Daniel Truong
ENGR 396

PWM shields, Motor Controllers, Low Pass Filters, Oh My!

The Adafruit 16-channel PWM/Servo Shield (or just PWM shield as Allison likes to call it) has a reliable history in the hardware box. Originally an auxiliary component for the old hardware box's motor controllers, it now is used for the new hardware box with an altered purpose. In this tutorial, I shall provide an overview of PWM and how it functions for both the old and new hardware box, explain how the PWM servo shield assists the main Arduino's job of sending signals to the motors controllers, and explain how to convert PWM signals to analog using a low pass filter. This documentation was written for audiences who are completely new to the concept of a motor controller and the different types of signals a motor controller can be controlled with. In some cases, it also acts as a log, commenting on the interesting and frustrating paths taken at this stage of the project. If you are no novice to the topics mentioned, skip to the "Putting it All Together" section. Otherwise, I hope I can enlighten you to more than just PWM shields in this tech doc.

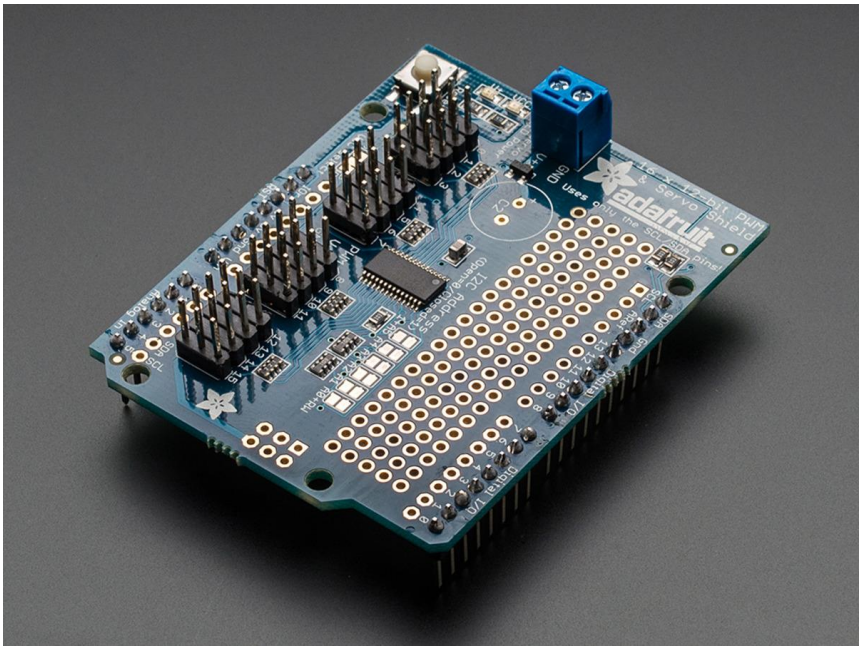

Figure 1: An Assembled PWM Shield

The Need for a Motor Controller

Our current thrusters operate at a maximum voltage of 36V. To control the thrusters, voltage with a certain magnitude and polarity needs to be sent to the thrusters which controls both the speed and direction respectively of the thrusters. For example, if you want to make the thrusters go half its speed forwards, you would send half its maximum voltage to the thrusters (18V) and make sure the polarity of the voltage is positive. To make it go a quarter of its speed in reverse, you would send a quarter of the maximum voltage (9V) and make sure the polarity of the voltage is negative.

Typically, part of the control systems for a motor uses a microcontroller, like an Arduino. However, the maximum voltage an Arduino can take is 12V. Even worse is an Arduino can at max output 5V when it sends out signals. Thus, if your thrusters were connected directly to an Arduino, you would go less than a quarter of the possible speed. (You are sending 5/36th of the

PWM shields, Motor Controllers, Low Pass Filters, Oh My!

possible voltage to the thrusters.) If you tried to send your 36V power supply through the Arduino in hopes of exceeding the 5V output, all you will get is the smell of fried electronics. How then can we control the voltage entering a large component like a thruster without breaking our microcontroller? Perhaps, the answer is a component, which scales a large input voltage (like 36V) based on a range an Arduino typically operates (0V-5V). Such a component is called a motor controller**. Motor controllers** take in a power sources voltage and current as an input and outputs a portion of the voltage depending on a signal a microcontroller like an Arduino sends it. Most motor controllers are also able of switching voltage polarity as well.

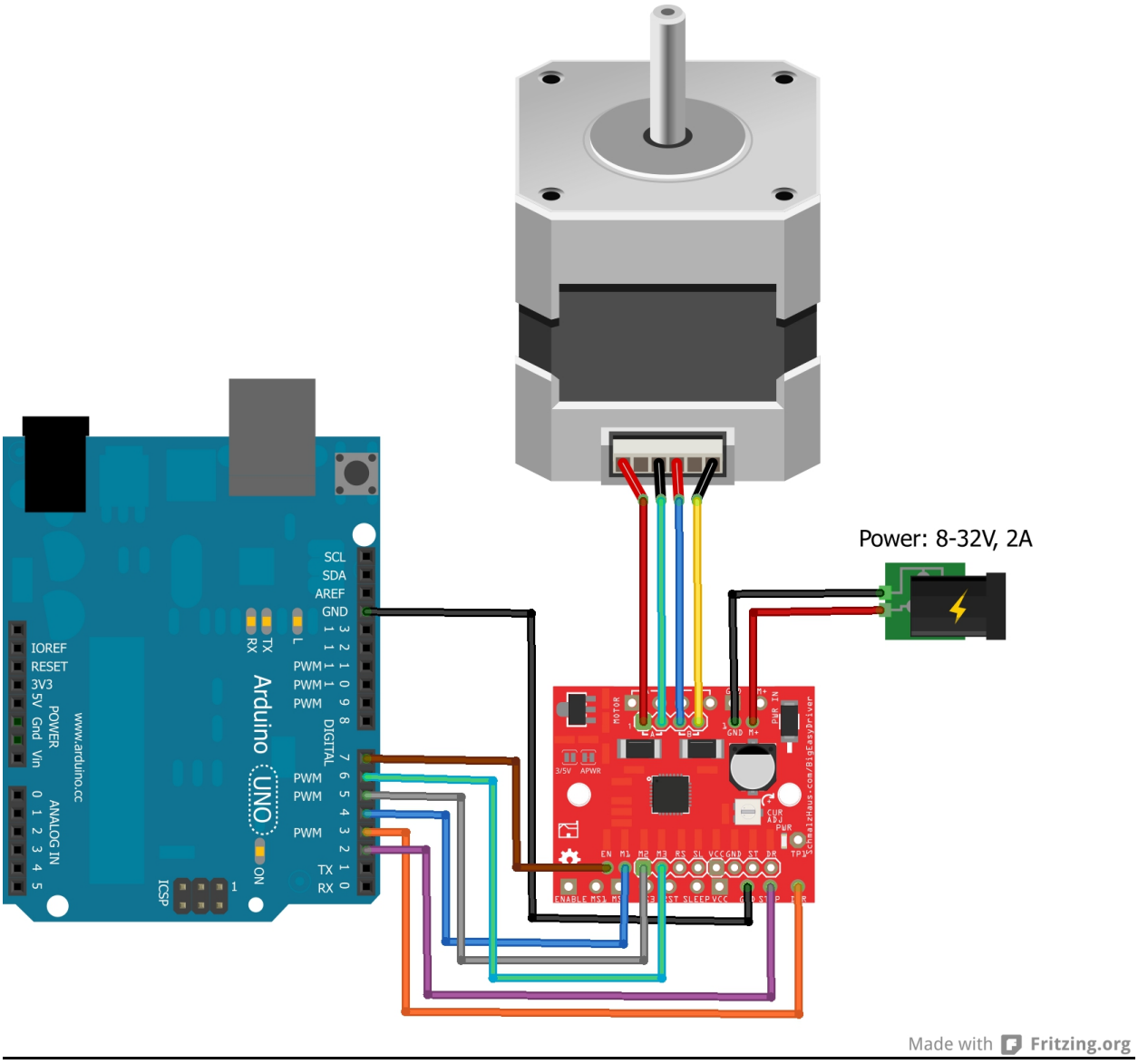

Figure 2: Typical arrangement of motors, microcontroller (Arduino), batteries, and motor controllers (red board).

Daniel Truong
ENGR 396

PWM shields, Motor Controllers, Low Pass Filters, Oh My!

The Old Hardware Box and an Intro to PWM signals

Despite being a relic of the past, it is useful to explain the function of the PWM shield in the old hardware box. My history with the PWM shield begins in June 2018, the month I joined team Kanaloa. At the time, the thrusters of the WAMV operated at a much lower voltage and life was much simpler. Motor controllers, like thrusters, have a maximum voltage for input. The old high current box used REV Robotics Spark Motor Controllers which had a maximum voltage of 12V. (same as our thrusters' max voltage at the time) In order to control the motor controllers in the old high current box, a PWM (Pulse Width Modulation) signal was required.

Before explaining what any type of signal is, lets provide a general definition for a signal; **signals** are typically variations in voltage over time that carry information. This definition will make more sense as you see the different types of signals. Back to PWM! The best way to visualize PWM signals is to first explain digital signals. **Digital signals** can best be described as a square wave. With a digital signal, there are 2 possible states represented as 2 plateaus: on (1) or off (0). An image of a digital signal is shown below. One thing to note is the length of a plateau can vary; duration the signal is in the "on" position does not need to match the duration the signal is in the "off" position. A **PWM** signal is like a digital signal switching between on and off at high frequency.

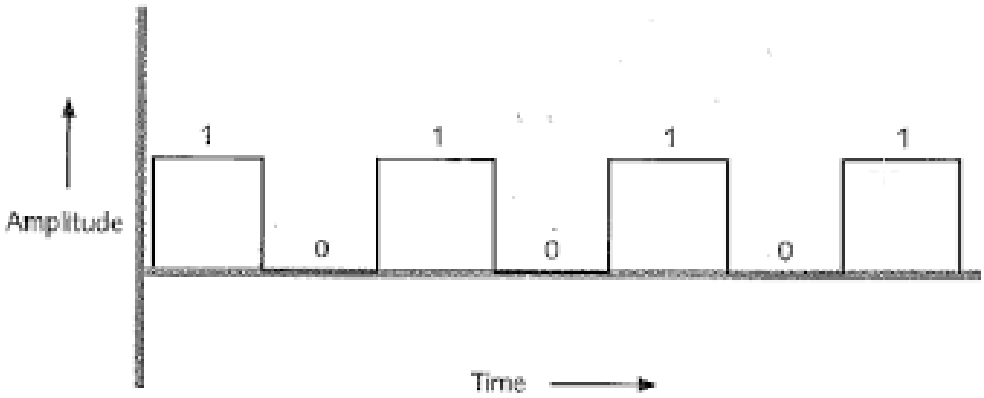

Figure 3: A digital signal. A digital signal has 2 states (plateaus), the on (1) state and the off (2) state. The on state is the max voltage for the digital logic (range of voltages possible). Typically this will be 5V. The bottom plateau typically is 0V.

There are 3 components to a PWM signal: amplitude, frequency, and duty cycle. **Amplitude** is the maximum voltage for the peaks of a PWM wave. Like a digital signal, typically this value is 5V. A **duty cycle** is the percentage of the time the PWM wave is at max voltage in a period. Looking at the digital signal from earlier, it appears that half the time the signal is in the "on" position and the other half of the time it is in the "off" position. For this wave, the duty cycle would be 50%. **Frequency** is the number of PWM waves which occurs per second. In the picture below, a period is the duration between 2 lines (total 5 periods). Frequency is the number of periods per second.

PWM shields, Motor Controllers, Low Pass Filters, Oh My!

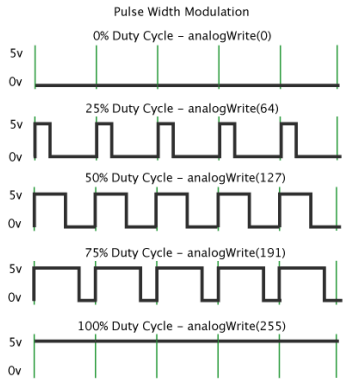

Figure 4: Various duty cycles. analogWrite is an Arduino function which controls PWM signals (more on that in a later section).

The main pinouts of an Arduino use digital signals, which has 2 states: max voltage and zero voltage. Using such signals for the **logic** (range of voltage values which convey information to a component.) of a motor controller would cause the thrusters to go either max speed or neutral. PWM allows an Arduino (and other microcontrollers) to reach those intermediate values of the motor controller logic. For example, if the thruster needs to go half speed, we must output a voltage that is half the Arduino's logic (2.5V) to the motor controller. To accomplish this, we must alter the duty cycle.

Sending an "on" signal using digital replicates the 100% duty cycle in PWM. To get 2.5V we should then change the duty cycle to 50%. Even though a PWM wave can be described as a digital signal switching between 5V or 0V at an orderly rate, typically the frequency of the signal is high enough to cause the device receiving the signal to perceive the average of all PWM waves entering. From the earlier example for 2.5V, because the receiving device perceives 5V only 50% of the time, the average would be 2.5Hz.

An Arduino on its own is capable of outputting PWM signals using analogWrite() with certain pins. For the Arduino Mega, these pins are 2 to 13 and 44 to 46. However, the frequency of PWM signals from an Arduino is difficult to configure; manipulation in code is required, usually with mixed results. Earlier experiments showed that the frequency of PWM signals from an Arduino Mega (the type of Arduino for controlling the motor controllers as of Fall 2018) typically is 490Hz (except for pins 5 and 6 which operates at 980Hz). The typical frequency of PWM inputs the Spark Motor controllers operated at was 50Hz. If we couldn't change the frequency of the Arduino, perhaps we need a separate component for PWM signals with configurable frequencies… Enter the PWM shield!

The PWM Shield

The **PWM shield** is a component which replaces an Arduino's PWM pins. As a shield it plugs into the pins on top of an Arduino. An Arduino can communicate with the PWM shield because it uses an interface called I2C. (Interfaces by the way are just a set of rules devices use to communicate with each other.) In a possible sequel to this guide, I will go in further depth about I2C; Devices that use I2C to communicate to an Arduino require 4 special pins: Power (5V or 3.3V), Ground (GND), SDA (Pin 20 on a Mega), and SCL (Pin 21 on a Mega). When you plug in the shield, ensure that these 4 pins on the shield correspond to the Arduino Mega.

PWM shields, Motor Controllers, Low Pass Filters, Oh My!

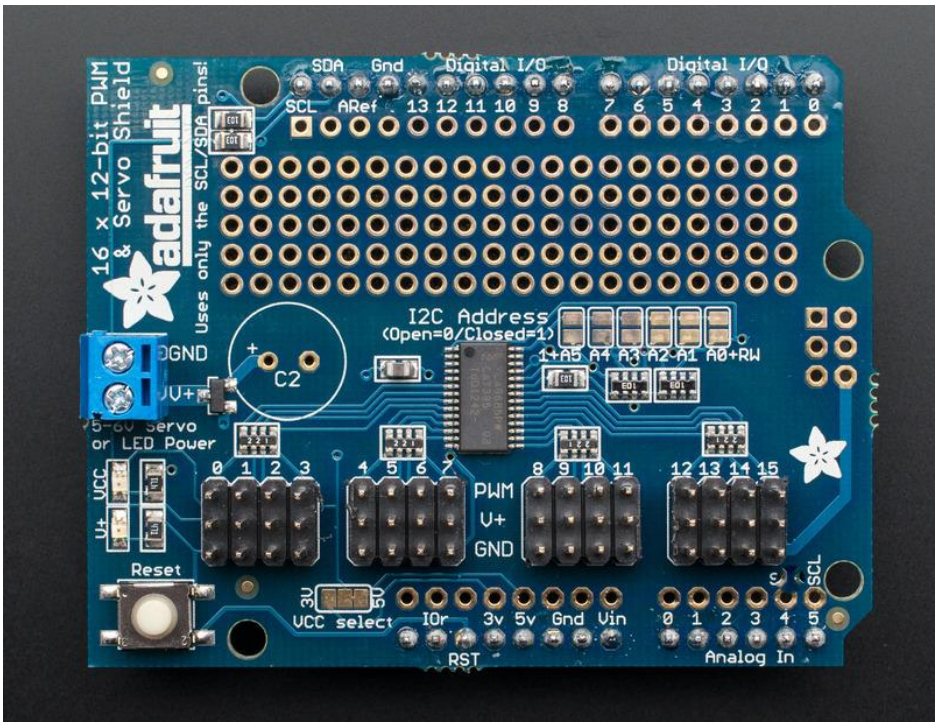

Figure 5: A top down view of the servo shield, when installing this component onto an Arduino Uno or Mega
ensure the 5 or 3.3V, GND, SDA, and SCL pins does go into those spots on your Arduino

In terms of coding, the PWM shield does have a library of ready-to-use functions for your needs. Before diving into these functions those, I will need to explain a new concept called resolution. **Resolution** is usually a term to describe how defined an image is. When browsing a Youtube video, you may notice that sometimes videos appear blurry or the opposite; sometimes a video looks almost like real life. The next time this happens, try pressing the little gear near the bottom right corner and check the quality. You will notice you have options like 240p, 1080p, 360p; all these options represent the number of pixels which make up each frame of that video. You may probably be able to guess that the more pixels used in the video, the more defined everything looks. This concept also applies to PWM signals.

Resolution in the context of PWM signals is the size of the intervals between the minimum value (0V) and maximum (5V). Let's look at an example. When an Arduino uses analogWrite to produce PWM signals, it has a resolution of 8 bits. 8-bit resolution means whatever range of values you have, like between 0 and 5, it is represented as a value between 0 and 255. (or 0 to 2^8-1) Representing 1 on the original scale would be 51 on the 8-bit resolution scale. If you look at the documentation for analogWrite on the Arduino website, the first value is what pin you want the signal to come out of, the second is called value. Value refers to resolution. If we want to output a duty cycle of 10% (or we want 0.5V for our PWM signal) the value we would choose 26 as our value. This is because 26 is approximately a tenth of 255. With the PWM shield, the resolution

PWM shields, Motor Controllers, Low Pass Filters, Oh My!

is 12 bits; this means a range of values can be broken down and represented as a number between 0 and 4095. (or 0 to 2^{12}-1) As a result, the PWM shield is a higher resolution than the Arduino in terms of PWM. Another way to describe it is this shield is able to have duty cycles 16 times smaller than with the Arduino alone. With that explanation out of the way, the image below is an example of using the PWM shield library. For a more in-depth look at each method called, I would recommend visiting the PWM shield's github.

```
#include <Adafruit_PWMServoDriver.h> //Library must be added either from website
                                     //Arduino IDE Library Tool
Adafruit_PWMServoDriver pwm;       //Declaring an object (C++ stuff), just remember
                                   //this line

void setup() {
  // put your setup code here, to run once:
  pwm.begin(); //Initialize I2C communication with PWM servo shield. (only needed once)
  pwm.setPWMFreq(50); //Sets frequency of PWM signal in HZ (optional line, default is 1000Hz)
}

void loop() {
  // put your main code here, to run repeatedly:
  pwm.reset(); //Resets the chip on the shield that produces PWM signals.
  pwm.setPWM(4,0,2048); //Sets PWM output for a (pin,lowest PWM value usually 0, max resolution value).
                        //Here duty cycle = 50% for pin 4 on the shield
  pwm.setPin(5,1024,true); //Another way to set a PWM output (pin #, max resolution value, true or false)
                           //Third parameter determines whether PWM signal needs to be inverted.
}
```

The image above features most of the key points in the comments. Some things worth mentioning though is for methods like setPin and setPWM, the pin in question is not on the Arduino, it is one of the columns with 3 male pins sticking out (More on this in the next paragraph). Also in retrospect, using pwm.setPin would have been more convenient than pwm.setPWM. The advantage of pwm.setPWM is if for some reason, you did not want to make the low value (bottom plateau) of the PWM signal not at 0V. This feature was not taken advantage of, which is why pwm.setPin would have been more convenient.

As for sending out PWM signals, this part is also relatively simple. All that is required is 3 wires with female headers. Near the bottom of the shield you may have noted a section with 4x3 blocks of pins sticking out. To get a PWM signal from one of these sections, you must first pick a column.

PWM shields, Motor Controllers, Low Pass Filters, Oh My!

Figure 6: Close up of the PWM pins on the shield

I will pick the 7th column. Each pin represents a component for the signal output. The top pin (pin closest to the number 7), is the actual PWM signal. The middle pin, (near the V+) is the reference voltage. This voltage is for determining the amplitude of the PWM signal and typically will be 5V. The bottom pin (with GND next to it) is ground and it is the same ground as the Arduino Mega. To transmit a signal, you simply put 3 female headers onto each of the pins in column 7, and then connect those wires to the corresponding parts on the motor controllers.

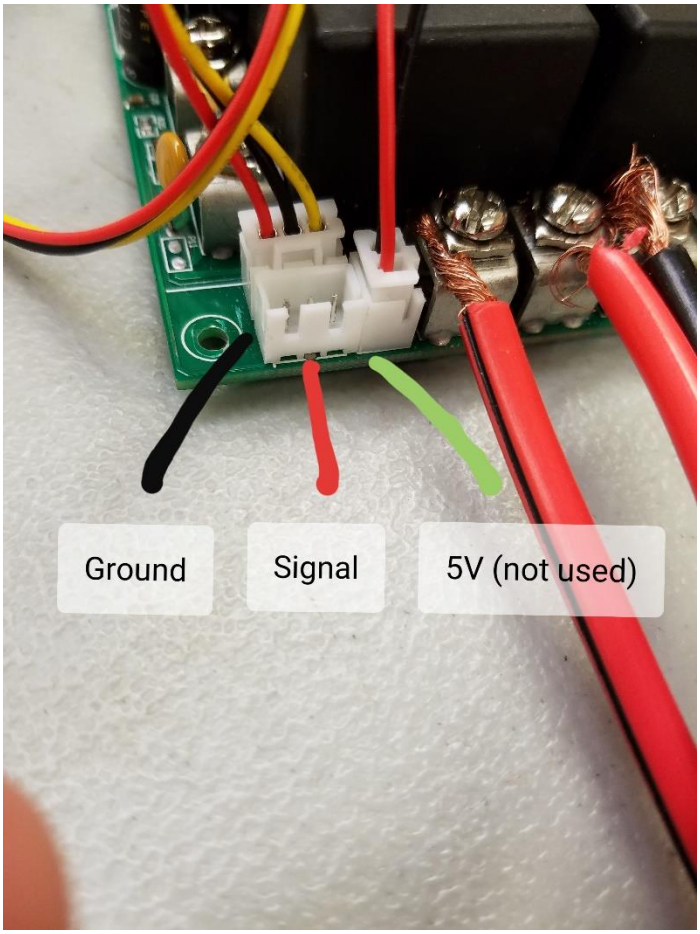

Figure 8: Example input port for a motor controller. Please ignore the not used message.

PWM shields, Motor Controllers, Low Pass Filters, Oh My!

The New High Current Box and Low Pass Filters

The current state of the new high current box is ambiguous at best. When writing the previous sections of this guide, the PWM shield was the method used for controlling the motor controllers. Recently it was removed and replaced with the control Arduino Mega's PWM pins. Why then would I spend so much time talking about the PWM shield if it isn't even being used in our system? In my opinion, the PWM shield provides superior control over PWM outputs compared to the Arduino's pins. As stated, the new high current box is still in troubleshooting, with components being added and removed constantly until a solution can be found for the problem (these issues will be later covered more in depth in another guide.). Its possible the PWM shield will return. Whatever options may be taken for components in the high current box, what remains constant is there will be a motor controller and that requires either PWM signals or analog signals to control. If the motor controllers use PWM signals to control, then it will be simple; it will be like the old high current box. But what if the motor controllers use an analog signal? In this section, I will cover low pass filters, which is a method for turning PWM signals into analog signals.

Analog signals are like the opposite to digital signals. If signals were people, digital signals are those type of people who only see the world as black and white; analog signals are more understanding. While digital signals can only have a set number of values, analog signals can have a whole range of values. The figure below is a representation of analog and digital signals. While the digital signal does resemble a square wave, the analog signal looks like the typical sine wave.

To those experienced or observant enough, you may have noticed the digital signal looks like a PWM signal; you would be correct. Way back in the first section, I mention that PWM is essentially a digital signal switching between high and low at high frequencies. You may have also noticed the variation in the pulse length corresponds to. To explain this phenomenon, it would be useful to note that PWM is a pseudo-analog signal, the Arduino's way of representing an analog signal digitally. The receiving device of a PWM signal would see the average of the pulse widths. Thus, when the PWM signal wants to represent a frequency close to ground, it will decrease its pulse width as much as possible.

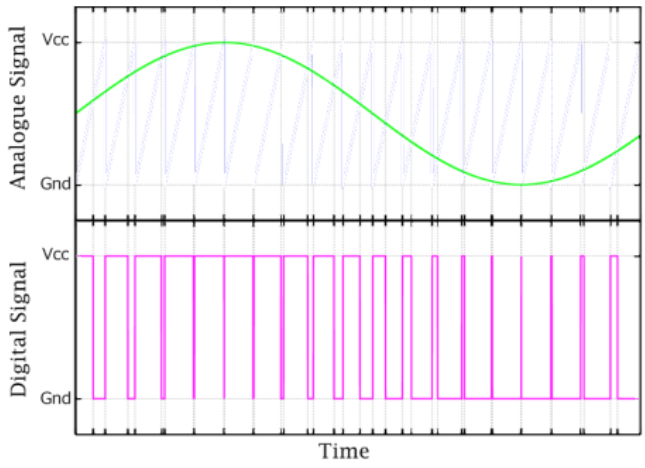

Figure 9: An analog signal and how that same signal would be represented digitally

PWM shields, Motor Controllers, Low Pass Filters, Oh My!

Why then is it necessary to have analog when PWM allows a digital signal to be represented as analog? For the purposes of this guide, the simplest answer is the internal circuitry of the motor controllers at the time of writing this was not designed for PWM signals. Thus, perhaps a device that can turn signals from digital signals to analog signals would be useful… Actually, we tried using Digital to Analog Converters before. It didn't end well. (Guide on that will be released on a future date.) The next best thing is to use a device called a low pass filter.

Low Pass Filter

Low pass filters are sometimes called "the poor man's digital analog converter". The construction of a low pass filter is relatively simple, but planning is required to properly create one for PWM signals. Below are the typical schematics of a low pass filter.

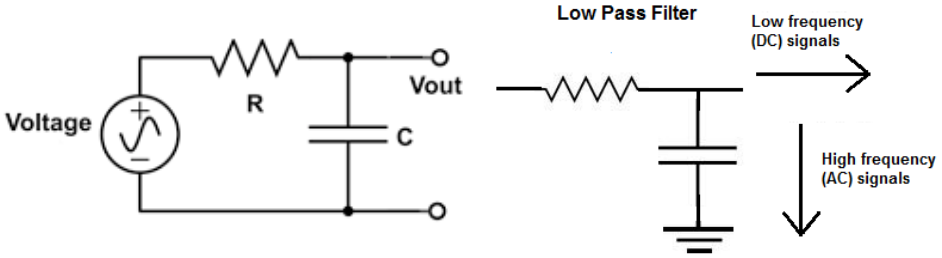

Figure 10: (Left Figure): A low pass filter with a signal source. Figure 11 (Right Figure): A low pass filters outputs

As you can see, all that is required is a resistor (squiggly with the "R" next to it in the left figure 9) in series with a signal source and a capacitor (two parallel lines with the "C" next to it in figure 9) connected in parallel with the source. How the low pass filter works (and to any dedicated EE's out there this is an oversimplification.) is the resistor limits the flow of current; this resistance is constant. The capacitor also can limit current flow, but its resistance is dependent on the frequency of the signal going through. As the right figure shows, high frequency signals can pass very easily through the capacitor, thus will be more likely to go down the path with the capacitor than the path labelled Vout. Charges that goes to the capacitor is stored in the capacitor (this fact is important later.) Frequencies that cannot make it to the path with the capacitor exit the path labelled Vout; this path is where the filtered PWM signal goes through. The frequency that the low pass filter begins filtering at is known as the cutoff frequency, which can be calculated using the formula below.

$$f_c = \frac{1}{2\pi RC}$$

The cutoff frequency (f_c), which is the frequency when filtration begins occurring, is inversely proportionally to the resistance (R in ohms) and capacitance (C in farads) of the resistor and capacitor respectively. The term "RC" is sometimes referred to as the time constant (τ); this value is how long it takes for the capacitor to charge up. When designing a low pass

PWM shields, Motor Controllers, Low Pass Filters, Oh My!

filter for a known frequency, there are nifty calculators online that let you determine either the required resistance or capacitance needed for a certain cut off frequency. (Links to such calculators will be in the further resources section). The strategy when using such calculators though is to look at what resistors and capacitors are available to you and then combine those values and resistances together to see if the cut off frequency is close to what you want. (For the sake of prototyping boards, you are trying to use a single capacitor or resistor).

Low pass filters are a concept that are not limited to PWM; they are useful to us though because they can filter out any high frequency components of a signal a output a non-oscillating signal. But what happens when you pass a PWM signal into a low pass filter? The figure below is what an oscilloscope might see when probing a PWM signal passed through a low pass filter. As mentioned earlier, when any signal is passing through the low pass filter, the high frequency component is more likely to go through the path with the capacitor. This causes the capacitor to charge up, which is the inclining part to this plot from 0 to 0.0005s. When the capacitor does charge up after 0.0005s, we see the voltage has plateaued but still shows some type of oscillation at a specific voltage. These oscillations are known as **voltage ripples**. In this case, the frequency of the signal is less than the cutoff frequency, so the signal is not completely filtered.

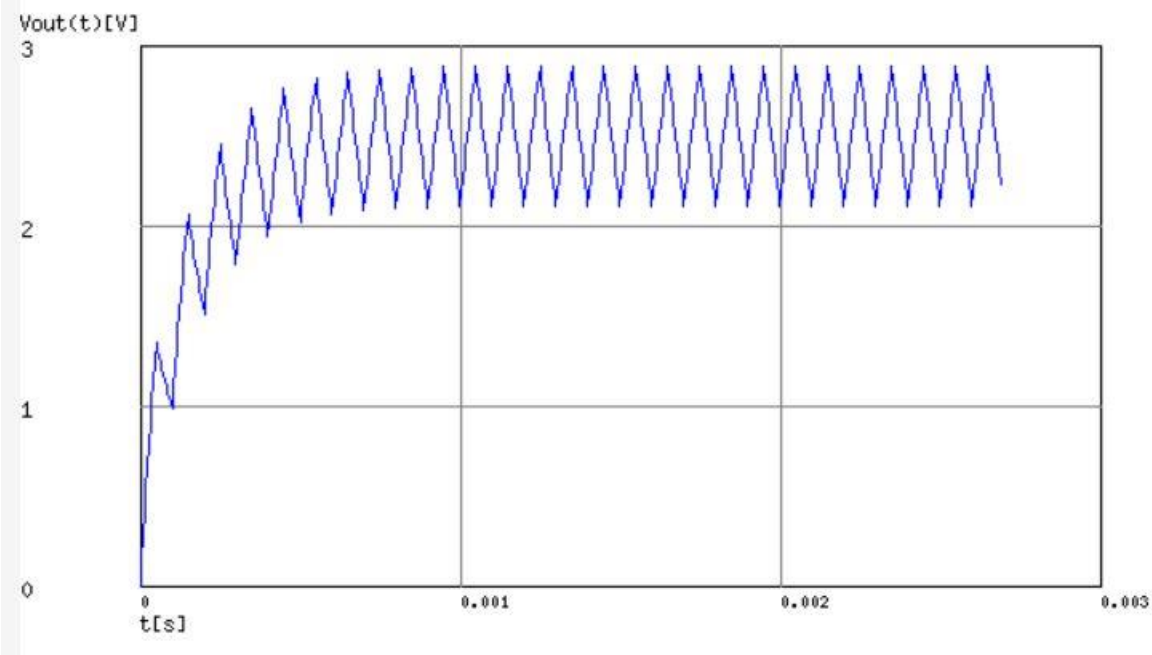

Figure 12: PWM signal with frequency less than the cut off frequency.

Below is another case, for a low pass filter with a cut off frequency designed to be below the signal frequency. The time constant here is greater than the previous case. The first thing to note is that the amount of time to charge the capacitor increases. Once it finishes charging though, we see the signal has been significantly filtered. It looks like an analog signal! By increasing the time constant further, the voltage ripples can almost be nonexistent. The trade off however is the initial charging period of the capacitor, which increases the amount of time it

PWM shields, Motor Controllers, Low Pass Filters, Oh My!

takes for the analog signal from reaching its intended voltage. When designing low pass filters for the WAMV though reader, this concern can be dismissed. The “electrical inertia” of the signal is significantly smaller than the mechanical inertia of the WAMV.

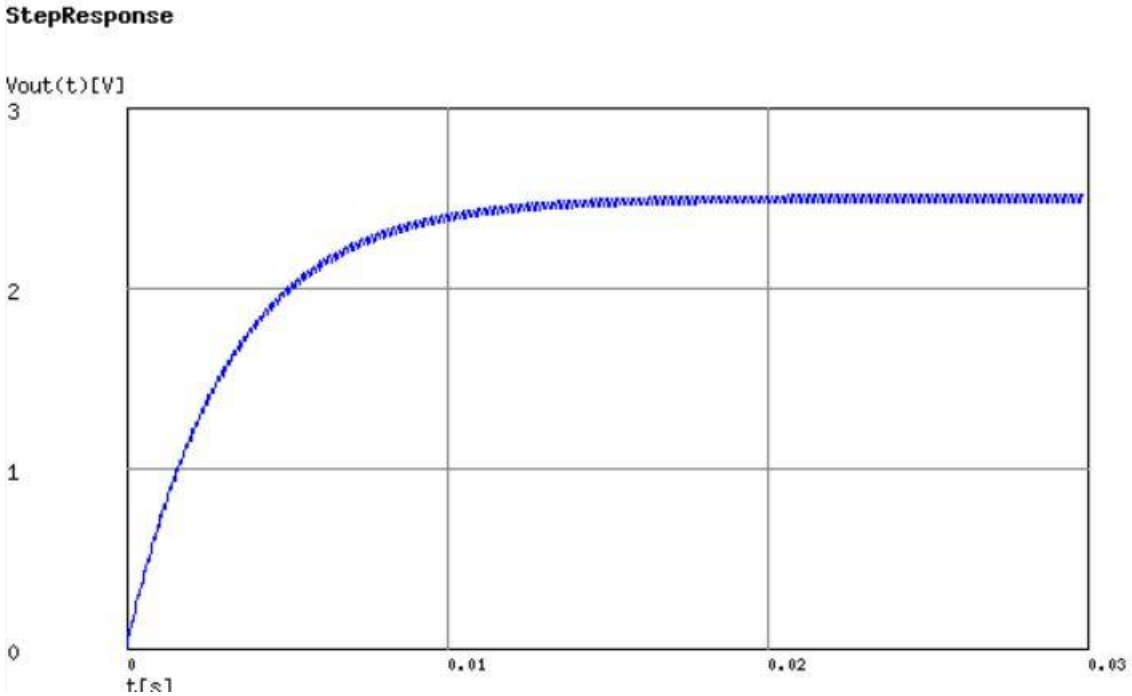

Figure 13: PWM signal with frequency less than the cut off frequency.

Putting it all together

Now that all the basic concepts of signals, the PWM shield and motor controllers has been covered, I can now run through how to control the motor controllers and give you a rough diagram of connecting components together for controlling the motor controllers (and thrusters). In most normal cases, it would make sense to talk about the actual system. However, because the WAMV’s hardware system is in the middle of massive changes this general overview should provide you with a solid background for working with whatever arrangement Brennan or other members of the hardware interoperability team may do.

Admittedly, connecting hardware components is not difficult. Its just making sure you connect the correct wire into the correct input. My fellow hardware interoperability team member most likely have documentation for that. I will not go in depth about the code the main Arduino uses to control the motor controllers for 2 reasons. First, the code during the trouble shooting process was completely rewritten. Second, the state the code was in before trouble shooting requires so much notes, that it warrant its own tech doc. Instead, I will provide 2 block diagrams with 2 scenarios, one with motor controllers that use PWM, and the other with motor controllers that require analog voltage and provide notes on how the components communicate.

Case 1: Analog controlled motor controllers

PWM shields, Motor Controllers, Low Pass Filters, Oh My!

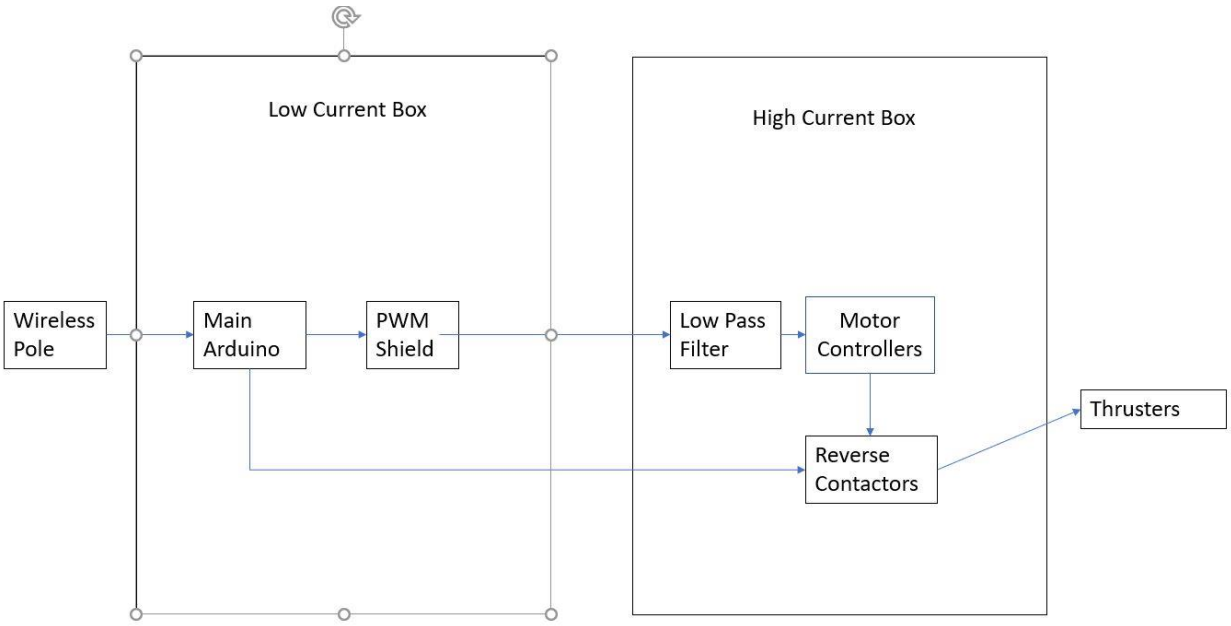

Figure 14: Block diagram of how signals are managed using an analog controlled motor controller.

The diagram above is a simplified block diagram of the order of components needed for turning the commands of a user into motion of the WAMV. This diagram only shows signals. Also, this process is outlined for controlling 1 thruster; there are 4 thrusters that need to be controlled

1.) Signals are received from an RC controller on shore by the Wireless Pole
2.) Main Arduino receives signals from wireless poles and process them for use for the hardware components.
3.) PWM shield is used by main Arduino to output PWM signals with 5V amplitude, 500Hz frequency, and varying duty cycles depending on whether the WAMV needs to go at full forward or full reverse.
4.) Low pass filter receives PWM signals at filters it to analog. This low pass filter does need to be designed for a cutoff frequency of 500Hz. Use formula or calculator in further resources for assistance on this task.
5.) Motor controller receives analog signal from low pass filter. The motor controllers control circuit should have 5V analog logic. In other words, the 50V going into the motor controllers is scaled depending on what value of analog voltage entering the control circuit on a scale of 0-5V. The max voltage of the analog signal can only be 3.6V because 3.6V corresponds to the max voltage of the thrusters, 36V.
6.) Scaled voltage from the motor controllers enters the reverse contactors (Jordan probably will have documentation on this.). While the motor controllers determine the magnitude of the voltage (and speed of the thrusters), the reverse contactor control the polarity of the voltage (and direction of the thrusters). The reverse contactors receive a digital signal from the main Arduino to know when to switch polarity.
7.) Thrusters receive voltage from the reverse contactors, moving WAMV.

PWM shields, Motor Controllers, Low Pass Filters, Oh My!

Case 2: Analog controlled motor controllers

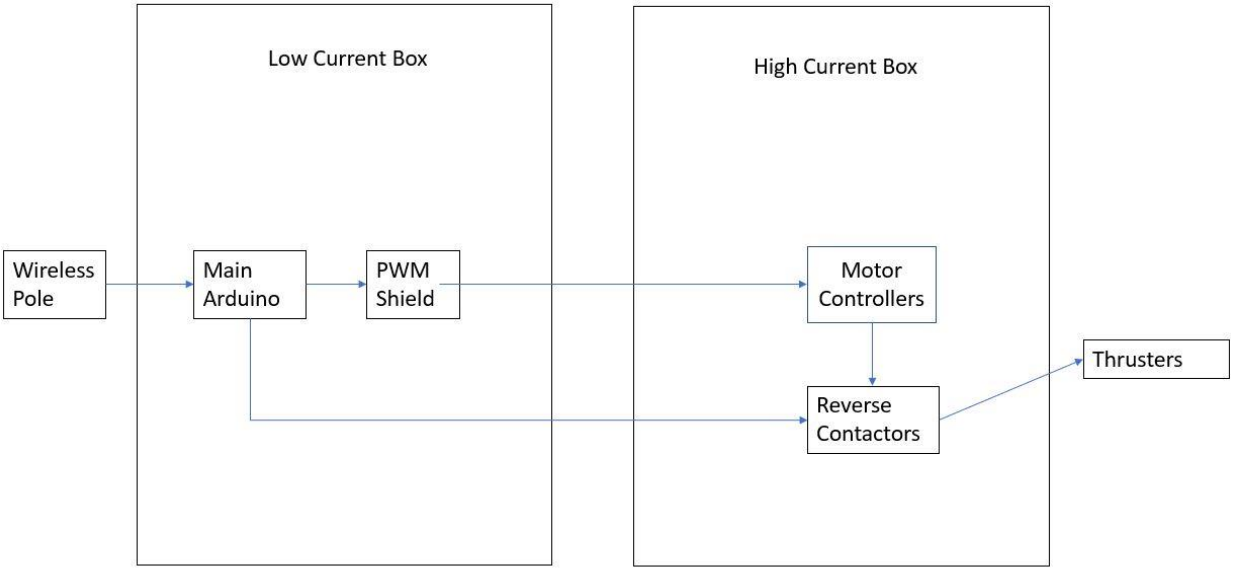

Figure 15: Block diagram of how signals are managed using an analog controlled motor controller.

The only major difference between this case and case 1 is the lack of low pass filters. A couple notes should be made on the PWM shield though. First, the frequency of the PWM shield is dependent on the motor controllers used. Check the data sheet for an appropriate PWM frequency to set the shield at. Second, although not depicted, the connection between the PWM shield and motor controllers have a component called an optocoupler. To prevent bloating out this guide more than it already is, it is basically a component to make sure there is no interference from the high current box affecting signals from the low current box. This fact is also true for case 1, but for the sake of simplicity that detail was removed.

Conclusion

In this guide, you learned about motor controllers, digital signals, PWM signals, the PWM shield, analog signals, and how to turn PWM signals into analog signals using a low pass filter. All this content will provide a strong foundation for sending a signal to other electronic components besides motor controllers. For more reads check out *DACs and I2C* and *Control the code: How a small board controls a big boat*. Coming to your nearest github soon!

Further Reading

Data Sheet on the PWM shield's chip (which really is the core of the PWM shield):
https://cdn-shop.adafruit.com/datasheets/PCA9685.pdf
A PWM Calculator
https://www.allaboutcircuits.com/technical-articles/low-pass-filter-a-pwm-signal-into-an-analog-voltage