UW BADGER Lab Quadcopter Exploratory Development

Prepared by: Jason Dekarske 6/28/2018

# Abstract

A quadcopter is a study of the challenge of digital controls and its demand for well understood physics. Each component must work together flawlessly, or the consequences could be catastrophic. The goal of this project was to construct a custom 4 rotor robot capable of following high level commands from a command station using position input from the BADGER Lab’s motion capture system. The motivation is to create a testbed for student development in ME 349: Introduction to Robotics. We sourced inexpensive parts with the intention of creating more than one unit for the use in the course and to control for broken devices. This document outlines the progress made toward this goal and the things the author has learned.

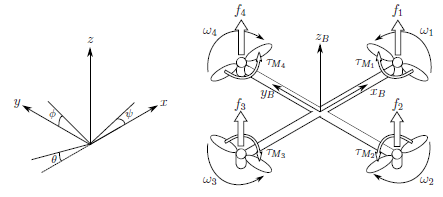
# Note

This writing is organized by quad theory, then component functional discovery and finally system integration and theory. It is important to note that most components may not be available exactly as described, but it should be possible to find a clone for similar price range. With that said, the greatest difficulty was sidestepping physical properties of components, as documentation was scarce, and proceeding with experimentation and trial and error. Hopefully the strategies laid out are helpful in recreating a functional device in the case an exact replacement cannot be found.

# Wait, this thing flies?

There are many examples in literature that outline quadcopter function at the PhD level with formality that is not necessarily important to get this thing in the air. Alternatively, hobby documentation of open source flight computers cannot be made to meld with the unique setup in the BADGER Lab. This is precisely the challenge presented here. Where quadcopter research labs have thousands to produce a robust, acrobatic system (see: coding and algorithms), an introductory course that spends a small unit on flight cannot compete. This will be touched later in design criteria and some choices of subsystems.

With basic understanding of Newton’s laws, we can construct a working device that will (hopefully) not destroy itself. From this base we can add toppings that improve the flavor and robustness to the system. Every write-up on quadcopters must include a form of the following nifty graphic (Figure 1). Notice the body frame is attached in line with the rotors. This simplifies the math greatly, since a movement in one direction is only dependent on one axis. Additionally, adjacent rotors spin in opposite directions to cancel torque; we will abuse this later.



Let’s get some equations rolling. First, we will model the quadcopter attached to a vertical pole: controlling only its altitude (Excuse my janky powerpoint shape free body diagrams please).

M1,M2

M3,M4

mg

**x**

**θ**

After accepting the brutal truth that we will not have a perfectly stable quadcopter we would find it’s thrust to be the following:

This means to accelerate up or down, we should modulate the force of the motors equally. Note the cosine terms include the angular correction for both horizontal axes. Not so tough, let’s look at keeping it stabilized. This time, we attach one axis to a horizontal pole (I actually did this, see later)

M1

M3

**x**

**θ**

In this plane, we are only looking at the rotational dynamics between opposite motors. We find the angular acceleration where is the inertia of each axis and is the length motor to motor (450mm):

Finally, we consider the final axis of the quadcopter: the yaw. Following a similar strategy, let’s only compare the relative forces of the motors.

M4

M2

M3

M1

As is the theme of this project, I was given no details of the physical properties of the propellers, so we need to take this part off-road a bit. We have intuition that rotating the M1,M3 pair faster than M2,M4 would result in the quad rotating clockwise (or negative Ψ) due to Newton’s second Law. If we assume the propellors are the same and everything is symmetrical, I conjecture:

In the coming sections we will revisit how these physical relations can be put into code!

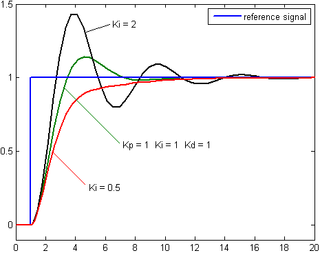
# I am in control!

Okay, so I’ve been pretty blunt about our lack of knowledge about our system. Surprise! Our engineering project isn’t ideal. The saving grace will be our controller, something that takes feedback from our sensors and adjusts accordingly. We’ll be using the popular PID scheme to correct errors. This requires an input, the motors, and an output, the position. This comes in two layers: stabilization and movement. Our inner layer, stabilization is priority number one. If we can keep the quadcopter stable, it ain’t goin nowhere. So, as our input, we command the angular position to be 0° for all axes. The controller will modulate the motors, as an output, to achieve this position with the help of the equations we developed earlier. Next, our outer later. Say we want to move our quadcopter 2 meters North (+y). Let’s command an angle ϕ = 2° and decrease to zero. This results in the quadcopter banking toward the north and stopping at 2 meters North. This all sounds good in theory, but let’s apply it.

I glossed over a crucial part of control in the last paragraph: PID. It stands for **P**roportional **I**ntegral **D**erivative, the three operations used in the algorithm. I’ll attack this in words and intuition first and equations after, feel free to switch between as you wish. These all work as a function of error, or the difference between the current input and the commanded input: , for example, where is the height of the quadcopter and is our desired height of the quadcopter. Looking at the proportional term, it will take the error and multiply it by a “gain”, often , and spit out an output, let’s say thrust. So, if we’re and want to be , with . Our output thrust is 4 which commands the motors to spin faster and ascend. A little hand-wavy, I know, we will get to how “thrust” interacts with the motors. As the quad approaches our desired height, the error will decrease and thus, the thrust. However, if the quad is moving too fast as it gets to the desired path it may overshoot! We are higher than the desired point and the thrust goes negative, and the motors slow down to descend. Let’s fix this with the **D**erivative term. As the name implies, we take the derivative of the error, , so we need to factor in a time variable. Since a computer is doing this math, we’ll have discrete sample points to calculate the error and time between each sample, perfect! Let’s find the difference in error between now and the last sample and divide by the time it took. How does this help us? If we are approaching our desired height too fast we will notice a large negative derivative term, because the error is decreasing quickly. We multiply this by a term and add it to the output. This decreases the output as we approach our desired point. Finally, we have the **I**ntegral term which is used for fixing steady state errors. The motors we have are quite sensitive, to stay within a ~cm of our setpoint, we need a low proportional term, or it will oscillate over and under the desired point. This is okay, but if the quad is very low, the term may not raise the output high enough to get to the desired point: a steady-state error. The integral term, as expected integrates, or adds, consecutive error between samples multiplies by the time between samples and you guessed it, , and tacks it on to the output. In summary, **P** fixes error, **I** keeps P under control, and **I** fixes outstanding error. That was a lot, let’s put it in maths and grapths.



Here we see the three familiar terms in the middle and a bunch of arrows pointing everywhere. r(t) is our desired point which stays the same, the summation circle subtracts the output, y(t) to give e(t), the error. Then, the PID terms do their thing and sum at the next circle to give our command to the motors, u(t). The motors move the quad, plant, and give us an output, position. Here are some examples of controllers with different gain values and how they interact. Note how red doesn’t overshoot, but it takes longer to reach the reference signal. These are the balances that must be made. Later we will work on finding what gains to use or “tuning”, this was the bulk of the time spent on the project.



# Design criteria

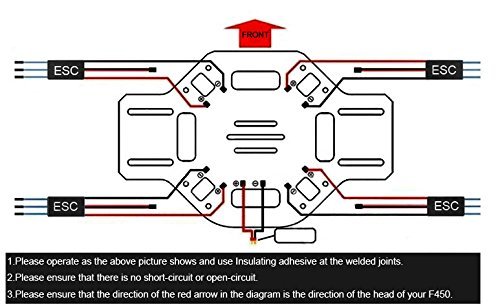
Because this was meant for a near graduate level course, it’s important to make the system accessible for tinkering and experimentation in the scope of robotics. The following criteria are in no way absolute for professional, experimental, or hobby quadcopter design. Starting with the tinkering theme, we need a microcontroller that can talk to the motion tracking system and the motors: an Arduino is a cheap solution that is very easy to use. Next, a large quad can claim slower dynamics by its increased mass and rotational inertia. Additionally, a larger footprint gives us a larger margin of error for the tracking markers because they can be placed further apart. Human sized wouldn’t fit in the lab so we’ll go with a small dog sized craft. Typically, these come in the flavor of a 450mm wingspan and a kilo or two. Consequently, we’ll need bigger motors powered by a bigger battery. Finally, we need communication between all our systems. I would have liked to use wifi for everything, but IT has the building on lockdown for router paraphernalia. We’ll make our own radio signals with a common 2.4GHz RF transceiver (it’s faster than wifi anyway!). I’ll breakdown each component in the following subsections.

## Frame

Let’s start with the part that looks pretty and holds things together. It took two iterations before I decided to use a tried and true frame. The first was some plexiglass screwed together in a not so balanced way, so I could get a feeling for where the components could go. Initially, I chose an “H” configuration. This setup allowed me to use the full length of the board for mounting and routing wires. This came at a cost of added weight and asymmetrical inertias. I was embarrassed by my poor craftsmanship, so I moved on to round two.

From a salvaged plexiglass sheet found in the design scraps, (SE corner ECB basement, seriously, go there, it’s a goldmine) I laser cut a beautiful four-piece frame with splines and everything. It was taller and had a two-tier mounting area. The lower tier contained the battery to lower the center of gravity while the upper tier held the controller. This time I went with the “X” configuration. I figured that if I didn’t assign which way was forward, it’s easier to play coy when it doesn’t cooperate. Also, having a single motor assigned to each axis with similar inertias makes the math much easier. Finally, since I was constructing a 3-dimensional device from 2-dimensional pieces, having perpendicular intersections made fitting the pieces easier. My beautiful artwork met its fate when a motor failed, and it dive bombed the hardened concrete floor. Tears were shed, but it was best I moved on to something guaranteed by Amazon to work.

Here is the “RipaFire®” 450 frame that I found on Amazon that is surprisingly still sold more than a year later. It is even made of “advanced engineering material”! I chose this in particular because it had a built in power distribution board. The bottom part of the frame is actually a PCB which routes power from the battery to the ESC’s (covered later). This saves me an extra part and consolidates wires, always a plus. Additionally, the motors fit the standard holes at the ends, the battery can fit in the lower portion, and I have plenty of room for a microcontroller on top.



## Motors

On to the star players! This is what really drove the decision making for the other components. Without a huge background in aeronautics, I was stumped for what to choose. I had to stay away from quality genuine motors to save the budget so generic ones from AliExpress had to do. I scoured forums of custom builds to see what parameters were typical for a quad of my size. I settled on a 920KV brushless motor. KV is the velocity constant of the motor – thousand RPM/Volt. <https://oscarliang.com/quadcopter-motor-propeller/> Generally, lower RPMs are suitable for larger propellers and quads. Since we have a rather large craft, I dutifully followed the table and stayed under 1000KV. When connecting to the ESC, the wires can connect in any combination; if the motor spins the wrong way, switch two of them.



## Propeller

I’m not sure this deserves a section with how little I spent researching. In hindsight, one of the most critical features did not see that much love. They certainly showed their displeasure by providing several casualties to my defenseless fingers. Following the table from above, I used a 1045 propellor which means 10” diameter and 45° pitch. I ordered a few packs figuring these would break often, I was, unfortunately, very correct. We visited the direction these should spin earlier, double check, as these are directional.



## Electronic Speed Controllers

What moves the motor you might ask? These convenient, drop and go, boards take PWM input from the controller and 12V from the battery, and transform to 3 phase output for the motors! Gorgeous! They just need a little programming. Most run with a standard firmware that is easy to change with an RC controller and a datasheet. The beauty of minimum spec designs meant I had neither! Fortunately, I could emulate a controller with the Arduino and after much experimentation with similar looking boards, I found a datasheet that kind of worked (See setup for details). The important parameter was finding an ESC with a high enough current rating. The only downside to choosing one with a higher rating meant a heavier piece. In this case, the weight of the ESC is negligible, so I went with a conservative 30A (compared to similar builds). Amazon gave me what I needed and a few extra after they burned out. I eventually found some that will cut the power if the motor stalls after getting the propellers caught under a desk. This and other features will be explored later. Note the extra is the output of the battery elimination circuit (BEC) which powers our microcontroller with a nice 5V.



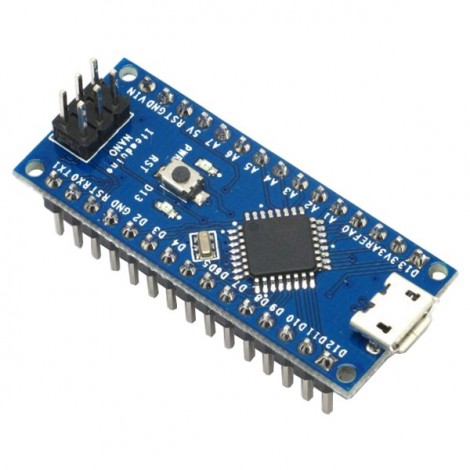
## Battery

Going down the line we see the heaviest chunk in the mix. The 4 cell LiPo was chosen based on reviews from Amazon and requirements for the motors capabilities. Out of all the parts, this is potentially the most dangerous. Starting Peter’s lab on fire was pretty high on the list of “do not’s”. I also grabbed a charger with a balancing feature. It’s crucial to note the cells should not be discharged lower than 3V. I tested the battery before and after sessions and when it was clear the motors weren’t working as hard as usual. The battery needed a discharge rate that would match our use. The “C” rating determines the nominal current available. With a 2.2 Ah capacity and 25C rating, our available current is 2.2\*25 = 55A. This means it should be tough to burn out our ESC’s at full power when the current is distributed to four motors.



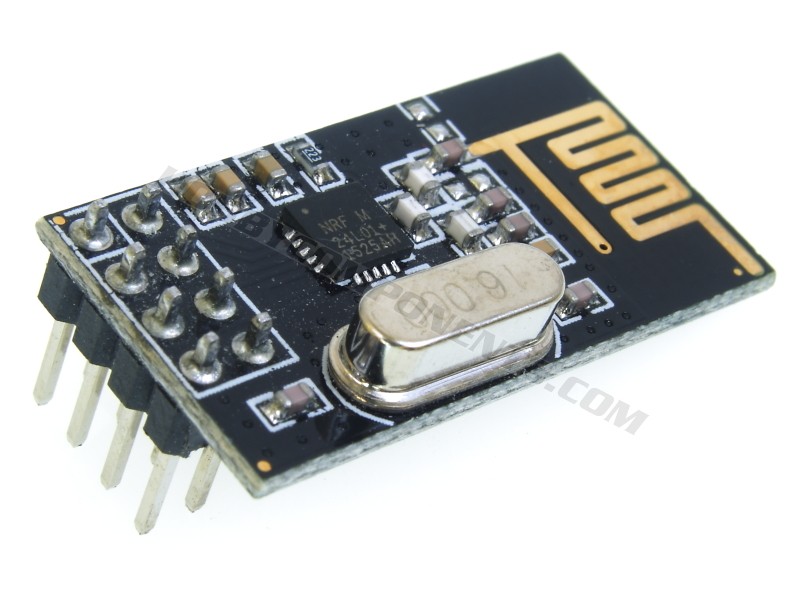
## Microcontroller

Being well versed in the language of DIY made this an easy choice. I needed 4 PWM channels, serial, SPI and I2C support, 5V power, and be on the small side. I already used an Arduino in tandem with the NRF24L01 (next section!) so implementation would be quick, or so I thought. I went with a clone of an Arduino Nano, fairly small, and it hits all the requirements.



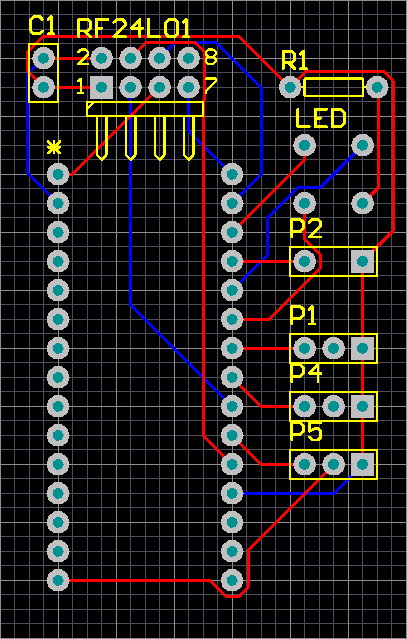
## RF Transceiver

This choice was part necessity, part personal preference. Like I said before, I knew wifi communication would be a battle I didn’t want to fight with IT so I sprung for the next best option. The NRF24L01 is a popular chip used by many hobbyists because of an extremely well documented and supported library for Arduino. I could have chosen Bluetooth, which is similar but I didn’t need to worry about acknowledgements or connection testing, so I threw that all out. I could easily send a string to be parsed on the other side with a simple function. The commands moved so fast, it wasn’t worth checking if they got there as a new command was already on its way. The only modification was a 47 uf decoupling capacitor soldered between power and ground.



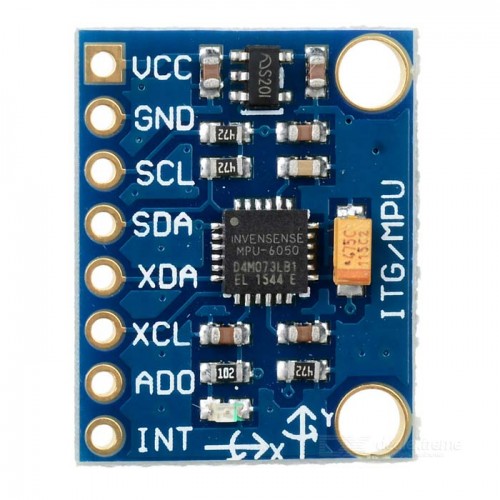
## PCB

The electronic glue that replaced my rats nest of a protoboard. Although wire wrapping is a great step up from breadboard prototyping, it doesn’t stand a chance against a perfectly (maybe) printed board. I discovered my digital signals just weren’t clean enough for clean handling. I began my trial by fire with Altium and found it wasn’t too bad (with a few components that is, the coals are burning my feet with my current project). I tried using the OtherMill at the makerspace to make my own, but they wouldn’t order double-sided boards for some reason. Forever a grudge I’ll hold. I sent my gerber files to the first search result which offered 5 boards for $20. I believe Osh Park is cheaper for one board, but hey, now we have 5. I included space for an RGB Led if you so wish. You’ll find them here <https://www.adafruit.com/product/314> or from Mike. You’ll find the files in the repo.



## Inertial Measurement Unit

Although not actually used, I do have one mounted on the frame and connected to the Arduino (the frame had a pair of screws that conveniently matched the breakout). I wanted to solely use the camera system for measurement, but the system could be completely robust if included. The cameras max out at 240Hz and there will be a delay through the system reaching the motors. The IMU on the other hand samples at 1000Hz and skips the lengthy communication gaps. With more time I would’ve introduced a stabilization inner loop with the IMU and a position control outer loop with the camera system. I have an MPU6050 breakout which communicates via I2C. Hopefully whoever is reading this can take on that challenge.



# Let’s count how many comments are in my code.

## Arduino

## Matlab

# “Figuring things out”

Let’s get down to business. I’ll be running though my workflow of tinkering with the various components.

ESC

Motor calibration curve

inertia

# Just make it fly already.

The following is a checklist I would run each time I began a session:

1. Setup motion capture
   1. Change all the cameras to 100Hz to match the matlab script.
   2. Hide all IR balls in the room and run the usual BADGER Lab calibration protocol with cameras focused on the flight area. I liked to use the forceplates as the ground plane as those are closest to level.
   3. On the streaming tab, enable and ensure axes match those in the matlab script. The status indicator on the lower left should be yellow (or green if matlab is taking streaming data).
   4. Add about 7 IR balls to the quad in an asymmetrical arrangement.
   5. Position the quad on the forceplate with the legs exactly along the length. This ensures the quad axes are aligned directly with the motors. (picture)
   6. Add as a rigid body (I don’t think it has to be named anything, leave it as the default).
2. Quadcopter
   1. Ensure battery is fully charged (12.6V).
   2. If desired, attach killswitch string to battery connector and wrist (proved successful multiple times).
   3. Plug in battery and wait for happy chime sequence, all motors should sound in unison, swap around ESC’s and motors to troubleshoot.
   4. Ensure propellers are screwed tight.
   5. Plug in homebase Arduino via USB.
3. Matlab
   1. In quad control callback check that framerate matches Motive.
   2. Run quadcontrolV4
   3. The GUI has been disabled to cut a ridiculous amount of latency. (future work: update every second or two rather than every frame. Or threading if you’re ambitious)
   4. Close the window and reopen to load the serial ports (bug) and choose the one connected to the home base Arduino
   5. With the quad battery unplugged, click “motors on” and move the quad manually to ensure the axes are correct and the motor commands react appropriately. You may have to add some print commands.
   6. The script should abort if the quad falls too low or rotates too extreme, you may test that now.
   7. With the script in “abort mode” plug in the battery and hold the quad in outstretched hand with sleeves and safety glasses on. Click “motors on”
   8. Guide the quad until the integral term catches up and provide small bumps to keep it on track. Soon you’ll start to feel cool powers where you think you can tune it by hand and spend an entire year doing so. Good luck!