-C, will be critical in the communication with the all sensors, save the GPS. I2C has commonly been implemented in communication when managing low speed peripherals. The fact that this interface is based on a master-slave coordination between an open I/O address on a microcontroller to a peripheral device, and add in that the Tiva C ™ microcontroller is has designated I/O that is unused elsewhere makes I2C interfacing the most promising means of communication as this project moves forward.

The ESCs control the motor speeds are controlled by pulse width modulated signals (PWM). Another great benefit of using the Tiva TM4C123G-H6PM microcontroller is the built in PWM ability. This makes the ESCs easily controlled by surface mounted short form pins that are straight memory mapped I/O from the microcontroller.

The chosen motors are three phase brush motors that are controlled by the ESCs. The ESCs produce an analog signal in each of the three lines leading to the motor. The motors take this signal and based on how out of phase these signals are, produce a certain amount of thrust.

The power system is fairly simple for this project due to the fact that it is digital, most components are low voltage and controlled from the PCB. The power will be distributed from the battery to the ESCs, followed by the motors, using simple banana jacks. Though this is simple and basic way to distribute power it is effective. The PCB will also be taking 15 volts by way of banana jacks and will regulate the signals then send out the proper power to off board components using the designated connectors on the board (for further information please see section 5.7).

Due to the fact that this project will not only contain PCB design and interfacing, but will also have a testing configuration that must be address. As far as the interfacing off of the PCB, nothing will change. However, the PCB will initially be a more powerful Tiva™ C Series TM4C123G Launchpad and a sensor stick containing the altimeter, gyroscope, digital compass, and accelerometer that will later be implemented in the PCB.

The sensor stick is a useful tool for initial implementation due to the fact that it’s operational ability has been verified. Secondly the sensor stick comes with a present communication set up of I2C. With this possible to work on in the initial testing phase reaffirms the decision made to move forward with the I2C interfacing.

The launchpad described is a Texas Instruments product that contains two microcontrollers, on board in circuit debug interface and USB for computer interfacing. On top of this it also provides preloaded RGB application forty on board I/O, switch selectable powering, 2 user switches, a reset switch, and test LEDs that will be useful in troubleshooting, but most will not be implemented on the PCB.

6.4. Coding Overview

As this project is a big undertaking from a programming stand point it needs to be something that the team is comfortable with. Due to the fact that the microcontroller is produced by Texas Instruments, Code Composer Studio version 5 is the best integrated development environment for this project. Code Composer Studios is not only the most comfortable compiler for the team, due to the fact that much of the programming done by the team has been through CCS, but it also provides debugging support and JTAG based development. The microcontroller has a JTAG programmable ARM processor in a reduced instruction set microprocessor powerful enough to perform all needs of the system. An added bonus is that Code composer includes a real time operating system, or RTOS for short.

A real time operating system is used to implement real time requests and return data output from the system. This in most cases is able to operate without lag or buffering delaying the data flow. Through the implementation of this program the testing will be more thorough. Due to the complexity of the sensor data that is used to maintain the entire flight system an ability to readily read the sensor responses will be crucial as the project moves forward. The ability to register real time data fed through the system will allow a better understanding of what will be the output to other components as well as the ability to maintain a stream of control data in order to diagnose any flight stability issues before test flight.

The system will be implementing a navigation computer that will be handle the autonomous flight and imaging as well as transmission of these images to the ground station. This system will be maintained on a Raspberry Pi model B. This runs on a Linux Raspbian operating system, a Debian variant operating system that is optimized for Raspberry Pi ™ hardware. Through the use of this powerful system, communication will be attainable with the microcontroller and the camera. Programs for this system will have the ability to run straight out of the command prompt. It will also fall upon this operating system to test out the DTN software during flight.

All code written for this project will be written in C++. This is a general purpose programming language that is one of the most popular programming languages used. It is a mid level language above assembly that CCS can interpret and compile into hardware language before sending to the device. With this language being the most dominant programming language as far as team understanding and implementation, it was an obvious choice with which to move forward.

7. Prototype Testing and Evaluation

7.1. Microcontroller RTOS and Peripheral Driver Testing

Making sure that the software for the Tiva C ™ software is ready for flight will require several phases of testing in order to show the predictability and accuracy of the software created and used. There are several elements of the software that will have to be tested individually before integrating them into a complete system. The Real Time Operating System (RTOS) and peripheral drivers are the subjects of interest for this section. The RTOS needs to be running and tested before integrating the peripheral drivers.

The reason why an RTOS is going to be used for this project is to prevent a single process from holding up the processor which would prevent other tasks from running. Making sure that all of the real time events that occur when implementing a digital control system do not get interrupted or blocked is critical to its operation. Before anything is done with any control algorithms or peripheral drivers, the RTOS must be running and tested. After installing the RTOS support software within Code Composer Studio™, the things that will be tested is its ability to schedule tasks, handle priorities, and do preemption. Each of these features will be tested separately using programs that will force the operating system to do just these tasks.

Scheduling needs to be tested so that tasks can run at fixed intervals. Testing this should involve more than one task. Initially one task will be created and should be set to run at a certain interval, such as once every second. An example would be toggling an LED. The interval in which the LED is on and off should match the specified schedule. This will be confirmed with an oscilloscope. The next thing that needs to be done is adding another task for the operating system to manage. This can be done with another LED or a GPIO pin set to toggle at a different rate. Again, the accuracy of the tasks can be measured with an oscilloscope. Different intervals will also be tested and perhaps a scheduling conflict will be tested too. The team will test certain intervals that overlap and cause conflicting schedules. This is so the team can observe the behavior of the TI-RTOS in order to understand what would happen in this situation.

Priorities are going to be tested at every level. The complete number of priority levels the RTOS supports will be tested. The highest priority task should always come before any other priority. Priorities are important when multiple tasks come in at the same time and the RTOS has to decide which one will get executed. Figure 55 shows a illustrative example of three processes: GPS, accelerometer, and magnetometer, being placed in the RTOS (represented by a funnel) at the same time. The one with the highest priority comes out first. This will be tested by creating two or more tasks with different priorities. Task one will have the highest priority and will turn on the red LED built into the evaluation board for one second. Task two will have a lower priority and will turn on the green LED for one second. They will both be scheduled to launch at the same time. This is easy to confirm just by seeing that the red LED comes on first. The same test will be repeated with both tasks at a lower priority level. The result should be the same. When both tasks have the same priority, the RTOS will have to pick one even though they were scheduled at the same time. This is another behavior that needs to be observed and documented.

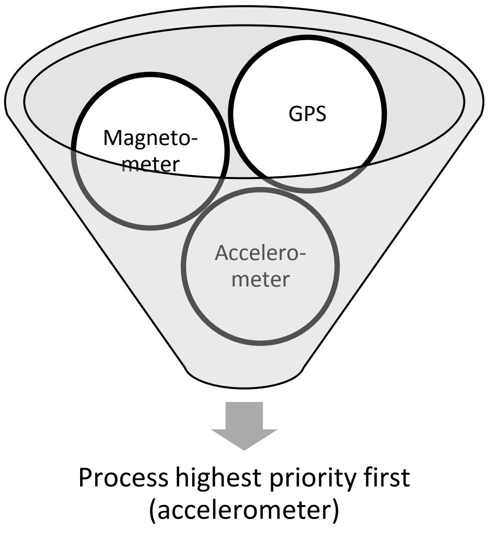


Figure 55 RTOS Priority Funnel

Preemption will be tested similarly to the priority testing above. The difference will be the lower priority task (task two) will turn on the green LED for five seconds and the higher priority task (task one) will turn on the red LED for one second. Task two will be scheduled first and task one will be scheduled one second after task two. If the RTOS preemption works right then the green LED will turn on for one second, then switch to red for one second, and finally back to green for the remaining 4 seconds. This test will also be ran again with task two at a higher priority than task one to make sure that preemption does not occur when the task with the higher priority is already running. The last test will be to have both tasks at the same priority level. The operating system will just run the task that came first.

Peripheral driver testing will be done through software and physically. At this point the microcontroller would have already been tested during the RTOS test since the team used the evaluation board to test it. Software has to be used to test the peripheral sensors since they will provide feedback through digital communication. The team will use the microcontroller for all of the peripheral devices. The accelerometer, gyroscope, magnetometer, and altimeter are all going to communicate via I2C. Each slave device should respond when a message is sent to it to retrieve data. None of the other devices on the I2C line should send any data if their address was not called from the master. The GPS and the Raspberry Pi ™ will communicate via UART. The 2.4 GHz remote control receiver is proprietary and setup like described in Section 5.3. The accelerometer, gyroscope, and magnetometer are on the same board. These will be tested separately when communicating with them and will have to be tested together physically. The altimeter, GPS, and Raspberry Pi ™ are physically independent. Each device will be tested to make sure they match the needed specifications for this project.

The accelerometer will be setup on the microcontroller to forward data to a laptop. After connecting the accelerometer to the microcontroller, the self test will be initiated. The accelerometer produces an electrostatic force that acts on the internal sensor in the same way that acceleration would. The self-test output is scaled based on the supply voltage. Using a 3.3V power source will scale the output by 1.77 on the X and Y axis, and 1.47 on the Z-axis. Since the accelerometer is set in the ±2 range and the power supply is 3.3V the output should fall within the following scaled parameters:

- $88.5 \leq$ X axis ≤ 955.8
- $-955.8 \leq$ Y axis ≤ -88.5
- $110.3 \leq$ Z axis ≤ 1286.3

If all of the axes are within the boundaries then testing can continue. If they are not then the test needs to be repeated. If the self-test continues then the other three gravitational force ranges should be checked to see if something may have been wrong with the initial parameters. The accelerometer will have to be replaced if the required ranges are not satisfied. Next MATLAB will be used to graph the data in real time. All three axes will be tested. The magnitude of each motion needs to be documented in order for the team to fine tune the sensor data into realistic values for the microcontroller to make determinations from. This test should be analyzed with movements in a similar magnitude that the accelerometer would feel on the quadcopter. The new data that comes through the FIFO and placed in the registers should be updated at a rate of 800 Hz.

The gyroscope does not have a self-test feature. It will be tested similarly to the accelerometer by having the microcontroller forward the data to a laptop and viewing the results in MATLAB. This data needs to be read from the upper and lower half registers of the X, Y, and Z axes. All directions of movement need to be tested and verified by rotating the gyroscope on each axis. The range and magnitude of each movement needs to be documented for fine tuning when it is on the quadcopter. The sign also needs to be verified. When the gyroscope is rotated clockwise on an axis it should be negative, and it should be positive when rotated counter-clockwise.

The magnetometer has a self-test option that is more involved than the accelerometer. The magnetometer internally generates its own magnetic field by pulling 10mA through the offset straps. The self-test sends a pulse, takes a single measurement, sources 10mA to generate about 1.1 Gauss, then takes a second measurement. The two measurements are then subtracted and stored in the X, Y, and Z output registers. The process of doing a positive self-test is shown in Figure 56. CRA and CRB stand for Configuration Register A and B respectively. The MR is the Mode Register. During self-test and flight, register 2 (MR) will always be in continuous-measurement mode. When self-testing the gain should start at 5 and if the data registers are not within the required limits, the gain will be increased. When the gain is increased, the next measurement needs to be skipped. This is why there would be two delays if the gain is increased. If the gain exceeds 7 then one or more axes did not pass the self-test. This test will be done for both positive and negative biasing.

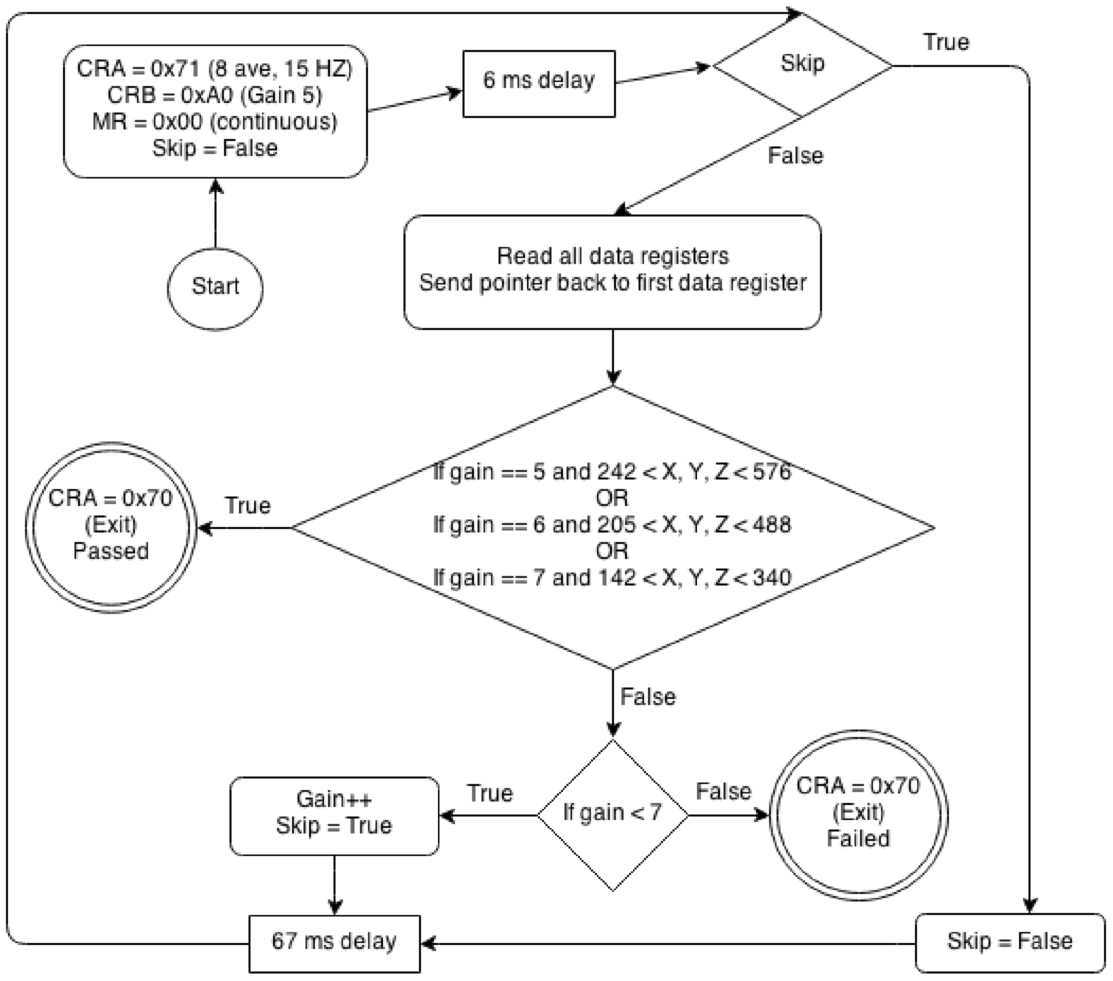


Figure 56 Magnetometer Self-test

Once the self-tests pass, the digital compass needs to be tested against a real compass. The values sent to the computer should show the magnetic direction of the earth. These should also change as the sensor is moved or rotated. The sensor can detect a magnetic field in three dimensions so the sensor should be rotated in every possible direction. North should be consistent throughout the test.

The altimeter will be tested the same way via I2C. This will require the team to use a tall building in order to see if the data provided from the device is accurate. The first thing to test with the altimeter is to read the calibration data from the E2PROM. In order to use the altimeter the algorithm discussed in Section 5.3 needs to be implemented. Once this is all working, the pressure should be compared to the actual pressure in the room with another device or by looking up the approximate pressure for the area the sensor is being tested in. The pressure then needs to be converted into height. At ground level the pressure will be read and set as a reference point of 0. The device needs to be moved up a floor to see a significant difference. This needs to be compared to the actual altitude and repeated a few times to get a set of data to represent how precise the measurements are. It should show a lower presser the higher the device is and vice-versa. This is good for general flight but the most critical time that altimeter is needed is during landing. If the quadcopter descends too slowly, it could take several minutes to land. If it is too fast then the quadcopter could be damaged. The ideal solution is for the quadcopter to slow down its decent as it gets closer. The altitude that needs to be tested the most is in the 5 to 0 foot range because that is when the quadcopter is about to land. Measurements will be taken every inch from 0 to 5 feet to see how accurate the sensor is at such a low altitude. The altimeter's datasheet claims that the altitude noise is 0.25m so this needs to be verified inside and outside.

The microcontroller will also forward the UART data from the altimeter to a laptop just like it did with the I2C testing. The GPS will mainly be tested outdoors so it has a line-of-sight signal to the satellites. This will be tested against the GPS in an Android phone and the data provided from Google Earth ™. It needs to be within ten feet of accuracy and tested in various locations. The response time needs to be tested on a clear day and a cloudy one. The output also needs to be observed when there is no signal. This checks if incorrect data is supplied when there is no signal. The six main NMEA sentences need to verified with each other to make sure the data the GPS is providing is consistent. The frequency at which each one occurs also needs to be noted. This will help decide which sentences to read the most or to reference from more frequently. The GPS needs to be tested when it is stationary so the latitude and longitude stay static and the speed is zero. It will then need to be tested while moving. This can be done in a car so the testers can use the speedometer as a reference. The things to check during these tests are the actual current location, speed, direction, changes in dilution of precision, and active satellites. Based on all of the testing, the team can find the optimal sentences to use for each state the quadcopter is in during the mission.

The Raspberry Pi ™ is also going to communicate through UART, but is not a sensor. The only thing that needs to be tested for the microcontroller is sending and receiving messages with the Raspberry Pi ™. This can easily be done by sending a number to the Raspberry Pi ™ and have it double the number and return it. The camera will be attached directly to the Raspberry Pi ™ and will not be accessed by the Tiva C ™ microcontroller. It will be tested by the Raspberry Pi ™ taking a picture with it every five seconds. The camera will be facing a clock with a visible second hand. The pictures will be stored on the SD card. The SD card will be removed and read from a computer. Each picture should show a time that is five seconds apart. This test proves the Raspberry Pi ™ can take pictures on command.

The receiver will have individual wires connected to the microcontroller. The handheld transmitter controller will have to be used to test this part. A program will have to be written for the microcontroller to send information about what it is being received from the receiver. Each movement with the joy sticks on the controller should be detected by the microcontroller and sent to the laptop. All ranges of motion with the controller should work with any speed of movement. There should not be any glitches. This is important to test for reliability and accuracy since these movements will all be mapped to what the microcontroller sends to the ESCs.

The final step in this section is to integrate the peripherals in the real-time operating system. Each peripheral will have a different schedule and priority, as discussed in Section 5.3. Out of all the peripherals, the accelerometer and gyroscope will have the highest priority and scheduled rate. The receiver is in second place followed by the altimeter. The UART Raspberry Pi ™ task has the second lowest priority but is still important since it tells the quadcopter what to do during the mission. The actual frequency of each access to the peripherals will have to be done experimentally. The integration of all this is running through both the RTOS and peripheral drivers test again with the peripheral drivers test within the RTOS test. Preemption, scheduling, and priorities should all work with the peripherals. This whole test section may seem inefficient since the RTOS and peripheral drivers could have been tested at the same time. The reason for this method of testing is to avoid any uncertainty when something does not work right. This alone could save time and prevent potential problems.

7.2. Filtering Algorithms & Control Loops Testing

In this section the testing of the filtering algorithms and the control loop testing will be given. These tests will be performed after the RTOS and peripheral drivers are proven to work properly. The least squares filter will have to be shown to reduce noise from the sensors and provide a proper estimate of position based on data. The control algorithm will also need to be tested as well as tuned.

For the least squares filter, data coming out of the sensors must be retrieved and plotted in order to show that the filter is working properly. The sensors will be read by the microcontroller where they will be sent serially to a computer to be processed in real-time as well as post-processed. As the data is coming in, there are two methods that will be employed to process the data. In order to test for accuracy and response time, MATLAB will be used to plot the data in real-time. Using this method, even small amounts of noise can be seen on the plot as well as any inaccuracies with the data output. The other method is to represent the data that is coming out of the data as a three dimensional cube. This method only works for the gyroscope, but it is good for determining if there is drift in the sensor, or jitter. The ability to track movements is also well represented using this method. Using these two methods, the filter should hopefully be shown to improve the quality of the data, while not increasing latency.

After the filters are working properly, the PID control algorithms must be tested. This will be done using various steps to show that the control loops are providing the right compensation based on the input. There are three parameters for each axis that must be tuned in order to make the quadcopter stable. The proportional gain, the integral gain, and the derivative gain for each axis need to be tuned. Before assembling the quadcopters, the control algorithm will be tested by reading the controller's PWM outputs on an oscilloscope and verify that they are showing the right control action based on the input. The full range of the PWM will be tested.

All of the filtering will be tweaked before implementing them on the microcontroller. From here small testing steps will be made to ensure everything is functioning properly. Just because the filter may appear to have made the sensor data smooth, does not mean the result of the data will function correctly in the quadcopter system. Other factors that may not have been considered will show themselves when everything is together. The control loops are also critical to tweak due to their nature.

When the quadcopters are ready, the control algorithms must first be tested while the each quadcopter is tethered. The goal is to get the algorithm working first and then transfer the program to the other quadcopter for use later. The quadcopter must be tethered for the initial testing in order to make sure that the system won't destroy itself when it flies for the first time. When the algorithms have been tuned enough so that the team feels comfortable enough to fly without a tether, un-tethered flights will begin.

7.3. Hardware Testing

The hardware testing involves all of the components and systems that are not programmable like the microcontroller, or semi-programmable like the sensors. This includes the motors, propellers, electronic speed controllers, lithium-polymer battery and charger, power breakout cable, Wi-Fi on the Raspberry Pi ™ and the laptop, and the power distribution system. In general throughout the hardware test plan, one component was added at a time. Each test built upon another test and that test the components in series like shown in Figure 57.

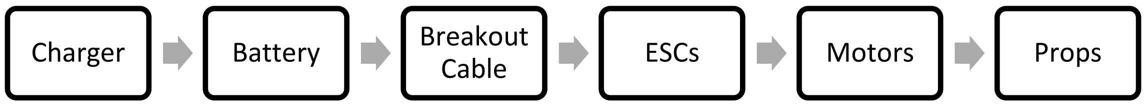


Figure 57 End-to End Power Testing Sequence

The battery is a 4 cell lithium polymer 5000mAh battery. The main plug needs to fit into the power breakout cable. The plugs on the end of the breakout cable should all fit in the ESCs. The battery also needs to fit the plug of the charger. This needs to be able to charge the battery. Using a voltage meter, a reading will be obtained when the battery is drained and the voltage should be less than 11V. Plug the battery into the charger and allow it to charge until it is complete. After unplugging the battery from the charger, it should have a voltage of 14.8V or higher. The temperature is something that should be monitored when first using the battery. This will be done while the battery is charging and discharging. If the battery gets above 50℃ when it is discharging, then the load is pulling too much current for this battery. A higher rating battery may be necessary to obtain. Charging is done at a slower rate than the rate at which the quadcopter would discharge it so the battery should not get very hot. If it does then this could be due to overcharging. Some cheap battery chargers are known to do this. If it is overcharged too long the battery will expand which causes permanent damage.

The motors and the electronic speed controllers (ESCs) will be tested together. The motor controllers will be tested using pulse-width modulation from the microcontroller. The motors need to be tested the entire voltage/speed range. The ESC should be able to drive the motor at full speed and completely stop the motor. This test's the functionality of the ESCs and the motors since they should work at any speed in-between the supported ranges. The total current of one motor and ESC combined should not exceed 18 amps. with or without the propellers attached. The speed and torque needs to be verified and nearly identical for each of the four motors with ESCs. The current they pull at each speed needs to be observed through the entire voltage range with the blades on so the team can have a current profile to calculate minimum and maximum current consumption and runtime. The thrust can be measured using a weight and a digital scale. The motor and propeller will be attached to the weight to hold it down. Once everything is on the digital scale the total weight can be subtracted by using the built-in tare feature on the scale. The thrust will be negative. It should reach around -0.75kg. Each motor will be compared for consistency. The propellers should not break during any of these tests if they do not come into contact with anything.

The USB Wi-Fi dongle in the Raspberry Pi ™ needs to connect to the laptop through its Wi-Fi card. This would require the laptop to host a network and the Raspberry Pi ™ needs to join it. Communication needs to be tested in both directions. The Raspberry Pi ™ should be able to send statuses and other information periodically. Next the range between the two devices will be tested. This test will not require DTN. The Raspberry Pi ™ will remain at ground level. One team member will walk away with the Raspberry Pi ™ connected to the laptop via Wi-Fi and see how far he can go before the connection is lost. This will be the approximate distance that the quadcopter can fly away from the ground station. This will be tested multiple times in different locations and environments. The flying results in the future may be better.

The Raspberry Pi requires a 5V power source and the Tiva C ™ microcontroller requires a 3.3V power supply. The only source of power on the quadcopter is the 14.8V Li-Po battery. This is why voltage regulators are added to the circuit. These need to be tested in order to prevent a brown or blackout for the processors. They will first be tested by connecting them to a fully charged battery. Next they need to have a test load that pulls a little more than what the maximum power consumption is on the microcontroller and the Raspberry Pi ™. The voltage shall remain at the voltage that should be provided from each regulator. The battery should then be discharged down to 10V or even lower. This should not cause the voltage to dip on the output of the regulators. This is important to test because having a low battery could cause the flight microcontroller to turn off and the quadcopter would crash.

All of these components need to be tested rigorously to ensure that there will be no problems during flight. The power train was the main system in this section to be tested. When dealing with large currents it is not a bad idea to check on how warm or hot all of the components are getting. Generally if something is wrong a component will be too hot to touch. The battery should be disconnected immediately and the problem needs to be investigated. Sometimes is a faulty component, other times it is the implementation.

7.4. Integrated Phase 1 Manual Flight Tests

A manual flight will be conducted with a RF controller that transmits movement commands to the quadcopter. This is how a typical remote control quadcopter that one could buy from a hobby shop is controlled. At this stage the quadcopter has all of the sensors it needs to stabilize itself. This would include the accelerometer and gyroscope. Other sensors such as the GPS, magnetometer, and altimeter are not required for self-stabilization. This test will demonstrate that the quadcopter can self-stabilize, move in a desired direction on all three axes of motion, function with an interrupted or weak signal from the controller, and fly outdoors. After that all of the peripheral will be tested on the quadcopter during flight. Figure 58 shows the hierarchical format of this test phase. Anything that can be performed in this test should prove that it is possible to do autonomously. Also autonomous expectations cannot exceed manual flight capabilities.

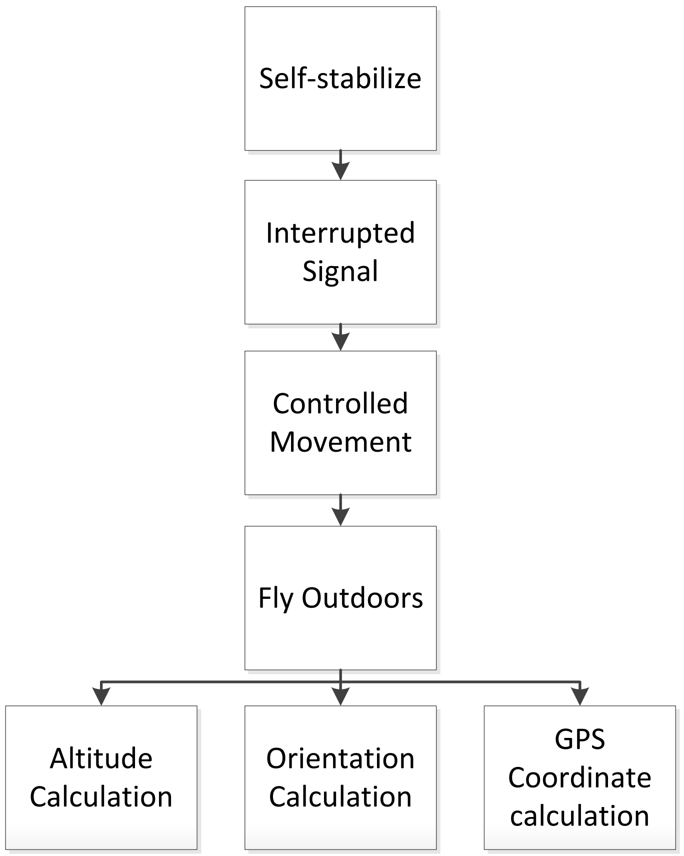


Figure 58 Manual Flight Test Plan

Self-stabilization is something that the quadcopter has to do autonomously regardless of whether or not the controller is transmitting movement instructions to it. The system has to continuously make fine adjustments to the motor speeds in order to stay level in one position at the current altitude (not precisely since the altimeter and barometer would not be required) based on the accelerometer's feedback when the controller is idle. This will first be tested indoors with provisions for crashing. This will be tested by turning on the quadcopter, providing thrust in order to get the quadcopter to lift of the ground and then back off the thrust to keep the quadcopter from gaining altitude. The user with the controller does not have to stabilize the quadcopter; he only has to move it in any desired direction including rotating. When nothing is being operated on the controller, the quadcopter should stay somewhat in its position in a controlled environment. It's normal for the quadcopter to start drifting as long as it is gradual and not drastic.

Movement will be controlled with the two joysticks on the controller. Each movement will be tested. This will first be tested indoors with provisions for crashing. The left joystick controls altitude and rotation. When it is pressed up, the quadcopter will move up the z-axis. When it's press down the quadcopter will move down the z-axis. This will mainly be used for lifting off and landing. When the joystick is pressed right, the quadcopter will begin to spin clockwise and when it's pressed left it will spin counter-clockwise. The right joystick controls leaning forward, backward, left, and right. Leaning in a certain direction causes the quadcopter to move in that direction. Pressing the joystick in any direction on the x-y plane will cause the quadcopter to correspondingly move in that direction. Once all of the directions are confirmed functional, multiple movements at the same time will be tested. This means the quadcopter should be able to perform x, y, and z axis movement at the same time, even with rotation. This will prove how robust the quadcopter's movements can be.

The next test is to fly the quadcopter with a weak or interrupted signal. The quadcopter should prevent itself from crashing when this happens by slowly lowering the thrust until it reaches zero. This is somewhat autonomous, but it is not sophisticated yet. That will be covered in detail in Phase 2. This safety feature will be used for three things: manual flight with a broken signal, a safe mode that the code will default to if something goes wrong in navigation (once the autonomous flight is implemented), and if a critical battery status occurs. This is mostly going to be used for manual flight tests. This will first be performed indoors with precautions in place for a crash landing. First the user will launch the quadcopter off from ground level with a good signal from the controller. Then the user will get the quadcopter in a hovering state. The controller will then be shut off. What should happen is the quadcopter stays level and slowly descends. This test will then be repeated using a signal that is intermittent, and lastly with a critically low battery. The intermittent signal will have different results than turning off the controller because the quadcopter should resume normal operation when signal is repaired and descend once the signal is broken. This transition should be smooth in order to prevent the quadcopter from quickly reacting which could cause it to crash. The critical battery status will cause the quadcopter to descend just like it did when it lost connection with the controller.

Once all of the above tests have been successfully performed and all of the revisions have been finalized, the quadcopter will be flown outside. This will be done to test the quadcopter's ability to fly in situations where wind or other environmental changes are present. First the team will fly the quadcopter outside with some wind. The quadcopter can be pushed around by the wind, but should not become unstable since the stabilization system should compensate for any involuntary movements. Next the team needs to confirm that the quadcopter can fly at a decent altitude and distance from the starting location. This will provide experience that can be applied to the autonomous flight control later. Range testing will be done at ground level. The quadcopter will be at rest on the ground and the team member with the controller will walk way. After about every 50 feet, the user will spin the blades of the quadcopter to confirm that the connection has not been broken. Once a distance is reached where the controller will no longer communicate with the quadcopter, the user will then walk 50 feet towards the quadcopter and see if he can fly the quadcopter to his location. That covers the preliminary flight test procedure needed for having a fully functional quadcopter.

Now that the quadcopter can function as it should, the team can test the other devices that will be used on the quadcopter. This will include the altimeter, magnetometer, and GPS. All of these sensors were already tested individually, but need to be tested on the quadcopter together in a flying environment. These are tested here in the manual flight section because they all have to be working in order to perform the autonomous testing in Phase 2.

The altimeter allows the microcontroller to know how high the quadcopter is flying. With this integrated on the board with the microcontroller, it should be able to read and adjust to it. The first test would be to see if the quadcopter can hover more accurately when commands are not being sent by the controller. Next, the team will hardcode an altitude to reach such as 50 feet and see if the quadcopter can recognize that it reached that height by turning on a red LED when it is flown to that height. The LED will stay on for visibility purposes, but using a tall building would be preferred. The next test will be to simulate a landing. This will simply blink an LED at a fixed rate when the quadcopter is above 5 feet. When the quadcopter descends, the LED will blink faster the closer it gets to the ground. When the altimeter detects that the quadcopter is within a foot off the ground, the LED will turn off. This helps analyze the accuracy for landing.

The magnetometer is easy to test since it only involves the quadcopter's rotation on the z-axis (when it is hovering level). When the copter is flying it will be setup to log the orientation of the flight the whole time. One of our team members will then write down the time and orientation every 30 seconds. This will also be logged every 30 seconds by the microcontroller. Once the flight is over the team will then verify that all of the readings were correct. The next step is to hardcode the orientation that the quadcopter should be at. The team will set the orientation of the quadcopter to always face north. Next the person with the controller will then spin the quadcopter on the z-axis and see that the green LED is only on when the quadcopter is facing north.

The GPS will allow the quadcopter to know where is on the earth. This will be tested by manually flying the quadcopter around and collecting GPS coordinates the entire time. When the flight is over the team will then verify that all of the data is correct. The team will then hardcode in GPS coordinates for the quadcopter to be manually flown to. This will have to be done within range. As long as the quadcopter is not at the correct coordinate, the red LED will be on. Once the quadcopter arrives at its destination, a green LED will turn on and the red LED will turn off.

All of the tests that were performed demonstrate all of the necessary functionalities that the quadcopter must have for this project. They should be tested multiple times in different ways and areas. Self-stabilization is the basic function that has to be mastered before anything else can be developed for the quadcopter. Knowing that each sensor is completely functional with the system as a whole means there should be no major issues implementing the autonomous flight.

7.5. Integrated Phase 2 Autonomous Flight & Imagery Flight Testing

Everything that was performed in the manual flight test should be done autonomously. The difference is the main board will have every sensor integrated and functioning. Autonomous flight can be very dangerous since the team would have to rely on the code they wrote to handle all of the operations without any controller input. This is why the team decided to have the controller on during all of these tests which enables manual override in the event of a potential crash. The functions that need to be tested are taking off and landing, movement, navigation, imagery, and a complete mission. Self-stabilization would have already been perfected in phase 1 testing. Figure 59 shows the hierarchical format of this test phase. Unlike Phase 1, it is completely linear.

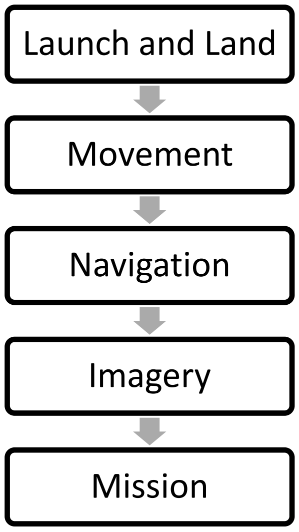


Figure 59 Autonomous Flight Test Plan

The first thing to do is see if the quadcopter can take off on its own. Simply setting the thrust to full power would prove that the quadcopter can lift off the ground. In reality it has to do two things: take off, and stop gaining altitude so the quadcopter can hover at a certain height. To test this, the quadcopter has to start quickly and gain altitude. Once the altimeter senses that the quadcopter is at least six feet in the air, the quadcopter should lower the thrust to a point where it is no longer gaining or losing altitude. At this point it should stay at this altitude for one minute. The next test for it to perform is landing. This will slowly descend the quadcopter until it touches the ground. The difference from the previous emergency landing from Phase 1 is the closer the altitude gets to zero, the slower it will descend for a soft landing. This will allow the quadcopter to land in a timely manner if it is very high.

Autonomous movement will be tested in a controlled indoor environment with provisions for crashing. Once the quadcopter has lifted off and is in hovering mode, it should then move to the left a few feet, and then back to where it started. This test will be repeated for right, forward and backwards movements. If the quadcopter successfully has done each of the four movements it will then fly in a horizontal square. The quadcopter will be facing the same direction the entire time. Next the quadcopter needs to spin on its z-axis. It can't move from its location when doing this. This will incorporate the magnetometer. It should start by facing north and then point east, south, west, and back to north. This will be tested again rotating counterclockwise. The next test would be to incorporate all of these movements into a more complex pattern such as flying in a horizontal figure-eight. This tested all of the movements in the x and y plane. Next is to test three dimensional movements. The z axis test would be for the quadcopter to fly in a cube pattern. After that the quadcopter will simulate a sine wave. Finally it should be able to fly in a vertical figure eight.

Navigation will require an algorithm that determines a route based on the destination. This part of the test will involve the ground station. During this entire navigation test the Raspberry Pi ™ will send its GPS information to the laptop via Wi-Fi. At this stage the DTN will be in place. Since the quadcopter will only be used in an open environment, meaning that there will not be any objects that it has to avoid at that specified height, the route can simply be a straight line. An easier way would be for the quadcopter to travel longitude first and then latitude, but that would reduce the size of the area to be captured since it would take longer to fly to that location. This test will require a destination and altitude to be entered on the laptop and sent to the Raspberry Pi ™ via Wi-Fi. Once entered, the Raspberry Pi ™ will calculate the route it must take. First the quadcopter will climb to the desired height to avoid any objects under the preset altitude. Then the Raspberry Pi ™ will send a vector to the microcontroller to go to its destination. It will use the GPS the entire time to make sure it does not fall off course. Once it reaches its destination, it will turn on an LED and land. This will be tested at different starting locations to ensure that it always goes to the same place. More testing will be done in different areas and destinations. Since the quadcopter is supposed to make a map, the next test is to give the quadcopter four coordinates and have it return. It will decide which coordinate it is closest to and fly two it. The quadcopter should then fly to each of the other three positions before landing back at its original starting location where the ground station is.

Imagery is handled by the Raspberry Pi ™. The camera will be connected directly to the Raspberry Pi ™ and the images will be stored on the SD card inserted into the Raspberry Pi's SD card slot. The Tiva C ™ microcontroller will communicate with the Raspberry Pi ™ board via UART. This will tell the Raspberry Pi all of the needed GPS information. The Raspberry Pi ™ will determine when to start and stop taking pictures. This will be tested by entering the GPS coordinates from the ground station for the quadcopter to fly to. When the quadcopter reaches the correct position, the Raspberry Pi ™ will tell the microcontroller to hover so it can take a level picture. This will be done at each photograph location. When the Raspberry Pi ™ has finished, it will tell the quadcopter to go to the next destination. When the miniature mission is complete, the only pictures on the SD card should be at that specified location.

Finally putting all of the autonomous testing together, the team needs to test a full mission with a single quadcopter. The quadcopter will be given four GPS locations to fly in a grid pattern and take pictures. The Raspberry Pi ™ is responsible for taking each picture at the required location. When the quadcopter finishes the entire grid, it will return to the ground station. The pictures will be taken off the SD card and put onto a laptop which will stitch them together to form a map. The map should not have any largely visible overlaps or missing portions of the area. This testing stage may require tweaking the amount of pictures that are taken during the mission. It is better to have too many pictures than too few. This test should also be performed at several different altitudes to figure out a formula that works best for determining how many pictures need to be taken based on the altitude and the camera lens angle. The higher the quadcopter is, the more area will be covered by the camera. If the quadcopter is too high then the pictures would be too far apart for the quadcopter to capture in the amount of battery life it has. Also the detail of the pictures will be reduced. If it is too low then there will be a large amount of pictures for a much smaller area. The optimal solution is best found through experimentation.

Once all of the testing in this phase is complete, then the project will be scalable. All of the small autonomous functions were tested before doing a single quadcopter mission. This testing should be done at multiple locations. Another factor to consider is the size of the area being mapped. The time it takes to map each area needs to be measured so the team can estimate the maximum area at the optimal height that the quadcopter is capable of doing. Once a mission test is large enough to drain more than 50% of the battery, it will be more difficult to test since the team has a limited number of batteries and they take a few hours to charge. There should not be any known bugs when this phase of testing is completed. The single quadcopter needs to be perfect before moving on to the next phase.

7.6. Integrated Phase 3 Dual Copter Autonomous Flight Testing

This is the final phase of testing. The testing outlined in this section involves the simultaneous interaction between the two quadcopters and the ground station. The second quadcopter SPOC is a replica of the first quadcopter KIRC. It will be ran through all of the same tests described in the beginning of Section 7. The parts that will be tested will be the division of an area, collision avoidance, and the wireless communication with DTN. Figure 60 shows the format of this test phase. Unlike Phase 1 and 2, it has no dependencies.

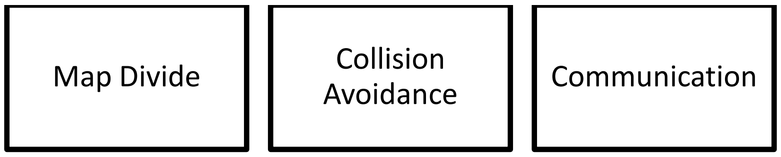


Figure 60 Dual Autonomous Flight Test Plan

Dividing the area between the two quadcopters will simply be cutting a rectangle in half. Each quadcopter will cover an equal sized square and they will both be sharing an edge. The laptop will just send the designated coordinates to each quadcopter and they will not know anything about the other quadcopter's path. This test should send out both quadcopters to survey the area. Once the mission is complete, the images from KIRC and SPOC will be stitched together as if one quadcopter took all of the pictures. If the final map is not seamless between KIRC and SPOC then the interval and placement of the pictures needs to be adjusted.

The quadcopters do not have any proximity sensors that are used to tell if they are too close to each other. The three potential collision stages during the mission are launching from the ground station, during surveying (particularly when they are capturing the edge they both share), and landing. These are shown in Figure 61. In order to prevent a collision during takeoff, they will not launch at the same time. KIRC will launch first then SPOC will launch about 30 seconds later. This delay is implemented on the ground station laptop so the quadcopters will have the same code onboard. They will also be placed several feet apart on the ground and this is where they should return once their mission is complete. They will not start at opposite sides of the map nor the seam of the map. KIRC will start at the top of its map and SPOC will also start at the top of its own map, which would be the side that both maps share. Figure 61(c) shows where KIRC would start, the green ring, and SPOC the blue ring. This contrasts with the collision in the figure since they should not collide if they started in the correct location. Another thing to consider when having those designated starting points is not having KIRK start much later than SPOC because SPOC will be almost done when KIRC is just starting. SPOC will not actually need to go all of the way to the edge since the pictures are spaced 50% apart anyways. Otherwise that would create redundancy, but it's better to prevent them from getting too close because a gust of wind could push one into the other.

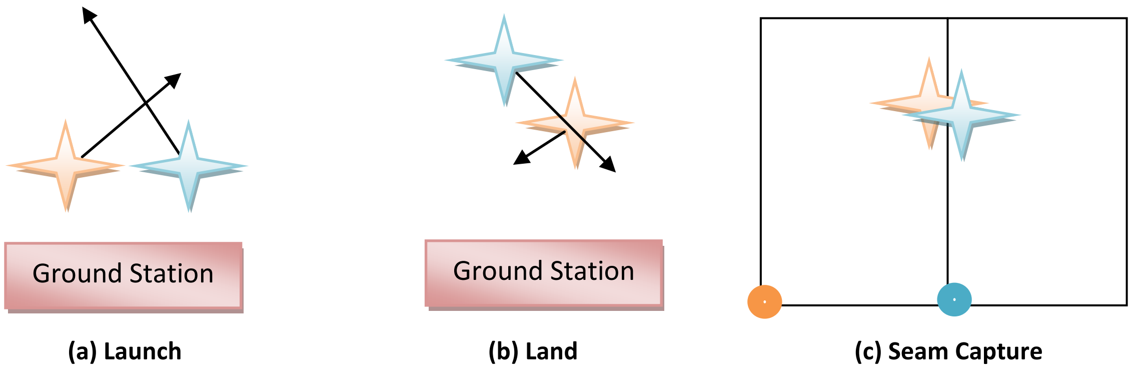


Figure 61: Three collision possibilities

The wireless communication between KIRC, SPOC, and the ground station should work

as described in Section 5.6. The quadcopters do not need to be flying in order to test this at first. The first test is the out of range test for both quadcopters shown in Figure 62. Two team members will walk away from each other and the ground station with KIRC and SPOC once they have connected to the ground station laptop. The signal should become weak or broken the further away they get. The status of each copter should still be able to update to the laptop when there is a connection since the network is supposed to handle the disruptions.

The next test needs to have KIRC in between SPOC and the ground station when SPOC is out of range with the laptop. This is shown in Figure 62. This will test if the data can hop from KIRC to the laptop and back instead of SPOC communicating directly to the ground station. It should also be tested with KIRC and SPOC swapped to see if the DTN works correctly for both quadcopters.

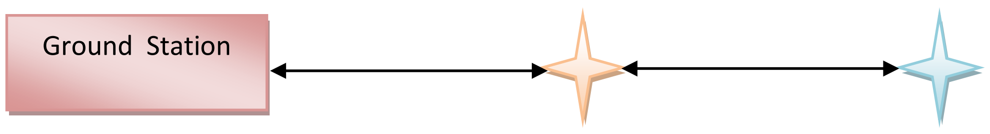


Figure 62 Dependent (Hop) Range Test

Multiple missions should be ran in different environments and locations. Each mission should be able to provide similar results. Time and battery life are the two variables that need to be watched closely during all missions. All of the prototype testing and evaluation is now complete. Everything necessary for the two quadcopters to function autonomously has been implemented. Tweaking is something that may never truly stop between missions. Each mission can provide more insight and there is always a lesson to be learned.

With these last things tested, everything should be thoroughly checked and the system should be completely ready for use on NASA earth science missions. The goals of repeatability, durability, predictability, and safety will be proven through thorough and incremental testing. Each phase of the project will have testing to do and objectives to meet.

8. Administrative Content

8.1. Milestone Discussion

As mentioned in the project description section, the Knight's Intelligent Reconnaissance Copter (KIRC) project is split into three prototyping phases. Each of the three phases represents a logical progression of how the team thought the project should be broken up. Throughout all the phases, there will be research, design, construction, and testing. Each of the phases are evaluated based on our abilities to achieve the goals within each respective phase. In this section, the goals, milestones, and schedules are defined for each phase is covered in detail for both Senior Design 1 and Senior Design II.

The first phase was longest phase of the project, as it ran the length of Senior Design 1 entirely. In this phase, one of the goals was to begin research into different quadcopter designs, parts, and software algorithms. Once an adequate microcontroller, inertial measurement unit (IMU), and a GPS were found, we decided to do a first round parts order with these parts in late September of 2013. When the parts arrived, the team began coding of the microcontroller software that controlled the motors and the stability algorithms that made the compensation. During this, research was also done for the physical quadcopter parts, such as the frame, motors, props, and batteries. Once the software was written, and the quadcopter parts came in, the team began integration of the quadcopter. After being fully integrated, the first prototype quadcopter will undergo rigorous stability, maneuverability, and durability testing under various conditions in manual mode only. The details of the testing phase, and the test plans associated with it are given in chapter 7 of this document. The goal is to have the Tiva C ™ microcontroller software completed and thoroughly tested before Senior Design II starts.

The second phase of the project starts at the very beginning of Senior Design II. With the quadcopter now capable of stable manual flight, the team will shift its focus towards making the quadcopter autonomous, creating a custom printed circuit board (PCB), and interfacing a camera to the quadcopter. For this part, the team will be split up into two separate task groups, which will hopefully speed up the progress during this phase. Two of the group members will be focused on creating a PCB, while the other two will be focusing on creating software to make the quadcopter autonomous with the embedded Linux computer. There will also be significant work towards researching an image stitching software for use on the ground computer, and also development of a ground station interface. At the end of this phase there will be doing more testing of the quadcopter, but this time on the autonomous capabilities and imaging software.

The third and last phase of the project will be to replicate the first quadcopter. Starting midway through Senior Design II, the team will replicate the first quadcopter, and begin simultaneous flight testing. Also during this phase the team will finalize the user interface, thoroughly test the dual copter system, and provide proof that DTN2 is working as NASA expected it to. There is about two weeks of buffer time during this phase to allow for scheduling delays.

Below is a list of milestones and dates that we came up with at the beginning of Senior Design II. This list does not include lost time, schedule delays, or other disturbances.

I.	Phase 1 (Senior Design 1): Build 1st Generation Copter	Aug 19th – Jan 6th
a.	Research and Design I	Aug 19th – Oct 18th
i.	Team Formation and Organization	Aug 19th – Sept 6th
ii.	Software Design	Sept 1st - Oct 1st
iii.	Hardware Design	Sept 1st -Oct 18th
iv.	1st Round Parts Acquisition *	Sept 16th – Oct 1st
v.	2nd Round Parts Acquisition **	Oct 1st – Oct 18th
b.	Software and Hardware Implementation	Oct 4th – Nov 15th
i.	Software: RTOS Implementation	Oct 4th – Oct 25th
ii.	Software: Peripheral Drivers	Oct 4th – Oct 25th
iii.	Software: Sensor Filtering & Control	Oct 18th – Nov 15th
iv.	Hardware: Assemble Frame & Motors	Oct 18th – Nov 1st
v.	Hardware: Make electronics mount	Oct 18th – Nov 15th
vi.	Use microcontroller to drive motors	Oct 18th – Nov 15th
vii.	Use microcontroller to read RC controller	Oct 18th – Nov 15th
c.	1st Gen Copter Integration	Nov 15th – Dec 10th
i.	Assemble the Copter	Nov 15th – 22nd
ii.	Tethered Flight Testing	Nov 15th – 22nd
iii.	THANKSGIVING WEEK OFF	Nov 25th – Dec 1st
iv.	Buffer Time	Dec 1st – Dec 10th
d.	1st Generation Copter Testing	Dec 10th – Jan 6th
i.	Test Manual Flight	
II.	Phase 2 (Senior Design 2): Build 2nd Generation Copter	Jan 6th – March 14th
a.	Research and Design II	Jan 6th – Feb 3rd
i.	Design PCB and send for fabrication	Jan 6th – Feb 3rd
ii.	Research image stitching software	Jan 6th – Feb 3rd
iii.	Ground Station Software Design	Jan 6th – Feb 3rd
iv.	Navigation Computer Software Design	Jan 6th – Feb 3rd
b.	Software and Hardware Implementation II	Feb 3rd – March 14th
i.	Software: Ground Station	Feb 3rd – March 14th
ii.	Software: Navigation Computer	Feb 3rd – March 3rd
iii.	Mount PCB and Navigation Computer	March 3rd – 14th
c.	2nd Generation Copter Testing	March 14th – 31st
i.	Test Autonomous Flight	
III.	Phase 3 (Senior Design 2) Replicate Copter and Testing	March 14th – May 2nd
a.	Copter Integration	March 14th – 31st
b.	Copter Testing	April 1st – April 14th
i.	Test Joint Autonomous Flight	
c.	Buffer Time	April 14th – May 2nd

** 1st Round Parts Acquisition Includes Microcontroller, IMU, and GPS part*

*** 2nd Round Parts Acquisition Includes Quadcopter Physical Parts*

**** This Milestone Outline Does Not Include Most Recent Dates and Scheduling*

The Schedules for Senior Design I and Senior Design II were organized from the milestone outline above into the two Gantt charts given below in Figure 63 and Figure 64 below respectively. In Figure 63, we have the Gantt chart for Senior Design I, which has been modified from the milestone outline above with more updated dates from our schedule used. In Figure 64 we have the projected schedule for Senior Design II. The major points that the team would like to achieve are: the construction of a stable prototype by the start of Senior Design II, a PCB with integrated sensors and processor by the end of February, a working autonomous quadcopter by mid-March, and two working quadcopters by mid-April.

Project Schedule		Senior Design 1															
Phase:		Phase 1: Build 1st Generation Copter															
Week:		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Research and Design I		🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨							
	Team Formation and Organization	🟨	🟨	🟨													
	Software Design			🟨	🟨	🟨	🟨	🟨	🟨	🟨							
	Hardware Design			🟨	🟨	🟨	🟨	🟨	🟨	🟨							
	1st Round Parts Acquisition					🟨	🟨	🟨	🟨								
Software & Hardware Implementation 1										🟨	🟨	🟨	🟨	🟨	🟨	🟨	
	RTOS, Peripheral Drivers, and Control Algorithm									🟨	🟨	🟨	🟨	🟨	🟨	🟨	
	2nd Round Parts Acquisition											🟨	🟨	🟨	🟨		
	Assemble Frame, Mount Motors and Electronics											🟨	🟨	🟨	🟨		
Hardware and Software Integration & Testing 1													🟨	🟨	🟨	🟨	🟨
	Use Microcontroller to Drive Motors & Read input												🟨	🟨	🟨	🟨	
	Assembly and Tethered Testing Copter													🟨	🟨		
1st Generation Copter Testing																	🟨

Figure 63: Senior Design I Schedule (Modified based on changes to schedule)

Project Schedule		Senior Design 2															
Phase:		Phase 2: Build 2nd Generation Copter						Phase 3: Replicate Copter and Testing									
Week:		17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
Research and Design II		■	■	■	■												
	Design PCB	■	■	■	■												
	Ground Station Software Research	■	■	■	■												
	Navigation Computer Software Design	■	■	■	■												
Software & Hardware Implementation 2					■	■	■	■	■	■	■						
	Ground Station Software Implementation				■	■	■	■	■	■							
	Navigation Computer Software Implementation					■	■	■	■								
	Fabricate PCB				■	■	■										
	Mount PCB & Navigation Computer									■	■						
2nd Generation Copter Feature Testing											■	■	■				
Replicate Copter and Testing											■	■	■	■	■	■	■
	Copter Integration										■	■	■				
	Dual Copter Testing													■	■		
	Buffer Time															■	■

Figure 64: Senior Design II Schedule

So far, the schedule has not gone completely as planned. One of the major events thus far that has slowed down the progress on the project was the government shutdown from October 1st through October 16th. With NASA providing most of the budget for the team, the government shutdown slowed down the acquisition of parts that was to occur from late September through late October. Some of the critical parts the team ordered themselves, as they did not wish to delay the project any further. Parts such as the Tiva C microcontroller and the 10 Degree of Freedom Inertial Measurement Unit were critical to the start of the software development so they decided to get the specific parts with their own money. The team is also planning for another possible government shutdown looming in late January. If this were to occur, the plan is to have all the necessary parts acquired by then so that the schedule will not be hindered by the possible shutdown.

In summary, the first phase, building a first generation quadcopter, covers from researching quadcopters to coding and testing manual mode. Phase two, making the quadcopter autonomous, begins during Senior Design II. During this phase, we would like for our quadcopter to have an embedded Linux computer on board and be able to autonomously fly to location(s). The last phase, replicating the quadcopter, begins midway through Senior Design II. In this phase, our goal is to completely replicate the first copter and have them cooperatively image an area. Not everything has gone to plan so far, as things have caused delay such as the government shutdown in October. This was not too detrimental to the progress of the project however, and only proved to be a minor setback over the long run. Overall, though, the team has tried to stick to the schedule as the guideline of when we should get certain items done.

8.2. Budget and Finance

Most of the budget for the KIRC project comes from an unofficial NASA sponsorship. The project has been given a rough budget of about $1000 to order parts necessary for outfitting the quadcopter, but the team has to buy the quadcopter parts themselves. These parts were eventually donated by Marc Seibert, the team's advisor from NASA. With the budget set by NASA, and the donated parts from Marc Seibert, the team has to allocate enough money for all parts. In this section, the budget allocation, the most up to date bill of parts, and any issues encountered so far with acquisition of parts.

With NASA's allocation of $1000 dollars, the team decided to allocate about $200 for avionics and control parts, $400 for quadcopter physical parts, and the rest of the budget for other miscellaneous parts. The goal was to make the cost of the quadcopters as little as possible while not limiting the functionality of the quadcopters themselves. In

Figure 65 below is the preliminary budget for the entire project. This budget does not include the exact parts and their costs, but it was based on a preliminary web search of parts and costs based on things needed for the project. Parts already acquired or freely available (via free samples), were not added parts into the total cost of the project. Each price estimate for each thing on the budget was rounded up to the nearest $5 and a quantity of each was given. Based on this estimate, the team came up with a rough cost of about $800 for the whole project. This included all the physical parts, the batteries, the avionics, the computers, and receivers. Keep in mind, however, that this is not the actual budget (provided later), but a preliminary estimate and allocation table based on initial estimates and research. Figure 65 on the next page shows the breakdown of the parts overall.

Category	Item	QTY	Price Ea. ($)	Total $	Need By Date	Status
Quad:ControlSys						
…	Microcontroller Launchpad	2	$15.00	$30.00	9/30/2013	To be acquired
…	IMU Sensor Unit	2	$25.00	$50.00	9/30/2013	To be acquired
…	GPS Unit	2	$50.00	$100.00	9/30/2013	To be acquired
Quad:FlightSys						
…	Speed Controller	8	$10.00	$80.00	11/4/2013	To be acquired
…	Motors	8	$20.00	$160.00	11/4/2013	To be acquired
…	Props	12	$4.00	$48.00	11/4/2013	To be acquired
…	Frame	2	$15.00	$30.00	11/4/2013	To be acquired
…	Li-Po Battery (4-5 A-h)	2	$40.00	$80.00	11/4/2013	To be acquired
…	RC Controller & Reciever	1	$50.00	$50.00	11/4/2013	To be acquired
Quad:GuidSys						
…	Embedded Linux Processor	2	N/A		N/A	Acquired
…	Power Cable	2	N/A		N/A	Acquired
…	SD Cards	2	N/A		N/A	Acquired
…	802.11G Wireless Card	2	N/A		N/A	Acquired
…	High Resolution Webcam	2	$50.00	$100.00	1/6/2014	To be acquired
Ground:GndStat						
…	Laptop	1	N/A		N/A	Acquired
Quad:PCBHardW						
…	Microcontroller Standalone	2	$10.00	$20.00	2/3/2014	To be acquired
…	Accelerometer	2	$5.00	$10.00	2/3/2014	To be acquired
…	Gyroscope	2	$5.00	$10.00	2/3/2014	To be acquired
…	Magnetometer	2	$5.00	$10.00	2/3/2014	To be acquired
…	Altimeter	2	$5.00	$10.00	2/3/2014	To be acquired
…	Passive Components		Free		N/A	Acquired
…	Active Components		Free		N/A	Acquired
TOTALS	All			**$788.00**	N/A	N/A

Figure 65: Generic Preliminary Budget Allocation

The parts will be ordered through a NASA contact, and for some parts there is paperwork that needs to be filled out. The team has to take into account, the purchase request process at NASA and allot at least a month of time before expect to receive any part requested. Parts will arrive at NASA KSC in the shipping and receiving building where they can be picked up by the NASA contact and be put in the team's custody for the remainder of the project. Parts bought by NASA will have to be handled with care not lose or destroy them, as they are still technically government property.

The most recent bill of parts is listed below in Figure 66, with status of each part, vendor website, and exact cost. Some of the parts were purchased directly from Texas Instruments, while others were from third party vendors such as Sparkfun or Amazon. Most of the quadcopter physical parts were acquired from Hobbyking, an online hobbyist retail store. The total cost so far is about $663.92 which is less than the initial estimate.

Category	Item	Part #	QTY	Price Ea. ($)	Total $
Quad:FltContlSys					
…	Tiva C Launchpad	TM4C123G	2	$12.99	$25.98
…	10 DoF IMU Sensor	Misc	1	$25.90	$25.90
…	50 Channel GPS Reciever	GP-635T	2	$39.95	$79.90
Quad:FltSys					
…	Electronic Speed Controller	9192000131-0	8	$12.49	$99.92
…	Prop Drive 900kv/270W Motors	NTM2830S-900	8	$14.99	$119.92
…	8x4.5 Propellers	17000055	12	$3.53	$42.36
…	Q450 Glass Fiber Frame (450mm)	297000028-0	2	$7.99	$15.98
…	4S 5000mAh Li-Po Battery	N5000.4S.45	2	$65.03	$130.06
	Turnigy Rx/Tx 9ch controller	TX-9X-M1	1	$54.00	$54.00
…	HXT 4mm to 4 X 3.5mm bullet Multistar ESC Power Breakout Cable		2	$3.65	$7.30
Quad:GuidSys					
…	Embedded Linux Processor		2	N/A	
…	Power Cable		2	N/A	
…	SD Cards		2	N/A	
…	802.11G Wireless Card		2	N/A	
…	RasPi 5MP Camera Board	7757731	2	$34.95	$69.90
Ground:TelStation					
…	Laptop	N/A	1	N/A	
Quad:PCB_HW					
…	Microcontroller Standalone		2		
…	Accelerometer		2		
…	Gyroscope		2		
…	Magnetometer		2		
…	Altimeter		2		
TOTALS	All				**$671.22**

Figure 66: Detailed Budget, Bill of Parts

In the course of the project, there have been some unforeseen issues associated with the acquisition of some parts. One, as mentioned in the previous section, was the government shutdown that delayed the procurement of the avionics parts. Another issue was the federal ban on the government purchase of parts from China. Unfortunately, this really limits what technological parts the team can purchase. One of the ways the team worked around this is that the ban is specifically related to computer parts manufactured in China, but there is an approved list of manufacturers. For the computer parts we used, specifically from Texas Instruments and Sparkfun, we filled out a request for investigation form for approval of parts from these companies. These forms were approved right before the government shutdown, and we were able to order these parts after the shutdown ended.

In summary, the budget allotment of $1000 from NASA is more than enough to cover the cost of the project. The goal to remain less than the allocated budget seems very realistic and attainable. The initial estimate for the total cost of the project was $778 and so far, the bill of parts comes out to under $700 for everything. The reason for the cost coming up much less than the initial budget is because the team found some vendors that have the parts they want for less than allocated for, and they also found some parts for free if sampled with the manufacturing company. The other cause is because the team also overestimated the budget on purpose in order to allow for any unforeseen cost. So far, though, the project has gone as planned from a budget perspective.

8.3. Team Organization

The KIRC project team is a group of four UCF seniors in engineering, with three electrical engineers and one computer engineer. The project has been divided into four areas, one for each group member. The areas of this project have been split into: Hardware, Software, Control Systems, and Team Lead. In this section, each team member’s job is explained and how the team has handled communication and collaborative file editing. Member profiles and brief bios are provided in the appendices for reference.

Nathaniel Cain, an electrical engineering senior, is the team lead. He is responsible for providing administrative leadership to the project as well as document tracking and editing. He is also responsible for the integration of the systems and subsystems that make up the project. Nathaniel is the NASA liaison for the project as is the NASA co-op/intern. His mentor is providing the budget for this project as well as guidance when necessary.

James Donegan, also an electrical engineering senior, is the hardware and power systems lead. He is responsible for quadcopter assembly, power distribution, wiring, and PCB design. James’ role is important in the assembly and integration of the quadcopter parts.

James Gregory, another electrical engineering senior, is the control systems lead for the project. His tasks include reading input from the RC controller, setting up a feedback controller, filtering the IMU sensors, and providing PWM output to the motors. This job is mostly software, and will require much collaboration with the software lead. James Gregory is also the backup hardware person behind James Donegan.

Wade Henderson, the team's only computer engineering senior, is the software lead for the project. He is responsible for software version management and organization, RTOS implementation, and peripheral drivers. Most of the other team members will have to coordinate their tasks with Wade because this project is mostly software. Wade is also the team lead backup, which means that he is team lead in the absence of Nathaniel.

During the first week, the team formed and exchanged phone numbers and email addresses in order to ensure contact with each member in the group outside of class. The team also formed a private Facebook ™ group where they could post updates and links amongst the members as an alternative form of communication. Another online tool utilized was Google Drive, a free online data storage service. This tool enabled the team to share files from a central online point and keep file management more organized. Despite all the technology making contacting and file sharing easier, the team still needed a weekly time to meet and share project updates. From the start, the team decided to meet on the UCF campus every Tuesday and Thursday from 8:30-9:00AM during Senior Design I.

Overall, the team is well organized into task groups that are able to tackle the problems and challenges associated with the project. The team's four members, each with their own diverse skill sets, have their own portions of the project that will be integrated into the final project. The team has multiple forms of communication, file sharing, and meeting options that are able to keep them organized and our information synchronized. In the end, it is the team organization, diverse skills, and the ability for the team to efficiently communicate and share information that will help team be successful.

Appendices

Appendix A – References and Other Projects Researched

1.) Hayes, M. H. "9.4: Recursive Least Squares." *Statistical Digital Signal Processing and Modeling*. New York: John Wiley & Sons. 1996..

2.) Ziegler, J.G and Nichols, N. B. "pages 759-768." *Optimum settings for automatic controllers*. 1942.

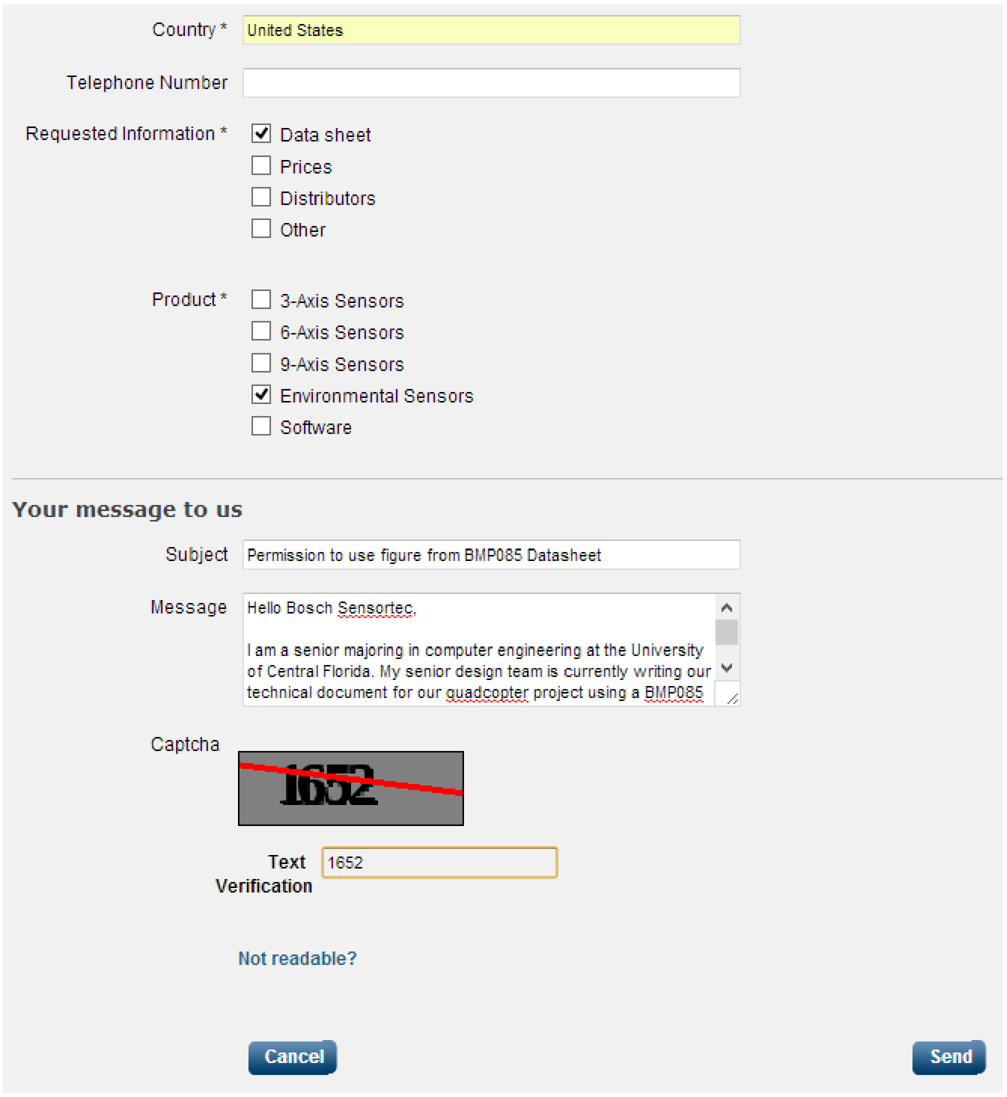
3.) Franklin, G. F. et al. "3.3: PID Control." *Digital Control of Dynamic Systems.* California: Addison Welsey Longman, Inc. 1998.

4.) http://eecs.ucf.edu/seniordesign/fa2012sp2013/g04/Final.pdf

5.) http://eecs.ucf.edu/seniordesign/fa2012sp2013/g17/Senior%20Design%202%20Report.pdf

Appendix B – Copyright Permissions

We were not able to write Bosch an email, we had to use this online form from:
http://www.bosch-sensortec.com/en/homepage/contact_1/contact/formedit/contact_form



Country *	United States
Telephone Number	
Requested Information *	✓ Data sheet ☐ Prices ☐ Distributors ☐ Other
Product *	☐ 3-Axis Sensors ☐ 6-Axis Sensors ☐ 9-Axis Sensors ✓ Environmental Sensors ☐ Software

Your message to us

Subject	Permission to use figure from BMP085 Datasheet
Message	Hello Bosch Sensortec, I am a senior majoring in computer engineering at the University of Central Florida. My senior design team is currently writing our technical document for our quadcopter project using a BMP085
Captcha	
Text Verification	1652

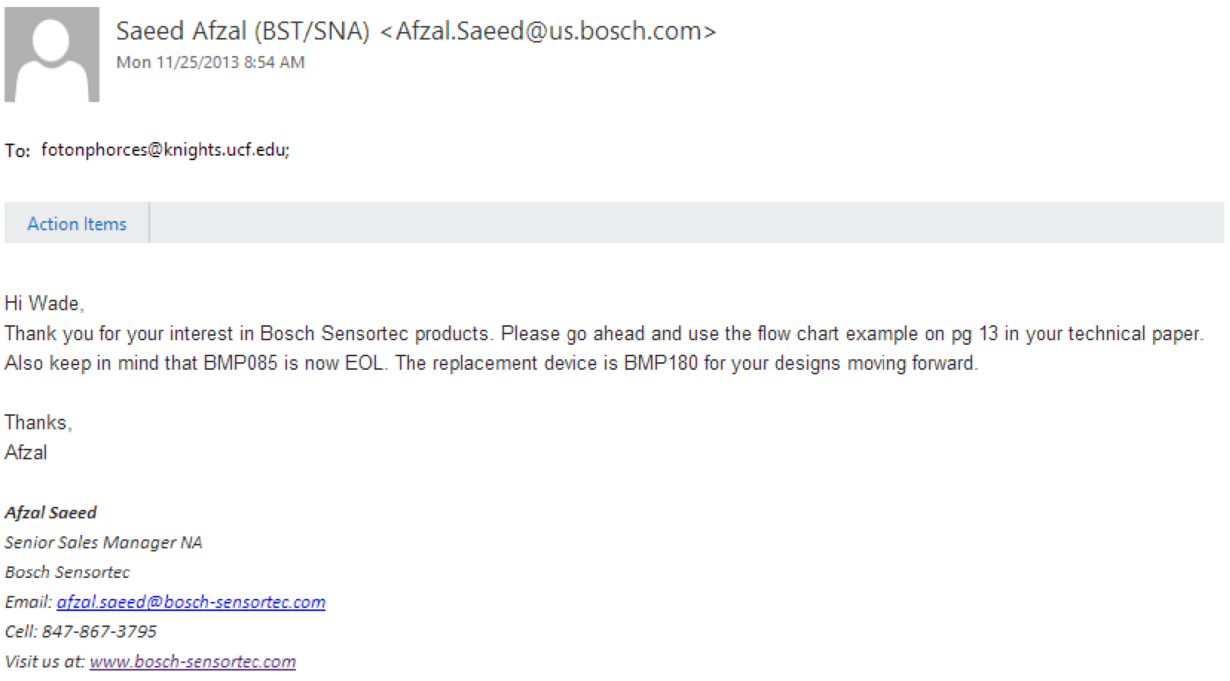
The actual text was:

Hello Bosch Sensortec,

I am a senior majoring in computer engineering at the University of Central Florida. My senior design team is currently writing our technical document for our quadcopter project using a BMP085 altimeter. We wanted to request permission to use the calculation of pressure and temperature figure on page 13 of the BMP085 altimeter document number: BST-BMP085-DS000-05 revision 1.2 our academic paper.

Best regards,
Wade Henderson

The response:



Saeed Afzal (BST/SNA) <Afzal.Saeed@us.bosch.com>
Mon 11/25/2013 8:54 AM

To: fotonphorces@knights.ucf.edu;

Hi Wade,
Thank you for your interest in Bosch Sensortec products. Please go ahead and use the flow chart example on pg 13 in your technical paper.
Also keep in mind that BMP085 is now EOL. The replacement device is BMP180 for your designs moving forward.

Thanks,
Afzal

Afzal Saeed
Senior Sales Manager NA
Bosch Sensortec
Email: afzal.saeed@bosch-sensortec.com
Cell: 847-867-3795
Visit us at: www.bosch-sensortec.com

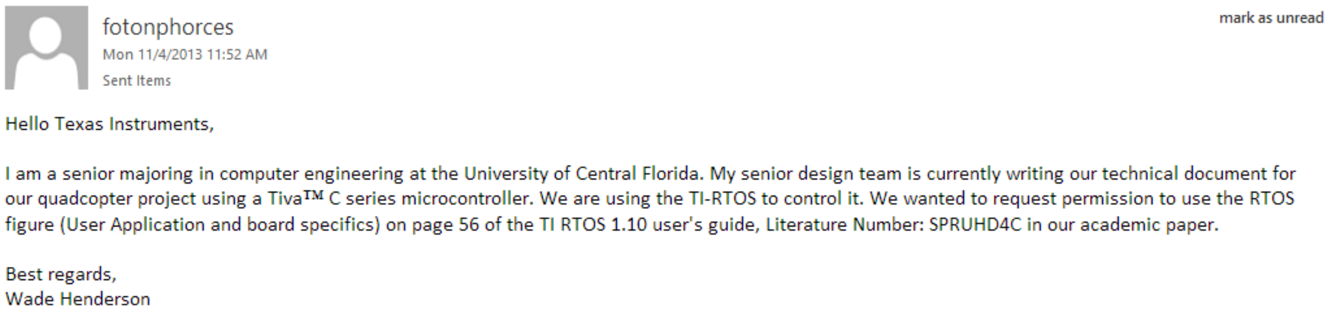
Actual text:

Hi Wade,
Thank you for your interest in Bosch Sensortec products. Please go ahead and use the flow chart example on pg 13 in your technical paper.
Also keep in mind that BMP085 is now EOL. The replacement device is BMP180 for your designs moving forward.

Thanks,
Afzal

Afzal Saeed
Senior Sales Manager NA
Bosch Sensortec
Email: afzal.saeed@bosch-sensortec.com
Cell: 847-867-3795
Visit us at: www.bosch-sensortec.com

TI RTOS figure request:



fotonphorces
Mon 11/4/2013 11:52 AM
Sent Items

mark as unread

Hello Texas Instruments,

I am a senior majoring in computer engineering at the University of Central Florida. My senior design team is currently writing our technical document for our quadcopter project using a Tiva™ C series microcontroller. We are using the TI-RTOS to control it. We wanted to request permission to use the RTOS figure (User Application and board specifics) on page 56 of the TI RTOS 1.10 user's guide, Literature Number: SPRUHD4C in our academic paper.

Best regards,
Wade Henderson

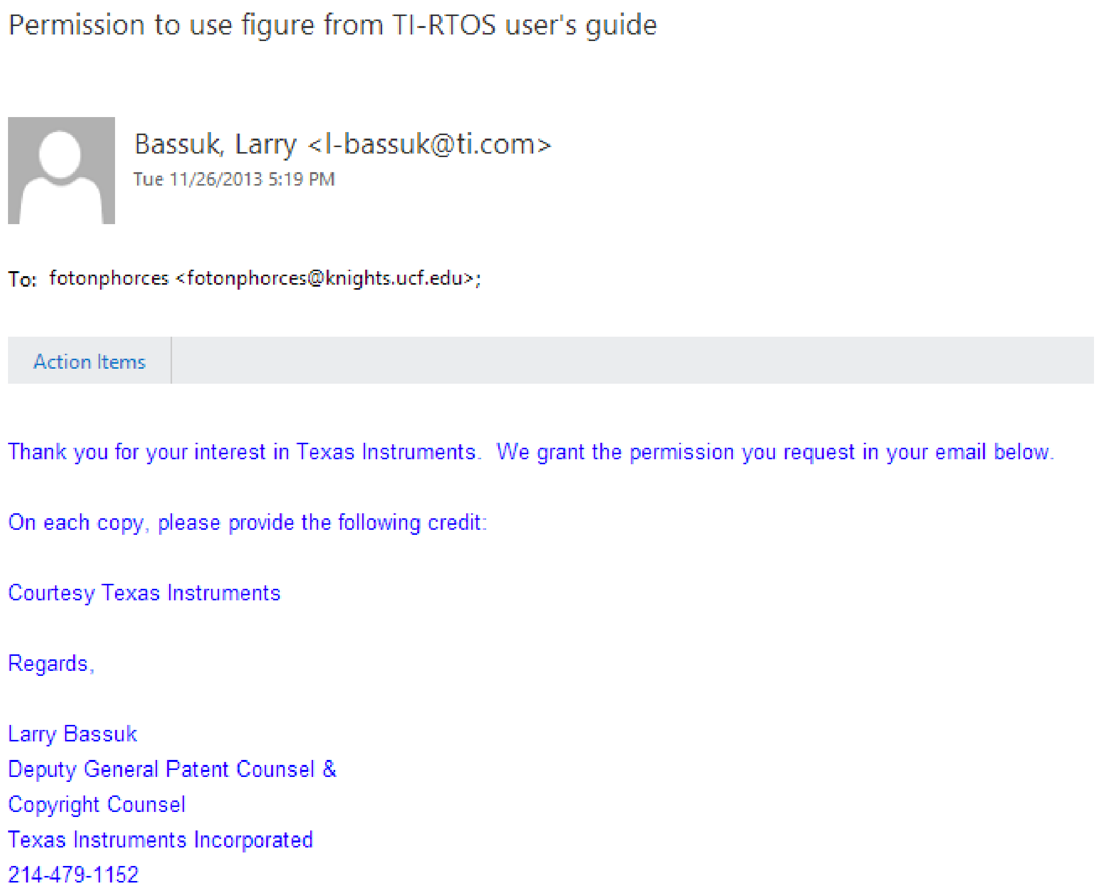
Actual text:

Hello Texas Instruments,

I am a senior majoring in computer engineering at the University of Central Florida. My senior design team is currently writing our technical document for our quadcopter project using a Tiva™ C series microcontroller. We are using the TI-RTOS to control it. We wanted to request permission to use the RTOS figure (User Application and board specifics) on page 56 of the TI RTOS 1.10 user's guide, Literature Number: SPRUHD4C in our academic paper.

Best regards,
Wade Henderson

Response:



Permission to use figure from TI-RTOS user's guide

Bassuk, Larry <l-bassuk@ti.com>
Tue 11/26/2013 5:19 PM

To: fotonphorces <fotonphorces@knights.ucf.edu>;

Action Items

Thank you for your interest in Texas Instruments. We grant the permission you request in your email below.

On each copy, please provide the following credit:

Courtesy Texas Instruments

Regards,

Larry Bassuk
Deputy General Patent Counsel &
Copyright Counsel
Texas Instruments Incorporated
214-479-1152

Actual text:

Thank you for your interest in Texas Instruments. We grant the permission you request in your email below.

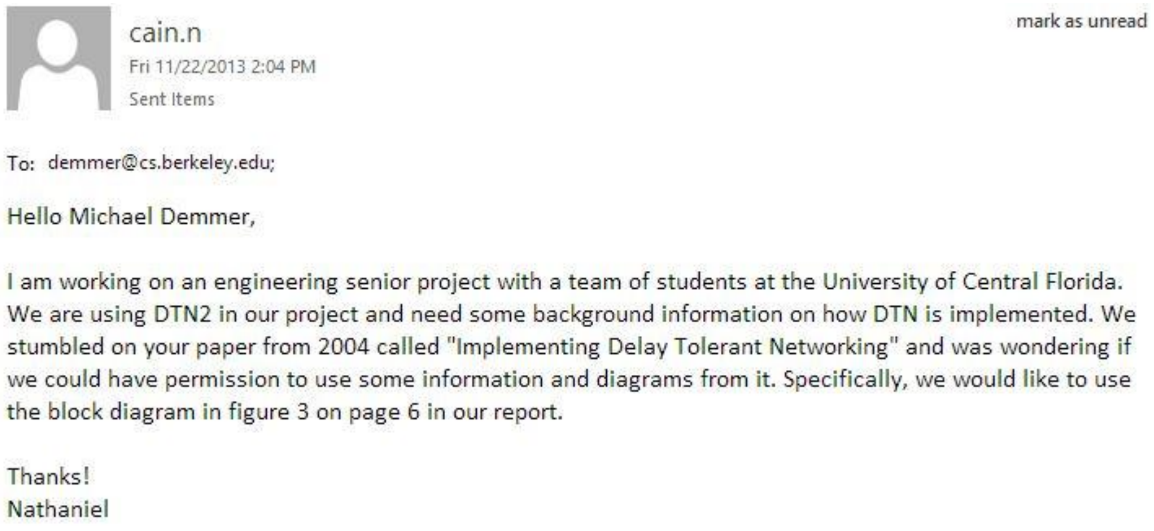
On each copy, please provide the following credit:

Courtesy Texas Instruments

Regards,

Larry Bassuk
Deputy General Patent Counsel &
Copyright Counsel
Texas Instruments Incorporated
214-479-1152

DTN figure request:



cain.n
Fri 11/22/2013 2:04 PM
Sent Items

mark as unread

To: demmer@cs.berkeley.edu;

Hello Michael Demmer,

I am working on an engineering senior project with a team of students at the University of Central Florida. We are using DTN2 in our project and need some background information on how DTN is implemented. We stumbled on your paper from 2004 called "Implementing Delay Tolerant Networking" and was wondering if we could have permission to use some information and diagrams from it. Specifically, we would like to use the block diagram in figure 3 on page 6 in our report.

Thanks!
Nathaniel

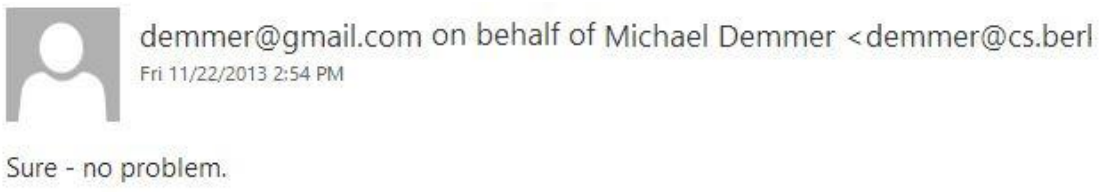
Actual Text:

Hello Michael Demmer,

I am working on an engineering senior project with a team of students at the University of Central Florida. We are using DTN2 in our project and need some background information on how DTN is implemented. We stumbled on your paper from 2004 called "Implementing Delay Tolerant Networking" and was wondering if we could have permission to use some information and diagrams from it. Specifically, we would like to use the block diagram in figure 3 on page 6 in our report.

Thanks!
Nathaniel

Response:



demmer@gmail.com on behalf of Michael Demmer <demmer@cs.berl
Fri 11/22/2013 2:54 PM

Sure - no problem.

Actual Text:

Sure-no problem.

Appendix C – Datasheets Referenced

1.) http://www.ti.com/tool/ek-tm4c123gxl

2.) http://www.ti.com/product/tm4c123gh6pm

3.) http://www.raspberrypi.org/technical-help-and-resource-documents

4.) http://www.hobbyking.com/hobbyking/store/__43709__Afro_ESC_20Amp_Multi_rotor_Motor_Speed_Controller_SimonK_Firmware_.html

5.) http://www.hobbyking.com/hobbyking/store/__25081__NTM_Prop_Drive_Series_28_30S_900kv_270w_short_shaft_version_.html

6.) http://www.hobbyking.com/hobbyking/store/__46308__8045_SF_Props_2pc_Standard_Rotation_2_pc_RH_Rotation_Red_USA_Warehouse_.html

7.) http://www.hobbyking.com/hobbyking/store/__24172__Q450_Glass_Fiber_Quadcopter_Frame_450mm.html

8.) http://www.hobbyking.com/hobbyking/store/__20791__Turnigy_nano_tech_5000mah_4S_45_90C_Lipo_Pack_USA_Warehouse_.html

9.) http://www.hobbyking.com/hobbyking/store/__8991__Turnigy_9X_9Ch_Transmitter_w_Module_8ch_Receiver_Mode_1_v2_Firmware_.html

10.) http://www.hobbyking.com/hobbyking/store/uh_viewitem.asp?idproduct=25483&aff=588847

11.) http://www.amazon.com/Raspberry-5MP-Camera-Board-Module/dp/B00E1GGE40/ref=wl_it_dp_o_pd_S_nC?ie=UTF8&colid=3AHXZEFR7P8ZM&coliid=I2D9TFB0V615PM

12.) https://www.sparkfun.com/products/10724

13.) https://www.sparkfun.com/products/11571

14.) https://www.sparkfun.com/datasheets/Components/General/BMP085_Flyer_Rev.0.2_March2008.pdf