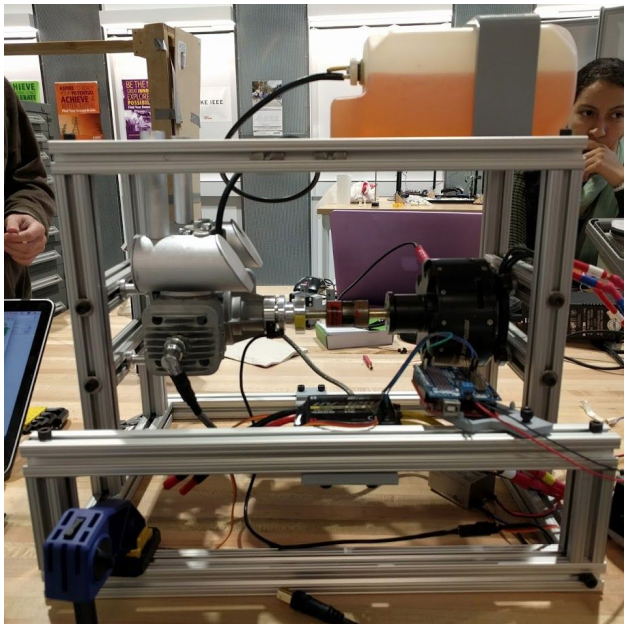
Hybrid Gas-Electric Power Generation for Heavy-Lift Drone

Garrett Andersen, Jihane Bettahi, Edward Kim, Tamra Nebabu, Henry Quach, Raiyan Sobhan



Overview

This report documents progress of the power electronics subteam of the Blue Devils Team in the 2017 Shell X-Prize. We have designed and constructed a proof-of-concept hybrid gas-electric engine light enough to be mounted on a eighteen-rotor drone and supply the rotors with power. We show that a twin-cylinder engine can be coupled to a brushless DC motor so that the motor may act as a generator to charge a battery. Thus, in a parallel hybrid configuration, either the engine or the battery it charges may be used to drive the drone.

Instructors: Dr. Martin Brooke, Dr. Doug Nowacek
ECE 495 - Ocean Engineering
Duke University

Table of Contents

The Contest **3**

Our Objectives **3**

The System **4**

The Big Picture 5

Circuit Design **5**

The Parts 6

Main Schematic 8

Main Connections 9

Rectifier Subcircuit 10

The Current Sensor 11

How to Run The System 16

Programming the ESC 17

Mechanical Frame Design **18**

Code **20**

Running the Code 21

Open the serial monitor 21

Calibrating the Servos 21

For the Future **22**

Documentation and Links **23**

The Contest

The 2017 Shell X-Prize is a contest to encourage innovations in ocean engineering and ocean discovery. The objective of the contest is to map the ocean floor using an autonomous system that is minimally invasive to marine life. The approach taken by the Blue Devil Team this year and in past years was to use a number of small submarine pods to collect sonar data. These pods would be deposited and picked up by a helicopter drone, that would communicate with the pods via WiFi.

Our Objectives

Because the drone must operate independently, it must be powered by an on-board system. Naturally, the power consumption is heavily dependent on the combined weight of the drone and its cargo. Of course, the power system itself (whether it consists of batteries, an engine and the drive train, or a hybrid of the two) will also contribute the weight of the system.

The issue arises of whether or not the system can be designed to be self-sustainable. Our objective as the power-electronics subteam was to design a proof-of-concept hybrid solution that satisfies the design constraints. One of the advantages of a using a hybrid solution is that the system is capable of switching between using the batteries or using the engine based on the circumstances to optimize performance. For example, suppose a few batteries are used to deliver power to the drone's rotor servos. If after some time the batteries are discharged significantly, the system can turn on the engine to supply power to drone. This can be done in one of two ways: either the engine can be used to directly power the rotors (in a parallel hybrid configuration), or the engine can be used to charge the batteries via a generator, and the batteries will be the only direct power supply for the rotors (in a series hybrid configuration).

For our project, we explored the feasibility of constructing a system where an engine could be used to generate power via an electric motor. The objectives for designing our proof-of-concept engine were as follows:

- Demonstrate that a twin-cylinder engine and electric motor could be coupled in such a way that the engine could be used to rotate the motor
- Design the circuitry that would allow the engine-coupled motor to generate DC power to charge a battery
- Demonstrate that the system is reliable (i.e. that the engine can be started on command and that it provides sufficient power through the motor to charge the batteries)

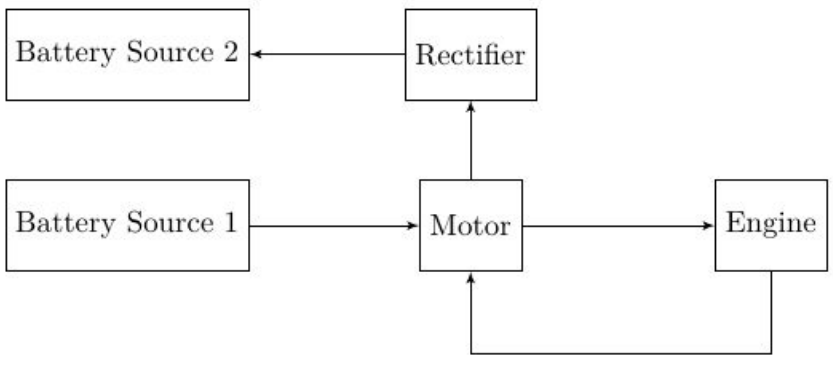
The System

In order to achieve the aforementioned goals one must first be able to start the engine. It should be noted that all engines require some initial rotation in order to get started. In vehicles, this rotation is achieved by using an electric starter motor to turn over the engine. This starter motor is turned on when the user turns the key in the ignition, or in the case of more modern cars, when the ignition button is pushed. We decided to design our system in an an analogous fashion whereby a coupled brushless DC motor would be used to start the engine and an Arduino microcontroller would take the place of the user. The DC motor would also be the mechanism of generating energy once the engine is running.

Of course there are a number of nuances that are involved in starting up the motor, coupling it to the engine, having the engine turn the motor, and ensuring that the motor can act as a DC generator. We endeavor to explain these nuances in this paper by talking about each component of the hybrid gas-electric engine, their purpose, and how they are used.

The Big Picture

Below is a block diagram for the general flow of power in the hybrid gas-electric engine.



The first battery source[1] is used to start the motor. When the motor rotates, it turns over the engine, thus allowing the engine to start. Once the engine starts, the battery is no longer used to power the motor and the motor now acts as a generator. However, because the motor generates a sinusoidally oscillating (AC) current, we need a rectifier to convert this to usable DC power that will charge a second battery source.

The reason we decided to go with a two battery-source architecture is to avoid running into conflicts where the flow of current to or from a battery is undetermined. By using two separate sources, we can guarantee that one source will always be used to start the motor (so current will always flow out) whereas the other source will always be charged by the engine (so current flows in). This second battery source can then be used to power the drone in parallel or series with the engine.

Circuit Design

The Parts

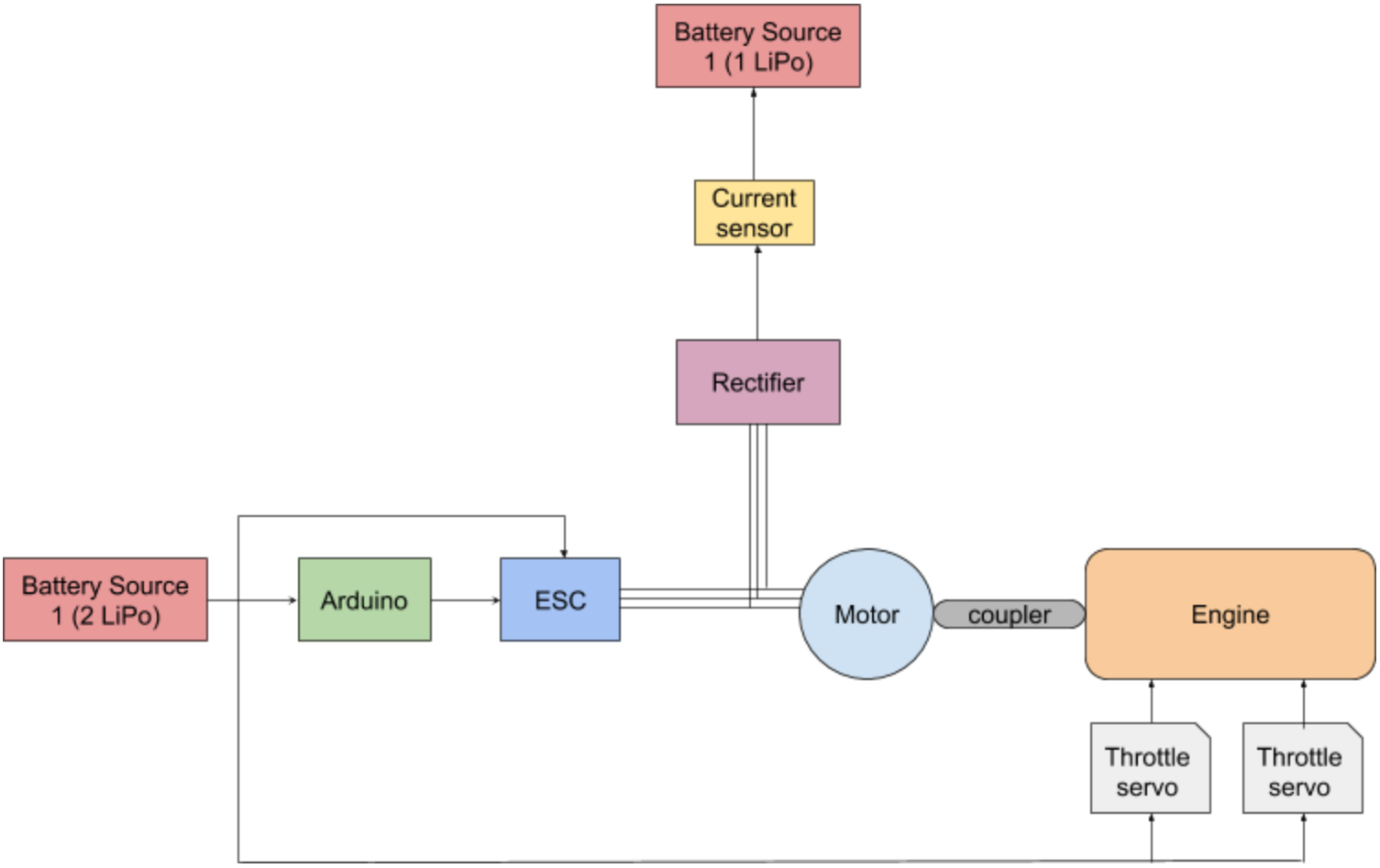
Below is a more detailed list of the components that make up the hybrid system, along with their purpose:

- **3 LiPo Batteries**: Two batteries run the motor (i.e. act as source 1), and one battery receives generated current (i.e. acts as source 2)
    - Two LiPo batteries were used as source 1 because of the difficulty of starting up the engine with only one LiPo. In general, the starting motor has to overcome the internal resistance of the engine when turning it over. This is a difficult process because of the vacuum generated in the cylinders. We found that we could only reliably start both cylinders if we used two batteries. However, if only one cylinder is used (by removing one of the spark plugs), the motor can reliably start the engine with one battery. This is ill-advised because removing one of the spark plugs will cause gasoline to spray all over the place.
- **DC Brushless Motor/Generator**: Used as the starting motor for engine as well as a generator
- **Twin Cylinder Engine**: a gas-powered two-cylinder engine that is used as the main power-generation component of the hybrid system

[1] The term “battery source” is used instead of “battery” because each source may be substituted with multiple batteries in series if needed. In fact, later we will talk about how we used two batteries as the first source to the motor.

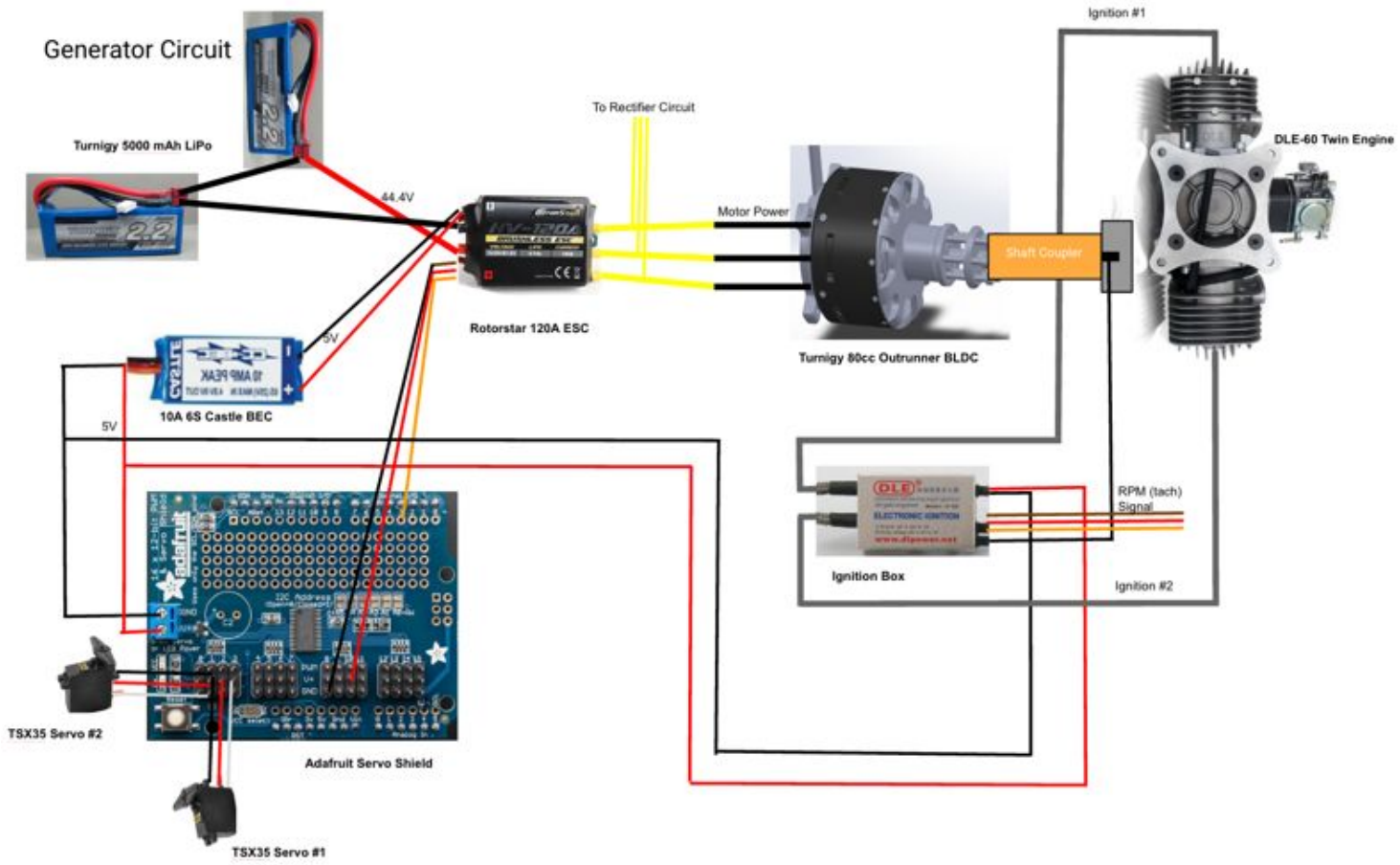
    - **Throttle valve**: controls the amount of gas in the air/gas mixture; a fully open throttle valve means the maximum amount of gas is in the mixture
    - **Choke valve**: controls the amount of air in the air/gas mixture; fully open means the maximum amount of air is in the mixture
    - **Tachometer**: measures the speed of the engine rotation; is internal to the engine but the signal may be retried from the ignition box
- **Ignition Box**: Fires the spark plugs based on the tachometer signal from the engine
- **Battery Eliminator Circuit (BEC)**: Provides 5 V power to the Arduino and ignition box
- **2 Servos (Throttle and Choke)**: controls the engine's throttle and choke valves which regulate the air/gas mixture to control the amount of combustion, which in turn controls the speed of the engine
- **Arduino Microcontroller**: the "brain" that controls the motor and engine and measures any diagnostic signals of the system
- **Adafruit Shield**: controls the I/O between the Arduino and the servos and ESC
- **Rectifier subcircuit**: converts 3-phase AC current generated by the motor into a DC current to charge a battery
- **Current Sensor**: Measures the current going from the generator to battery and rotors and outputs a signal voltage that is proportional to the measured current

Main Schematic



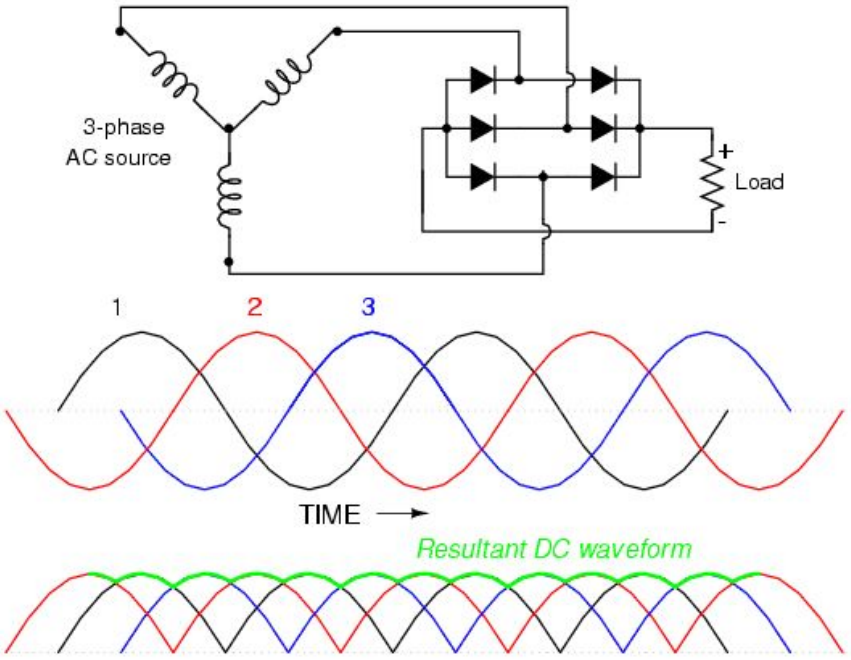
Main Connections

Below is a reference for how most of the components are connected (current sensor not pictured).



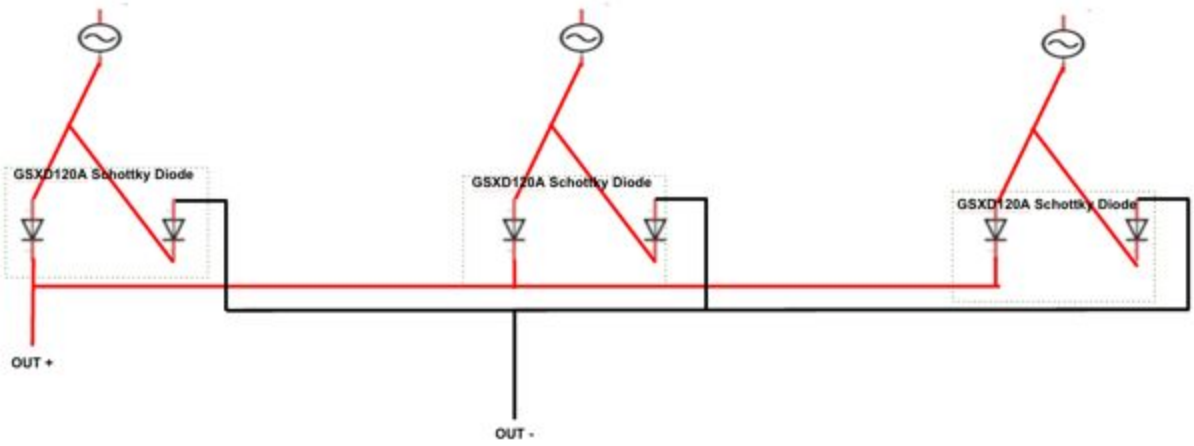
Rectifier Subcircuit

The rectifier serves the purpose of converting the sinusoidal current coming out of the motor to a DC current. A general schematic of a three-phase rectifier as well as its expected output is shown below.

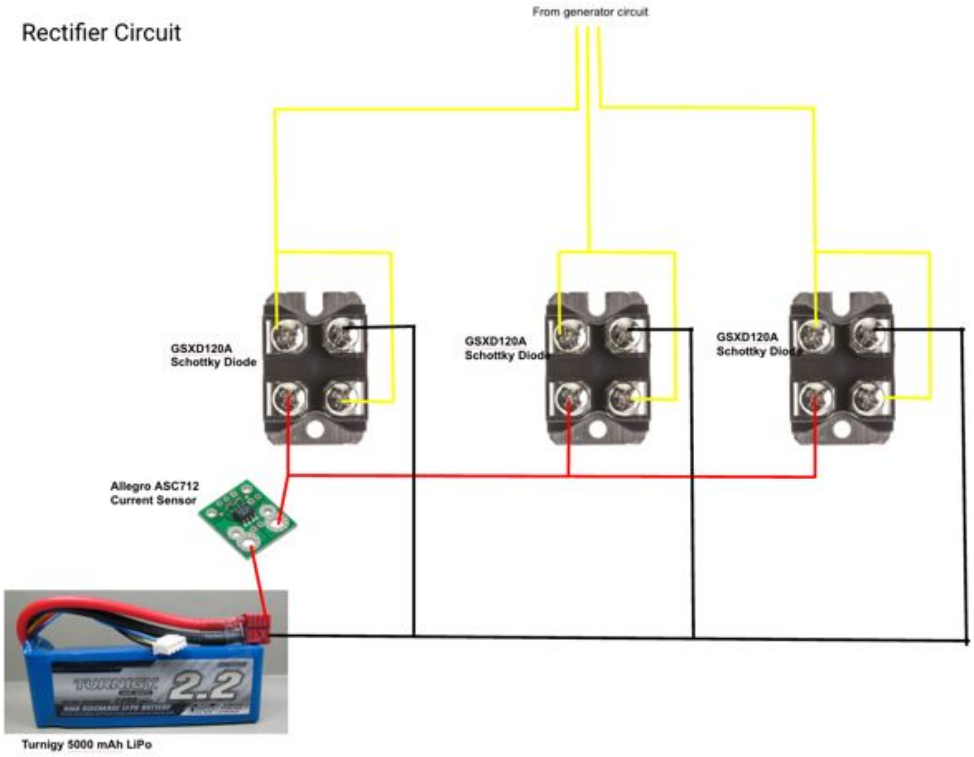


Source: http://i.stack.imgur.com/t37gL.gif

We used high-current Schottky diodes to construct our three-phase rectifier. The circuit diagram is shown below:

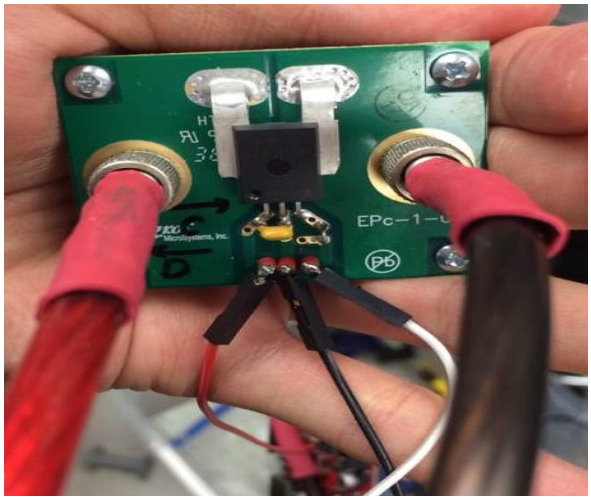


An alternate circuit diagram that shows the connections of the actual diode components and battery that we used is displayed below.



The Current Sensor

The main purpose of the current sensor is to ensure that the generator does not send too much current to the battery, which can potentially cause a fire. We used a bidirectional current sensor that is capable of measuring up to 200 A. It requires a separate power source to operate. The sensor is pictured below.

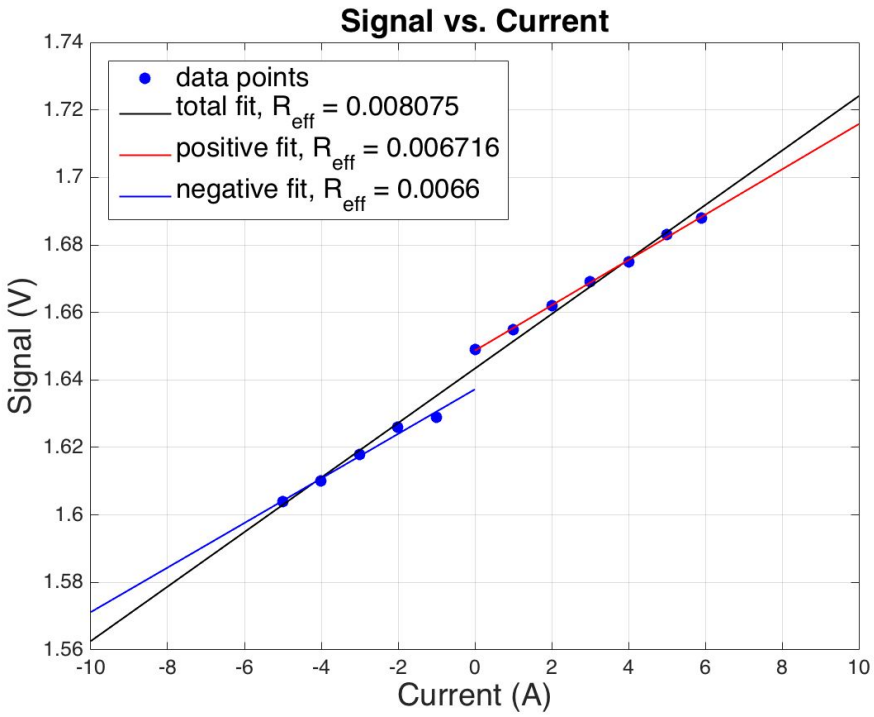


The data sheet for the current sensor can be found at this link:

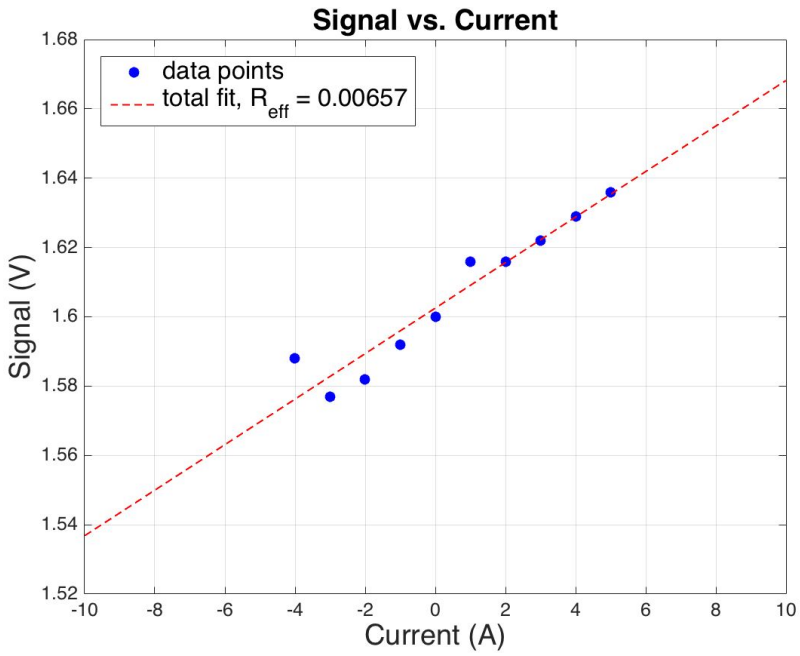
http://www.digikey.com/product-detail/en/allegro-microsystems-llc/ASEK759ECB-200B-T-DK/620-1611-ND/4867632

Characterizing the sensor

We tested the current sensor's output by connecting it between a LiPo battery and a LiPo charger. We used the charger to charge or discharge the LiPo battery, and selected the amperage. Then we compared the current sensors outputs for different charging and discharging currents. The results for trials on two different days are shown below. In both trials a 3.3 V power source was used. A linear fit of the signal voltage and the known charging or discharging current should yield a linear fit that shows the effective resistance R_{eff} of the sensor.



The plotted for the data collected on Trial Day 1. Here we show 3 different linear fits--one for positive current (discharging), one for negative current (charging), and one for all of the data.



The plotted for the data collected on Trial Day 2.

It appears that the effective resistance of the resistor inside the current sensor is ~0.007 Ω at a supply voltage of 3.3 V. This value should be used to convert the value of the voltage signal into the current that is flowing through the sensor. All of our data is documented in a MATLAB script which can be found on our github (see Documentation and Links).

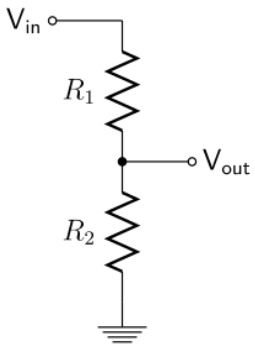
How it operates (known information + speculation)

A current sensor operates by essentially measuring the current going through a very small resistor. This resistor must have a low impedance so that little voltage is dropped across the resistor so as to not affect the circuit one is measuring. This small voltage is amplified and provided as the measuring signal of the sensor. Of course, there a lot more complicated parts of a current sensor including but it is not necessary to go into detail about this.

While testing the sensor we discovered one very important piece of information: The “base” voltage signal—i.e. the signal voltage when *no* current is flowing through the sensor—is half the supplied power. This relationship is fairly consistent, but begins to diverge at higher supply voltages above 8 V. It should be noted that the data sheet recommends a supply voltage of 3.3-3.6 V.

The current sensor likely works as follows: like most active elements, **the maximum voltage it can deliver is limited by the voltage it is supplied**. So if we provide 5 V relative to some ground to power the sensor, it can output a maximum of 5 V relative to that same ground. If it was a unidirectional sensor, it could simply map its maximum current reading of 200 A to the

maximum voltage it can output, which in this case is also 5 V. Then 0 A would correspond to a signal of 0 V. However, because our current sensor is bidirectional, it needs to essentially split up the voltage range it can provide so that half the range corresponds to a positive voltage reading and half the range corresponds to a negative voltage reading. This is analogous (BUT NOT EQUIVALENT TO) to having a voltage divider where $R_1 = R_2$, and the supplied power to the voltage divider is the power you supply to the current sensor, 5 V.



For this analogy, this means the the sensor's *signal* is related to V_{out}. In essence, this makes the zero-current voltage reading half the supplied voltage, and then maps the maximum current it can read (200 A) to the supplied voltage. So for a supplied 5 V, the zero-current reading is 2.5 V, the 100 A reading is 5 V, and the -5 A reading is 0. **So the current sensor will never provide a negative voltage signal, making it ideal for reading it with an arduino.**

This hypothesis is likely to be true but we were not able to verify it for larger currents (>6 A) because the LiPo charger would take too much time to "startup" charging or discharging at this current. One can theoretically check what values the current sensor gets for different generator speeds, but our group never got around to it.

Things to Note:

- Our team did not get around to integrating the code that reads in the current signal with the rest of our code that runs the hybrid engine. However, this can be done relatively easily by integrating the appropriate code from the `CurrentSensorTest.ino` into `GeneratorControl.ino` (see Code section).
- When conducting one's own characterization test, to measure charging current as a positive current, connect the thick red wire to the positive wire of the LiPo battery.
- The three thin wires should be connected to the Arduino. The red wire is Vcc(3.3v), black is ground, and white is current measurement output in a form of Analog voltage signal ranging from 0 to 3.3V.
- There is no need to connect resistor or capacitor to the white wire. As to the mapping between analog voltage from the thin white wire and actual current flowing through the thick wire is linear as shown in p.15 of the datasheet. Refer to

our attached MATLAB code to access the data we have collected on this current sensor.

- Data Collection : we collected these data by connecting the current sensor in series with a LiPo battery and LiPo charger. We charged and discharged the battery at various current levels.

How to Run The System

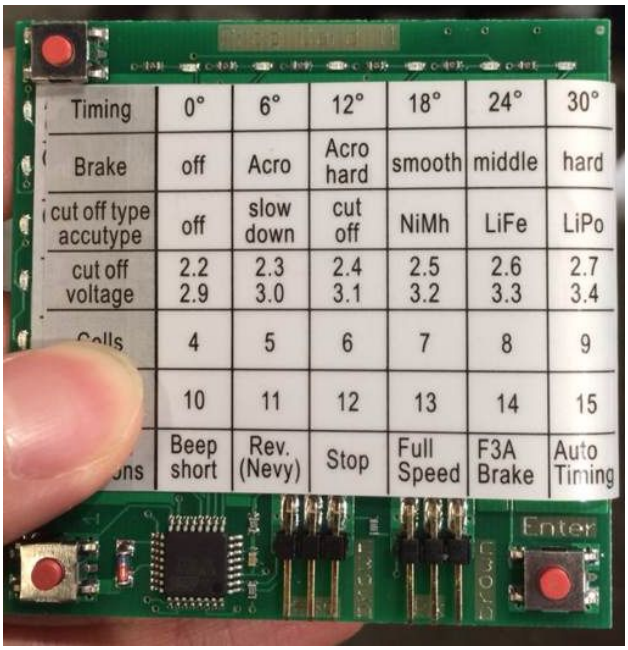
1. Connect the components as shown in the main circuit diagram EXCEPT FOR THE TWO SUPPLY BATTERIES GOING TO THE ESC. This will be saved for the final step.
    a. Make sure the ESC is connected to analog pin 3 on the Adafruit shield. Make sure to use pin 5 for the throttle and pin 6 for the choke pin.
        i. The reason these pins are used is because only certain pins on the shield operate at frequencies that are suitable for producing the pulse-width-modulation (PWM) signals that control ESCs and servos.
    b. Supply battery power to BEC and connect the BEC to the Adafruit shield's two screw terminals and the ignition box via the Y-cable. Alternatively, one can omit the BEC and use an external 5 V power supply.
2. Set the throttle and choke valves to produce a rich gas mixture to start the engine. This means the choke should be fully closed and the throttle should be fully open. This can be done by manually moving the valves by hand, or by using the servos to control them. However, if the servos are used, they must be calibrated first (see Servo Calibration section).
    a. Note: This setting corresponds to a rich mixture because it reduces the amount of air entering the cylinders which "richens" the air/gas mixture, which helps in starting the engine when it is cold. When the engine warms up, the choke should be scaled back so that the engine runs more efficiently.
3. Supply USB power to the Arduino with a computer. Both green and red lights on arduino should be on.
4. Open up the `GeneratorControl` script which can be found on our github. Upload just to make sure it is loaded onto the Arduino and open up the serial monitor.
5. Now one can connect the two series supply batteries to the ESC. Make sure you hear the correct tone from the ESC. For two 6s (6 cell) LiPo batteries, there should be 3 beeps.
6. Watch the serial monitor to view the initialization sequence. The engine should start before the completion of the sequence. IT IS IMPERATIVE THAT ONE IMMEDIATELY STOP SENDING SIGNALS TO THE MOTOR VIA THE ESC WHEN THE ENGINE STARTS. This can be done by by manually disconnecting the ESC and motor or shortening the initialization sequence in the code so that it is finished by the time the engine starts.
    a. IMPORTANT NOTE: If the engine is run without the rectifier and its battery load, it will fry the ESC!

b. Note: The electric motor can be run in either direction by switching two of the wires from the ESC. Make sure the electric motor is spinning clockwise to that it turns the engine in the proper direction.

Programming the ESC

An ESC must be programmed to operate with a specific battery and with specific settings based on what it is to be used for. It only needs to be programmed once but if one is using a new ESC or if the settings need to be changed, one needs to program it. We provide instructions on how to program the ESC below.

If the ESC is powered with at most six-cell batteries, then one can program the ESC using a joystick using the simple protocol illustrated in “ESCprogramming.pdf” in the “Our Documentation” folder of our Google drive folder (see Documentation and Links). However, because we use two 6s batteries (a total of 12 cells) we need to use ESC programming card which is shown below. This card is easy to use, and is ideal even in the case of using ≤ 6 battery cells.



Connecting the Programming Card

There are two 3-pin connectors as shown in the image (at the bottom of the program card). The left 3-pin connector is for the ESC receiver cable (which is the shorter of the two 3-pin female connectors on the ESC!) The right 3-pin connector on the card is to supply power to the card. Connect the middle pin to V_{cc} power and the right-most pin to the ground. The left-most pin is should be left unconnected.

Using the Programming Card

The three buttons are used to select the different settings based on the horizontal and vertical position of the blinking light. Make sure to select 12 cells for two LiPo batteries. Two helpful instructional videos are given in the links below:

- https://www.youtube.com/watch?v=4Kg7MzGwvdw
- https://www.youtube.com/watch?v=a7kbMhBaUA4

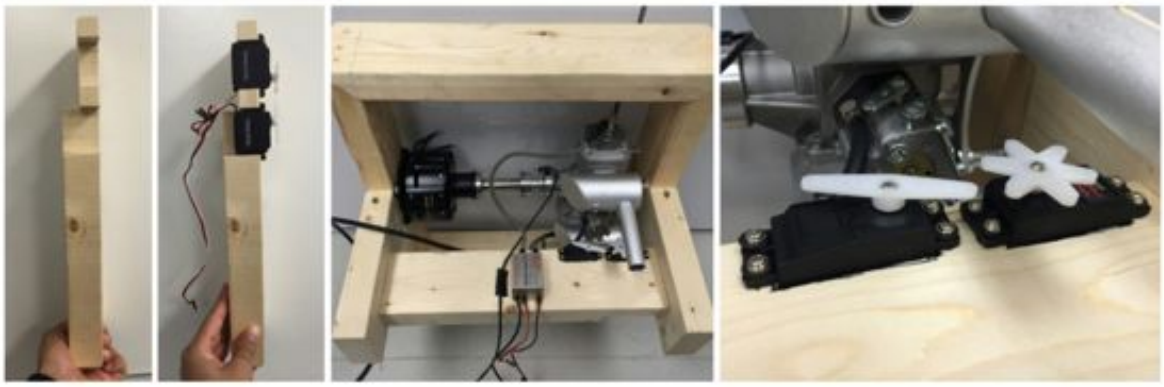
Make sure to program the ESC to turn its brake off. **This prevents the ESC from forcibly halting the motion of the motor when the engine begins to turn the motor and signals are no longer being sent by the ESC.**

Mechanical Frame Design

While the frame we have designed for our proof-of-concept hybrid engine is fairly lightweight, this was not the primary objective of the frame design. The main purpose of the frame in this rendition of the hybrid engine is for safety, stability, and testing jig[2]. This was designed with the following in mind:

- Resist the torsion of an engine spinning at 4000 rpm
- Be modular and easily modifiable for removal or addition of hardware (especially the engine and motor)
- Be fire-proof and sturdy

The previous design was constructed from wood. Its weaknesses were multifaceted - unsturdy, asymmetric, missing bolts, and susceptible to fire damage.



Predecessor from 2015.

[2] It is important to design the frame to withstand the vibrations of the engine and motor.

Features of the new frame include:

- **Sturdy 8020 aluminum extrusion construction**. T-Slots were cut and tapped by 8020.net, and thru holes were made at the Duke Machine shop. Joints were made with a washer and ¼"-20 set screw assembly or economy t-nut and ¼-20" socket cap bolt
- **Engine Mount**. The square, aluminum engine mount must be re-drilled so that the thru-hole diameter is 0.25". This was secured with the the aluminum standoffs going into the t-nuts. Note that this assembly may need to seek tightening.
- **3D Printed Mounts**. Mounts were modeled in SolidWorks and were fastened with ¼-20" socket cap bolts.
    - 1L Gasoline Fuel Tank - modeled for a loose sliding fit
    - DLE V2 Ignition Box - modeled for a firm fit
    - Double Servo Mount - modeled to be screwed in with #3 machine screws and nut
    - Arduino Mount - modeled to be mounted at three positions in the board
    - ESC Mount - modeled to be snug with the 120A ESC
- **Lovejoy Coupling**. A 10 mm inner bore jaw was used for both shafts. A 18-8 cone point set screw was used to plastically deform the shaft to couple both shafts together. This set screw was purchased separately as the cup point set screw was worn out after an initial trial. Standard nitrile was used for the spider. Of note, the coupling does fail because too much torque is generated. This coupling needs to be replaced.

The models for these components are available in a separate folder of our google doc (see Documents and Links section.) For the eventual use of this assembly on the actual 18-rotor copter, the following assembly has been proposed. The justification or the aluminum extrusions being used is that carbon fiber would not be able to withstand the torsional load. The penalty for this is that 5 additional pounds of weight must be added.

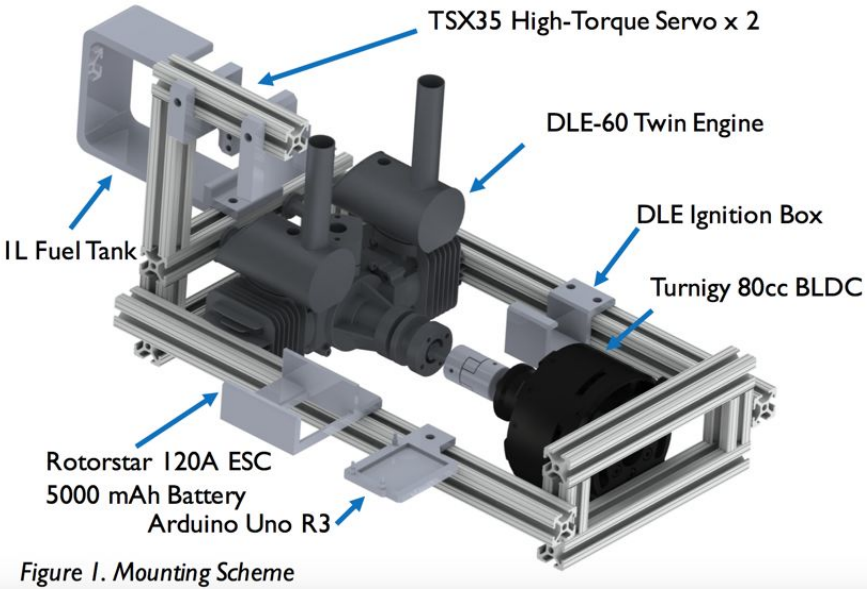


Figure 1. Mounting Scheme

Code

We have written a number of scripts to test and run the hybrid engine, as well as a MATLAB script to record and plot the data we acquired when testing the current sensor. They can all be found at our github (see Documents and Links section).

Here is a basic outline of what each script does:

1. **`CalibrateServo`**: code to calibrate the movement of the throttle/choke servos. By finding the pulse lengths/angular positions that correspond to the throttle/choke valves being fully open and closed, one can find what pulse length one needs to send to get any degree of "openness."
2. **`CurrentSensorTest`**: code to read the signal voltage from the current sensor
3. **`GeneratorControl`**: starts up the entire system (without readout from the current sensor)
4. **`MotorTest_Adafruit`**: tests sending PWM code to the ESC using the Adafruit servo library; has since been rendered obsolete
5. **`MotorTest_Arduino`**: tests sending PWM code to the ESC using the Arduino servo library; was successful and was incorporated into **`GeneratorControl`**
6. **Old Code**: folder that contains all of the code from the previous year's power electronics team.
7. **`ServoTest`**: meant to test the servos; based on the minimum and maximum pulse lengths found in the calibration, this tests to see how the servos perform when sending a percent from 0% to 100% (fully closed or fully open)
8. **`Tachometer`**: meant to view the signal coming from the tachometer readout of the ignition box
9. **`AllegroCurrSensorData`**: a record of the data and plots that we obtained from the current sensor in our final system
10. **`TBScurrSensorDataRun`**: a record of the data and plots that we obtained from the first current sensor that we used; given that it was unidirectional and difficult to use, we decided not to use it

Running the Code

How to run any of the code (in order!!!):

1. Upload to the Arduino
2. Plug in the supply batteries (source 1)
3. Open the serial monitor

Calibrating the Servos

There are two servo arms which allow the servos to control the throttle and choke valves on the gas engine. To calibrate, one has to figure out what angles/PWM pulse lengths correspond to the throttle and choke being fully open and fully close. These can then be used to map a given percentage of "openness" to the range between fully open and closed. This mapping function is already provided in the code.

To calibrate one servo:

1. Plug one servo into Arduino pin 6.
2. Run the servo calibration code. Input various pulse lengths into the serial monitor in MICROSECONDS from 1000 to 2000.
3. Find the pulse length at which the valve transitions to being fully closed, i.e. when no more physical movement can occur to make it more closed.
4. Find the pulse length at which the valve transitions to being fully opened, i.e. when no more physical movement can occur to make it more open.
5. Use these as the limits for the PWM pulse in `ServoTest` or `GeneratorControl`. These scripts now map the limits to 0% and 100%. Note that a 50% pulse in these scripts (which is inputted into the serial monitor as 50) corresponds to a pulse length that is halfway between the limiting pulse lengths.

For the Future

This is a list of things we would have liked to have done but didn't get around to:

1. Modifying the `GeneratorControl` code to stop the ESC from sending signals to the motor once the gas motor has started so that it doesn't have to be manually disconnected each time. This would involve using the tachometer signal to figure out when to tell the ESC to stop sending signals.
2. Integrating the current sensor readout into the code.
3. Optimizing the servo mount and servo pins for better control of the valves.

4. Figuring out how to control the speed of the engine using the
5. Characterizing the power output of the generator.
6. Implementing an emergency kill-switch in case too much current is sent to the charging battery (source 2).
7. Rebuilding the frame to match our design for incorporation into the drone.

Documentation and Links

All of our code may be accessed at: https://github.com/tamran/Team-Hydra

The documentation of our meetings, links to references, and our videos and pictures can be found on our Google drive folder here: https://drive.google.com/open?id=0B9PuS159UEkGa3ZVZEJ0bGZIQVk

Engine manual: http://manuals.hobbico.com/dle/dleg0060-manual-v2.pdf

TEAM HYDRA