Self-Stabilized Monopod – A Reaction Wheel Inverted Pendulum with High Powered Brushless Motors

Kyle Bartholomew

Advisor: Nolan Tsuchiya PhD.

June 13, 2017
Senior Project
Mechanical Engineering
Cal Poly Pomona

Table of Contents

Purpose

The goal of this project was to create a standalone self-stabilizing monopod. The idea for this project came from the Core77 product design blog. A writer there had seen a video of a 2-axis reaction wheel pendulum, and was excited by the possible applications of it. One of the ideas they had was a monopod for taking selfies that could stand on its own just like a tripod. Upon reading this, I also thought that was a fantastic idea, so I decided to make this my senior project. This project is exciting because it is simple enough to be completed in a 2-quarter senior project, it has possible business applications, and it just looks super cool.

Final setup overview

The final electronics setup that I got working consisted of a microcontroller, an angle sensor, a hobby-grade brushless motor, a servo-grade brushless motor controller, and a lead acid battery. The final hardware setup consisted of skate-bearings, an aluminum tube, and 2 custom machined mounts. The bearings are an important part of the final setup, because this version was just a single axis. With 2 stabilized axes, no bearings are required.

Initial Plan

This is the timeline that I set out for myself at the beginning of the project:

Outline of Steps:

1. *Create a MATLAB simulation of the device using Simulink and Simscape Mechanics. The initial control system will be a simple feedback LQR controller.*
2. *Determine the approximate power and speed requirements for the motor.*
3. *Build a simple inertial dynamometer to test motors.*
4. *Test multiple motors to find the appropriate motor for the application. For each motor, a 3D surface showing the motor characteristics will be developed (RPM vs Torque vs Input)*
5. *Build a desktop-sized 1 axis inverted pendulum.*
6. *Build the full-size (40”-60”) inverted pendulum using the chosen motors.*
7. *Tune the system by hand for the best performance with an LQR controller.*

The goal is to complete these steps evenly spaced over the first quarter. This large buffer gives room for issues to come up, and room to do more in-depth exploration in the second quarter if time allows.

Second Set of Steps (Quarter 2 if time allows):

1. *Implement an LQG controller.*
2. *Manufacture a super lightweight and not-rigid monopod, and implement a controller that dampens vibrations in the monopod.*

Adjustments to Initial Plan

The initial plan ended up working very well for this project. I have essentially followed all of the steps in the first set of steps. I got to step 5 in the first quarter, but I was having issues stabilizing my desktop pendulum with my chosen motor, so in the second quarter I took a few steps back and focused on understanding the motors better and figuring out the proper type of motor to use. I have not created a full size 2-axis monopod, but I have created a working nearly full size 1-axis monopod. Unfortunately I was unable to get to any of the steps in the second set, but I may be able to revisit them at a later date. From

my experience with the system, an LQG controller doesn't seem necessary, so I would likely pursue the second step, which is a vibration damping controller with a non-rigid monopod.

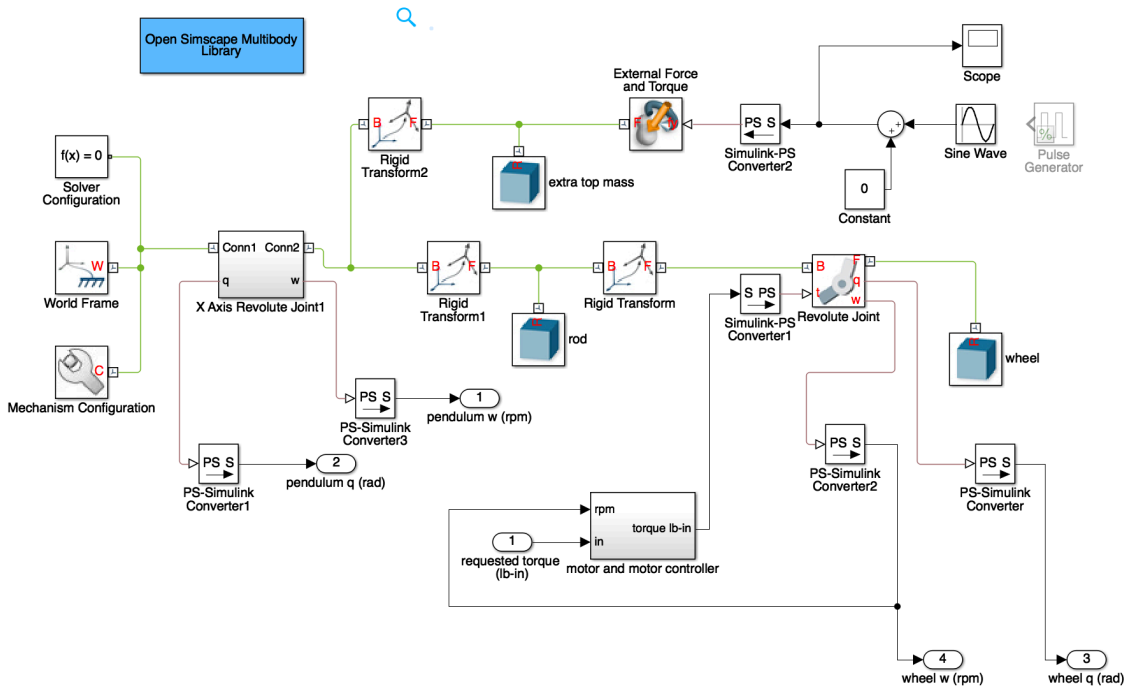
Theory of Operation

The reaction wheel pendulum operates in a way that is not immediately intuitive to most people. A mass is mounted to the end of a motor, which is attached at the top of a rod. The rod is able to rotate freely in 1 or 2 axes, depending on the design of the reaction wheel pendulum. The axis of the motor is parallel to the axis of rotation of the rod that it controls. In the case of a 1-axis reaction wheel pendulum, the motor shaft is parallel to the shaft at the bottom of the rod.

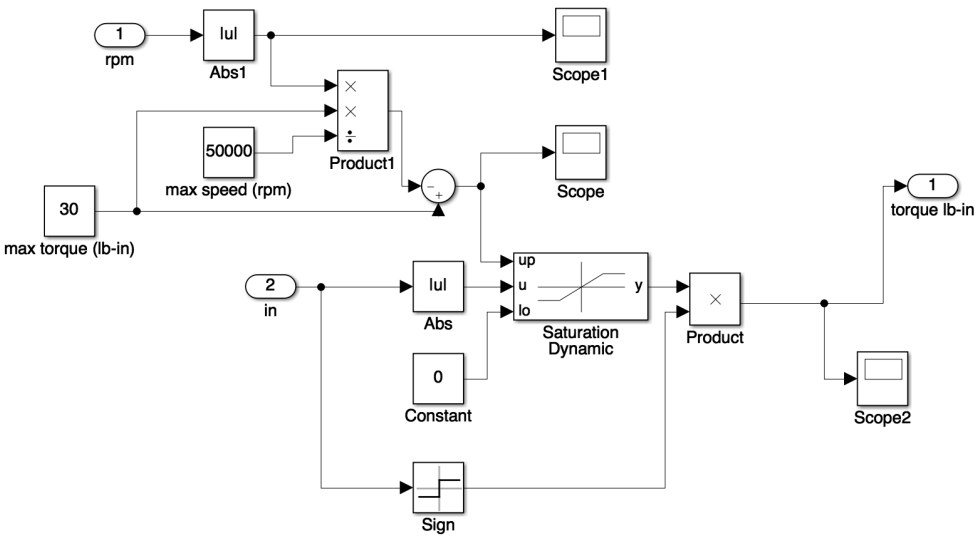
If the motor is spinning at a constant speed, the motor is not creating any torque. But if the motor changes speeds, then torque is transferred to the pendulum. This torque is the result of Newton's Third Law: For every action there is an equal and opposite reaction. Torque is required to change the speed of the motor and wheel. The reaction to this torque then goes into the pendulum. Since the pendulum is not mounted rigidly, it moves. This movement is what we are using to prevent the pendulum from falling over. If the pendulum is falling to one side, the motor changes speed, which creates a torque, which causes the pendulum to move in the opposite direction. The controller modulates this to keep the pendulum as close to the top as possible.

Matlab Simulation

The first step of this project was to create a Matlab simulation of the mechanical system and the control system. Making the simulation was helpful for understanding the system better, getting some real data about the requirements of the system, and learning how to simulate mechanical systems. While this system is easy enough to simulate with a simple ODE, I decided to use Simscape Multibody, because it is a useful skill to know for future, more complex projects. Most systems are easier to simulate in Simscape Multibody rather than developing non-linear ODEs. Simscape has 2 main methods of creating the physical model. You can either create blocks of joints, solid bodies, sensors, etc from scratch, or you can import a CAD assembly. I experimented with the CAD method in a previous project. For this project, I created the physical model using blocks from scratch. This was relatively simple and allowed me to adjust the parameters of the system entirely within Simscape, as opposed to having to reimport the entire model anytime a dimension changed.



The model shown above consists of a rod attached to a point mass that simulates the mass of the electronics, a wheel attached to the top through a revolute joint, and an unconstrained 1-axis revolute joint on the bottom of the rod. The wheel's revolute joint takes a torque input and it outputs the wheel's angle and angular velocity. The rod's revolute joint doesn't have any inputs and it outputs the rod's angle and angular velocity. Various different disturbance forces are applied to the extra top mass, to examine the response of the controller. I also attempted to simulate the output capabilities of a brushed DC motor, since that was what I was originally planning on using.



This model of a brushed DC motor doesn't take into account dynamics or frequency response. It simply limits the maximum torque output based on a linearly decreasing torque curve.

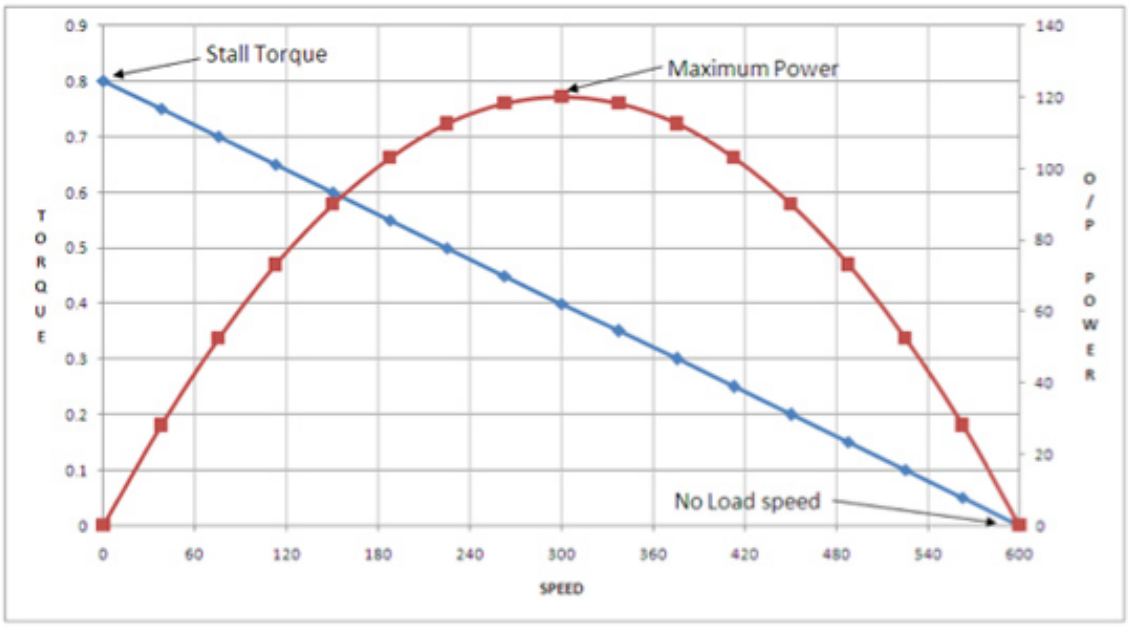
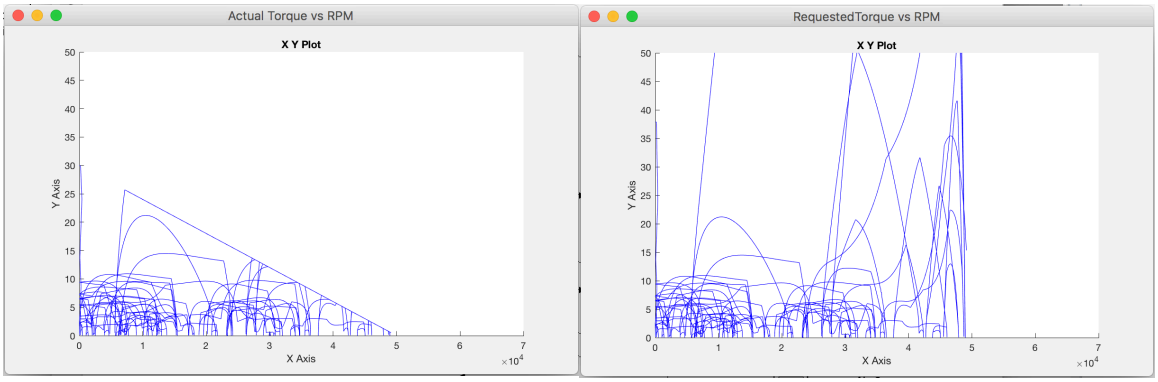
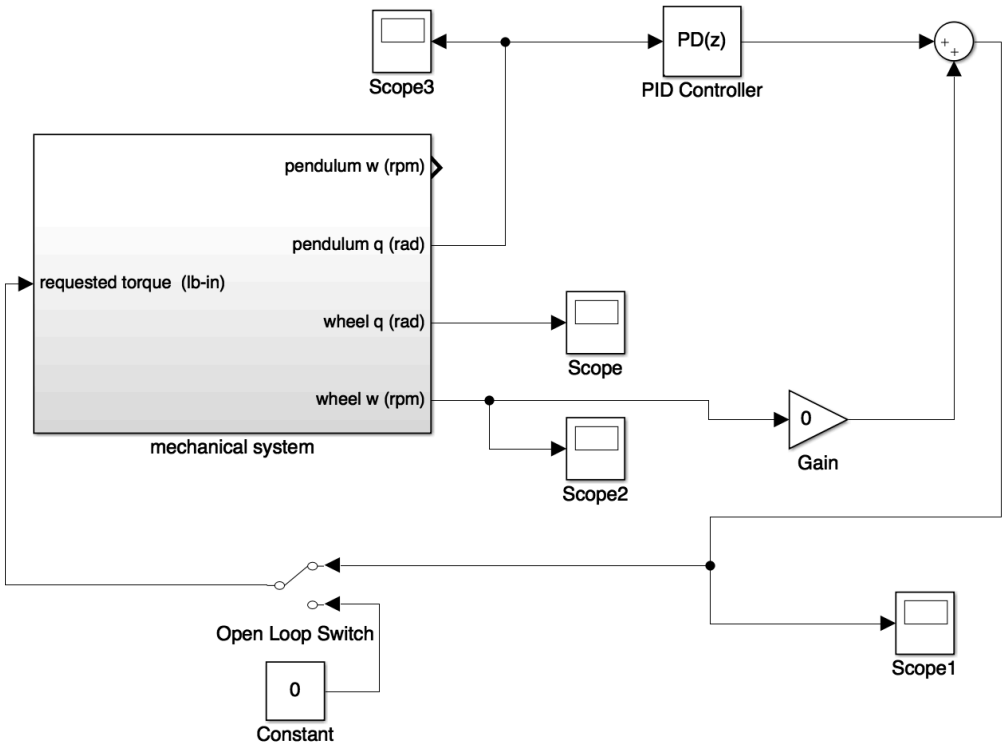


Figure 5: Speed-Torque-Power curve

This graphs shows a simplification of the capabilities of a brushed DC motor. The motor can supply max torque at "stall" (zero RPM), and zero torque at the "No Load Speed" (max possible RPM).



These graphs show the actual torque applied to the motor in the simulation and the requested torque from the PDS controller. The simulation that created these graphs had band-limited white noise applied as a force to the top of the pendulum. Even though the torque is significantly limited at many points in the simulation, the pendulum did not fall. The gains were essentially just smaller when the motor was being saturated, so this resulted in a slower response time. The power level of the white noise was set so that the pendulum would not fall over, but with slightly higher amplitude white noise, the motor saturates for a longer period of time and the pendulum falls.



To control the simulated system, I used a PDS controller. The PD part was accomplished with a PID block. This allowed me to use Matlab's PID tuning tools to find gains that would result in certain response times. I also tried applying experimental gains directly to the angle and angular velocity of the rod, which worked exactly the same way. The S part of the controller is a gain applied to the speed of the wheel. This acts like friction on the wheel, to slow it down slowly. Since the simulation is perfectly ideal, there is no friction on the rod. This meant that slowing down the wheel was a torque that caused the rod to fall over. In real life, there is some friction on the rod, so slowing the wheel down very slowly won't cause the rod to move. The S term is important in real life, because all motors have a maximum RPM. At some point the motor will be spinning close to that max speed, so it will not be able to impart any torque on the rod. To prevent this, the motor is slowed down with the S term so that the maximum torque is available as often as possible.

Running the simulation that I created required a handful of Matlab toolboxes that do not come with the base installation. My installation of Matlab R2016a has every toolbox, but of course not every toolbox is required for this simulation. In order to use Simscape Multibody to simulate the system, Matlab, Simulink, Simscape, and Simscape Multibody are required. I also highly recommend the Control System Toolbox and the Curve Fitting Toolbox. The former is useful for the simulation and the latter is useful for analyzing data from the physical system.

Implemented Control Algorithm

The simulation of the system was important for finding the type of control algorithm needed to stabilize the system and reject disturbances. It was clear that a P (proportional) controller was not enough, since there is no damping in the simulated model, and nearly zero damping in real life. In the simulation and in testing, a P controller caused oscillations. If the P gain was low enough, the system would be

marginally stable, with a constant amplitude oscillation. If the P gain was any higher, the oscillations would grow and the pendulum would fall over.

A PD controller was used in the simulation with great success. In a simulation with an infinitely powerful motor, the PD controlled system was able to reach arbitrarily fast response and settling times. Since the inverted pendulum is a linear system within ±15deg from vertical, a PD controller is suitable.

One of the potential extensions of this project was to use a super lightweight rod for the pendulum, so vibrations would be induced in the rod from the motors and from random disturbances such as air currents. The rod would be optimized for strength, not for stiffness, so these vibrations would be significant. A controller would be developed that would measure the vibrations and use the motors to damp them out. This controller would add considerable complexity to the project, and is not necessary for a reasonably lightweight, but rigid, pendulum.

Motor Requirements

The biggest challenge of the project was finding the right type of motor. The reaction wheel pendulum has a very specific need that most control systems do not have: high power-to-weight ratio. Many control systems have the motor rigidly mounted to the ground or to a chassis. This means that the motor body doesn't contribute to the dynamics of the system. It is common to see people recognize the importance of low rotor inertia because the rotor is part of the dynamic system. Most systems that employ servos do not care about the mass of the system, except for what I will call "vehicles". In this context, a vehicle is a control system where the entire system is self-contained and moves. Some good examples of this are cars, drones, and rockets.

Cars must accelerate the engine along with the rest of the car. A 10-ton 1000 Horsepower industrial engine would be of no use to a car, because, ignoring size constraints, it would barely be able to accelerate at all.

A drone, or quadrotor, needs to be lightweight in order to stay in the air longer. The energy storage (the battery) of a drone makes up a significant percentage of the overall mass of the drone, so doubling the size of the energy storage only increases the flight time by a small percentage.

Rockets have the same problem as drones, but much worse. Rockets need to escape the gravity of the Earth. This requires a massive amount of energy, but the energy needs to be carried on the vehicle, and energy in the form of chemical bonds has mass. So this extra mass must be brought out of the Earth's gravity as well. This ends up being a loop, the limit of which ends up with a massive amount of fuel to bring a small amount of mass to space. In general, 90% of the weight of a rocket is fuel. That massive amount of fuel means that the rocket must have a large physical structure to carry the fuel. In the end, only a tiny amount of the rocket is the payload: generally between 1% and 5%.

I classify a reaction wheel pendulum as a vehicle, because, in order to stabilize itself, it must accelerate and decelerate its entire mass. If the motor weighs more, the motor must accelerate more mass, which means the motor must be bigger, which means the motor must accelerate more mass, etc. For this reason, the overall mass of the motor must be minimized, while the power must be maximized. Additionally, the dynamics of the reaction wheel inverted pendulum are very non-linear, so if the pendulum swings too far, the moment on the pendulum will increase dramatically. This means the motor must be powerful enough to keep the pendulum in the linear range, which is approximately ±15 degrees from vertical.

This brings up the issue of power vs torque. Technically all that is required is torque, even at a low power. However there are two issues with that. If you were to use a low power, high torque motor, such as a brushed DC motor with a gearbox, then the motor would need to stay at low rpm, since
*Power = Torque * RPM*

Additionally,
$Torque = I * \alpha$
where:
α *(Alpha)* is angular acceleration
I is moment of inertia

In order to decrease *power*, *RPM* must be decreased, since *torque* needed cannot be changed. In order to decrease *RPM*, *alpha* must be decreased, because *RPM* is just the integral of *alpha*. In order to maintain the same level of *torque*, the inertia of the wheel must be increased. This means that the overall weight of the assembly will increase, which will increase the torque required. This cycle results in a very large motor.

We run into another issue if we use a low power, high torque motor. Such a motor is only efficient in a small range of speeds. Beyond those speeds, the motor ends up with 0 torque. Since the only force on the motor is the inertial load of the wheel, the speed of the motor will constantly increase. Therefore, the amount of time that a motor can sustain a given torque is determined by its power. A low power motor could likely stabilize a system with very small disturbances, but as soon as the angle of the pendulum increases, the motor will run out of torque before it can bring the pendulum back to the top.

These considerations for a low power, high torque motor only affect the use of a brushed DC motor. Most other motor types can be scaled up to high power, while still being lightweight. A brushed DC motor is uniquely disadvantaged, because the control circuitry is built into the hardware of the motor. This control circuitry is a set of carbon brushes and copper commutator bars. These components must be fairly large, since they handle a lot of current, over a fairly high resistance junction. A brushless DC motor, on the other hand, uses tiny mosfets to commutate the motor. These mosfets account for a very small portion of the weight of the motor system. Additionally, it is very difficult to find brushed DC motors that have been built in a lightweight manner. Because of the inherent increased weight of brushed motors, manufacturers do not try to optimize for weight. There are some high power, relatively lightweight brushed DC motors available, but they are just as expensive as high end brushless motors (including the control electronics), which have a better power to weight ratio, so there is not much of a reason to use a high-end brushed DC motor.

I did do some testing with a cheap brushed DC motor, but it was clear that it would not work for this application. This small motor was heavier than the brushless motor I ended up using, but it actually started smoking after about 5 seconds with power and an inertial load applied. This motor was not capable of outputting constant high power due to heat building up in the brushes and commutator bars. A larger brushed motor would have been able to deal with the required load, but it would have been far too heavy, leading to the cycle discussed earlier. I did not do any testing with high-end brushed motors, though they likely would have worked ok.

The first quarter of this project I attempted to use a class of brushless motors and motor controllers typically designed for drones. These motors are revolutionary, because they have an extremely high power density and they are very low cost. A 1 Horsepower motor and controller combo costs less

than $70, and weighs less than 12oz (.75lb). This means that quadrotors can carry more batteries and increase their flight time. While these motors are perfect for quadrotors, they cannot be used in most robotics applications. The motor controllers are designed to be used at high speeds only, so they cannot be used below approximately 1000 RPM. The bigger issue is that these speed controllers cannot control torque. They can only control speed. That means that when there is a direction change, they allow a massive amount of current to flow. This can cause the mosfets to explode, or just a motor jerk accompanied by an awful sound. Torque control is often the most important control mode for a robotics application. These controller issues are due to hardware limitations, not software limitations.

I knew that some of these limitations existed when I started the project, but I have seen other people using these motors in inertial-loaded hobby robotics before, so I figured I would try it. If this did work, then the cost of a stabilized monopod would be incredibly low. I picked the best controllers for the application, which have open source firmware. There are many settings in the firmware that can be adjusted, so I figured I could adjust the firmware to work in this application. In my testing I found that while I could get the motor to start up reliably from zero speed, the controller was not controlling the torque, but rather it was slamming the motor to the requested speed as quickly as possible. I wasn't sure how that would affect the balancing system, so I pushed on with that testing. Throughout the project, I have done my testing with a P controller, then moving to a PD controller. In both cases, the process variable was the angle, and the refresh rate was fixed at 100hz. The cheap brushless motor and controller combination was not able to stabilize the pendulum. The motor controller couldn't keep track of the rotor position, so at any given time, it would do one of three things. It would not move at all, it would spin ok, or it would slam the motor into a different direction with a massive amount of current. Most of the time it was the last case, which resulted in an uncontrolled oscillation. Since the motor controller controls speed, decreasing the P gain results in the same amount of torque applied, but for a shorter period of time until the motor reaches the commanded speed. For these reasons, no set of gains resulted in a stable system.

The failure of this system resulted in a significant shift in the aim of the project. I started doing research into other types of motor controllers that would be able to control the torque of a brushless motor. I figured that the issue was not the brushless motor itself, but just the motor controller. I focused on trying to find a controller that I could get working with a cheap, lightweight drone brushless motor.

The first controller I found is called the VESC, an acronym for Vedder Electronic Speed Controller. This controller is fully open source, and was named after its creator, Benjamin Vedder. The VESC implements current sensing and FOC control, which I will explain more later. The VESC was designed to be used for robotics and other general uses, but it ended up being picked up by the electric skateboarder community, because it performed better and was cheaper than other skateboard motor controllers. I immediately bought a VESC from a DIY electric skateboard parts supplier and tried to get it working with the brushless motors I had. I ran into a few issues getting the VESC to operate the motors properly, because the model I had purchased was setup for massive brushless motors. The current sensing on the VESC, and many other controllers, is accomplished with current shunts. Those shunts need to be sized to give the maximum resolution and the maximum current range needed. Since mine was setup for motors that were capable of drawing over 150 amps, my motors were barely registering any current at all. The motors I had available were capable of 20 amps max, so there was not enough resolution in the VESC current sensing circuitry. After spending a few days working with members of the VESC forum, I tried a few solutions, but nothing really worked. I'm quite confident the motor would have worked just fine with the correct size current shunts, but I didn't want to resolder on new current shunts. Additionally, the VESC only has 2 current shunts and it assumes the value of the third current based on Kirchhoff's

current law. This makes it more susceptible to noise. And finally, only one individual is actively developing the VESC, so any additional issues would have to be fixed by myself. For these reasons, I decided to look for other controller options.

I finally found an excellent option that I think has the potential to be revolutionary in the robotics world. This is the Texas Instruments InstaSPIN system. TI has a line of DSP-optimized microcontrollers that were originally aimed at general usage. They recognized the need for an advanced brushless motor controller, so they wrote open source code for these controllers that turn them into high speed motor controllers. In addition to the open source code, they also developed some proprietary code that gets loaded into read-only ROM on the microcontrollers. The proprietary code estimates the rotor position and state, and measures the electrical parameters of the motor. To make this system more accessible, TI has a few application boards available that use the InstaSPIN system. These application boards can easily be turned into custom controller boards, with the specific power requirements of the given application.

The InstaSPIN system does well in some of the weak points of the VESC. The InstaSPIN system supports 3 current shunts, which allows for cleaner current measurements. Their proprietary motor parameter estimation also works better than the VESC's, which allows using motors with very different winding styles. They also have a team of engineers and an active forum available for support. Finally, the TI system is much easier to implement into a custom board, because they have numerous reference designs, each setup for different use cases.

I got started with the TI Launchpad F28069M, and the DRV8305 Boosterpack. This setup can control a motor at up to 45V and 15A continuous (20A max). That works perfectly for the motor size I have available, which ended up being the right size for this project. I ran into a lot of issues getting the system to work, but these were entirely due to poor documentation, not the system itself. Eventually I figured out how to use the example files and had motors spinning. One of the more frustrating challenges of this project has been writing code for this microcontroller. The supplied code only controls a motor and does not include any communication systems. In order to actually control a motor, the desired signals have to get into the controller somehow. They had the most code available for a simple serial connection, so I used this as my communication method. I had to dive into the 1,200 page manual for this specific microcontroller to figure out how to get serial communication working, but eventually I did get it working. This allowed me to do testing with the motor and eventually to add it to the control system.

Commutation Types

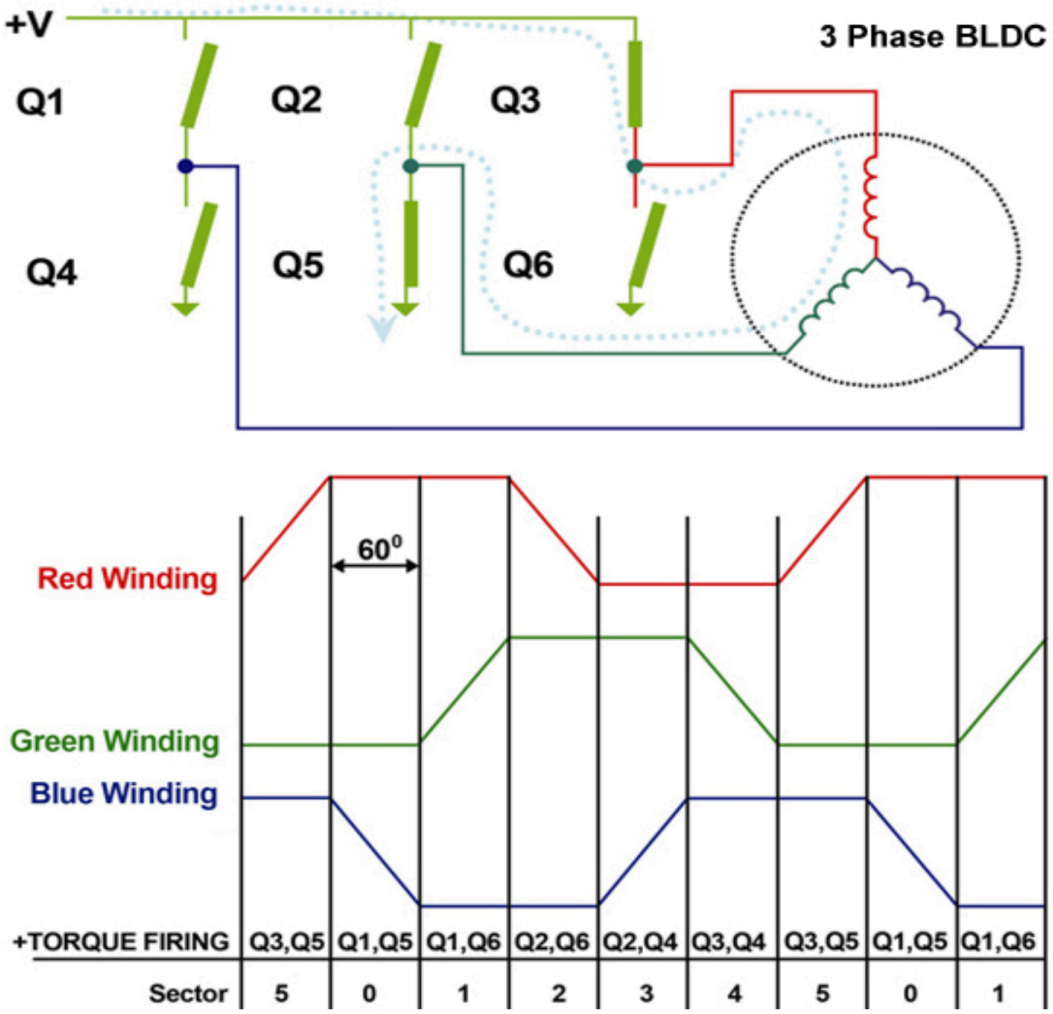
In motor terminology, commutation refers to the synchronization of the power applied to motor coils with the magnets in the motor. There are many different ways to do this. The simplest is with brushes in a brushed motor. The brushes contact the commutator bars and power up the coils. Then as the motor turns, the brushes lose contact and move to another set of commutator bars. This creates the rotating magnetic field that makes the motor spin.

In a brushless motor, the powering of the motor coils is done with electronics. In the past, all brushless motors had a set of magnetic sensors called hall effect sensors that sensed the position of the magnets in the motor. When the magnets moved to the next hall effect sensor, the motor controller would switch the power to the next set of coils. The combination of hall effect sensors and mosfets was a simple replacement for brushes that offered slightly better performance and much longer life.

More recently, the hall effect sensors were replaced with circuitry that senses the voltage spikes coming from the motor. This tells the controller how fast the motor is spinning, and effectively replaces the hall effect sensors. This method is only reliable at high speeds, since the voltage generated by the

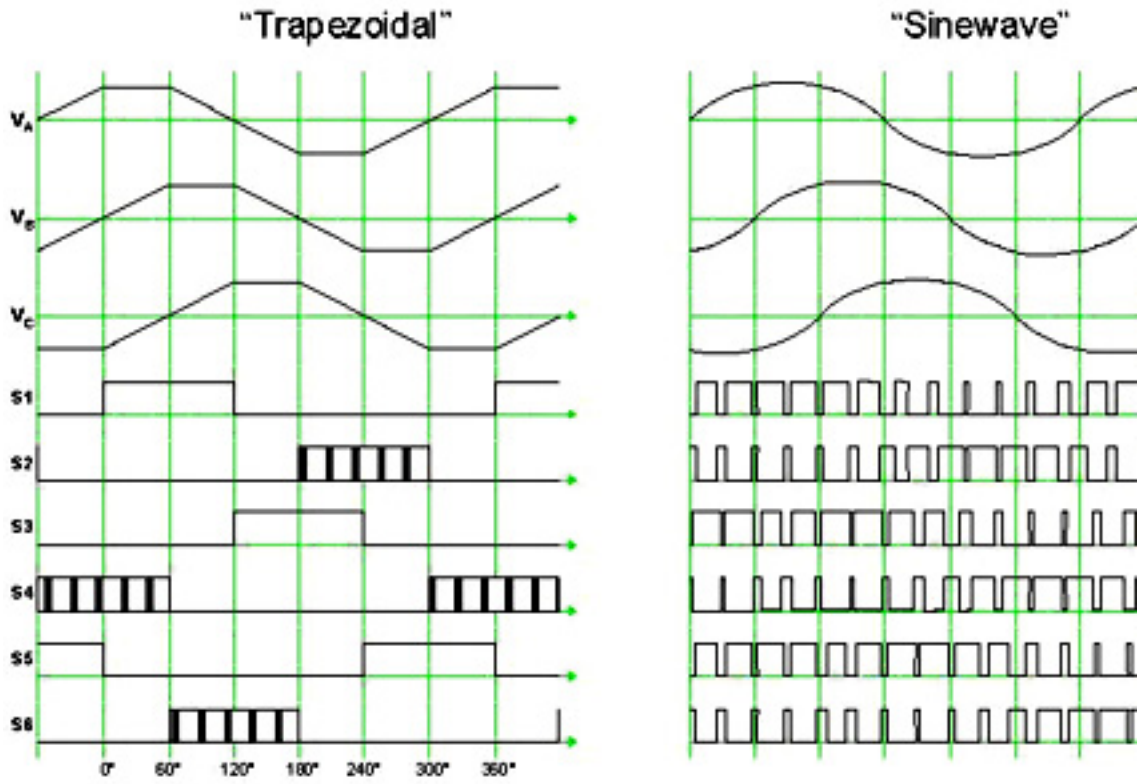
motor is proportional to the speed of the motor. The position of the magnets is unknown at low speeds (often below 1000 RPM), so the motor controller sends a big pulse of voltage into the motor to get it spinning. Once the motor is spinning, the motor controller knows where the magnets are located and can commutate the coils at the correct time. This is the method that quadrotor ESCs use.

All of these methods use what is called trapezoidal commutation, where the waveform being sent to the motor coils is a trapezoid. This is extremely easy to generate with a cheap, low-speed microcontroller or even with a simple integrated circuit. The trapezoidal commutation is very similar to the commutation produced by brushed motors, so it was a logical next step.



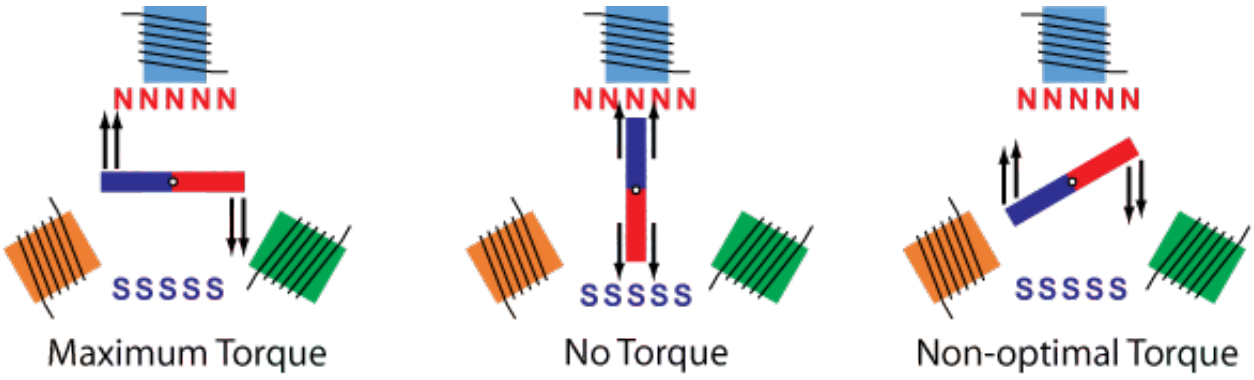
An example of trapezoidal commutation.

While this commutation method does work, there are other methods. One of these is sinusoidal voltage commutation. This is an extension of the trapezoidal method, but instead of using trapezoids, the waveform sent is a sine wave. The sine wave is created using voltage pulse width modulation (pwm). This makes the motor smoother, but since the sine wave is sent in voltage, the motor has a similar efficiency and torque curve as one using trapezoidal commutation.



Trapezoidal Commutation vs Sinewave Commutation, with the PWM signal for both displayed on the bottom.

A dramatically better method of commutating a brushless motor is Field-Oriented Control (FOC). This method is also sometimes called Vector Motor Control. For this method to work, the motor controller needs to know the exact position of the magnets, not just the sector that the magnets are currently in. The controller uses this information to point the magnetic field in the optimal position at all times. It uses a mathematical transformation to convert the 3-phase coil currents into a single vector, where one component is parallel to the magnet's field and one component is perpendicular. It applies current only to the component that is perpendicular, which creates the maximum amount of torque possible.

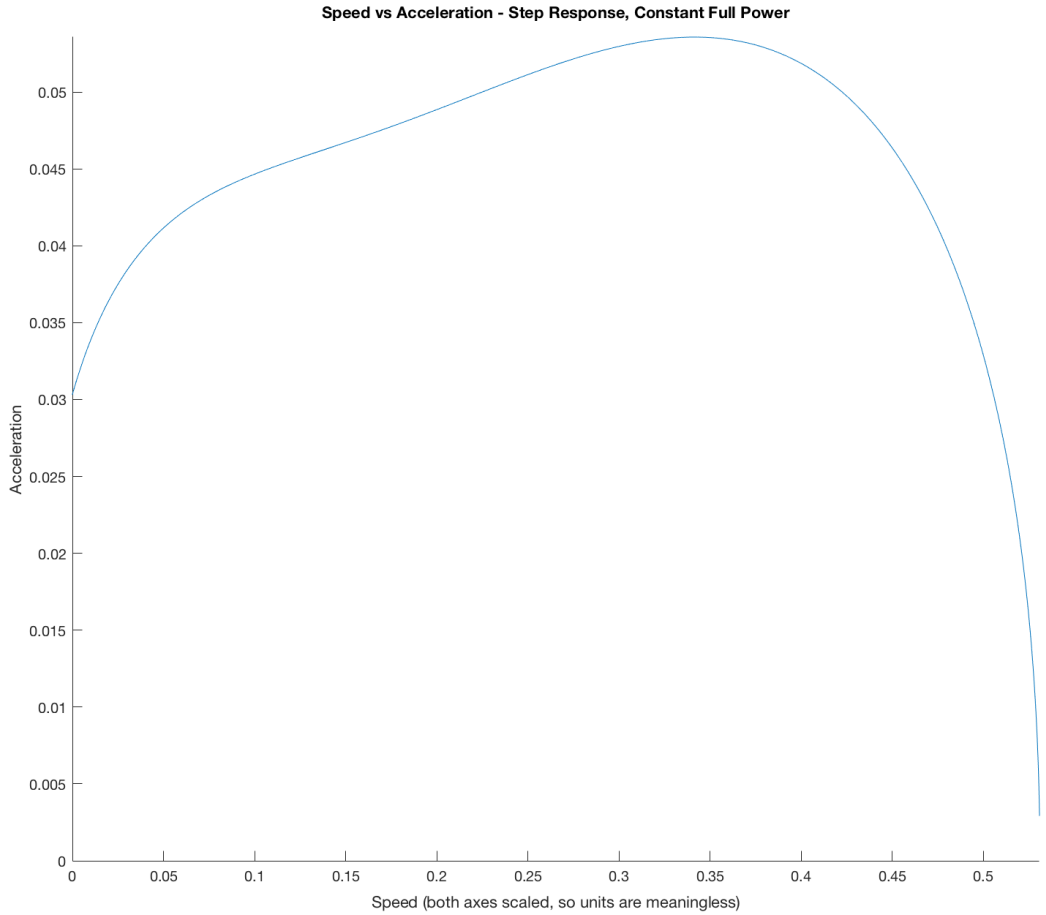


An example of how the magnetic field vector can be pointed to create different amounts of torque.

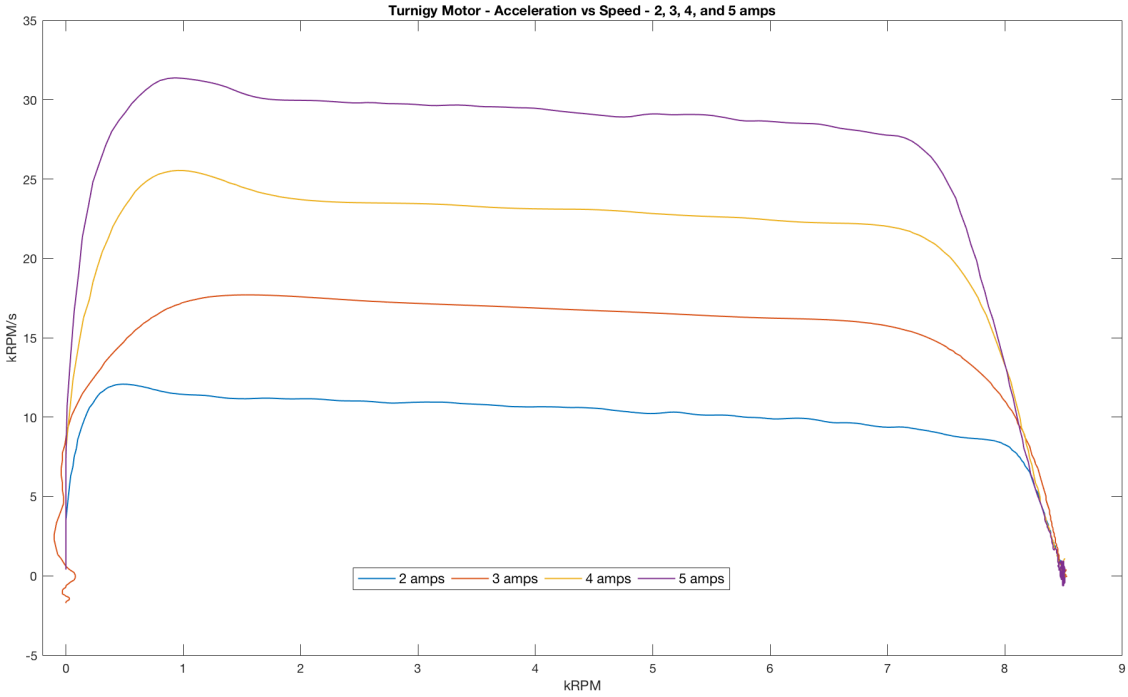
FOC commutation also creates a sine wave, however, unlike with basic sine wave commutation, FOC outputs the sine wave in current. Since there is inductance in the motor, the current is not directly related to the voltage. As the motor spins faster, the current lags the voltage. Current is what creates a magnetic field, so when the current and voltage are out of sync, a voltage command does not result in optimal torque. FOC directly controls current, so it is essentially sine wave commutation of the coil current.

Motor Testing

I ran step response tests for each of the motors I had operating, in order to get an acceleration vs speed curve. Since the only load on the motors is inertia, the acceleration of the motor is directly related to the torque. I don't have the data anymore for the brushless motor with a quadrotor ESC, but here is the data from a brushed DC motor and from the FOC-controlled brushless motor.



Brushed DC Motor Torque Curve



FOC Brushless Motor Torque Curve, at multiple current levels

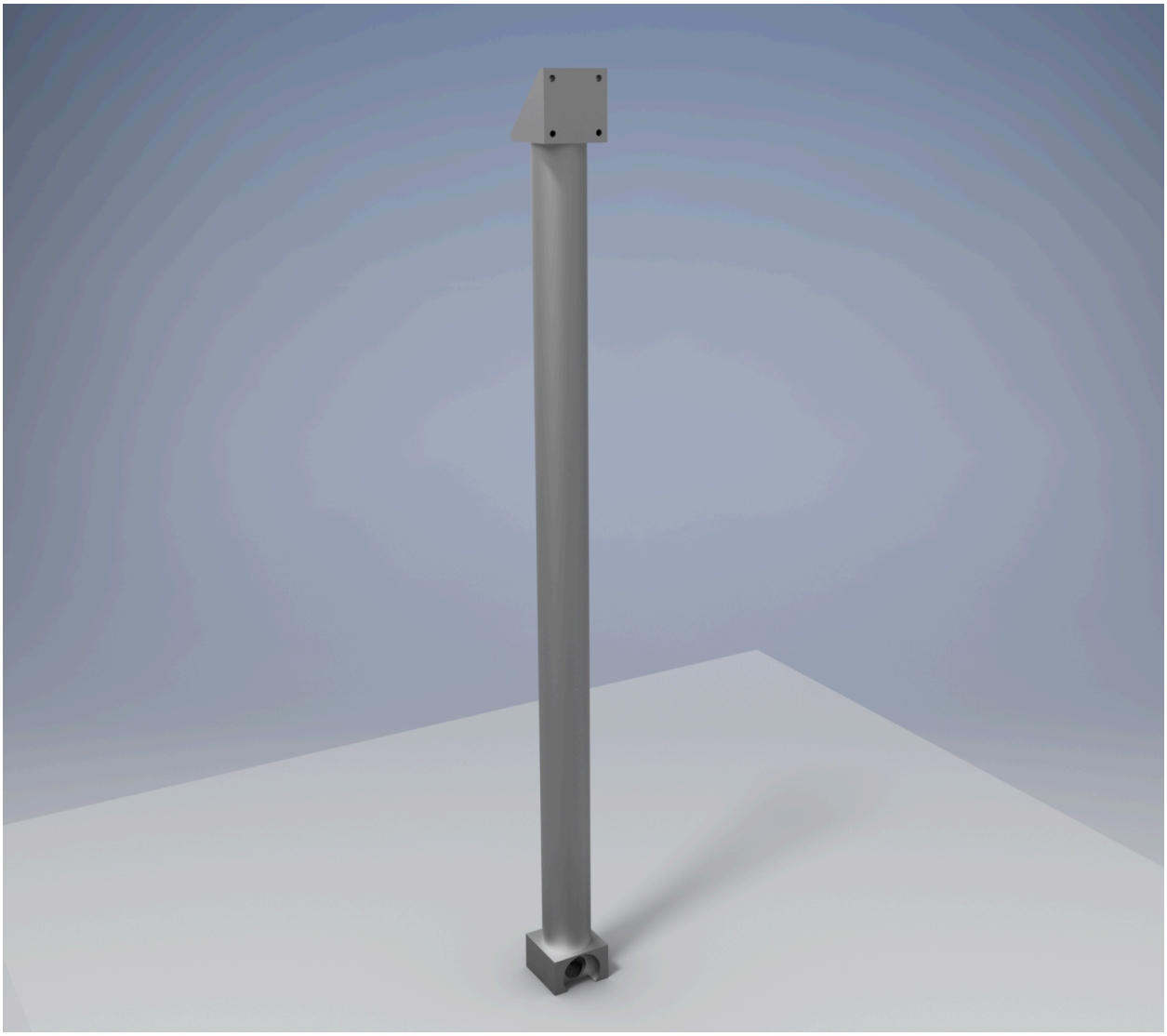
Final Bill of Materials
(all are quantity of 1, unless specified otherwise in notes)

Item Name	Supplier/ Manufacturer	Price (each)	Notes
Teensy 3.5	Amazon/PJRC	$29	The main microcontroller. A slower controller such as the Teensy LC would likely work fine with a PD control algorithm.
MPU6050 GY-521 Breakout Board	Amazon/Phantom YoYo	$6	The breakout board for the Invensense MPU6050 chip.
TI Launchpad F28069M	Mouser/TI	$27	The logic board for the motor controller. This board can control up to 2 motors.
TI DRV8305 Boosterpack	Mouser/TI	$49	The power board for the motor controller. Each Boosterpack powers 1 motor.
Sealed Lead Acid Battery (12V, 7Ah)	Frys/Tenergy	$30	The power supply for the system.
Small Breadboard	NA	NA	Connects the MPU6050 and the Teensy
Jumper Wires	NA	NA	22Ga Solid-core wire to connect the Teensy to the Launchpad
Power Wires	NA	NA	14-20Ga wire to connect 12V power to the
4500μF 16V Capacitor	NA	NA	An extra capacitor attached to the Boosterpack at the top of the pendulum.
Aluminum Tubing and Square Stock	Industrial Metal Supply	$15	These parts were machined by me to make the pendulum and the mounts.
608 Bearings	NA	NA	Double sealed, medium quality bearings. 2 bearings were used.
5/16” Threaded Rod and Nuts	NA	NA	The shaft for the bearings. 5/16” is close enough to 8mm, so it can be used with 608 bearings.

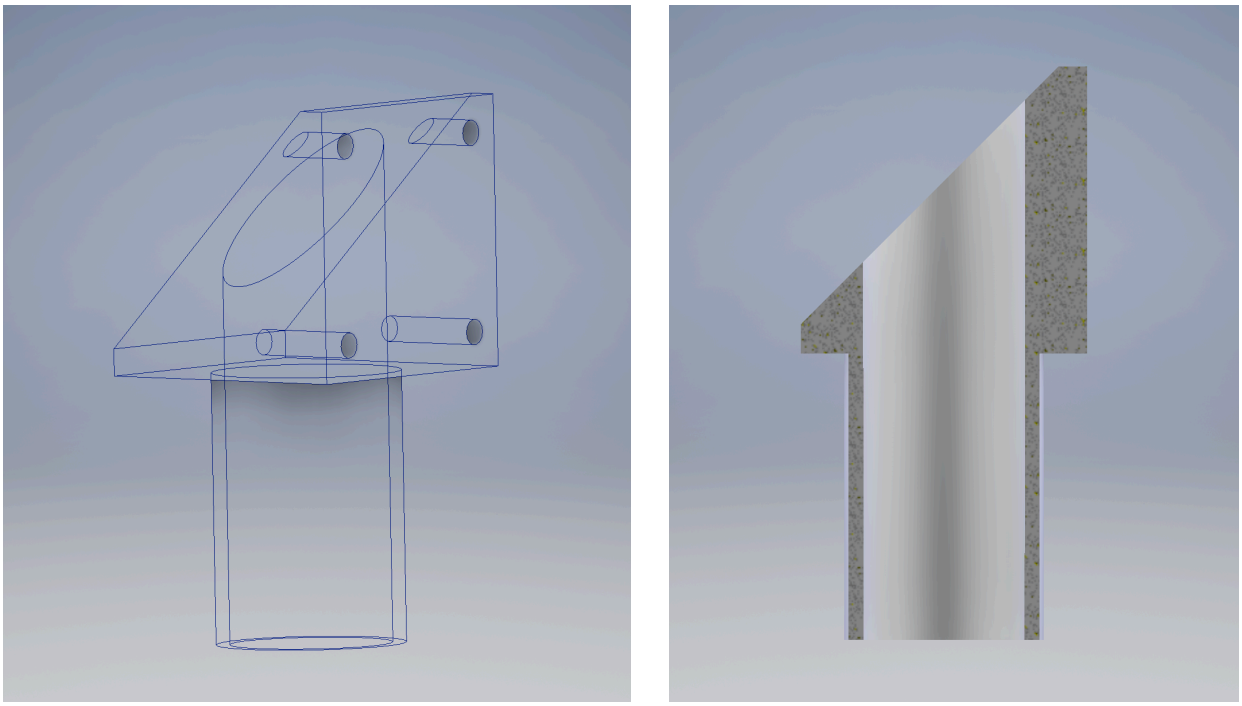
Total Cost (not including taxes, shipping, and parts I already had available [parts with NA]): $156

Mechanical Assembly

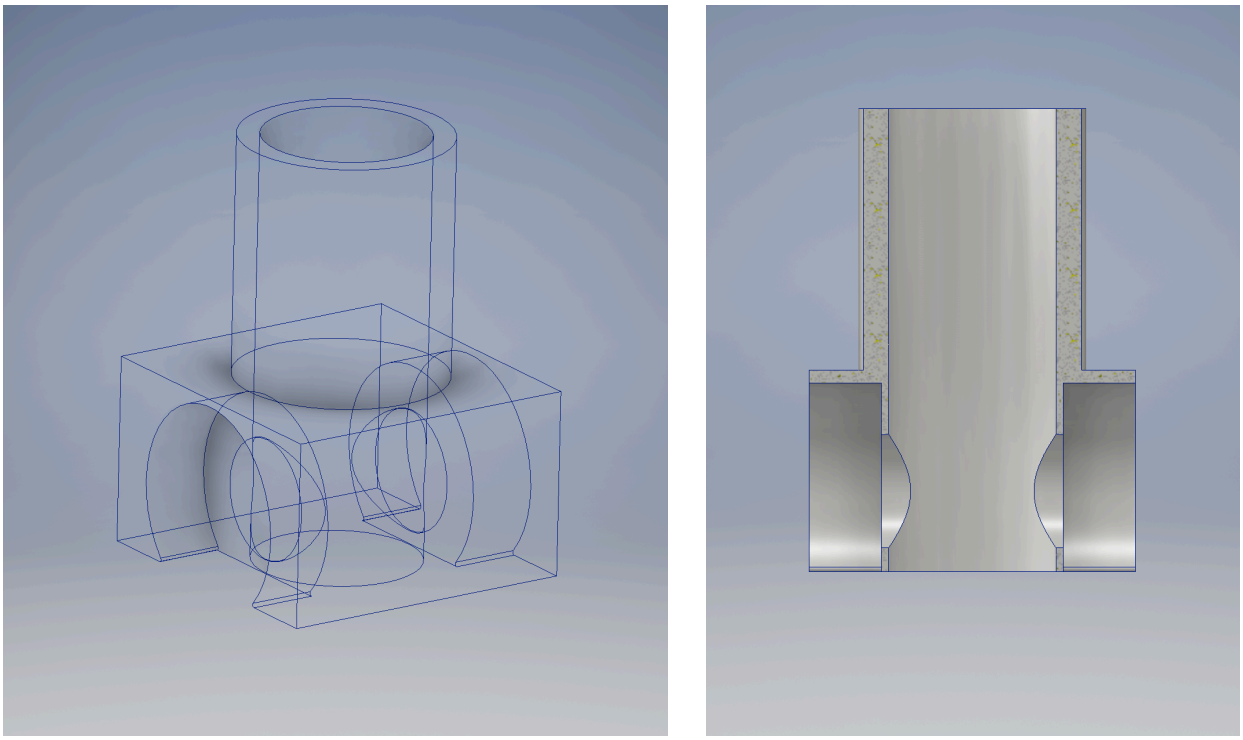
The mechanical assembly was designed using Autodesk Inventor. Here are images of the assembly in CAD.



A rendering of the complete assembly.



A wireframe and a section view of the top motor mount.

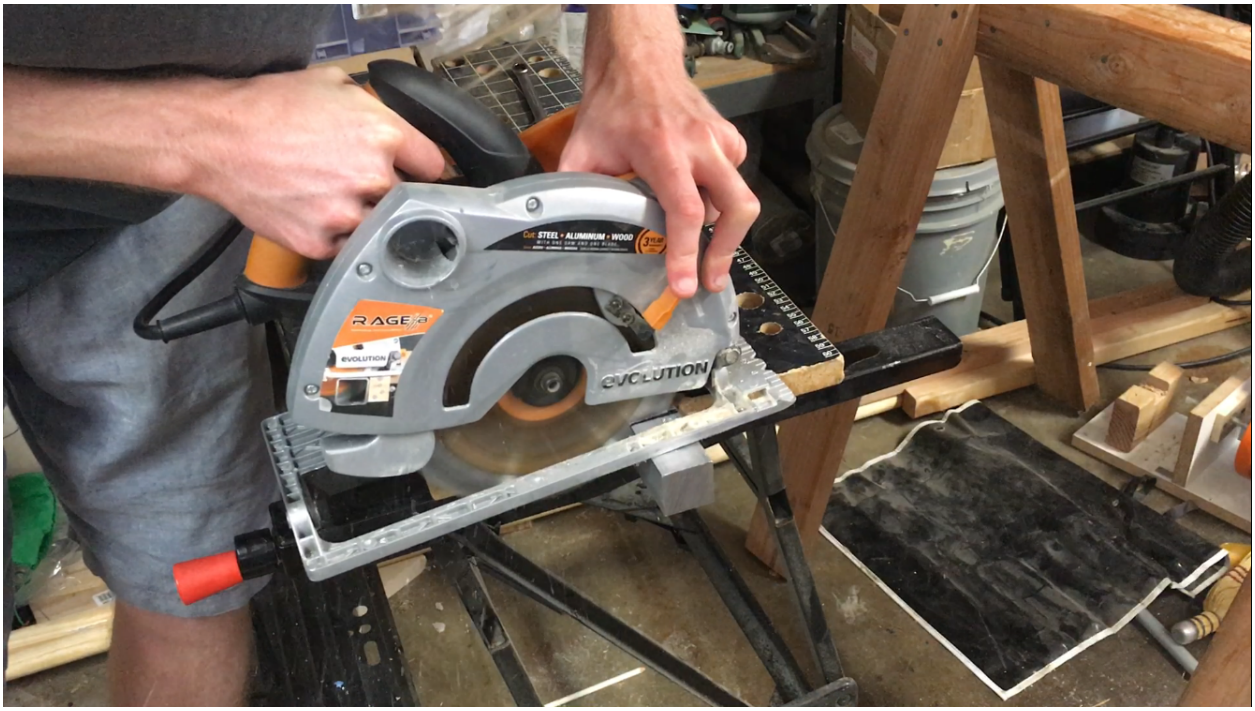


A wireframe and a section view of the bottom bearing mount.

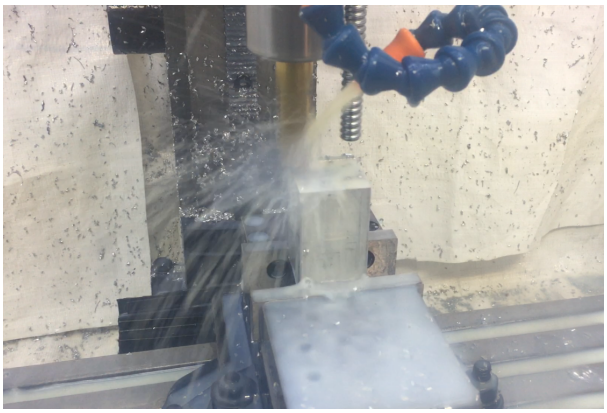
The mechanical system is composed of a length of aluminum tube, a motor mount, and a bearing mount. The motor and bearing mounts press into the aluminum tube and are a friction fit. During operation the forces on the assembly are small and none result in the mounts being pulled out of the tube. Using a press fit also allows the length of the tube to be changed at any time. I was able to use this feature to test 2 different lengths of tubing. The system was designed to be extremely rigid, so that low frequency vibrations don't affect anything during initial testing. The system would likely work very well with a mechanical system that is less than half the weight. These parts would have worked just fine as 3D printed parts, but I didn't have immediate access to a working 3D printer at the time, so I decided to use the tools I had and machine the parts. I could have machined them out of plastic, but it isn't any cheaper than aluminum, and aluminum looks better.

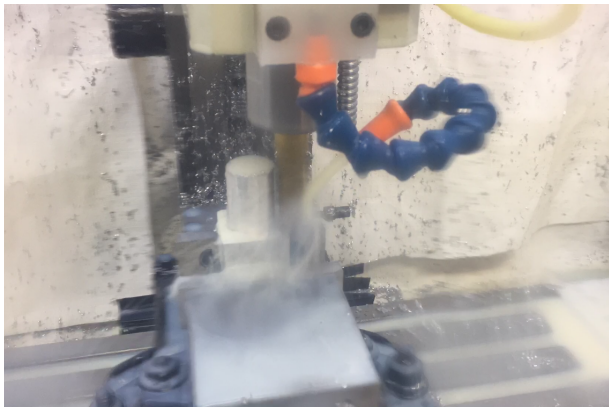
I machined the aluminum mounts out of solid pieces of aluminum, using a small CNC mill and a small manual lathe. The parts required many different tools and operations, moving between the mill and the lathe. The lathe was primarily used for getting the final finish on the tube press fits and for drilling holes that are concentric with that press fit axis. The lathe was also used to get a light press fit for the bearing mounts. The tapped holes in the motor mount were drilled with a drill press, and tapped by hand, with an M3-0.5 tap.

Here are images of some parts of the machining process:



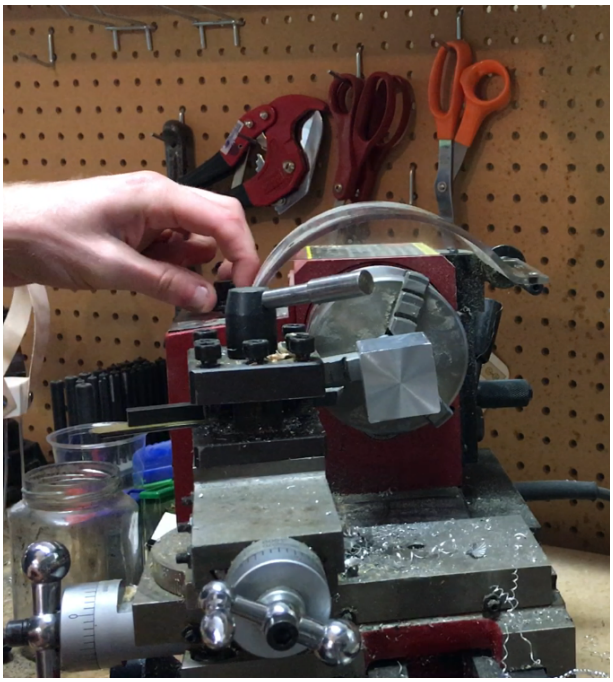
Cutting off a piece of material roughly to size, using a circular saw with a metal cutting blade.



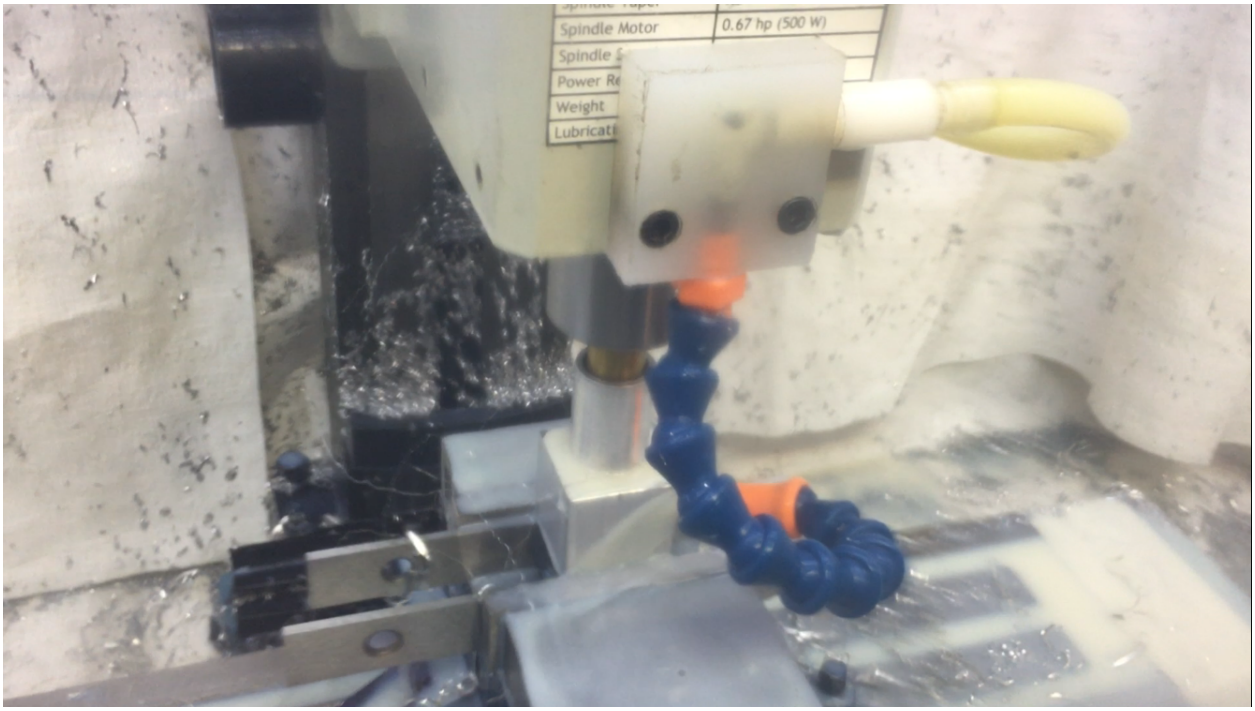


Beginning the roughing cut and finishing the roughing cut.

Removing the burrs from the machined part.



Bringing the motor mount to its final length on the lathe.



Drilling the deep hole going through the mount using an end mill. The majority of the material was removed on the lathe with the largest drill bit I had available, ½".

Electrical System

The electrical system is composed of a microcontroller, a motor controller logic board, a motor controller power board, and a battery. The microcontroller used is a Teensy 3.5, running Arduino code. The Teensy is a great option because it is cheaper than official Arduinos, but it is just as easy to use. The Teensy 3.5 is much faster than other Arduinos: 120Mhz vs the Arduino standard 16Mhz. This speed increase was not very important for this application, but if I had implemented a more complex control algorithm then it would have helped. With the simple PD control system, a cheaper microcontroller would have been more than sufficient. A good replacement would be the Teensy LC, since it is essentially the cheapest board-mounted microcontroller available, it runs at 48Mhz, it has multiple hardware serial ports, and it runs Arduino code.

I used the Invensense MPU6050 to find the current angle of the pendulum. This is the most important part of the system, because it gives the signal that all decisions are based on. The MPU6050 is a widely used IMU (inertial measurement unit), consisting of a 3-axis accelerometer, a 3-axis gyroscope, and a digital motion processor (DMP). Communication with the MPU6050 is done over I2C. You can get raw values from the accelerometer and gyroscope, but you can also get a clean angle value from the DMP.

Neither accelerometers nor gyroscopes produce angle outputs, so they must be converted. An angle can be found from the accelerometer by assuming that the only force on the accelerometer is gravity and using trigonometry to convert to an angle. This results in a very noisy angle, because there are always vibrations or large forces that create “fake gravity vectors”. Accelerometers only know about linear forces, so directly converting to an angle will result in a lot of random noise. An angle can also be taken from the gyroscope. The gyro outputs an angular velocity measurement (deg/s). That can be integrated with respect to time to get an angle. The issue is that this is a relative angle. If the device is upside down when you turn it on, it will think that is 0°, even though it is actually 180°. Additionally, if the gyroscope heats up slightly, it will start outputting a slightly higher voltage. The microcontroller will have no choice but to interpret that as a constant angular velocity. Integrating a constant angular velocity will result in the angle increasing, even when nothing is moving. This is called gyro drift.

The solution is to combine the angle value from both sensors to get a low-noise, drift-free measurement. The gyro is low-noise and the accelerometer is drift-free, so combining them gives a correct angle measurement. A simple way of doing this is with a complementary filter. This essentially just takes the average of two sensor values, but with a low-pass filter applied to the accelerometer and a high-pass filter applied to the gyroscope. This can work very well, but it needs to be tuned to get good results. Usually, some small amount of noise or gyro drift will still make it into the final angle. The better way of doing this is with a Kalman filter. I don’t have a strong understanding of how a Kalman filter works and it is very complex to implement. It requires data about the output characteristics of each sensor, and significant processing power to do the computations.

Instead of implementing this myself, Invensense has done it for me. They created something similar to a Kalman filter that runs on the MPU6050 chip. Since the chip has been specifically designed for this purpose, it doesn’t require a high speed microcontroller to do the computations. Plus, since Invensense knows the exact characteristics of their sensors, they can create a much better Kalman filter than I could. These calculations run in the DMP and are accessed as a final angle value over I2C. In addition to the angle value, I also used the raw gyro data, because the D term of the PD controller needs to know the angular velocity of the pendulum. Taking numerical derivatives can introduce noise, so it is much simpler to use the low-noise values directly from the gyroscope.

The motor controller logic board is a TI Launchpad F28069M. This is essentially a breakout board for the TI F28069M microcontroller chip. The chip runs C code that is compiled using TI's Code Composer Studio. The chip runs at 90Mhz, so there is some overhead available to add extra functionality. I could have implemented the controller code directly on the TI chip, but it would have required porting the I2Cdev library from Arduino code to TI supported code. This library is what communicates with the Invensense MPU6050. It is a fairly large library, so porting it would have taken a long time. Adding a supported microcontroller to talk to the MPU6050 was much easier.

The TI Launchpad works with the TI DRV8305 Boosterpack to control the brushless motor. The Boosterpack consists mainly of a DRV8305 chip, a Buck converter to regulate the high voltage input down to 3.3v, and a set of 6 mosfets to power the motor. The DRV8305 chip has a SPI bus to communicate with the Launchpad, a set of current shunt amplifiers, and a set of mosfet drivers. The current shunt amplifiers take the low voltage signal from the current shunts and amplify them to 0-3.3v. The mosfet drivers provide the necessary higher voltage to switch the mosfets.

The motor controller system uses the TI InstaSPIN code to do sensorless FOC motor control. The InstaSPIN system has an open source portion and a closed source portion. The majority is open source, which allows for a lot of freedom in development. The closed source portion is responsible for taking the motor coil voltage and current measurements and determining the position and state of the motor. This information is fed into the open source FOC code to create the output signals. The closed source portion also has a handful of optional features, like a motor parameter estimator and a servo controller. The servo controller was not used in this project.

The last important piece of the electrical system is the power supply. I did some initial testing using a computer PSU. I got a low-cost PSU that can supposedly output up to 42A at 12v. It seems that it can output at least half of that, so it works quite well for powering motors in testing. The issue is that when a motor is spinning and the controller tries to slow it down, that energy needs to go somewhere, so it actually goes back through the controller and into the power supply. This is desirable behavior, because that energy can be stored and reused. The issue is that if the energy is not stored or dissipated, it will cause the voltage at the input to increase. Power supplies don't like being fed energy, so they trip their fuse. This is what I experienced whenever I commanded the motor controller to slow down the motor. It didn't hurt the power supply because it had protection against this issue, but it just turned off. Controlling the pendulum requires slowing the motor down, so a regular power supply won't work. Some power supplies that are designed for motor control have a circuit built in that can dissipate energy, but these are more specialized and expensive. That circuit can also be added to any system, in between the power supply and the motor controller. The circuit is pretty simple; if the voltage on the input goes above a certain value (maybe 13v for a 12v nominal system), the circuit disconnects the power supply from the system and shorts the motor controller input lines through a large resistor. This switching has to be done with a power transistor of some type, because once the power lines are shorted the voltage will quickly drop back to 12v, so the power supply will be connected again. I will likely explore this option in the future, since it is relatively simple, but for this project I decided to go with a simpler method.

Instead of using a main's connected power supply, I used a lead acid battery. The battery I got can supply enough current for the system in short bursts, and it has enough capacity for well over 30 minutes of continuous testing (I didn't actually test this, but that's a guess based on how often I needed to charge it). When the voltage rises above the nominal voltage, instead of turning off, the battery just charges. This is better than a power supply with a circuit, because that power is saved and will be reused

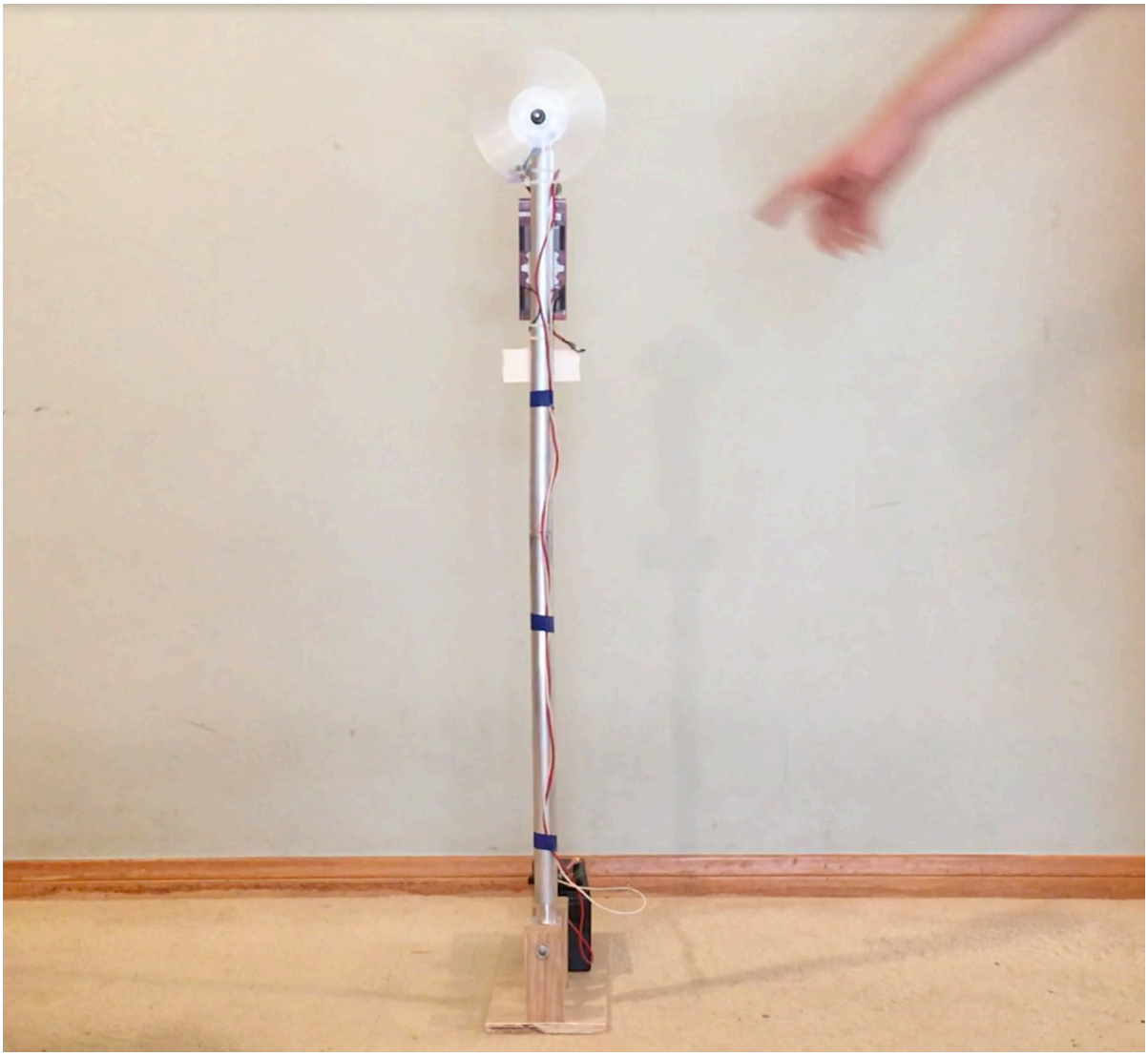
later, instead of being dissipated as heat through a large resistor. Additionally, the battery is cheaper than a power supply and it makes the system portable.

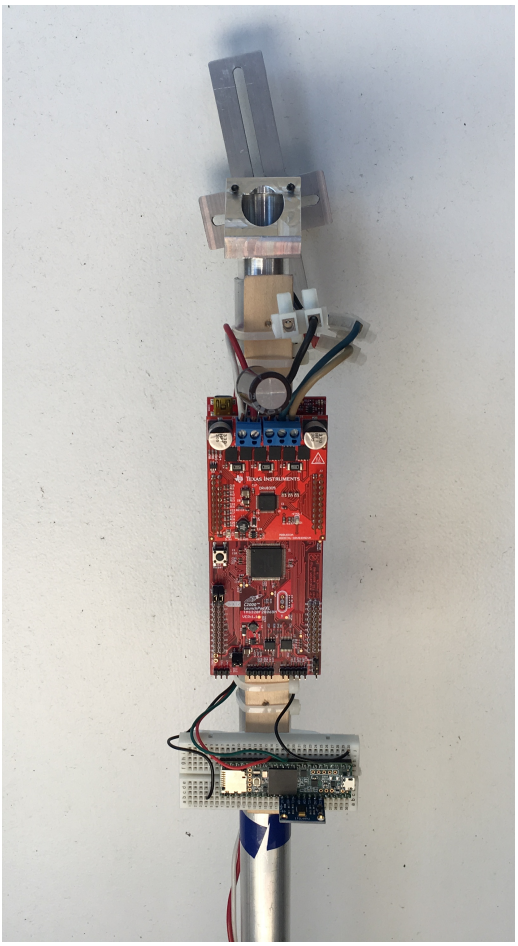
Final System Performance and Future Improvements

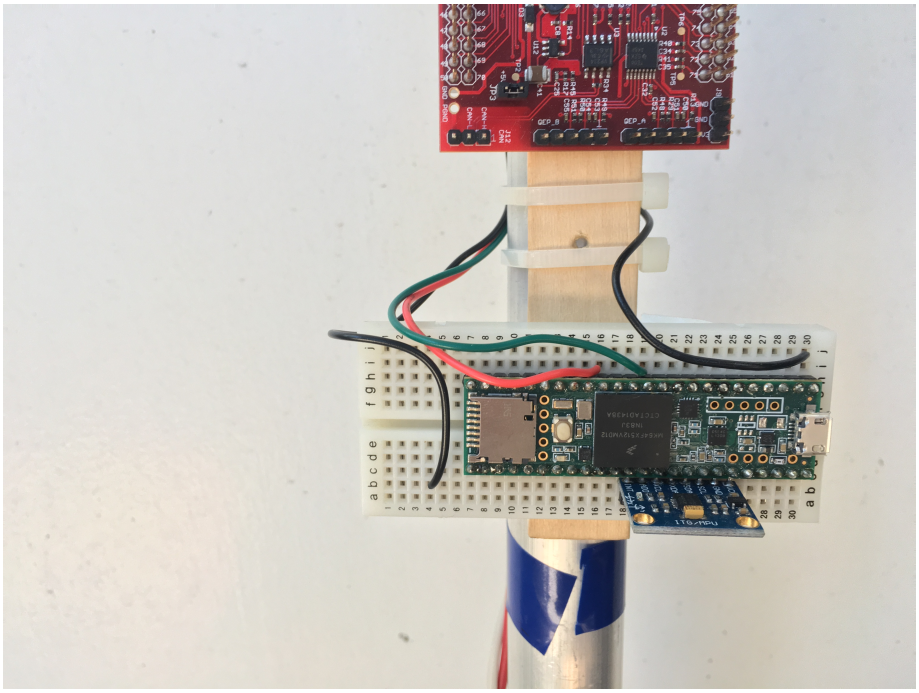
The system was able to perform very well with the motor, motor controller, and control algorithm outlined here. The system responded to disturbances and re-centered itself in about 1 second. With a taller system, the pendulum takes about 3 seconds to fully come to a rest. If the system is pressed hard enough, the pendulum will fall, but that is expected behavior. It appears that the pendulum can handle 5-10 degrees from vertical.

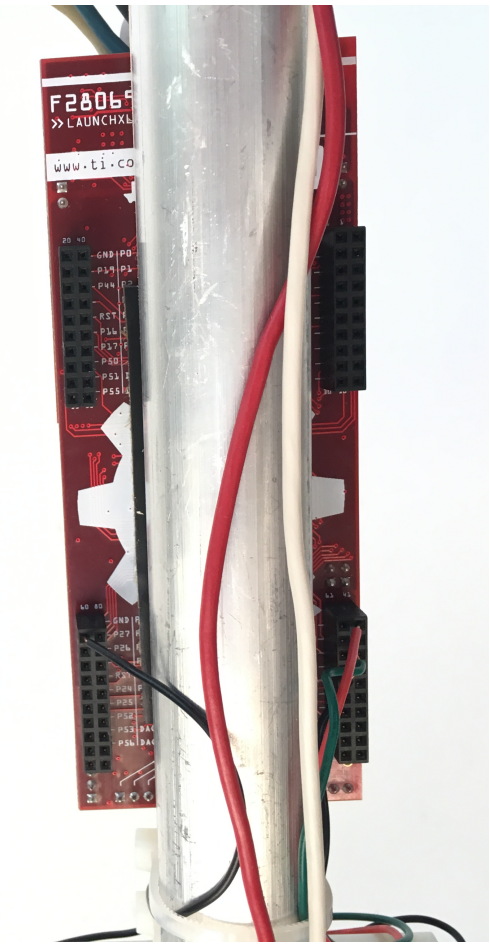
There are some improvements that could be made to the code. One is to implement the S part of the PDS controller. At the time of writing, only a PD controller has been implemented, so the motor often stays at a high speed for longer than is necessary. A setpoint adjustment algorithm would also be helpful. If the setpoint is not within 0.1 deg of vertical, the motor will not stop spinning even when the pendulum is balanced.

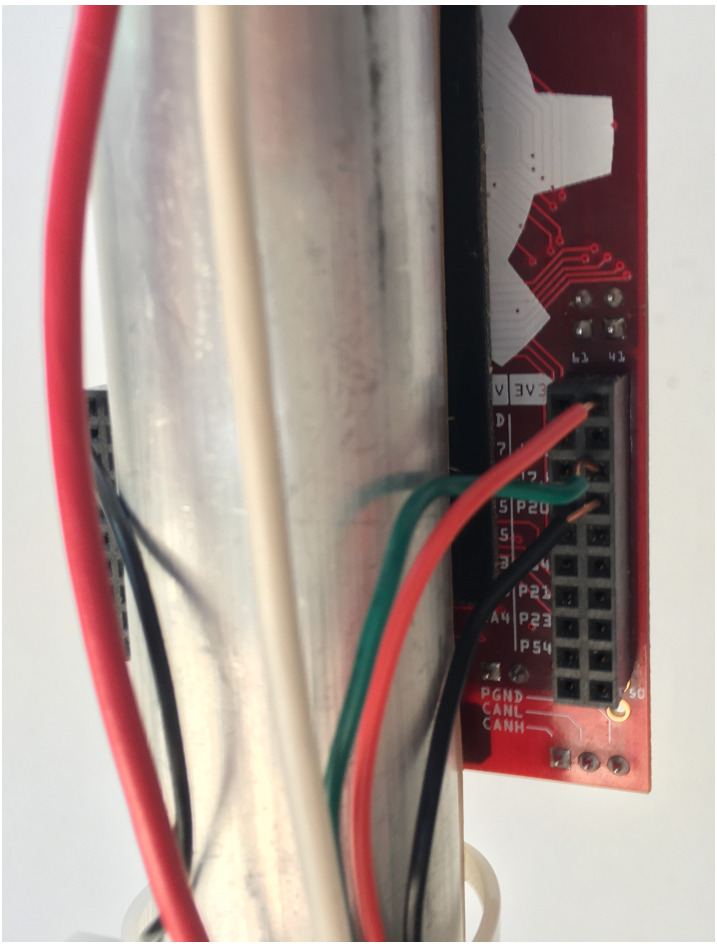
Photos








F28065
>> LAUNCHXL
www.ti.co


3V3
P20
P21
P23
P54
PGND
CANL
CANH

Works Cited

Allain, Rhett. "Modeling the Thrust From a Quadcopter." *Wired*. Conde Nast, n.d. Web. 13 June 2017. <https://www.wired.com/2014/05/modeling-the-thrust-from-a-quadcopter/>.

"BOOSTXL-DRV8305EVM User ’s Guide." *TI Reference Designs*. TI, n.d. Web. 13 June 2017. <http://www.ti.com/lit/ug/slvuai8/slvuai8.pdf>.

"Could We Add Feedback to the Flight Controller through SPI? · Issue #6." *GitHub*. N.p., n.d. Web. 13 June 2017. <https://github.com/sim-/tgy/issues/6>.

"Electronic Projects with Components Available Worldwide." *PJRC*. N.p., n.d. Web. 13 June 2017. <https://www.pjrc.com/>.

"Field Oriented Control of Permanent Magnet Motors." *TI Training*. Texas Instruments, 07 Mar. 2017. Web. 13 June 2017. <https://training.ti.com/field-oriented-control-permanent-magnet-motors>.

Fisher, Patrick. "High Performance Brushless DC Motor Control." TI, May 2014. Web. 13 June 2017. <http://www.ti.com/lit/an/sprt703/sprt703.pdf>.

Guan, Charles. "Tuning the SimonK Firmware for Robot Drive." *Equals Zero*. N.p., n.d. Web. 13 June 2017. <http://www.etotheipiplusone.net/?p=3985>.

Harrington, Aaron M., and Christopher Kroninger. "Characterization of Small DC Brushed and Brushless Motors." *Army Research Laboratory*. N.p., Mar. 2013. Web. <www.dtic.mil/get-tr-doc/pdf?AD=ADA577582>.

"High-Speed Sensorless-FOC Reference Design for Drone ESCs." *TI Reference Designs*. TI, n.d. Web. 13 June 2017. <http://www.ti.com/lit/ug/tiducf1/tiducf1.pdf>.

"Home." *Control Tutorials for MATLAB and Simulink*. U. of Michigan, n.d. Web. 13 June 2017. <http://ctms.engin.umich.edu/CTMS/index.php?aux=Home>.

Kalouche, Simon. "DESIGN FOR 3D AGILITY AND VIRTUAL COMPLIANCE USING PROPRIOCEPTIVE FORCE CONTROL IN DYNAMIC LEGGED ROBOTS." Carnegie Mellon U., 2017. Web. 13 June 2017. <http://www.ri.cmu.edu/pub_files/2016/8/kaloucheThesis.pdf>.

"LAUNCHXL-F28069M Overview." *TI*. N.p., n.d. Web. 13 June 2017. <http://www.ti.com/lit/ug/sprui11/sprui11.pdf>.

"Leading and Lagging Current." *Wikipedia*. Wikimedia Foundation, 19 Apr. 2017. Web. 13 June 2017. <https://en.wikipedia.org/wiki/Leading_and_lagging_current>.

Luque, Hector Hernandez, and Nicholas Oborny. "4.4 to 30 V, 15 A, High Performance Brushless DC Propeller Controller Reference Design." *TI Reference Designs* (n.d.): n. pag. *TI*. Web. 13 June 2017. <http://www.ti.com/lit/ug/tiduak1/tiduak1.pdf>.

"Meet the TMS320F28069M Launchpad." *TI*. N.p., n.d. Web. 13 June 2017. <http://www.ti.com/lit/ml/sprui02/sprui02.pdf>.

"MotorWare™ Software." *TI.com*. TI, n.d. Web. 13 June 2017. <http://www.ti.com/tool/motorware>.

Noe, Rain. "Inventor Creates Self-Balancing Stick, We Dream Up the Perfect Application." *Core77*. N.p., n.d. Web. 13 June 2017. <http://www.core77.com/posts/55211/Inventor-Creates-Self-Balancing-Stick-We-Dream-Up-the-Perfect-Application>.

"Open Source Software for ATmega-based Brushless ESCs." *SimonK's Tgy*. N.p., n.d. Web. 13 June 2017. <http://0x.ca/tgy/?tag=2015-09-12&obj=&target=afro>.

Page, Matt. "Understanding D.C. Motor Characteristics." MIT, n.d. Web. 13 June 2017. <http://lancet.mit.edu/motors/motors3.html>.

Rouleau, Mike. "Self Balancing Stick - Dual Axis Reaction Wheel Inverted Pendulum." *YouTube*. YouTube, 29 Sept. 2015. Web. 13 June 2017. <https://www.youtube.com/watch?v=woCdjbsjbPg>.

"SimonK Firmware." *GitHub*. N.p., n.d. Web. 13 June 2017. <https://github.com/sim-/tgy>.

"TI InstaSPIN™ Motor Control Solutions." *TI.com*. TI, n.d. Web. 13 June 2017. <http://www.ti.com/ww/en/mcu/instaspin/index.shtml>.

"TMS320x2806x Piccolo Technical Reference Manual." *TI*. N.p., Apr. 2017. Web. 13 June 2017. <http://www.ti.com/lit/ug/spruh18g/spruh18g.pdf>.

"Turnigy Aerodrive SK3 - 4240-620kv Brushless Outrunner Motor." *Hobbyking*. N.p., n.d. Web. 13 June 2017. <https://hobbyking.com/en_us/turnigy-aerodrive-sk3-4240-620kv-brushless-outrunner-motor.html>.

Vedder, Benjamin. "VESC – Open Source ESC." *Benjamin's Robotics*. N.p., 2016. Web. 13 June 2017. <http://vedder.se/2015/01/vesc-open-source-esc/>.