Autonomous Quadrotor Project: Anzhelka

ALPHA DRAFT

Cody Lewis
srlm@anzhelka.com

Luke De Ruyter
ilukester@anzhelka.com

April 23, 2012

And when he [Herod] had apprehended him [Peter], he put him in prison, and delivered him to four quaternions of soldiers to keep him; intending after Easter to bring him forth to the people.
–Acts 12:14, King James Bible, Cambridge Edition

Revions Here is where you will place the revisions of this document.

9:55pm - All of the spelling was checked and fixed for the first 2 secitions writen by Luke.

Table of Contents

1 Introduction

1.1 Executive Summary

This senior design project's goal was to create an autonomous quadrotor that would be used for outdoor sports like mountain biking, snowboarding, etc. The quadrotor would have a video camera mounted onto a gimble that would always point at a subject/object in which the tracking device was placed. The tracking device would have its' own GPS and IMU in order to be able to determine the location and heading of the subject/object.

Our quadrotor has many features including: 3-Axis Gyro, 3-Axis Accelerometer, 3-Axis Compass, Voltage and Current monitoring of each motor, a separate battery for logic operations for emergency recoveries, ability to control up to 8 Servos, able to track the exact speed and position of each motor, monitor 8 additional analog inputs, and mounting holes for all components and future upgrades.

This senior design project was built around being Open. Open Source. Open Hardware. The frame that was used for the quadrotor was designed by Ken Gracey, an employee at Parallax, and is called the "Elev-8". There is a Git repository hosted by Google Code in which you can find all the information used and created during this senior design project. There is also a Blog which goes into some of the extensive detail that was put into this project. The main processing board for this quad rotor features a Parallax Propeller, overclocked to 100Mhz, and a fully custom board that measures in at 4inches by 3inches.

Testing something that is so dynamic in nature is a great challenge in its own. One of the test that has to be done in order to calculate some of the gains for the control algorithms is the motor's torque and thrust. This meant that there had to be a test stand that would test both of these constants. Another test that had to be completed was on the control board that was designed for the quadrotor.

The project was started without any expertise or experiance with autonomous flying or quadrotors, but with will to learn, create and develop a system that could even be understood by people with experience in these fields. Software is one of the groups expertise. Both team members have had years of experience in multiple programming languages. Hardware on the other hand is a bit more difficult. Only Luke had any real experience with designing PCB's and circuits, however nothing to the extent of this project. The team believed that this project was worth aprroximately 16 credits. This was determined by the amount of hours that were being put into the project each week. Approximately 30 hours per person per week were devoted to this project.

NEEDS ACHIEVEMENTS

1.2 Design Objectives and System Overview

This project was designed to be open ended and to be used in an unprecedented amount of ways. After watching countless videos from Go Pro cameras from a single perspectives and partial views of the subject, project Anzhelka was created. Anzhelka would allow users to be able to capture video angles that were once unattainable without costing thousands of dollars. Anzhelka would allow users to capture video with the same ease as using a Go Pro camera, but without the single perspectives and gitter from traditional methods.

The quadrotor will have at least a 15minute run time, with the abilitly to carry a 2 pound payload. It could also be controlled by a human from up to a mile away with line of sight. The control loop to keep the platform stable will run at 300Hz and be asynchronous.

On this project the team members of Anzhelka decided to keep responsibilities as open as possible, however SRLM did most of the coding and wood working while ILUKESTER did most of the embedded hardware and wiring. Everything else was either split evenly or worked on jointly.

1.3 Background and Prior Art

There have been several different renditions of this project but to the best of the teams knowledge there has not been a project that has contained the 3 main parts of this project. Autonomous Quadrotor, Object tracking, and Open.

1.4 Development Environment and Tools

The Anzhelka project software was developed for the most part using a Linux system. All code and other project files are hosted on the project Git repository (code.anzhelka.com). The Propeller code was written in Gedit, compiled and downloaded with the BSTC compiler, and interfaced with using picocom.

1.4.1 Directory Structure

The project uses the following directory structure:

/hardware /frame /pcb /software /spin /src /lib /test /tool /config /java /src /lib /test /tool /config /doc /datasheet /reports /figures /notes /tests /extra /extra

Hardware Subfolders will store the major components of the project. For example, the frame has several .dxf files that are sent to the laser cutter, so that will all go into a subfolder called frame. The project may have several PCBs made as well, and so each should go into a subfolder under pcb.

Software The software is separated by language into separate folders. This makes sense because each processor in the project will have only one language running, but separate processors that are running the same language may share components (library files, for example). Each language has a number of subfolders:

* src is where the source code for the project is stored. Subfolders as appropriate.

* lib stores all general purpose library files (code) such as Propeller Obex objects.

* test stores the test harnesses such as unit tests and Spin code to test a particular module (the latter case would have a 'main' type method and would be self supporting when running on the Propeller).

* tool holds all the relevant development tools for that language (bstc for Spin, for example).

* config stores any sort of relevant compile time or testing configuration files.

The files that are in the Software folder should be used only for what runs onboard the quadrotor. Test programs or desktop PC client programs should instead go into the Extra folder. Note that these programs may still access the lib and tool subfolders in the software directory.

Documentation This folder stores all the relevant datasheets in the datasheet subdirectory, and any other project documentation that is deemed to fit. Note that most documentation probably belongs in the anzhelka wiki.

The datasheet and reports folder contain the reference datasheets for each component and the various generated reports of the project, respectively. The figures folder holds an images that is used in the documentation. The notes folders holds papers that are interesting and relevant to the project, such as the cited research papers. The tests folder holds the test results data, and any associated data processing scripts. The extra folder holds other documentation material such as project logos and fonts.

Extra This folder contains other associated programs for the project. Since the Software folder is dedicated exclusively to software that is intended to fly the quadrotor other programs need to be stored in the Extra folder. This folder stores the Anzhelka Terminal files and the Thrust/Torque test stand files for example.

All code is compiled with the BSTC compiler. This is available in [REFERENCE HERE]. This compiler was selected because it is Linux compatible and it has several optimizations over the Parallax provided compiler. In addition, it has a command line interface which makes it easy to integrate into a compilation cycle script. BSTC will also download the compiled code via the USB COM port to the Propeller.

1.4.2 Development Cycle

To facilitate the development cycle a simple compilation script was developed. A template form of this script can be found in $/software/spin/tool/bst_temp late.sh$. This script will compile the Spin program, and if no errors are found it will attempt to download to the Propeller. If successful it will open picocom (terminal

program) on the USB port to listen for data. Using this script while programming dramatically decreases the write-compile-test debug cycle.

All code was written in Gedit. Gedit is a simple text editor with a few features built in, including syntax highlighting. For most of the languages in this project, Gedit has provided suitable syntax highlighting. Propeller Spin, however, is not available by default. Syntax highlighting was accomplished by writing a language definition file ($/software/spin/tool/spin.lang$) and placing it in $/usr/share/gtksourceview-3.0/language-specs$. Gedit will then automatically highlight the Spin files.

All project files are hosted on the project Git repository, hosted by Google Code at code.anzhelka.com. A Git repository was selected so that all developers would have equal access to the source files, changes would be logged and trackable, issues would be trackable, and so that concurrent work on files could be easily merged. All project files are open source under the MIT license and can be downloaded freely.

1.5 Definitions and Acronyms

Needs lots more of deffinitions

ESC	Electronics Speed Control
IMU	Inertial Messurement Unit
PWM	Pulse Width Modulation
PCB	Printed Circuit Board
RISC	Reduced Instruction Set Computing
PASM	Propeller Assembly
RAM	Random Access Memroy
GUI	Graphical User Interface
AT	Anzhelka Terminal
$ADxxx	Anzhelka Terminal Data Type
OOP	Object Orianted Programming
EEPROM	Electrically Erasable Programmable Read-Only Memory
PID	Proportional-Integral-Derviavtive

2 Requirements Specifications

This section describes issues that need to be addressed or resolved prior or while completing the desgn as well as issues that may influence the design process.

2.1 Assumptions

Have you ever mounted a camera on to your helmet and rode down a mountainous trail? What about trying to capture yourself while water skiing? Watching the video usually turns out shaky and in one perspective. Would it be nice to be able to see what you did wrong that caused you to fall of your bike? Most of the time you can't see what went wrong. What if you could do all of that

while still capturing amazing views? All this can be accomplished while being as simple as powering on a couple of devices.

2.2 Realistic Constraints

Every system has constraints and Anzhelka is no exception. Our quadrotor can only travel so fast, about 10 mph, therefore it wouldn't be able to keep up with anything much faster than that. Large gust of wind could cause problems with keeping the quadrotor platform stable enough to keep the camera from shaking. Wet weather would be large problem and would cause unknown and uncalculated problems.

2.3 System Environment and External Interfaces

To be able to accomplish all of these tasks there is a lot of interfacing between many different devices. Our main control board must control all 4 ESC's, communicate with the IMU via I2C, servos must be controlled via PWM signals, voltage and current of each motor is monitored via a specially designed circuit.

2.4 Budget and Cost Analysis

Unfortunately there was no money that was given to the team in order to support the project. All of the funding had come from the team members own personal accounts. Below is an exert from our spreadsheet with the cost analysis and money spent on this project.

INSERT COST TABLE HERE

Update table before adding

2.5 Safety

When dealing with any autonomous system one must take extreme cautions in order to insure the safety of everyone. Autonomous systems are dangerous because there is no one behind the controls of the system and can become unpredictable in the event of a system failure.

During the design process of the frame the team made sure to use proper spec fasteners, washers, and nuts. Also, all threaded components were secured using blue lock tight to ensure that nothing would loosen on its own.

Whenever flying an aerial vehicle be sure to wear safety glasses to protect your eyes in the event that the propeller has a failure and is detached/released from the motor(s).

2.6 Importance of Team Work

Being able to work in a team is both a skill and a challenge. Working on a project in a group helps you split up the work load and possibly get more work

done in less time, however being able to work together with others on the project could present a greater challenge than the project itself.

This was a foreseen challenge and the team set up a Git repository for all code, data, images, and presentations. There was also an official blog set up where we could go in great detail on what we were working on and we had yet to complete. With these two resources set up and with the help of keeping an open schedule the team has come to realize how important team work really is.

3 System Design

System Designthis level of design includes the entire system, including the people and processes involved, and not just the software architecture. Cody

3.1 Experiment Design

3.2 Experiment Results and Feasibility

4 Software Introduction

4.1 Propeller

4.1.1 Architecture

The Anzhelka project uses the Propeller P8X32A microcontroller from Parallax. This microcontroller is a 8 core 32 bit RISC processor. The Propeller is $8 per chip, plus approximately $2 for support components. For this project the Propeller has been overclocked from the default 80MHz to 100MHz. This additional speed facilitates more complex computations without sacrificing output rate. Each core, called a COG, is identical with equal access to all chip resources. The Propeller has a central RAM area called the HUB which is COG accessible in a round robin fashion.

The Propeller is distinctly different from most other microcontrollers by it's lack of built in hardware. The Propeller does not have any hardware level serial ports, analog to digital or digital to analog ports, or any pulse width modulation ports. Instead, the Propeller is designed to be able to use software for these common interfaces. This is typically done through what is known as bit banging. The only exception is the built in video generation hardware that assists in creating NSTC, PAL, or VGA signals. To make development easier for the programmer, Parallax hosts a source code website that provides code for common tasks such as serial or PWM.

The Propeller has two built in languages: a high level language called Spin and the assembly language called PASM. PASM is executed directly by a COG. Spin is executed by a built in PASM Spin interpreter that can be dynamically loaded into one or more COGs. Other high level languages available for the

Propeller generally operate in a similar manner to Spin. This project uses Spin and PASM exclusively.

The Propeller has three different memory locations: 2KB of COG RAM, 32KB of HUB RAM, and external 64KB of I2C EEPROM. Upon startup, the Propeller copies the contents of the EEPROM to HUB RAM, loads a Spin interpreter into COG 0, and begins execution. Any PASM code, including the Spin interpreter, must fit in 496 instructions or less in order to fit into the COG RAM. The Propeller does not have provisions for fetching PASM instructions from other locations besides COG RAM. For Spin code the compiled interpretable bytes are stored in the HUB RAM, and are fetched and decoded by the Spin interpreter.

4.1.2 Programming

The Propeller is programmed in PASM and a high level language called Spin. In general, PASM is used when speed is required. At 100MHz, the Propeller can execute 25,000,000 assembly instructions each second (each instruction takes 40ns). By contrast, the interpreted Spin language is about 100x slower. Because of this contrast Spin is used where ease of programming is important, and PASM is used where speed is important.

Spin is officially called an object oriented programming language, but there are some subtle differences compared to mainstream object oriented terminology. Spin OOP is used to organize code into logical blocks much like an import statement in C++ or Python. Spin objects do not use techniques such as class instances, inheritance, subtype polymorphism. A Spin object is used to group related functions together into a unit that can be included in multiple Spin programs. Typically, Spin objects are used for interfacing with external devices. For example, a typical Spin program might have an object for serial communication, an object for VGA signal generation, and an object for I2C communication. The Parallax Object Exchange (obex.parallax.com) hosts Parallax written objects and community written objects under the open source MIT licence.

Spin syntax is very similar to Python. To denote a new scope Spin uses indentation instead of curly braces (familiar to C/C++ and Java programmers). Spin code is divided into blocks: CON (constant), VAR (variable), OBJ (object), DAT (data), PUB (public function), and PRI (private function). Most typical programming constructs are a part of the Spin language: conditional IF statements, FOR loops (called REPEAT), boolean conditions, and so on.

All variables in the Propeller are integers. Variables are typically 32 bit signed integers called longs, but in Spin it is possible to create 16 bit or 8 bit sized variables as well (called words and bytes, respectively). Global variables are declared in the VAR block, and global constants are declared in the CON block. PRI and PUB functions can declare local variables, along with function parameters.

4.2 Other Software Stuff

Cody... ie things such as the python and the possibilty of future expansion (BeagleBoard)

4.3 Rationale and Alternatives

5 Quadrotor Prototype Construction

5.1 Hardware

Luke

5.1.1 Frame

The frame is constucted out of a material called Derilin made by DuPont^{TM}

5.1.2 Motors/Propellers

5.1.3 PCBs

6 Constant Identification

6.1 Thrust Torque Test Stand

A good autonomous quadrotor needs to be able to measure, in real units such as kilograms and seconds, important aspects about itself such as orientation, motor thrust, acceleration, and so on. It's fairly easy to make a remote controlled quadrotor platform since the human in the loop can intuitively correct for many small errors, and our eyes are very good at collecting the necessary raw information. An autonomous quadrotor does not have this luxury, and must explicitly define each kinematic and dynamic equation. Among others, the quadrotor must know the propeller torque and thrust constants.

[EQUATIONS HERE]

Above, we have the two equations that define how the propellers affect our quadrotor system. The form that they are in now makes it convenient for us to measure the constants K_T and K_Q: if we can somehow measure the terms on the right hand side then we can figure out what the constants are.

From these equations, it is clear that we need to measure thrust, torque, air density, rotation speed, and the propeller diameter. Measuring rotor diameter, motor speed and air density are straight forward, and so they are not covered here. The real challenge comes from measuring motor thrust and torque.

Thrust is measured by mounting the motor on the end of the lever arm, then measuring the torque that the propeller exerts as it spins. A pressure sensor rigidly mounted between the lever arm and a stationary base can measure this torque.

Calculating torque is a bit more complicated: the motor body needs to be mounted on a rotating axis that is directly in line with the motor shaft, and the torque along this axis needs to be measured. Most measurement test stands seem to only measure thrust, and ignore yaw. We, however, rely on the torque to yaw the quadrotor vehicle.

To measure torque, our test stand has the motor mounted to a rod, which then has a lever arm attached that presses on a scale. In a same way as thrust the force pressing on the scale can be read, and with the length of the lever arm torque can be calculated.

Our test can measure thrust and torque simultaneously and automatically. To do this we are using the Flexiforce pressure sensors ([REFERENCE]). These sensors vary the resistance based on the amount of pressure, and resistance is very easy to measure with a microcontroller.

For motor speed we will be using a Eagle Tree brushless RPM sensors ([REFERENCE]). Our main control board is the quadpower board that we have developed for our quadrotor. This has the advantage of being identical to what we will be flying, it will have the motor current and voltage sensing built in.

7 Implementation Details

7.1 Anzhelka Data and Command Exchange Protocol

7.1.1 Introduction

The Anzhelka project uses a protocol called $ATXXX to facilitate data and command exchange between different subsystems. $ATXXX is very similar to NMEA-0183 where data is exchanged via sentences prefaced with a sentence code. This allows for the Propeller to send real time information about the running state to other onboard processors or to offboard computers, and to receive commands. The protocol was developed to facilitate predictable and reliable exchanges.

7.1.2 Implementation Details

All strings are encoded in standard ASCII. Numbers are converted to their ASCII equivalent and are in base 10 unless otherwise noted. If a number is decimal then it will have the ASCII decimal point included in the sentence string. By having all the data be in ASCII a normal terminal program (such as picocom or cutecom) can be used to receive and send data.

Each string is independent of all other strings. There is no order required to send or receive strings. If multiple strings of the same type are received, the listening devices should assume that the last received is the most recent. If an unknown string or other serial data is encountered then it should be ignored.

Data strings are prefaced with $ADxxx (short for Anzhelka Data type xxx).

Note that the only defined whitespace in a string is a single space after the sentence code, and a newline (ASCII characters LF and CR) after each string.

For clarity of notation the newline is not written in the string lists below.

7.1.3 Example Usage

This protocol is used in the thrust/torque test stand. The test stand measures the thrust and torque using force sensors, and measures the rotations per second of the motor. The test stand then outputs the $ADRPS, $ADPWM, $ADMTH, and $ADMTQ strings to a USB serial port. A desktop computer reads these strings from the serial port and displays the variable on-screen in a GUI that updates in real time as new strings are received. In this way the user can watch as testing proceeds.

This protocol could also be used to log parameters of the quadrotor as it is flying. The Propeller that is controlling the quadrotor could send these strings via a serial port to an external SD card. If an SD card is not available then the system could perhaps use a wireless transceiver such as an XBee. In the event of a system failure, the developer could look at the logged strings in order to help with debugging.

7.1.4 Protocol String Formats

$ADRPS	Most recent rotations per second for each motor Format: $ADRPS m1,m2,m3,m4
$ADMIA	Most recent motor current in milliamps Format: $ADMIA m1,m2,m3,m4

$ADMVV Most recent motor voltage in millivolts Format: $ADMVV m1,m2,m3,m4
$ADPWM Most recent motor PWM command, in microseconds (uS)
Format:
$ADPWM m1,m2,m3,m4
$ADMKP Current motor PID loop proportional constant (KP). No units.
Format:
$ADMKP m1,m2,m3,m4
$ADMKI Current motor PID loop integral constant (KI). No units.
Format:
$ADMKI m1,m2,m3,m4
$ADMKD Current motor PID loop derivative constant (KD). No units.
Format:
$ADMKD m1,m2,m3,m4
$ADMTH Most recent motor thrust in units of Newtons.
Format:
$ADMTH m1,m2,m3,m4
$ADMTQ Most recent motor torque in units of Newton-Meters.
Format:
$ADMTQ m1,m2,m3,m4

$ADDRP

Most recent motor rotations per second set point. This is the speed that motors are trying to achieve, and is fed into the motor PID loops.

Format:

$ADDRP m1,m2,m3,m4

7.2 High Level Hardware Design

Luke

7.3 High Level Software Design

Cody

8 User Interface

8.1 Anzhelka Terminal

The Anzhelka project uses a PC based GUI platform called Anzhelka Terminal (AT) to display realtime system states and to allow for parameter tuning. AT is written in Python and uses the WxWidgets Python branch WxPython for the GUI. AT is cross platform.

The AT GUI was written in WxPython and developed in part with the WYSIWYG editor WxGlade. WxGlade was suitable for most of the general layout tools, but was insufficient in two areas. First, WxGlade could not specifically layout the dynamic graphs since the graph class is a non standard class. Secondly, WxGlade was not used to layout the motor parameter table. This table has a row for each motor (normally 4 motors, but possibly 3 or more) and a column for each parameter of interest. Since either number should be easily changeable without excessive GUI editing this section was hand coded.

AT will listen on the specified serial port for $ATXXX strings, and will make these strings available for reading. Within the program, different threads will search the received strings list and extract the strings of interest, then display them onscreen. If an unknown string is encountered then AT will ignore it.

There are two GUI screens that are currently implemented: the COM port setup tab and the motor variable tab. Additional screens planned for the future include a inertial measurements tab, a control parameters tuning tab, and a high level command tab.

The motor control tab has several features that are important. At the top of the pane is a dynamic graph that will display in real time it's received information. This feature was one of the deciding factors to use Python instead of some other language. The graph can be automatically or user scaled on either axis, and is updated behind the scenes with the received serial strings.

Below the graph is the motor parameter table. This table displays all relevant information about each motor, including: motor number rotations per second desired rotations per second voltage current electronic speed controller

PWM command thrust torque K_P of the motor PID controller K_I of the motor PID controller K_D of the motor PID controller The table is updated in real time based on the received strings. Some fields are editable as well (such as K_{PID} and DRPM), and when changed AT will send these values through the serial port as a $ATXXX string (Note: this feature is still in development). This table is generated at runtime so it is easy to change the number of motors and the motor parameters. This was done in particular to support the same AT program for hexrotors and octorotors as well as quadrotors.

9 Testing

TestingYou are expected to have a detailed test plan, including Unit testing Integration testing Acceptance testing If you are unfamiliar with these terms, look them up. Search engines are marvelous inventions.

9.1 Unit Testing

9.2 Integration Testing

9.3 Acceptance Testing

9.4 Hardware Testing

Because Hardware is a physical entity it is subjected to many more points of failure. Parts can be flawed from the factory, assembled wrongly, or miscalculated.

9.4.1 Frame

The frame is ridged and is made up from nearly the same exact parts for all 4 booms. The center of the frame is the only thing that is not duplicated, however the top side and bottom side are nearly identical with minor mounting differentces. Each boom needs to be identical in both size and weight to insure proper balancing.

Once each boom is assembled it should weigh in within an ounce of other booms. If there is to much of a differnce in the weight of the boom this could cause the center of mass to change to dramtically. Therefore the booms would have to be matched.

9.4.2 Motors/Propellers

The motors and propellers would also need to be tested and matched to insure the best performance. We can assemble all of our motors and propellers and then we can test them for torque and thrust ratings. Once we have all of the results we can then match the motors with nearly the same characteristics together.

One slight nick in a propeller can cause a large inbalancing problem within the propeller that could make it unmatchable. If this is the case we would have to throw away the propeller and begin testing with another.

We know that most motors made by the same manufacture with the same specifications should be nearly identical. From this we can assume that all motors are same and can be treated as such.

9.4.3 PCBs

PCBs can have many points of failure and need to be tested at many different parts of the build phase to insure the most amount of yeild. The PCBs must be tested before they are even produced.

Each fabrications house has its own set of limitations on what it can produce as far as trace widths, via sizes, spacing, etc. All fabrication houses with provide you with these details before you place an order so that you know what you can and what you can't do in your design. You must also keep in mind the amount of current or power that will have to travel through your traces that you have designed onto your board. If you are trying to push 10 amps through a 6mil trace you are going to blow that trace and ruin your board. Once you have sent off your design to the fabrication house to be produced, the product must then be electrially tested.

Electrical testing is a key factor in product yeild. What is electrial testing? Electrial testing is a process in testing that a pad that is supposed to be electrially connected to another pad is actually connected. To some this sounds quite simple, however in practice boardsc can have thousands of points that need to be tested. Some fabrication houses will provided 50% to 100% electrial testing, however some will not provide testing.

In the event that you cant get your boards fully tested you will need to create an overlay that will touch all the pads and check to see what is connected and what is not. In practice this is not acceptable to be done outside of the fabrication house or a special electrial testing board house.

The physical components that will be soldered onto the PCB have usually been tested and have an acceptable failure rate that wont have a great effect on the yeild. With that being said, placement of the components is essential. If a component is misplaced by 20mils this could cause a bridge between two pins causing a short deaming that board unusable.

Once the PCB has been fully populated it now must be tested to insure that all the components are in working order. One of the easiest ways to accomplish this is to develop test code that will run on the hardware and will give you a set of known outputs. Having something that will test all components several times under different inputs and outputs is key to insure that the PCB is in good working order.

10 Maintenance Plan

You are expected to produce a maintenance plan, which states how you plan to keep the software solution current with future environment changes. Here, we assume that you will maintain your project following the 10 week course period, even though we know that this is not really the case.

10.1 For the next ten weeks

This senior design project is no where it needs to be as for a finished product and will require a lot more time to get there. In the following weeks we hope to acomplish the following tasks.

Write the code for the test stand
Write motor PID object
Calibrate the pressure sensors for the test stand
Characterizing the Motors and Propellers
Create Eagle Tree wiring harnesses
Finialize mechanical assembly
Mount the IMU onto the quadrotor
Write the IMU interface object
Setup wireless interfacing
Balancing of the propellers
Build Roll and Pitch test stand
Write Motor control object
Finish all electrial wirring for the quadrotor
Write attitude control object
Build yaw test stand
Write altitude control object
Test for physical limit of payload capacity

These are just some of the many items that we hope to complete in the following ten weeks.

10.2 For the next year

Since the members of this project are so financially invested into it, we plan on contuining this project for a very long time. It is quite possiable that we may even turn this project into company/business.

One of the hopes is to present the project to the investor community(Kickstarter, IndieGoGo, etc.) and to see if there is enough interest in the project to have them help fund this idea to its full potential.

With the right funding we could add things like object advoidance, quadrotor acrobatics, and user generated ideas.

11 Engineering Effort and Societal Impacts

Engineering Effort and Societal ImpactsEach project must consider realistic constraints on time and money, and should consider safety, reliability, aesthetics, ethics, and other possible social impacts. Produce a short (1-2 page) statement that describes how your group has addressed these key issue.

Essay time!!!

11.1 Project Management

11.2 Requirements Traceability Matrix

11.3 Packaging and Installation Issues

11.4 Design Metrics to be used

11.5 Restrictions, Limitations, and Constraints

12 Conclusions

ConclusionsAssume that you write this just before handing in your document. Summarize briey what you did for you project. Address the following; what most surprised you about the process of creating the project. What would you do differently if you had to do it again?

13 References

ReferencesEvery, book, paper, webpage you used must be cited in a standard format. You could use the American Psychological format [1, 2], or the IEEE standard [3], or any other format so long as you are consistent and complete. You should also reference any standard template libraries used in your code. [1] Burgess, P., S. (1995). A Guide for Writing Research Papers based on Styles Recommended ! ! by The American Psychological Association. [2] Coppola, L. (2000). The APA Citation Format. Rochester Institute of Technology, Wallace [3] Institute of Electrical and Electronics Engineers (2000). Computer science style guide.

14 Appendices

AppendicesInclude the following: Source code printed in the 2 pages per page format. A printed copy of the slides used for your nal presentation. If you use ! ! animation in your slides, then you need to sanitize the slides so that they are readable when printed.. A

professional quality one-page resume for each member of the group. Use the ! ! ! same template for each resume.

15 Acknowledgements

AcknowledgementsThank anyone who helped your group with the project.