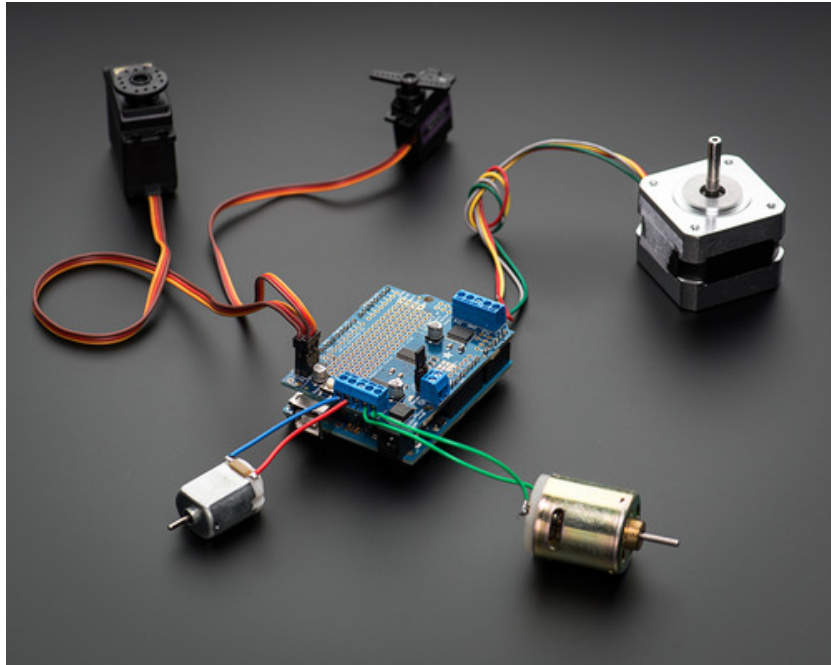




Adafruit Motor Shield V2 for Arduino

Created by lady ada



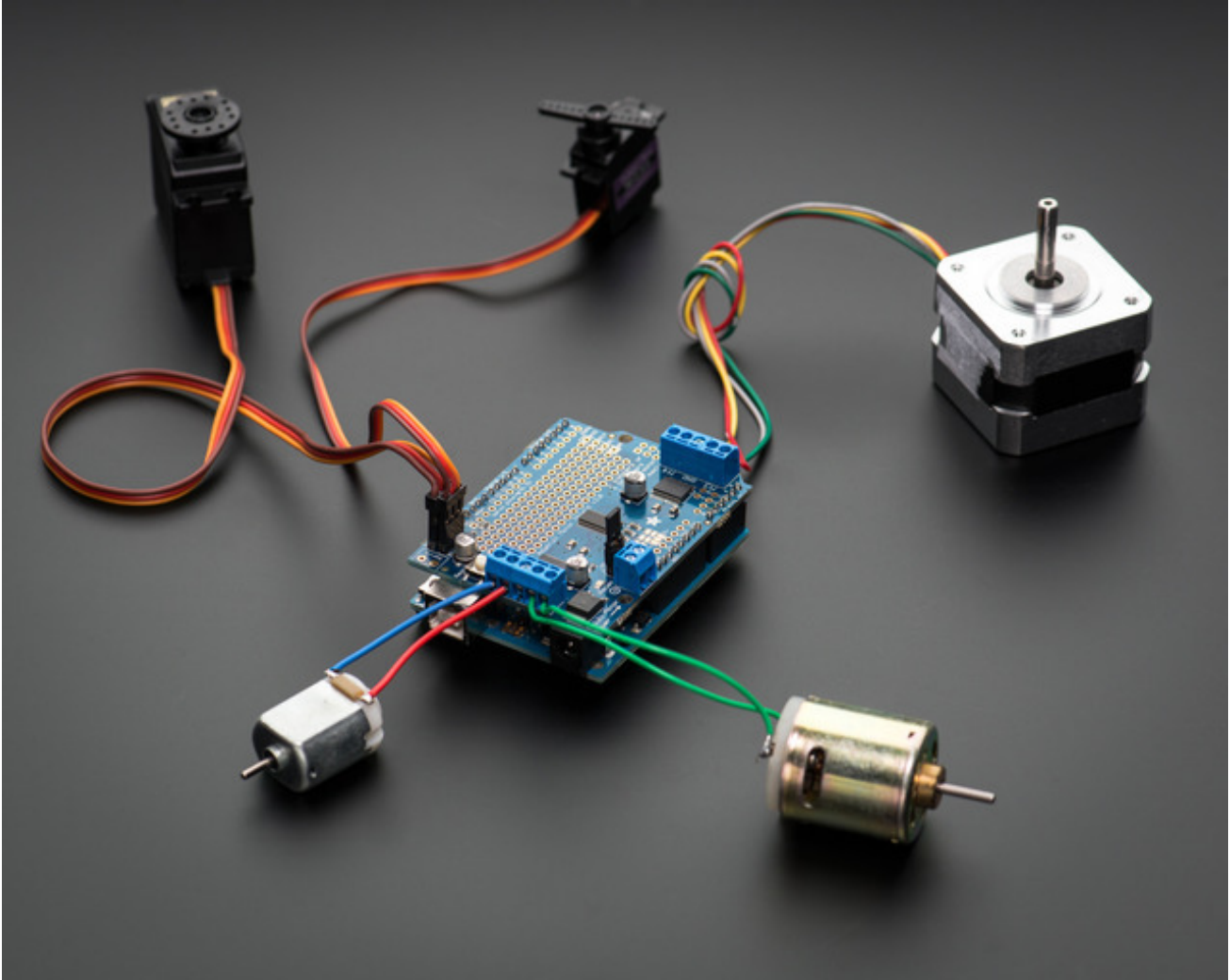
Last updated on 2014-11-02 05:15:12 AM EST

Guide Contents

Guide Contents	2
Overview	4
FAQ	7
Install Headers & Terminals	11
Installing Standard Headers	11
Installing Terminal Blocks and more	14
Installing with Stacking Headers	18
Install Software	22
Running the Example Code	22
DC Motor	22
Stepper Motor Test	24
Powering Motors	27
Voltage requirements:	27
Current requirements:	27
Setting up your shield for powering Hobby Servos	27
Setting up your shield for powering DC Motors	27
If you would like to have a single DC power supply for the Arduino and motors	28
If you would like to have the Arduino powered off of USB and the motors powered off of a DC power supply	28
If you would like to have 2 separate DC power supplies for the Arduino and motors.	29
Using RC Servos	30
Powering Servos	30
Using DC Motors	32
Connecting DC Motors	32
Include the required libraries	32
Create the Adafruit_MotorShield object	33
Create the DC motor object	33
Connect to the Controller	33
Set default speed	33

Run the motor	33
Using Stepper Motors	34
Include the required libraries	35
Create the Adafruit_MotorShield object	35
Create the stepper motor object	35
Set default speed	35
Run the motor	35
Stacking Shields	37
Addressing the Shields	38
Writing Code for Multiple Shields	39
Resources	40
Motor ideas and tutorials	40
Library Reference	41
class Adafruit_MotorShield;	41
Adafruit_MotorShield(uint8_t addr = 0x60);	41
void begin(uint16_t freq = 1600);	41
Adafruit_DCMotor *getMotor(uint8_t n);	42
Adafruit_StepperMotor *getStepper(uint16_t steps, uint8_t n);	42
void setPWM(uint8_t pin, uint16_t val); void setPin(uint8_t pin, boolean val);	42
class Adafruit_DCMotor	42
Adafruit_DCMotor(void);	43
void run(uint8_t);	43
void setSpeed(uint8_t);	43
class Adafruit_StepperMotor	44
Adafruit_StepperMotor(void);	44
void step(uint16_t steps, uint8_t dir, uint8_t style = SINGLE);	44
void setSpeed(uint16_t);	45
uint8_t onestep(uint8_t dir, uint8_t style);	45
void release(void);	45

Overview



The original Adafruit Motorshield kit is one of our most beloved kits, which is why we decided to make something even better. We have upgraded the shield kit to make the bestest, easiest way to drive DC and Stepper motors. This shield will make quick work of your next robotics project! We kept the ability to drive up to 4 DC motors or 2 stepper motors, but added many improvements:

Instead of a L293D darlington driver, we now have the TB6612 MOSFET driver: with **1.2A per channel and 3A peak current capability**. It also has much lower voltage drops across the motor so you get more torque out of your batteries, and there are built-in flyback diodes as well.

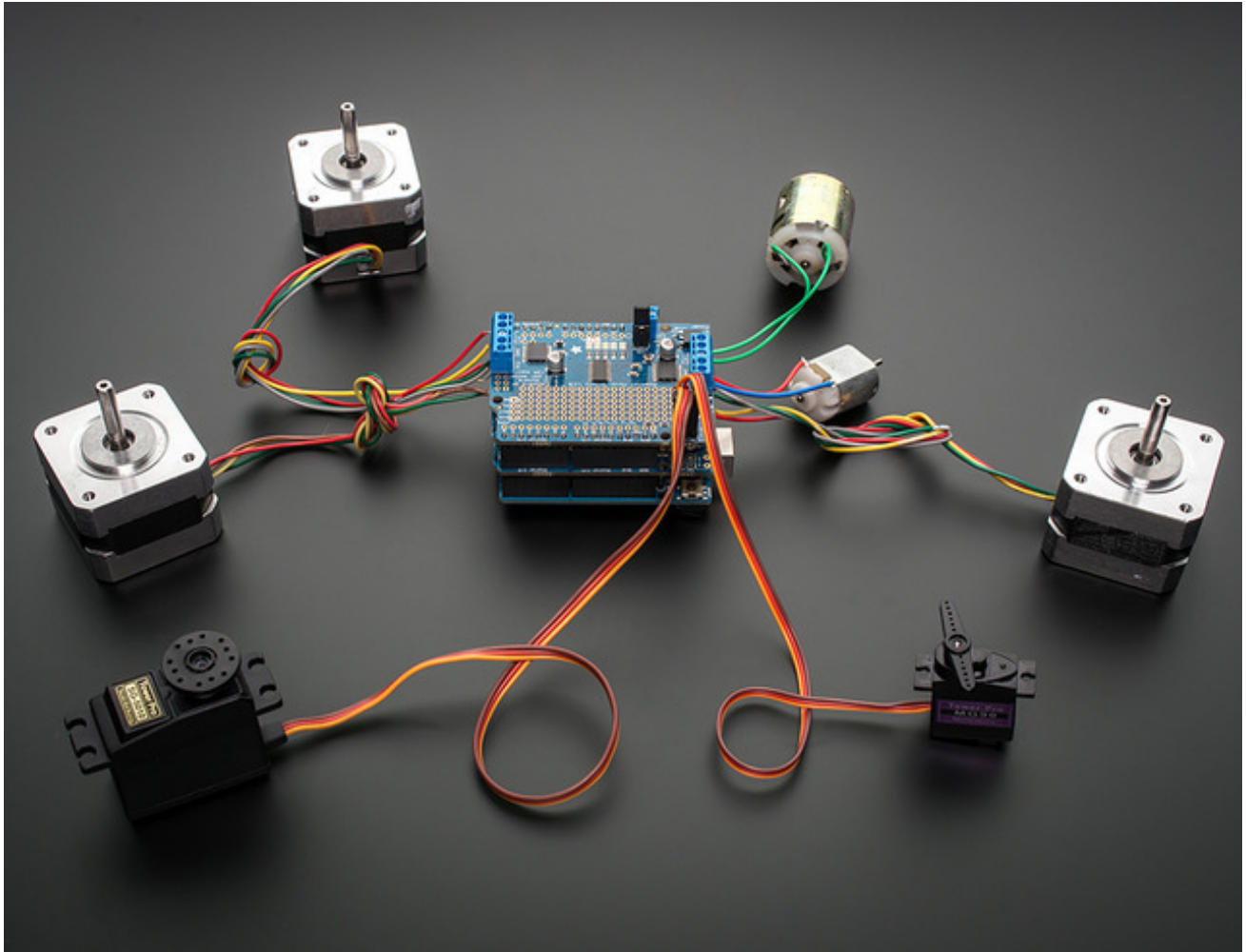
Instead of using a latch and the Arduino's PWM pins, we have a **fully-dedicated PWM driver chip** onboard. This chip handles all the motor and speed controls over I2C. Only two pins (SDA & SCL) are required to drive the multiple motors, and since it's I2C you can also connect any other I2C devices or shields to the same pins. This also makes it drop-in compatible with any Arduino, such as the Uno, Leonardo, Due and Mega R3.

Completely stackable design: 5 address-select pins means up to 32 stackable shields: that's 64 steppers or 128 DC motors! What on earth could you do with that many steppers? I have no idea but if you come up with something send us a photo because that would be a pretty glorious project.

Lots of other little improvements such as a polarity protection FET on the power pins and a big prototyping area. And the shield is assembled and tested here at Adafruit so all you have to do is solder on straight or stacking headers and the terminal blocks.

Lets check out these specs again:

- **2 connections for 5V 'hobby' servos** connected to the Arduino's high-resolution dedicated timer - no jitter!
- 4 H-Bridges: TB6612 chipset provides **1.2A per bridge** (3A peak) with thermal shutdown protection, internal kickback protection diodes. Can run motors on 4.5VDC to 13.5VDC.
- **Up to 4 bi-directional DC** motors with individual 8-bit speed selection (so, about 0.5% resolution)
- **Up to 2 stepper motors** (unipolar or bipolar) with single coil, double coil, interleaved or micro-stepping.
- Motors automatically disabled on power-up
- Big terminal block connectors to easily hook up wires (18-26AWG) and power
- Arduino reset button brought up top
- Polarity protected 2-pin terminal block and jumper to connect external power, for separate logic/motor supplies
- Tested compatible with Arduino UNO, Leonardo, ADK/Mega R3, Diecimila & Duemilanove. Works with Due with 3.3v logic jumper. Works with Mega/ADK R2 and earlier with 2 wire jumpers.
- Download the easy-to-use Arduino software library, check out the examples and you're ready to go!
- **5v** or **3.3v** compatible logic levels - jumper configurable.



As of Arduino 1.5.6-r2 BETA, there is a bug in the Due Wire library that prevents multiple Motor Shields from working properly with the Due!

FAQ

How many motors can I use with this shield?

You can use 2 DC hobby servos that run on 5V and up to 4 DC motors or 2 stepper motors (or 1 stepper and up to 2 DC motors) that run on 5-12VDC

Can I connect more motors?

Yes, by stacking shields! Every shield you stack on will add 4 DC motors or 2 stepper motors (or 1 more stepper and 2 more DC motors).

You will not gain more servo connections as the servo contacts go to pin #9 and #10 on the Arduino.

What if I also need some more servos?

Check out our lovely servo shield, also stackable with this motor shield and adds 16 free-running servos per shield <http://learn.adafruit.com/adafruit-16-channel-pwm-slash-servo-shield> (<http://adafru.it/ciQ>)

What Arduinos is this shield compatible with?

It is tested to work with Duemilanove, Diecimila, Uno (all revisions), Leonardo and Mega/ADK R3 and higher.

It can work with Mega R2 and lower if you solder a jumper wire from the shield's SDA pin to Digital 20 and the SCL pin to Digital 21

For use with the Due or other 3.3v processors, you must configure the board for 3.3v logic levels. Find the set of 3 pads labeled "Logic". Cut the small trace between the center pad and 5v and add a jumper from 3.3v to the center.

As of Arduino 1.5.6-r2 BETA, there is a bug in the Due Wire library that prevents multiple Motor Shields from working properly!

I get the following error trying to run the example code: "error: Adafruit_MotorShield.h: No such file or directory...."

Make sure you have installed the Adafruit_MotorShield library

How do I install the library?

Check the tutorial page on the subject here <http://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/install-software> (<http://adafru.it/ciO>)

HELP! My motor doesnt work! - HELP! My motor doesnt work!...But the servos work FINE!
Is the power LED lit? The Stepper and DC motor connections will not work if the onboard green Power LED is not lit brightly!

You must connect 5-12VDC power to the shield through the POWER terminal blocks or through the DC barrel jack on the Arduino and VIN jumper.

What is the green Power LED for?

The LED indicates the **DC/Stepper motor power supply is working**. If it is not lit brightly, then the DC/Stepper motors will not run. The servo ports are 5V powered and does not use the DC motor supply

What pins are/are not used on the motor shield?

The shield uses the SDA and SCL i2c pins to control DC and stepper motors. On the Arduino UNO these are also known as A4 and A5. On the Mega these are also known as Digital 20 and 21. On the Leonardo these are also known as digital 2 and 3. Do not use those pins on those Arduinos with this shield with anything other than an i2c sensor/driver.

Since the shield uses I2C to communicate, you can connect any other i2c sensor or driver to the SDA/SCL pins as long as they do not use address 0x60 (the default address of the shield)

If you want to use the servo connections, they are on pins #9 and #10. If you do not use the connector then those pins are simply not used.

You can use any other pins for any other use

How can I connect to the unused pins?

All pins are broken out into 0.1" spaced header along the edges of the shield

My Arduino freaks out when the motors are running! Is the shield broken?

Motors take a lot of power, and can cause 'brownouts' that reset the Arduino. For that reason the shield is designed for separate (split) supplies - one for the electronics and one for the motor. Doing this will prevent brownouts. Please read the user manual for information about appropriate power supplies.

I'm trying to build this robot and it doesn't seem to run on a 9V battery....

You **cannot** power motors from a 9V battery. You must use AA batteries or a lead acid battery for motors.

Can this shield control small 3V motors?

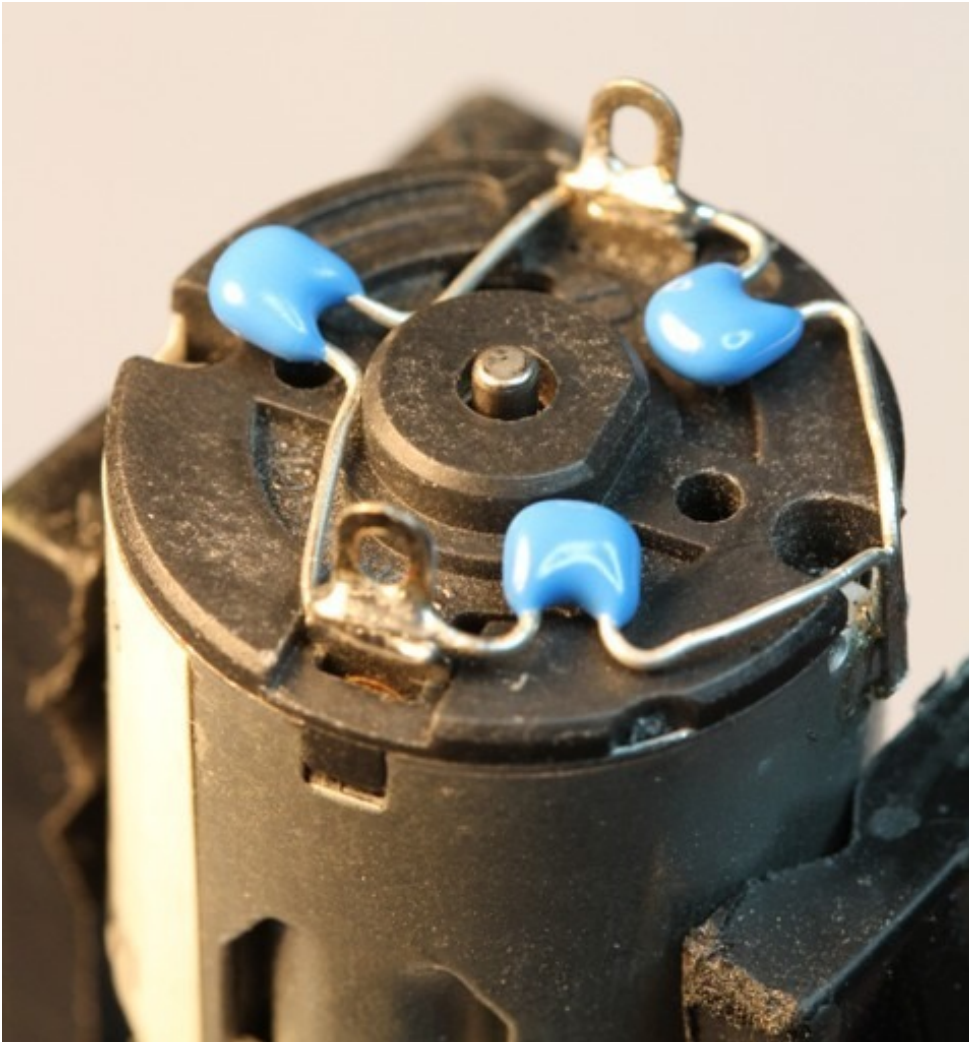
Not really, its meant for larger, 5V+ motors. It does not work for 3V motors unless you overdrive them at 5V and then they will burn out faster

I have good solid power supplies, but the DC motors seem to 'cut out' or 'skip'.

Try soldering a ceramic or disc 0.1uF capacitor between the motor tabs (on the motor itself!) this will reduce noise that could be feeding back into the circuit (thanks [macegr \(http://adafru.it/clc\)!](http://adafru.it/clc))

I'm using a 4WD robot platform and I can't get anything to work.

The motors used in the 4WD robot platforms from some distributors have a lot of "brush noise". This feeds back into the Arduino circuitry and causes unstable operation. This problem can be solved by soldering 3 noise suppression capacitors to the motor. 1 between the motor terminals, and one from each terminal to the motor casing.



But my motor already has a capacitor on it and it still doesn't work.

These motors generate a lot of brush noise and usually need the full 3-capacitor treatment for adequate suppression.

Why don't you just design capacitors into the shield?

They would not be effective there. The noise must be suppressed at the source or the motor leads will act like antennae and broadcast it to the rest of the system!

Why won't my stepper motor go any faster?

Since the shield is controlled by i2c, the maximum step rate is limited by the i2c bus speed. The default bus speed is 100KHz and can be increased to 400KHz by editing the library file in your Arduino installation folder. The file can be found in

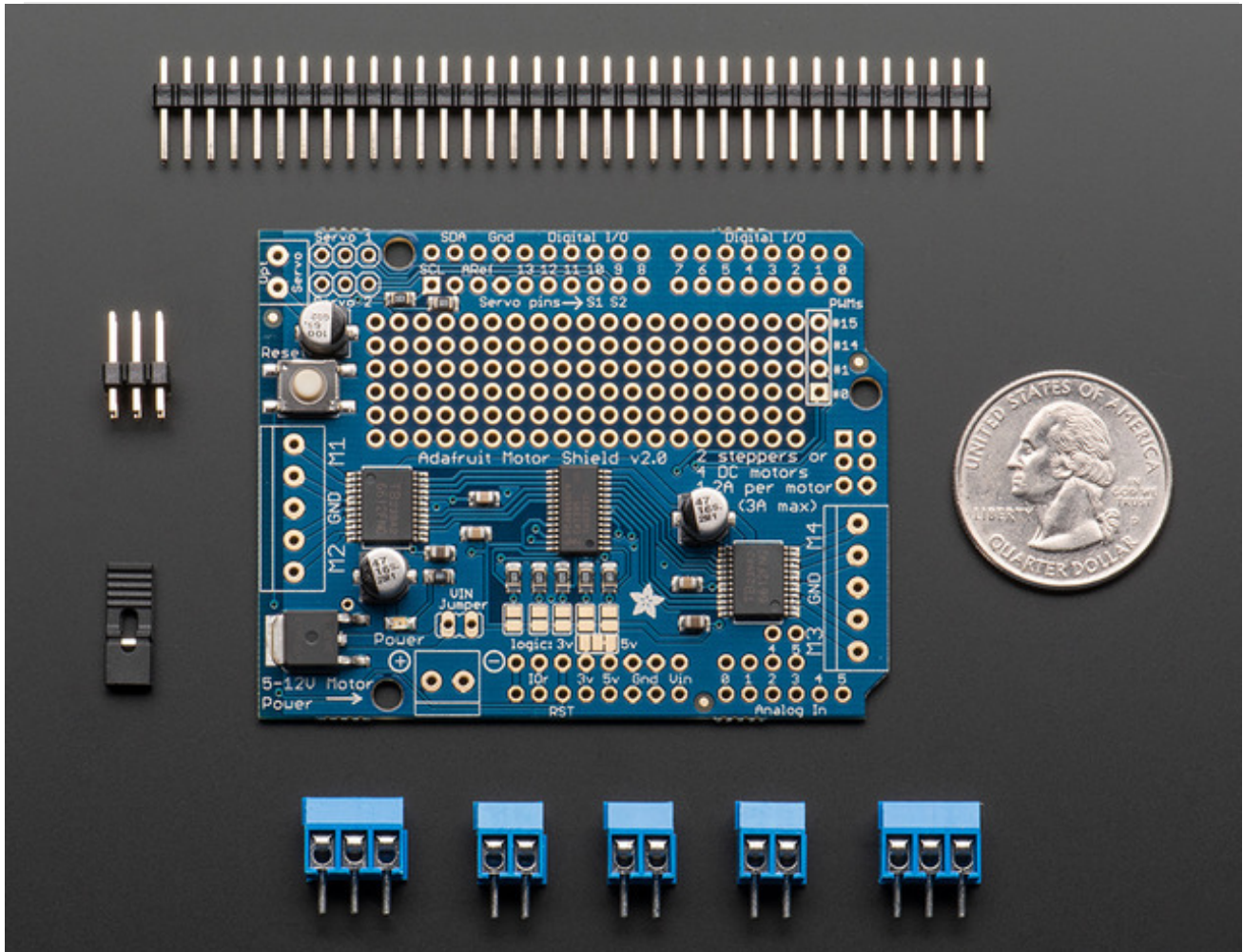
hardware/libraries/wire/utility/twi.h.

Find the line with: `"#define TWI_FREQ 100000L"`
and change it to `"#define TWI_FREQ 400000L"`

Or, you can add the following code to your `setup()` function:

```
TWBR = ((F_CPU / 400000L) - 16) / 2; // Change the i2c clock to 400KHz
```

Install Headers & Terminals

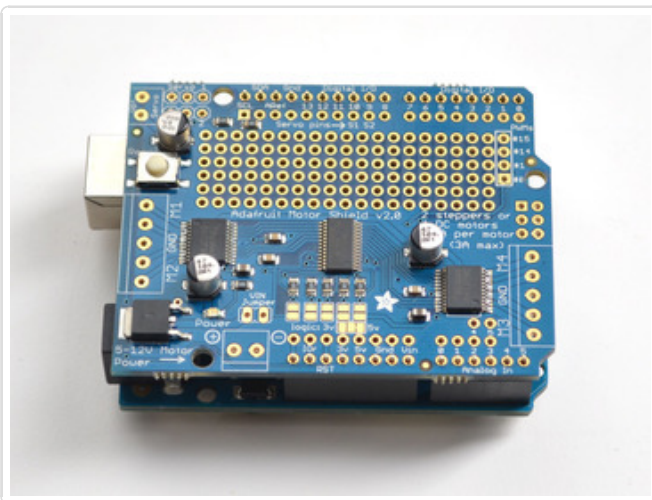


Installing Standard Headers

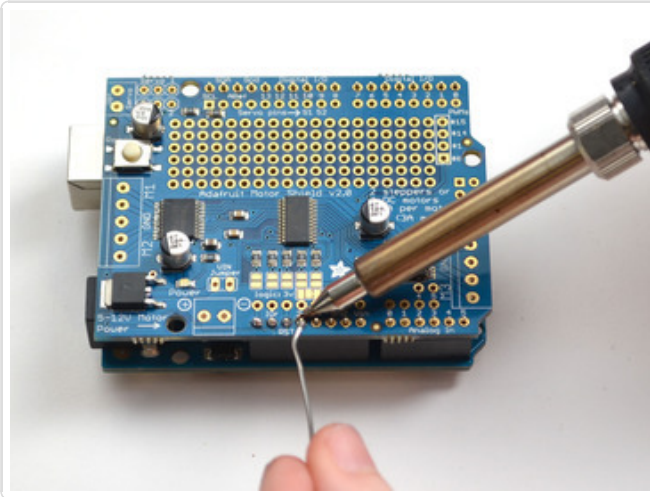
The shield comes with 0.1" standard header. Standard header does not permit stacking but it is mechanically stronger and they're much less expensive too! If you want to stack a shield on top, do not perform this step as it is not possible to uninstall the headers once soldered in! Skip down to the bottom for the stacking tutorial



Break apart the 0.1" header into 6, 8 and/or 10-pin long pieces and slip the long ends into the headers of your Arduino

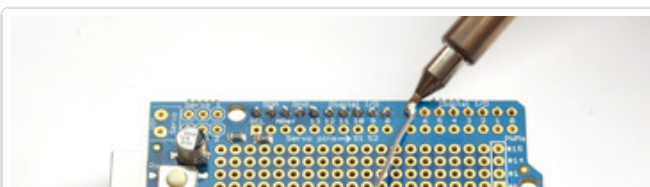
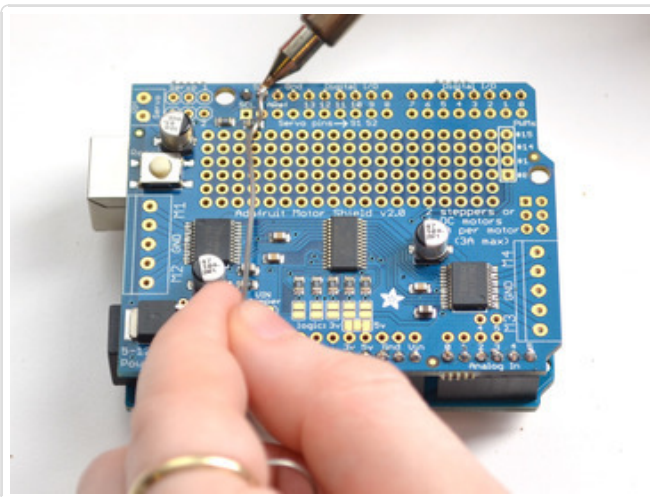
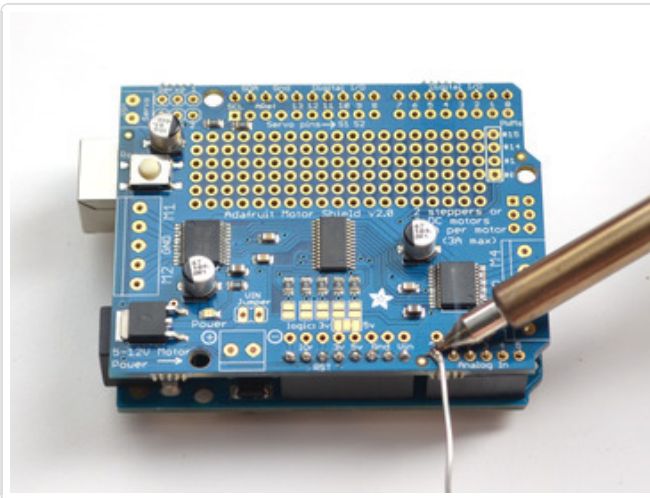


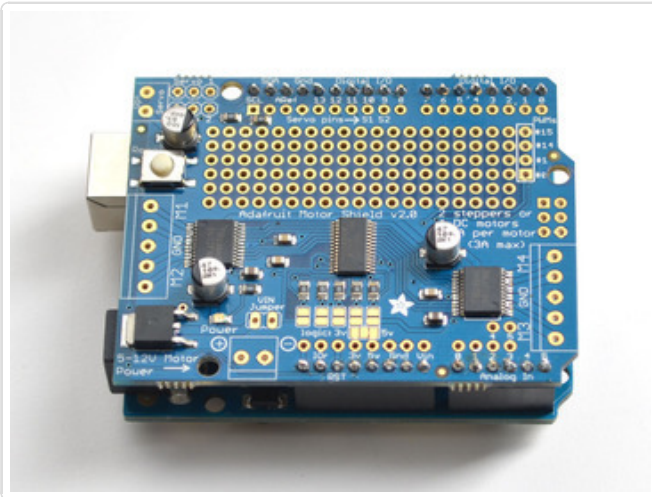
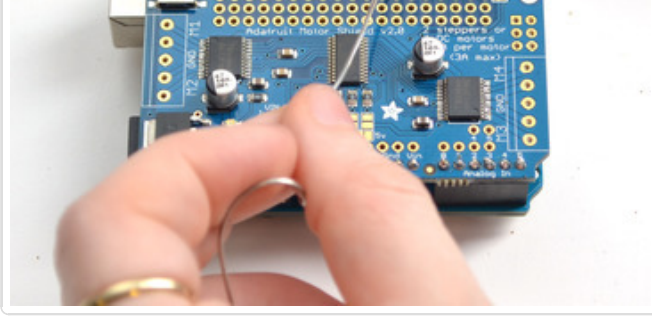
Place the assembled shield on top of the header-ed Arduino so that all of the short parts of the header are sticking through the outer set of pads



Solder each one of the pins into the shield to make a secure connection

Next, you will attach the terminal blocks, power jumper and servo connections

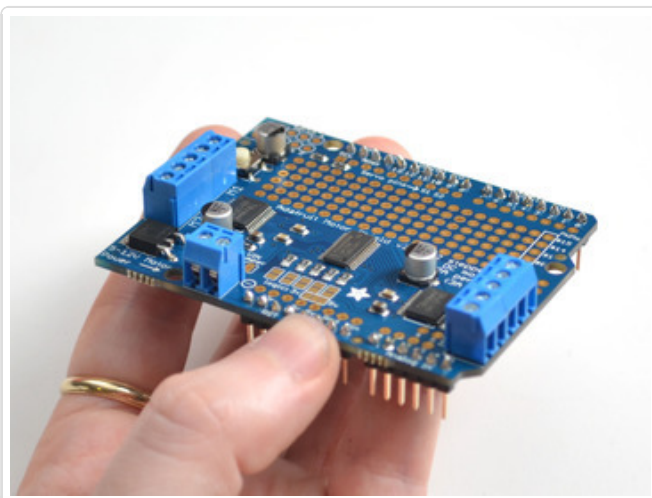




That's it! Now you can install the terminal blocks and jumper...

Installing Terminal Blocks and more

After you have installed either normal or stacking headers, you must install the terminal blocks.



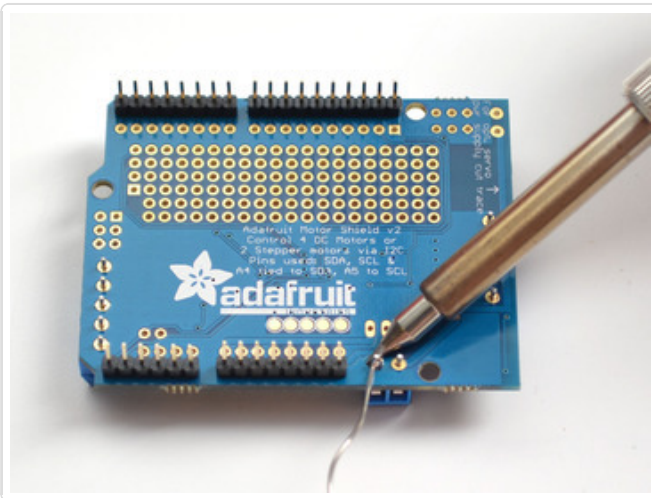
Next we will install the terminal blocks. These are how we will connect power and motors to the shield. They're much easier to use than soldering direct, just use a small screwdriver to release/attach wires!

First, though, we must solder them in.

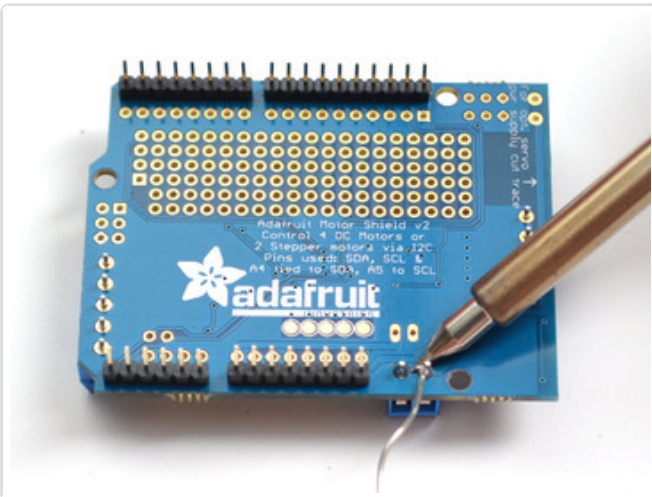
Slide the 3-pin terminal blocks into 2-pin terminal blocks so that you have 2 x 5-pin and 1 x 2-pin blocks. The two 5-pin sets go on either side. The 2-pin piece goes near the bottom of the shield. Make sure that the open holes of the terminal blocks face **out**!

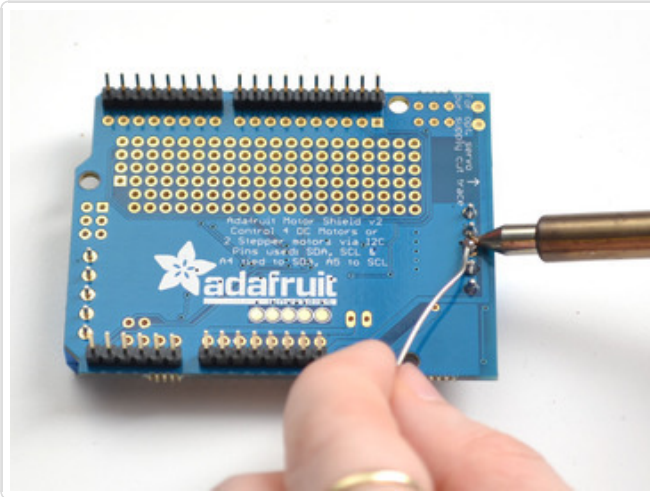


Flip the board over so that you can see & solder the pins of the terminal blocks

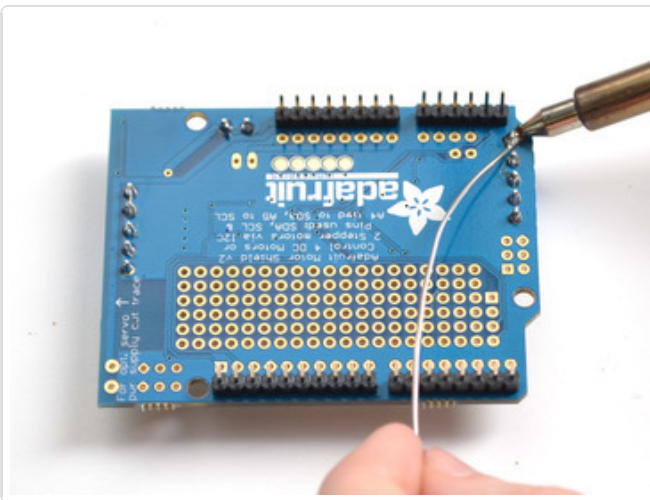


Solder in the two pins of the external power terminal-block

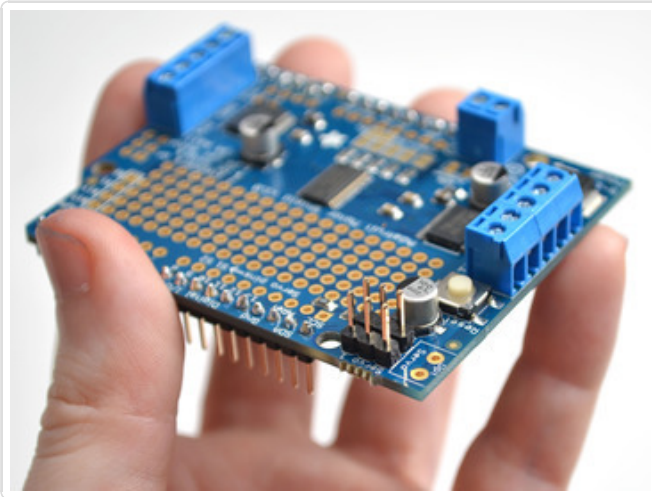




Solder in both motor blocks, 5 pads each

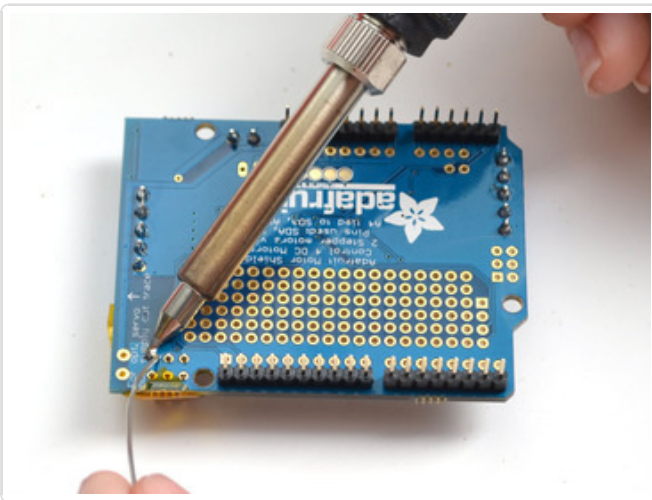


That's it for the terminal blocks. Next up, servo connections.

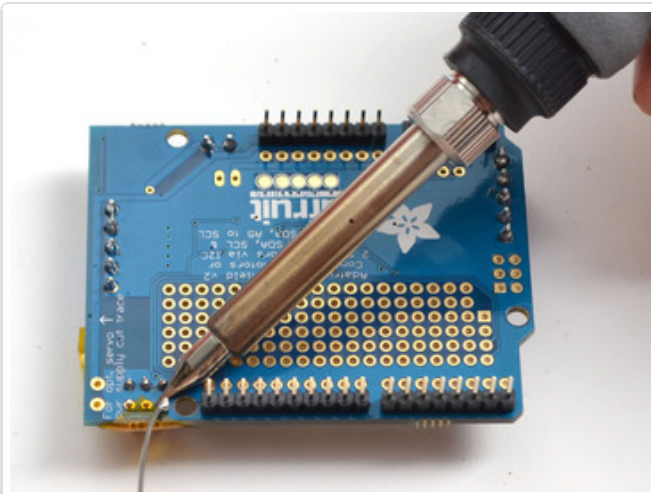


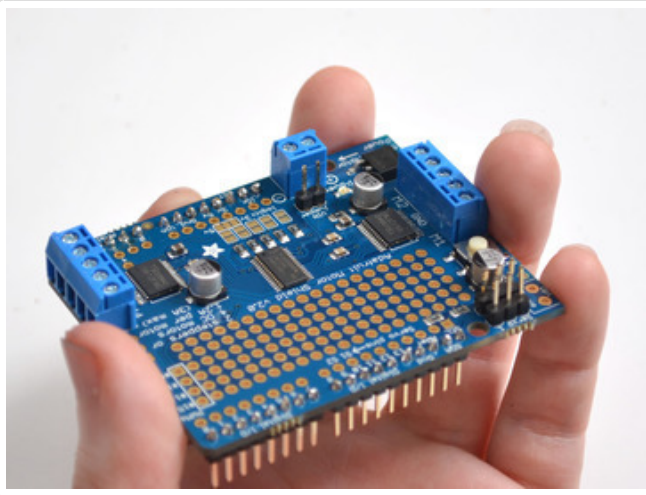
OK next up take the **2x3 pin header** and place it with the short legs down into the top corner where it says SERVO 1 and SERVO 2

You might have to sort of angle the part a little to get it to fit into both sets of 3-pin holes. we did this so it wont fall out easily when you turn it over!

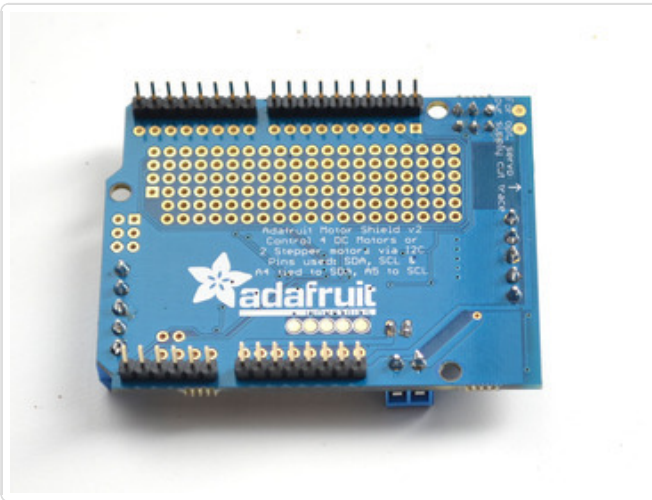


Then flip the board over and solder the 6 pins

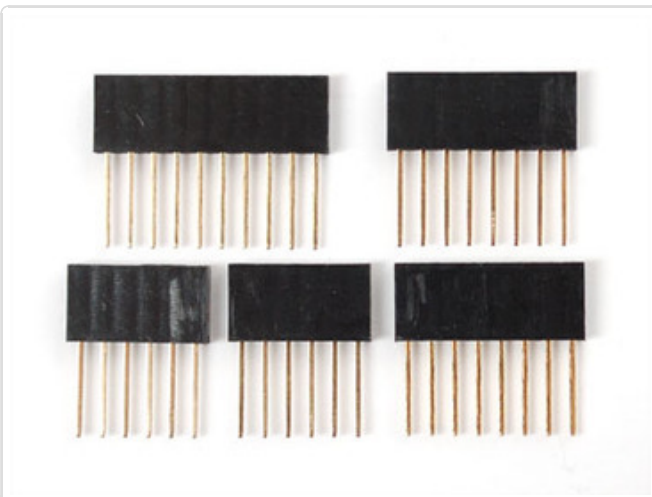




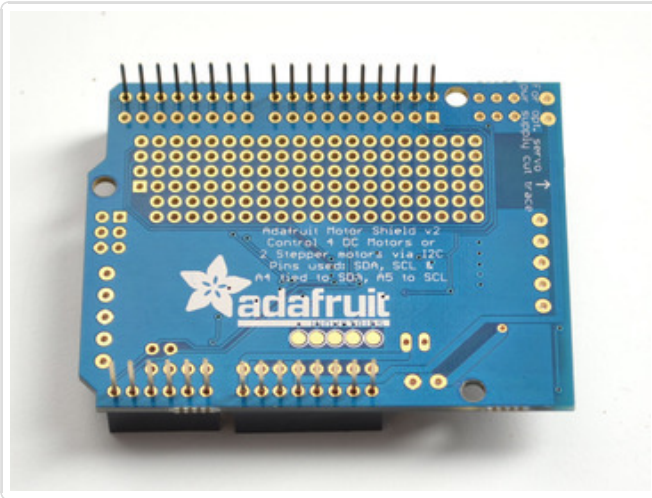
Finally, break off a 2-pin piece of header and place it next to the POWER terminal block, short legs down, tape it in place if necessary and solder it in.



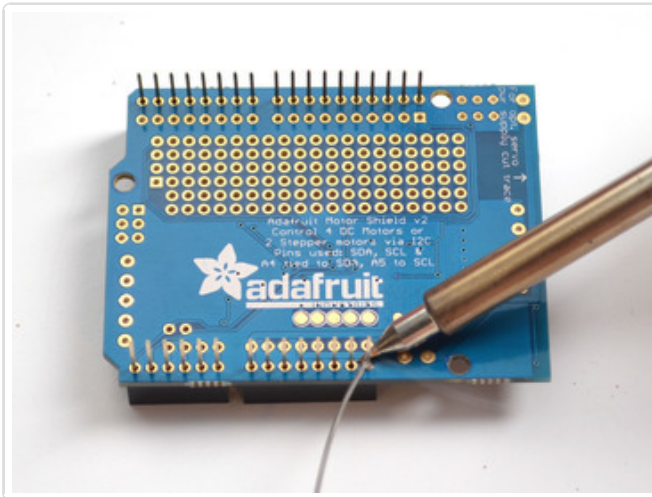
Installing with Stacking Headers



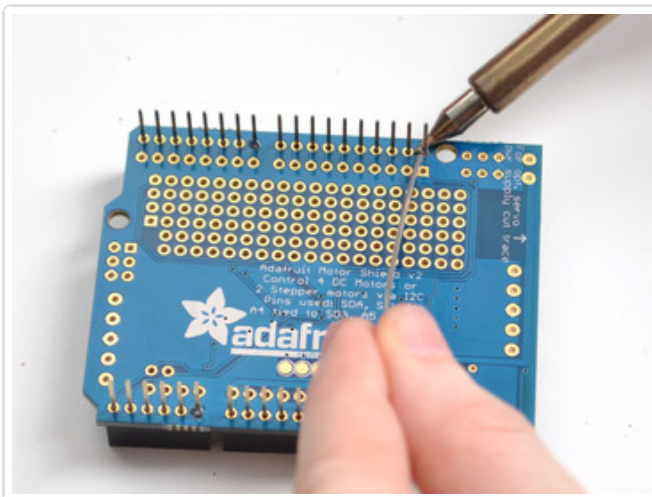
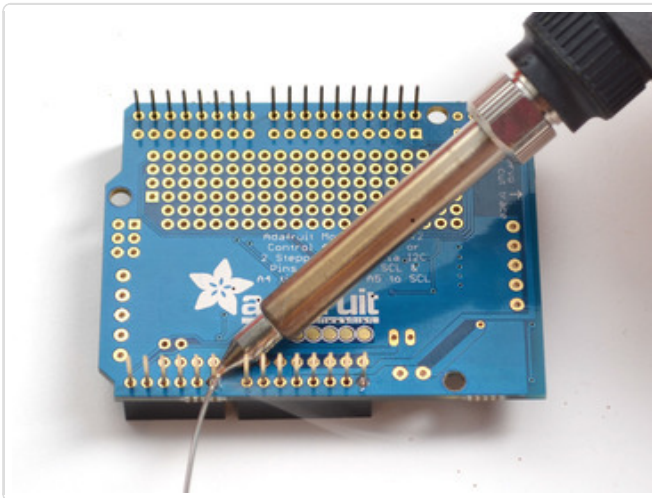
You will need to purchase Arduino stacking headers for this step, the shield does not come with them. (<http://adafru.it/85>)

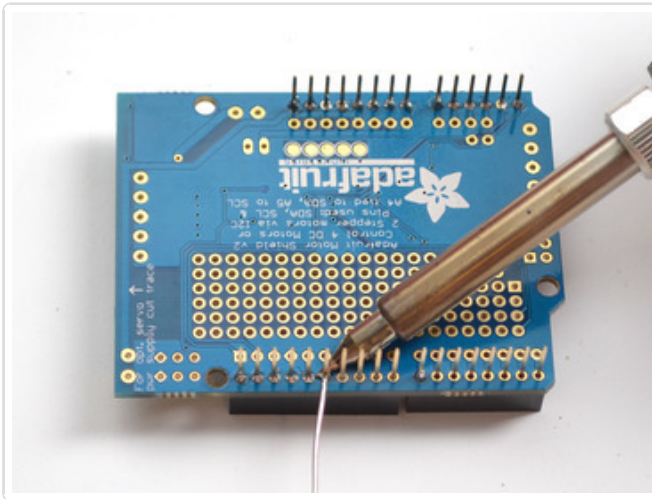


Start by sliding the 10 pin, 2 x 8 pin and 6-pin stacking headers into the outer rows of the shield from the top. Then flip the board over so its resting on the four headers. Pull on the legs if necessary to straighten them out.

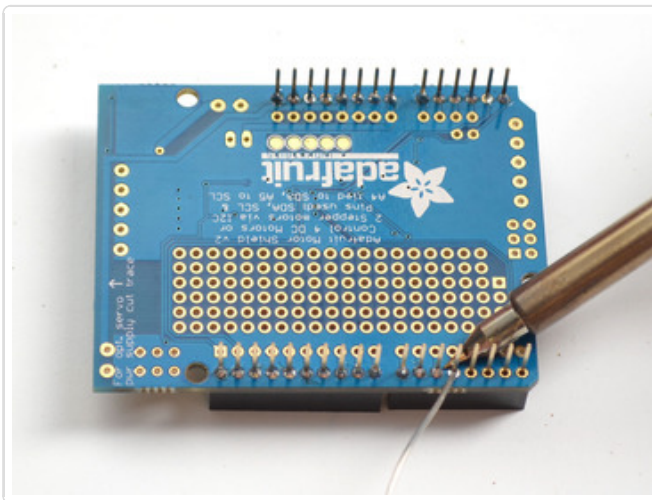


Tack one pin of each header, to get them set in place before more soldering. If the headers go crooked you can re-heat the one pin while re-positioning to straighten them up





Once you've tacked and straightened all the headers, go back and solder the remaining pins for each header.



Install Software

To use the shield on an Arduino, you'll need to install the Adafruit Motorshield v2 library. **This library is not compatible with the older AF_Motor library** used for v1 shields. However, if you have code for the older shield, adapting the code to use the new shield isn't difficult. We had to change the interface a little to support shield stacking, & we think its worth it!

Start by downloading the zip of the [library from the Github repository \(http://adafru.it/ciN\)](http://adafru.it/ciN). You can also just click the button below

Download Latest Adafruit Motor
Shield V2 Library

<http://adafru.it/cBx>

Copy the folder inside the zip file to the Libraries folder inside your Arduino Sketchbook folder and re-name it to Adafruit_Motorshield
[For more details on how to install Arduino libraries, check out our detailed tutorial! \(http://adafru.it/aYM\)](http://adafru.it/aYM)

If you plan to use .AccelStepper for acceleration control or for simultaneous control of multiple stepper motors, you will also need to download and install the AccelStepper library:

AccelStepper Library

<http://adafru.it/dh1>

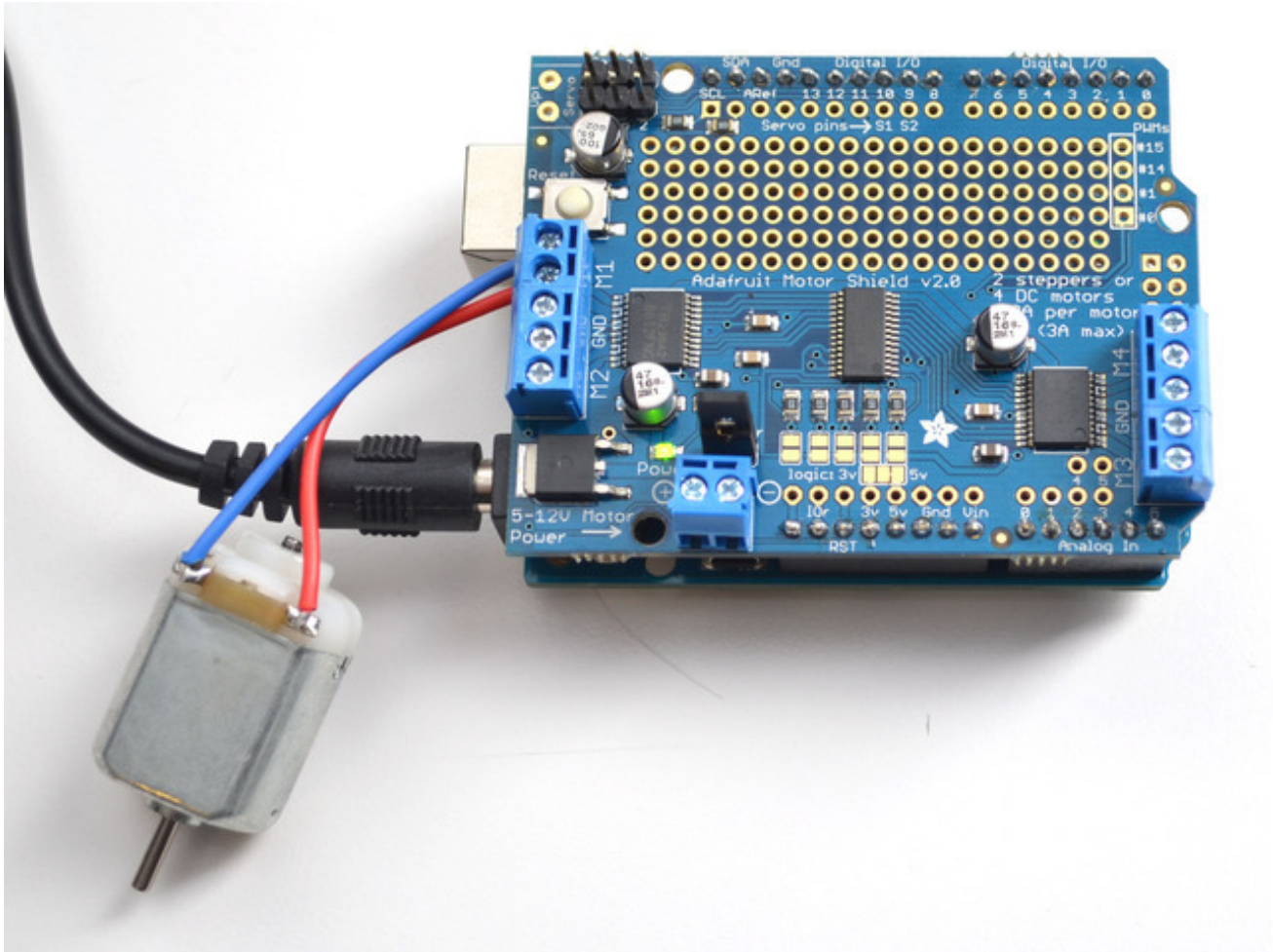
Running the Example Code

DC Motor

The library comes with a few examples to get you started up fast. We suggest getting started with the DC motor example. You can use any DC motor that can be powered by 6V-12VDC

First, restart the IDE to make sure the new library is loaded.

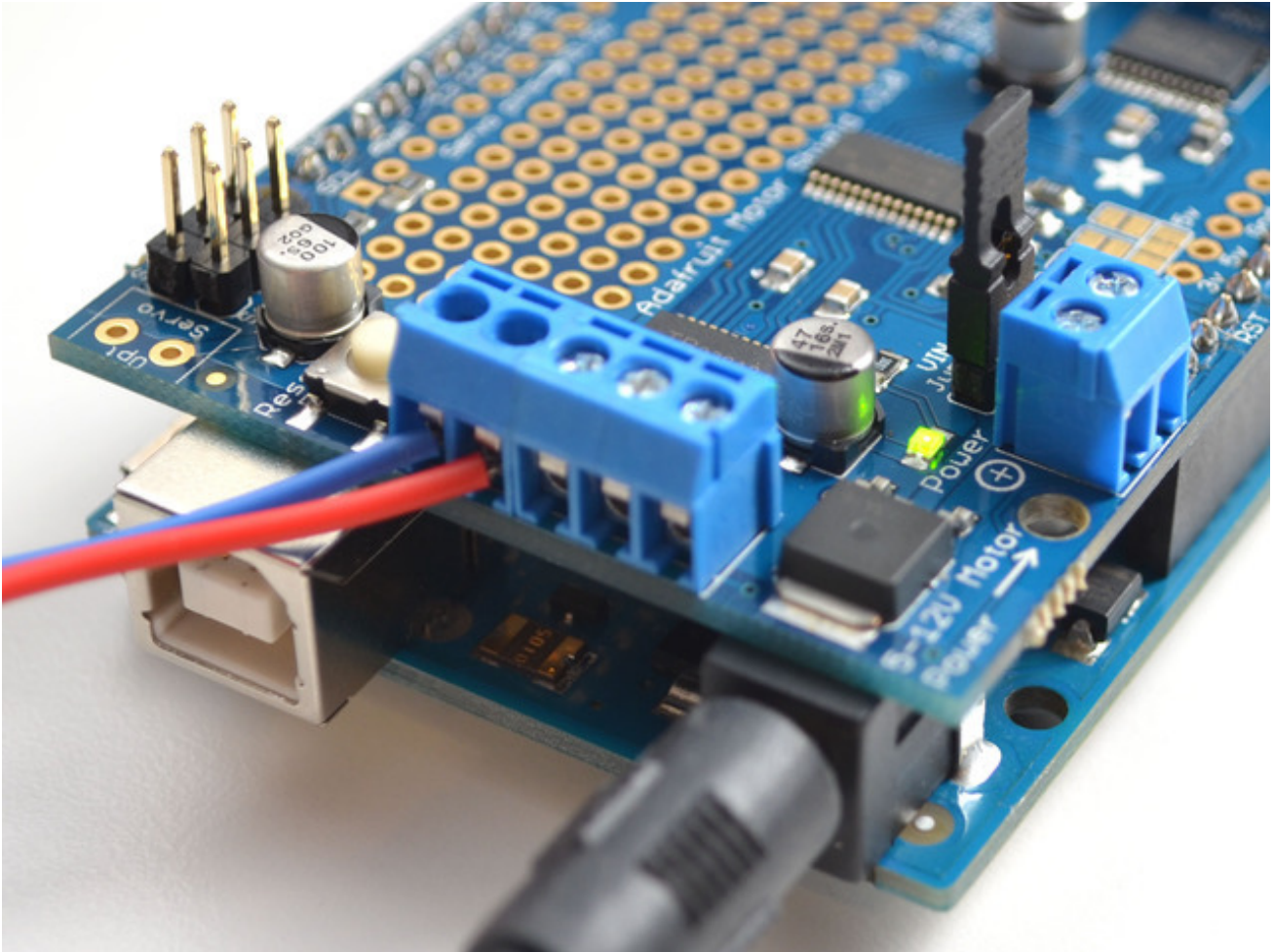
Plug the shield into the Arduino and connect a DC motor to **motor port 1** - it does not matter which wire goes into which terminal block as motors are bi-directional. Connect to the top two terminal ports, do not connect to the middle pin (GND) See the photo below for the red and blue wire example. Be sure to screw down the terminal blocks to make a good connection!



You must also supply 5-12VDC to power the motor. There are two ways to do this

1. You can power the Arduino via the **DC Barrel Jack** and **insert the VIN Jumper** shown as the tall black handle right next to the green Power LED below
2. You can power the Arduino via the DC Barrel jack **or** USB port. Then Power the shield via the 5-12VDC motor power terminal port, the double terminal block next to the green Power LED **and remove the VIN jumper**

If the Green LED isn't lit up brightly **do not continue** - you must power it via the VIN jumper **or** the terminal block



Once you have verified the motor is connected properly **and** you have the power LED lit up brightly, we can upload our code.

In the IDE, load **File->Examples->Adafruit_MotorShield->DCMotorTest**

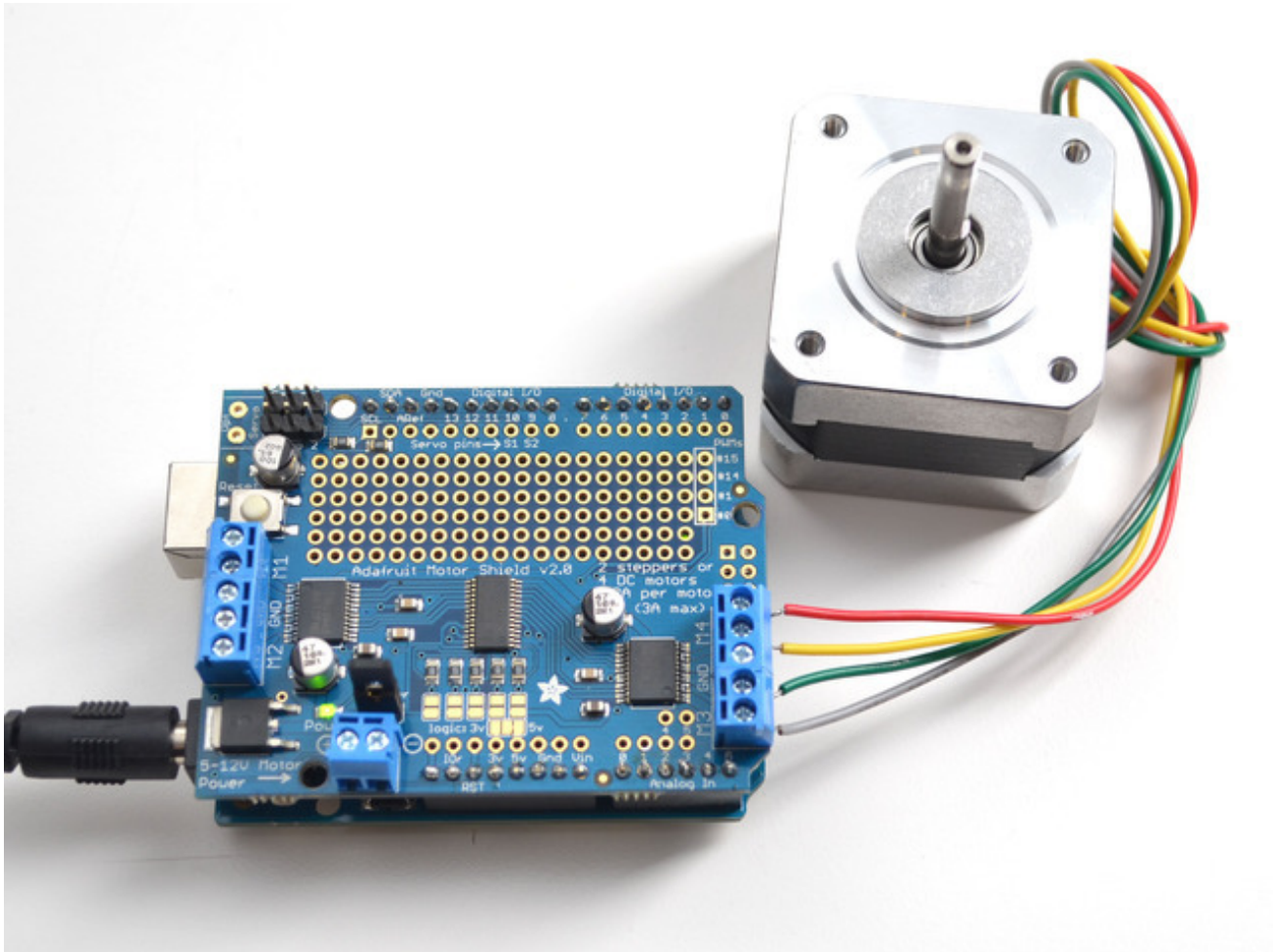
You should see and hear the DC motor turn on and move back and forth, attaching a slip of paper or tape as a 'flag' can help you visualize the movement if you have trouble seeing the movement

Stepper Motor Test

You can also test a stepper motor connection with the shield. The shield can run **unipolar** (5-wire and 6-wire) and **bipolar** (4-wire) steppers. It cannot run steppers with any other # of wires! The code is the same for unipolar or bipolar motors, the wiring is just slightly different.

Plug the shield into the Arduino and connect a stepper motor to **motor port 2** - unlike DC motors, the wire order does 'matter'. Connect to the top two terminal ports (coil #1) and the bottom two terminal ports (coil #2).

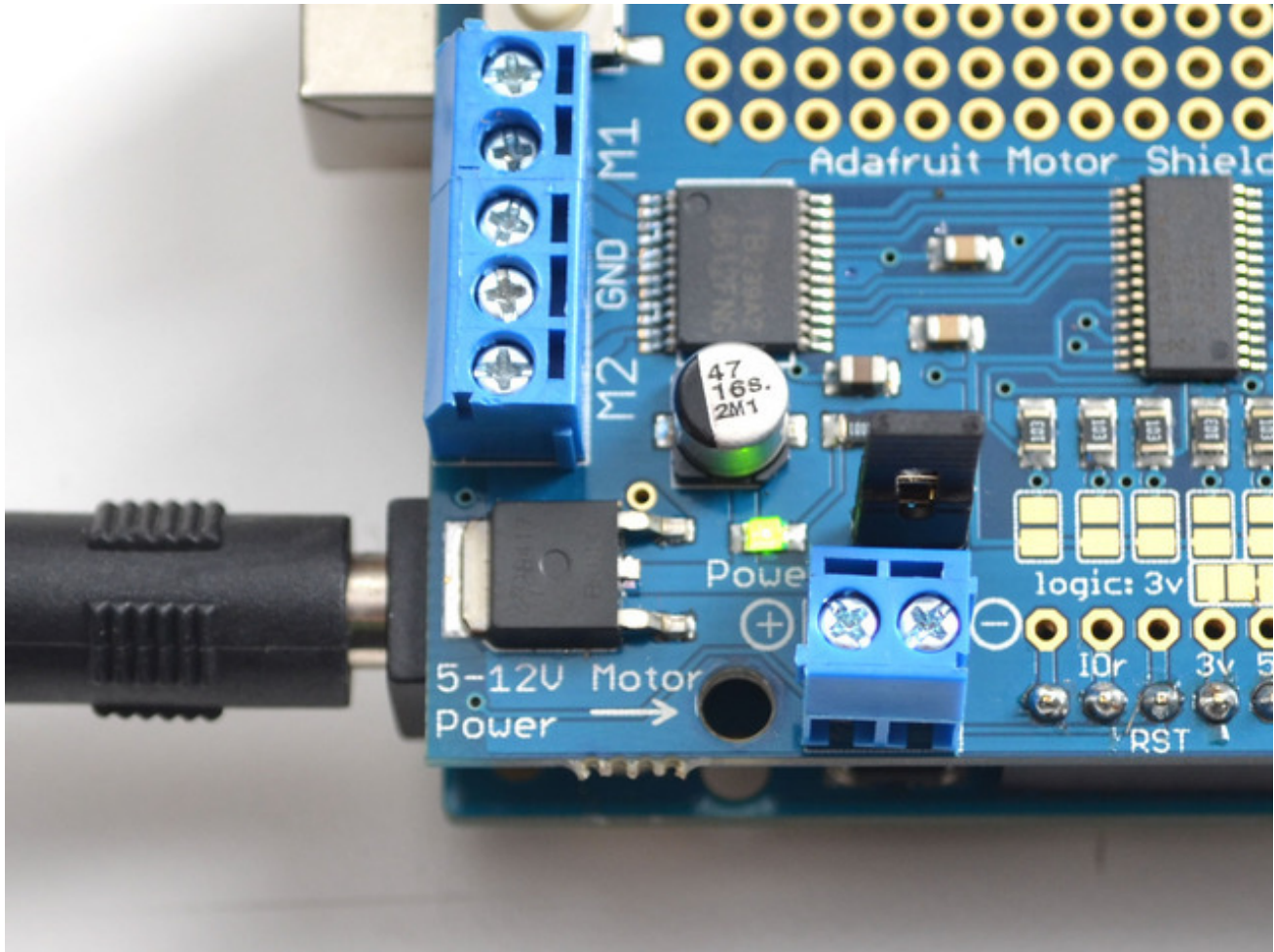
- If you have a bipolar motor, do not connect to the middle pin (GND).
- If you are using a unipolar motor with 5 wires, connect the common wire to GND.
- If you are using a unipolar motor with 6 wires, you can connect the two 'center coil wires' together to GND



You must also supply 5-12VDC to power the motor. There are two ways to do this

1. You can power the Arduino via the **DC Barrel Jack** and **insert the VIN Jumper** shown as the tall black handle right next to the green Power LED below
2. You can power the Arduino via the DC Barrel jack **or** USB port. Then Power the shield via the 5-12VDC motor power terminal port, the double terminal block next to the green Power LED **and remove the VIN jumper**

If the Green LED isn't lit up brightly **do not continue** - you must power it via the VIN jumper **or** the terminal block



Once you have verified the motor is connected properly **and** you have the power LED lit up brightly, we can upload our code.

In the IDE, load **File->Examples->Adafruit_MotorShield->StepperTest**

You should see and hear the stepper motor turn on and move back and forth, attaching a slip of paper or tape as a 'flag' can help you visualize the movement if you have trouble seeing the movement. There are four ways to move a stepper, with varying speed, torque and smoothness tradeoffs. This example code will demonstrate all four.

Powering Motors

Motors need a lot of energy, especially cheap motors since they're less efficient.

Voltage requirements:

The first important thing to figure out what voltage the motor is going to use. If you're lucky your motor came with some sort of specifications. Some small hobby motors are only intended to run at 1.5V, but its just as common to have 6-12V motors. The motor controllers on this shield are designed to run from **5V to 12V**.

MOST 1.5-3V MOTORS WILL NOT WORK

Current requirements:

The second thing to figure out is how much current your motor will need. The motor driver chips that come with the kit are designed to provide up to 1.2 A per motor, with 3A peak current. Note that once you head towards 2A you'll probably want to put a heat-sink on the motor driver, otherwise you will get thermal failure, possibly burning out the chip.

You can't run motors off of a 9V battery so don't waste your time/batteries!

Use a big Lead Acid or NiMH battery pack. Its also very much suggested that you set up two power supplies (split supply) one for the Arduino and one for the motors. **99% of 'weird motor problems'** are due to noise on the power line from sharing power supplies and/or not having a powerful enough supply! Even small DC motors can draw up to 3 Amps when they stall.

Setting up your shield for powering Hobby Servos

Servos are powered off of the same regulated 5V that the Arduino uses. This is OK for the small hobby servos suggested. Basically, power up your Arduino with the USB port or DC barrel jack and you're good to go. If you want something beefier, cut the trace going to the optional servo power terminal and wire up your own 5-6V supply!

Setting up your shield for powering DC Motors

The DC motors are powered off of a 'high voltage supply' and NOT the regulated 5V. **Don't connect the motor power supply to the Arduino's 5V power pin.** This is a very very very bad idea unless you are sure you know what you're doing! You could damage your Arduino and/or USB port!

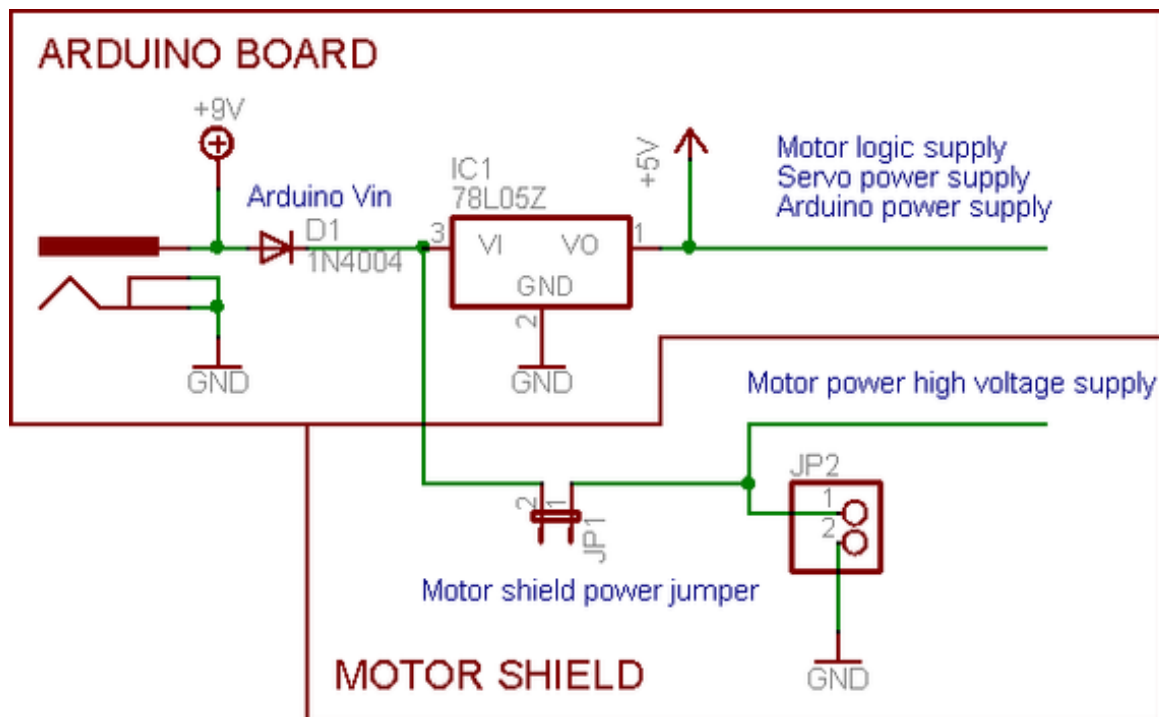
There are two places you can get your motor 'high voltage supply' from.

1. One is the DC barrel jack on the Arduino board
2. The other is the 2-terminal block on the shield that is labeled **DC Motor Power 5-**

12VDC.

The DC Jack on the Arduino has a protection diode so you won't be able to mess things up too bad if you plug in the wrong kind of power. The terminal block as a protection FET so you will not damage the arduino/shield if you wire up your battery supply backwards, but it won't work either!

Here's how it works:



If you would like to have a **single DC power supply for the Arduino and motors**

Say a wall adapter or a single battery pack with 6-12VDC output, simply plug it into the DC jack on the Arduino or the 2-pin power terminal block on the shield. Place the power jumper on the motor shield.

Note that you may have problems with Arduino resets if the battery supply is not able to provide constant power, so it is not a suggested way of powering your motor project. You cannot use a 9V battery for this, it must be 4 to 8 AA batteries or a single/double lead acid battery pack.

If you would like to have the **Arduino powered off of USB** and the **motors powered off of a DC power supply**

Plug in the USB cable. Then connect the motor supply to the power terminal block on the

shield. Do not place the jumper on the shield.

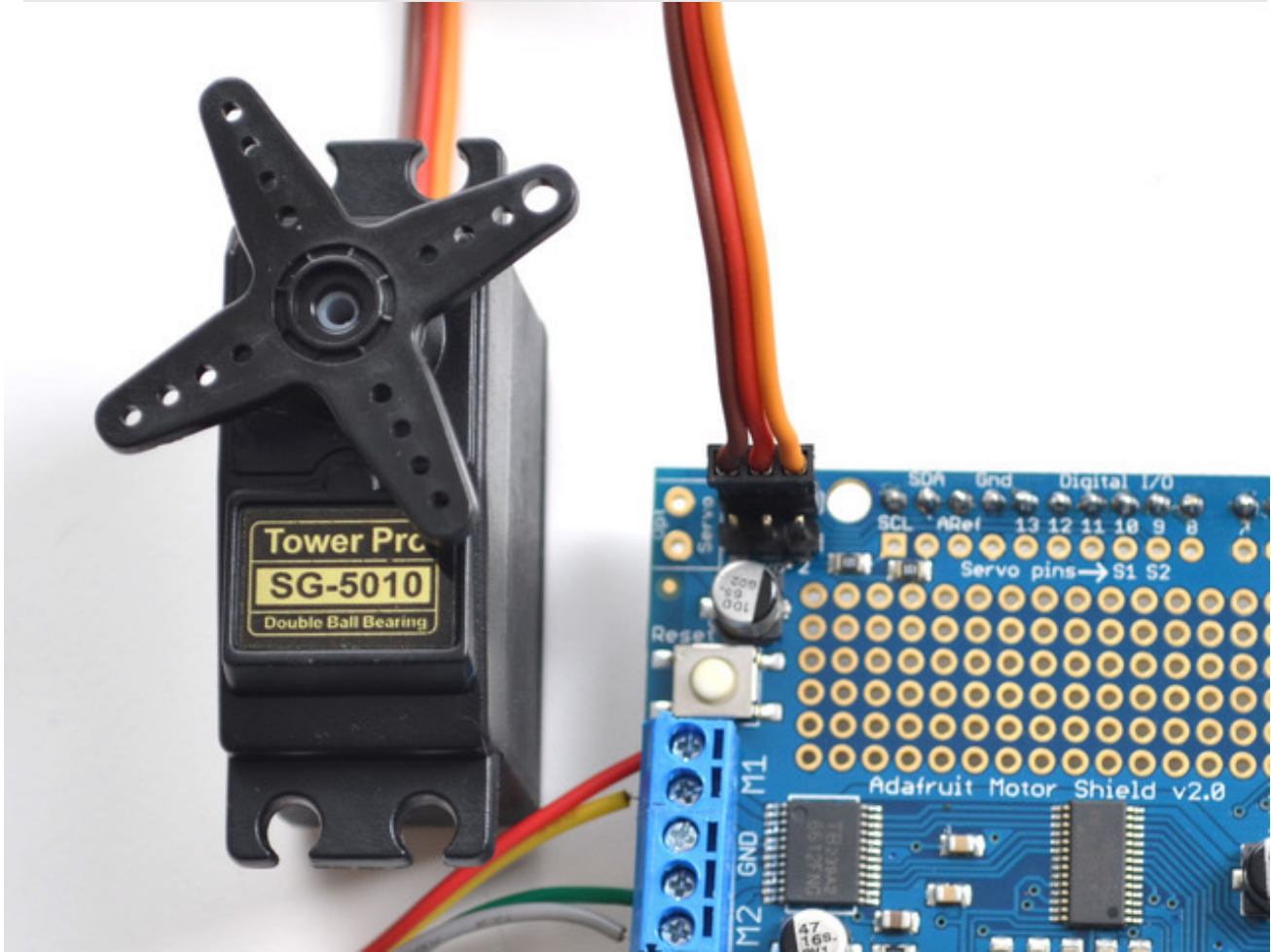
This is a suggested method of powering your motor project as it has a split supply, one power supply for logic, and one supply for motors

If you would like to have 2 separate DC power supplies for the Arduino and motors.

Plug in the supply for the Arduino into the DC jack, and connect the motor supply to the power terminal block. Make sure the jumper is removed from the motor shield.

No matter what, if you want to use the DC motor/Stepper system the motor shield LED should be lit indicating good motor power

Using RC Servos



Hobby servos are the easiest way to get going with motor control. They have a 3-pin 0.1" female header connection with +5V, ground and signal inputs. The motor shield simply brings out the PWM output lines from Arduino pins 9 and 10 to two 3-pin headers so that its easy to plug in and go. They can take a lot of power so a 9V battery wont last more than a few minutes!

The nice thing about using the onboard PWM is that its very precise and goes about its business in the background. You can use the built in **Servo** library

[Using the servos is easy, please read the official Arduino documentation for how to use them and see the example Servo sketches in the IDE \(http://adafru.it/aOD\).](http://adafru.it/aOD)

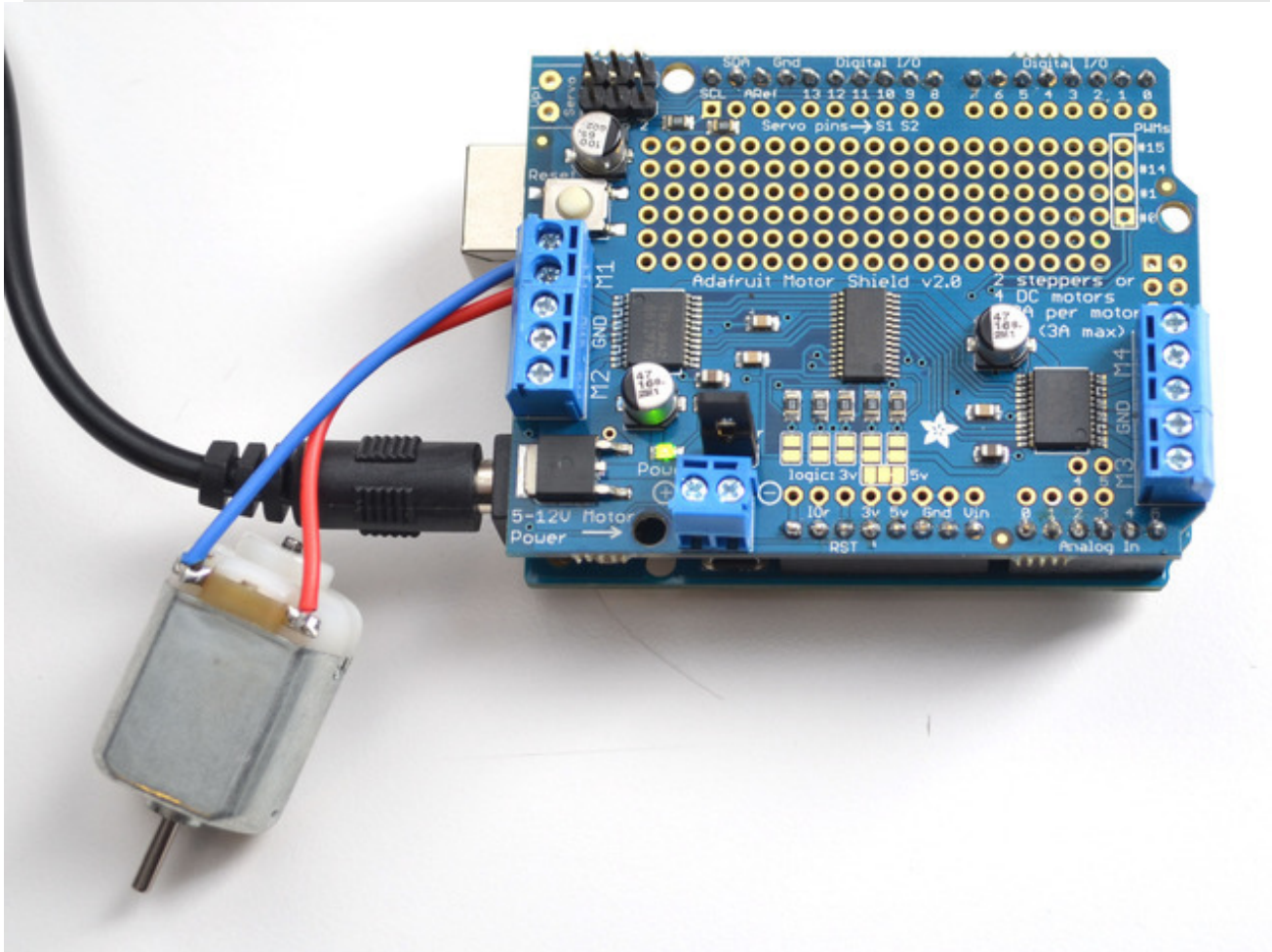
Powering Servos

Power for the Servos comes from the Arduino's on-board 5V regulator, powered directly from the USB or DC power jack on the Arduino. If you need an external supply, cut the 5v trace on the bottom of the board and connect a 5V or 6V DC supply directly to the **Opt Servo** power input. Using an external supply is for *advanced*

users as you can accidentally destroy the servos by connecting a power supply incorrectly!

When using external servo power, be careful not to let it short out against the USB socket shell on the processor board. Insulate the top of the USB socket with some electrical tape.

Using DC Motors



DC motors are used for all sort of robotic projects.

The motor shield can drive up to 4 DC motors bi-directionally. That means they can be driven forwards and backwards. The speed can also be varied at 0.5% increments using the high-quality built in PWM. This means the speed is very smooth and won't vary!

Note that the H-bridge chip is not meant for driving continuous loads over 1.2A or motors that peak over 3A, so this is for small motors. Check the datasheet for information about the motor to verify its OK!

Connecting DC Motors

To connect a motor, simply solder two wires to the terminals and then connect them to either the M1, M2, M3, or M4. Then follow these steps in your sketch

Include the required libraries

Make sure you **#include** the required libraries
#include <Wire.h>

```
#include <Adafruit_MotorShield.h>
#include "utility/Adafruit_PWMServoDriver.h"
```

Create the Adafruit_MotorShield object

```
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
```

Create the DC motor object

Request the Stepper motor from the Adafruit_MotorShield:

```
Adafruit_DCMotor *myMotor = AFMS.getMotor(1);
```

with **getMotor(port#)**. **Port#** is which port it is connected to. If you're using M1 its **1**, M2 use **2**, M3 use **3** and M4 use **4**

Connect to the Controller

In your setup() function, call 'begin()' on the Adafruit_MotorShield object:

```
AFMS.begin();
```

Set default speed

Set the speed of the motor using **setSpeed(speed)** where the **speed** ranges from 0 (stopped) to 255 (full speed). You can set the speed whenever you want.

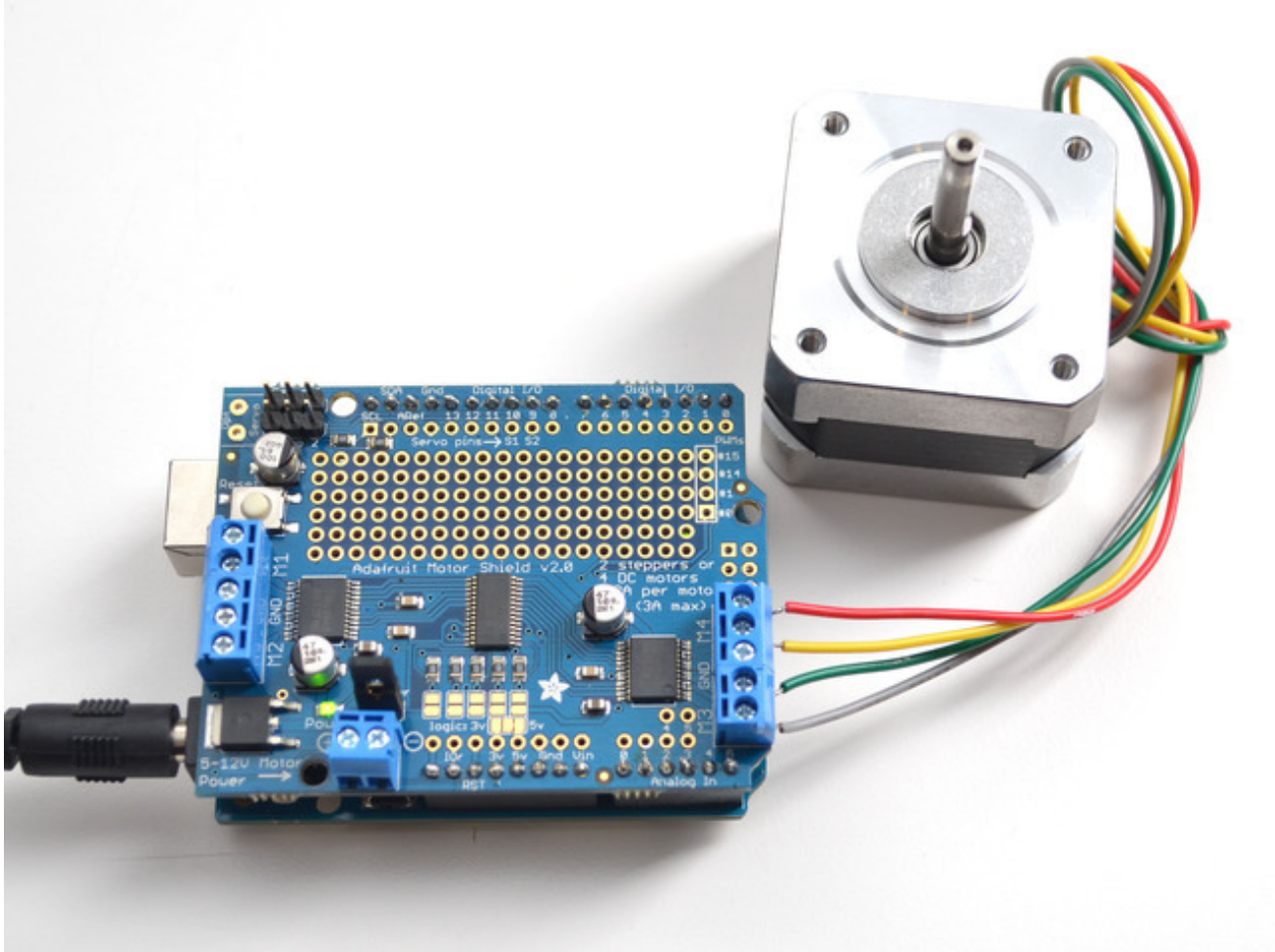
```
myMotor->setSpeed(100);
```

Run the motor

To run the motor, call **run(direction)** where **direction** is **FORWARD**, **BACKWARD** or **RELEASE**. Of course, the Arduino doesn't actually know if the motor is 'forward' or 'backward', so if you want to change which way it thinks is forward, simply swap the two wires from the motor to the shield.

```
myMotor->run(FORWARD);
```

Using Stepper Motors



Stepper motors are great for (semi-)precise control, perfect for many robot and CNC projects. This motor shield supports up to 2 stepper motors. The library works identically for bi-polar and uni-polar motors

For unipolar motors: to connect up the stepper, first figure out which pins connected to which coil, and which pins are the center taps. If its a 5-wire motor then there will be 1 that is the center tap for both coils. [Theres plenty of tutorials online on how to reverse engineer the coils pinout. \(http://adafru.it/aOO\)](http://adafru.it/aOO) The center taps should both be connected together to the GND terminal on the motor shield output block. then coil 1 should connect to one motor port (say M1 or M3) and coil 2 should connect to the other motor port (M2 or M4).

For bipolar motors: its just like unipolar motors except theres no 5th wire to connect to ground. The code is exactly the same.

Running a stepper is a little more intricate than running a DC motor but its still very easy

Include the required libraries

Make sure you **#include** the required libraries

```
#include <Wire.h>
#include <Adafruit_MotorShield.h>
#include "utility/Adafruit_PWMServoDriver.h"
```

Create the Adafruit_MotorShield object

```
Adafruit_MotorShield AFMS = Adafruit_MotorShield();
```

Create the stepper motor object

Request the Stepper motor from the Adafruit_MotorShield:

```
Adafruit_StepperMotor *myMotor = AFMS.getStepper(200, 2);
```

with **getStepper(steps, stepper#)**. **Steps** indicates how many steps per revolution the motor has. A 7.5degree/step motor has $360/7.5 = 48$ steps. **Stepper#** is which port it is connected to. If you're using M1 and M2, its port **1**. If you're using M3 and M4 indicate port **2**

Set default speed

Set the speed of the motor using **setSpeed(rpm)** where rpm is how many revolutions per minute you want the stepper to turn.

Run the motor

Then every time you want the motor to move, call the **step(#steps, direction, steptype)** procedure. **#steps** is how many steps you'd like it to take. direction is either **FORWARD** or **BACKWARD** and the step type is **SINGLE**, **DOUBLE**, **INTERLEAVE** or **MICROSTEP**.

- "Single" means single-coil activation
- "Double" means 2 coils are activated at once (for higher torque)
- "Interleave" means that it alternates between single and double to get twice the resolution (but of course its half the speed).
- "Microstepping" is a method where the coils are PWM'd to create smooth motion between steps.

Theres tons of information about the pros and cons of these different stepping methods in the resources page.

You can use whichever stepping method you want, changing it "on the fly" to as you may

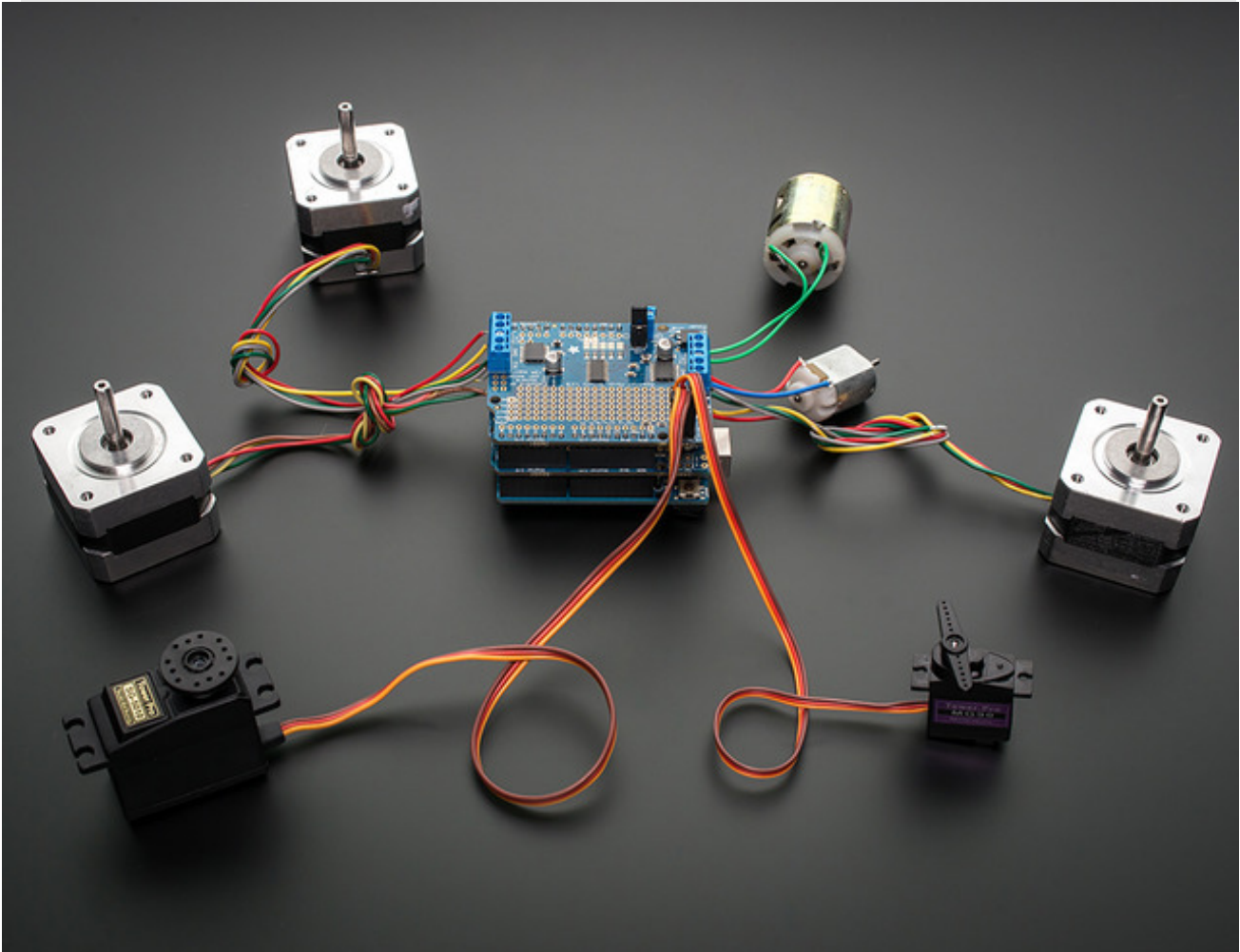
want minimum power, more torque, or more precision.

By default, the motor will 'hold' the position after its done stepping. If you want to release all the coils, so that it can spin freely, call **release()**

The stepping commands are 'blocking' and will return once the steps have finished.

Because the stepping commands 'block' - you have to instruct the Stepper motors each time you want them to move. If you want to have more of a 'background task' stepper control, [check out AccelStepper library \(http://adafru.it/aOL\)](http://adafru.it/aOL) (install similarly to how you did with Adafruit_MotorShield) which has some examples for controlling three steppers simultaneously with varying acceleration

Stacking Shields



One of the cool things about this shield design is that it is possible to stack shields. Every shield you stack can control another 2 steppers or 4 DC motors (or a mix of the two)

You can stack up to 32 shields for a total of 64 steppers or 128 DC motors! Most people will probably just stack two or maybe three but hey, you never know. (PS if you drive 64 steppers from one of these shields send us a photo, OK?)

Note that stacking shields does not increase the servo connections - those are hard-wired to the Arduino digital 9 & 10 pins. [If you need to control a lot of servos, you can use our 16-channel servo shield and stack it with this shield to add a crazy large # of servos.](#) (<http://adafruit.it/1411>)

Stacking shields is very easy. Each shield you want to stack on top of must have stacking headers installed. [Check our instructions for how to do so.](#) (<http://adafruit.it/ciP>) The top shield does not have to have stacking headers unless you eventually want to put something on top of it.

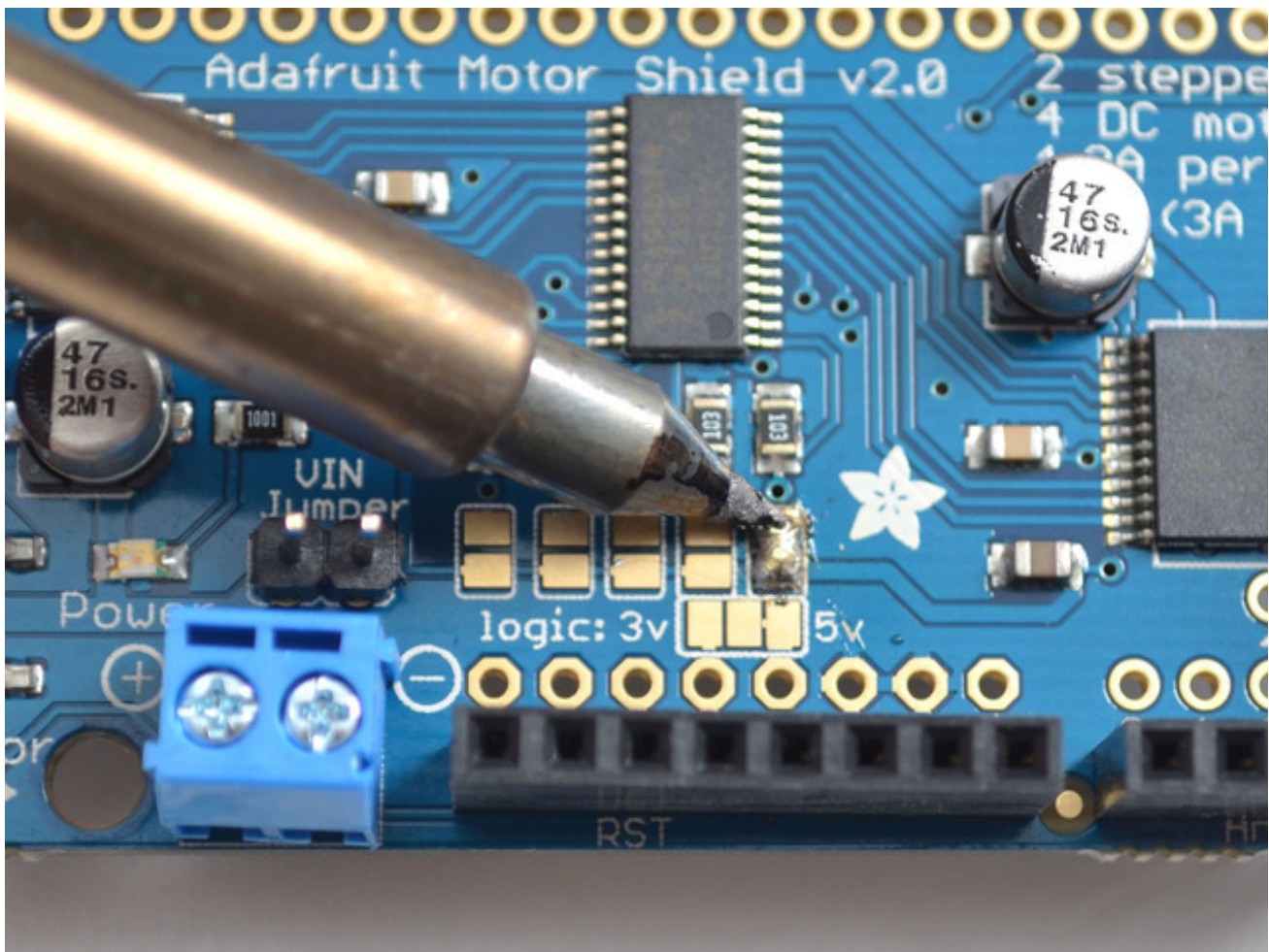
The only thing to watch for when stacking shields is every shield must have a unique I2C address. The default address is **0x60**. You can adjust the address of the shields to range from 0x60 to 0x80 for a total of 32 unique addresses.

Addressing the Shields

Each board in the chain must be assigned a unique address. This is done with the address jumpers on the lower edge of the board. The I2C base address for each board is 0x60. The binary address that you program with the address jumpers is added to the base I2C address.

To program the address offset, use a drop of solder to bridge the corresponding address jumper for each binary '1' in the address.

The right-most jumper is address bit #0, then to the left of that is address bit #1, etc up to address bit #5



Board 0: Address = 0x60 Offset = binary 0000 (no jumpers required)

Board 1: Address = 0x61 Offset = binary 0001 (bridge A0 as in the photo above)

Board 2: Address = 0x62 Offset = binary 0010 (bridge A1, to the left of A0)

Board 3: Address = 0x63 Offset = binary 0011 (bridge A0 & A1, two rightmost jumpers)

Board 4: Address = 0x64 Offset = binary 0100 (bridge A2, middle jumper)

etc.

Writing Code for Multiple Shields

The Adafruit_MotorShield library has the ability to control multiple shields, unlike the older AF_Motor library. First we must create a Motor Shield Controller for each shield, with the assigned address.

```
Adafruit_MotorShield AFMSbot(0x61); // Rightmost jumper closed
Adafruit_MotorShield AFMStop(0x60); // Default address, no jumpers
```

One motor shield is going to be called AFMSbot (bottom shield, so we remember) and one is AFMStop (top shield) so we can keep them apart. When you create the shield object, specify the address you set for it above.

Then we can request the motors connected to each one

```
// On the top shield, connect two steppers, each with 200 steps
Adafruit_StepperMotor *myStepper2 = AFMStop.getStepper(200, 1);
Adafruit_StepperMotor *myStepper3 = AFMStop.getStepper(200, 2);

// On the bottom shield connect a stepper to port M3/M4 with 200 steps
Adafruit_StepperMotor *myStepper1 = AFMSbot.getStepper(200, 2);
// And a DC Motor to port M1
Adafruit_DCMotor *myMotor1 = AFMSbot.getMotor(1);
```

You can request a stepper or DC motor from any port, just be sure to use the right AFMS controller object when you call **getMotor** or **getStepper**!

Then, both shields must have **begin** called, before you use the motors connected

```
AFMSbot.begin(); // Start the bottom shield
AFMStop.begin(); // Start the top shield
```

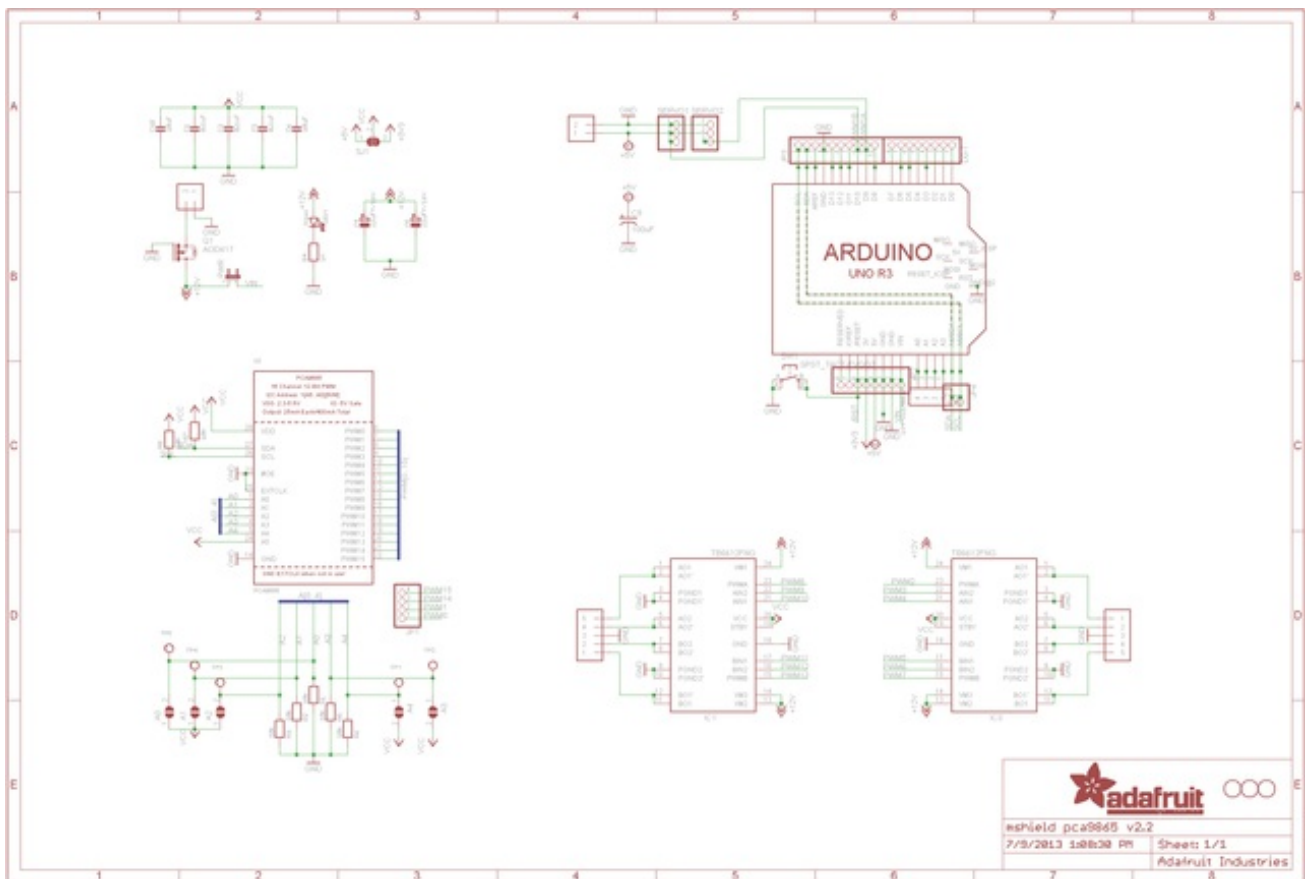
You can try out this code for yourself by setting up two shields and running the **File->Examples->Adafruit_MotorShield->StackingTest** example

Resources

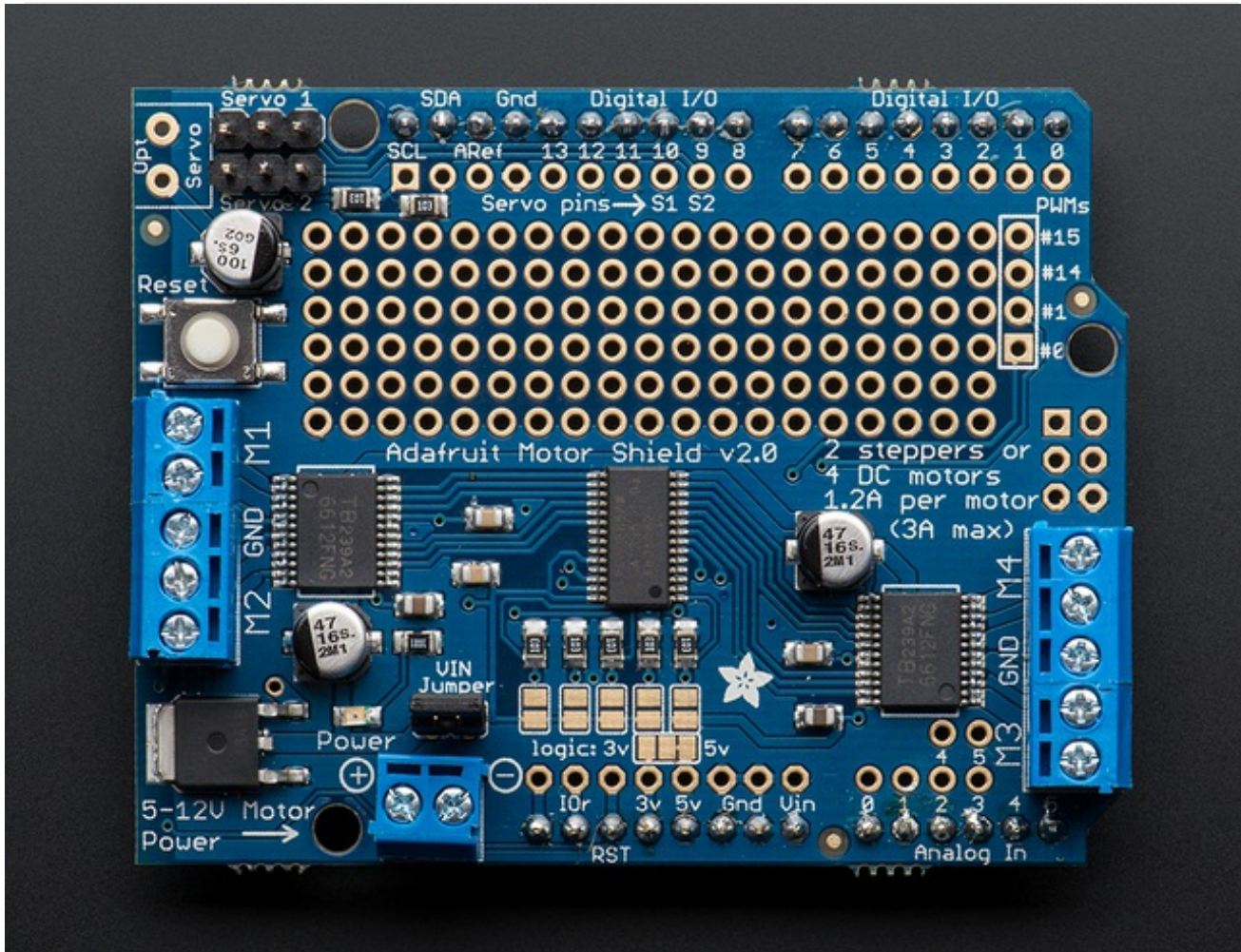
Motor ideas and tutorials

- [Wikipedia has tons of information \(http://adafru.it/aOF\)](http://adafru.it/aOF) on steppers
- [Jones on stepper motor types \(http://adafru.it/aOH\)](http://adafru.it/aOH)
- [Jason on reverse engineering the stepper wire pinouts \(http://adafru.it/aOI\)](http://adafru.it/aOI)

Schematic, click to embiggen



Library Reference



`class Adafruit_MotorShield;`

The `Adafruit_MotorShield` class represents a motor shield and must be instantiated before any `DCMotors` or `StepperMotors` can be used. You will need to declare one `Adafruit_MotorShield` for each shield in your system.

`Adafruit_MotorShield(uint8_t addr = 0x60);`

The constructor takes one optional parameter to specify the i2c address of the shield. The default address of the constructor (0x60) matches the default address of the boards as shipped. If you have more than one shield in your system, each shield must have a unique address.

`void begin(uint16_t freq = 1600);`

`begin()` must be called in `setup()` to initialize the shield. An optional frequency parameter can be used to specify something other than the default maximum: 1.6KHz PWM frequency.

Adafruit_DCMotor *getMotor(uint8_t n);

This function returns one of 4 pre-defined DC motor objects controlled by the shield. The parameter specifies the associated motor channel: 1-4.

Adafruit_StepperMotor *getStepper(uint16_t steps, uint8_t n);

This function returns one of 2 pre-defined stepper motor objects controlled by the shield.

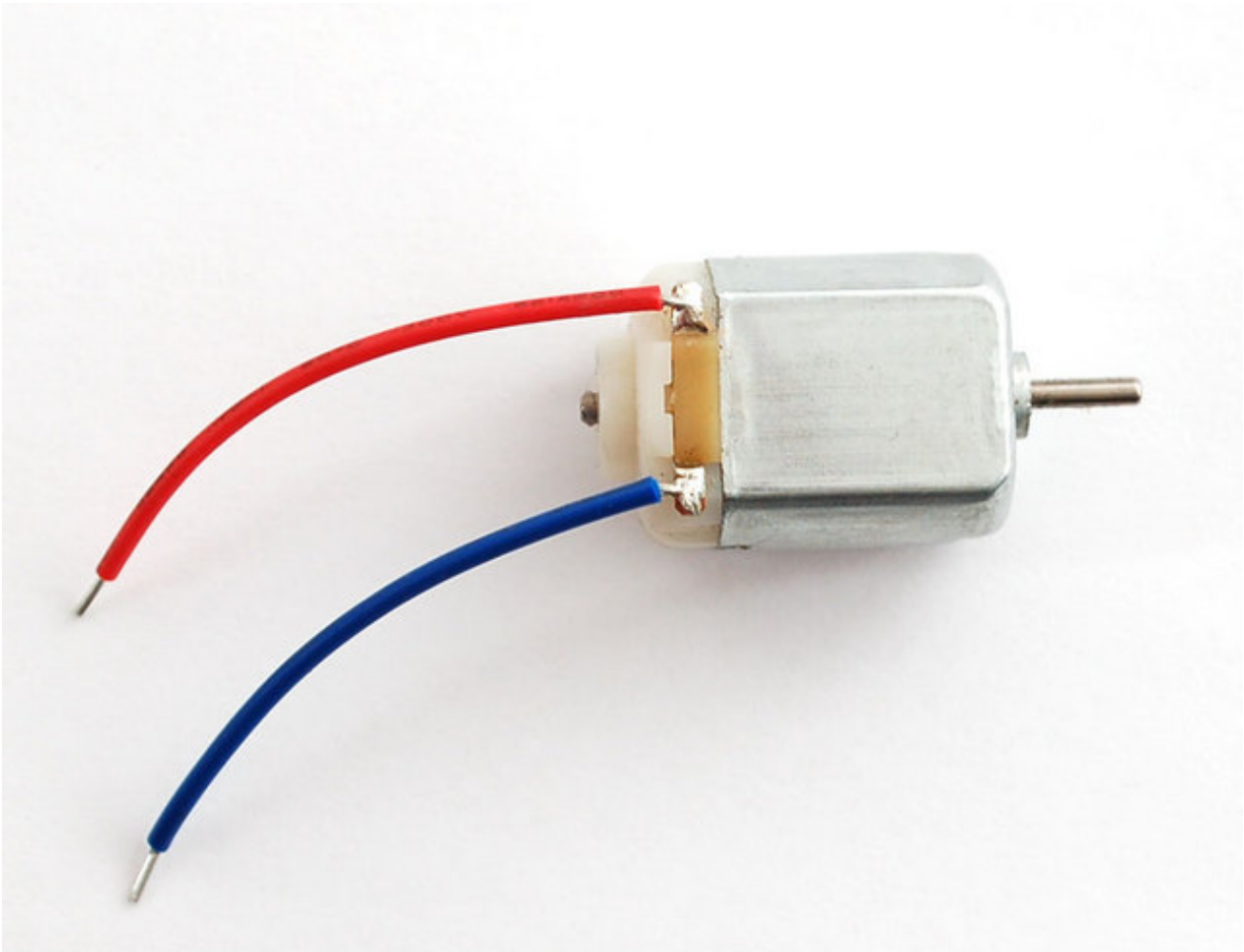
The first parameter specifies the number of steps per revolution.

The second parameter specifies the associated stepper channel: 1-2.

void setPWM(uint8_t pin, uint16_t val);

void setPin(uint8_t pin, boolean val);

These are low-level functions to control pins on the on-board PWM driver chip. These functions are intended for internal use only.



class Adafruit_DCMotor

The Adafruit_DCMotor class represents a DC motor attached to the shield. You must declare an Adafruit_DCMotor for each motor in your system.

Adafruit_DCMotor(void);

The constructor takes no arguments. The motor object is typically initialized by assigning a motor object retrieved from the shield class as below:

```
// Create the motor shield object with the default I2C address
Adafruit_MotorShield AFMS = Adafruit_MotorShield();

// Select which 'port' M1, M2, M3 or M4. In this case, M1
Adafruit_DCMotor *myMotor = AFMS.getMotor(1);
// You can also make another motor on port M2
Adafruit_DCMotor *myOtherMotor = AFMS.getMotor(2);
```

void run(uint8_t);

The run() function controls the motor state. The parameter can have one of 3 values:

- **FORWARD** - Rotate in a forward direction
- **REVERSE** - Rotate in the reverse direction
- **RELEASE** - Stop rotation

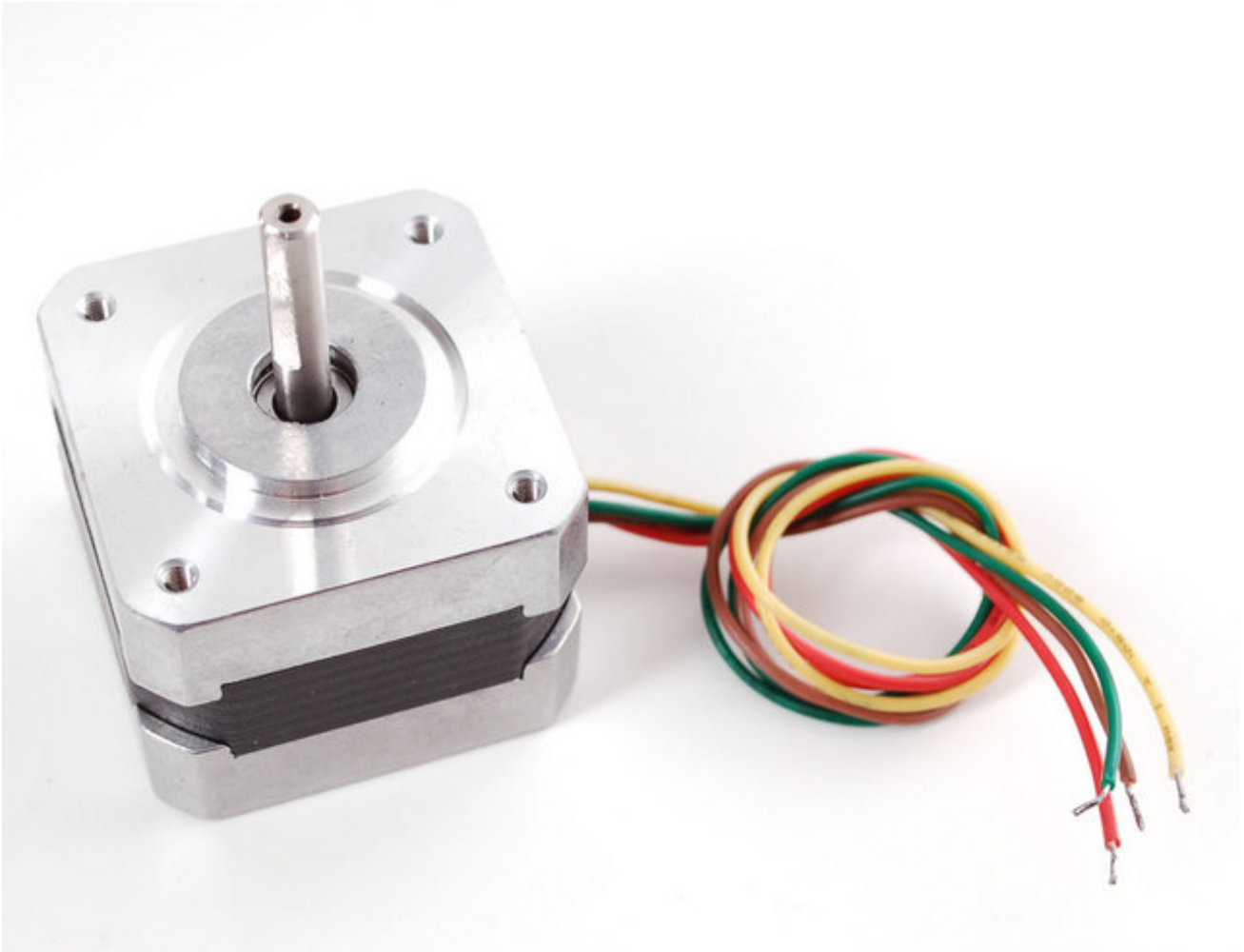
Note that the "FORWARD" and "REVERSE" directions are arbitrary. If they do not match the actual direction of your vehicle or robot, simply swap the motor leads.

Also note that "RELEASE" simply cuts power to the motor. It does not apply any braking.

void setSpeed(uint8_t);

The setSpeed() function controls the power level delivered to the motor. The speed parameter is a value between 0 and 255.

Note that setSpeed just controls the power delivered to the motor. The actual speed of the motor will depend on several factors, including: The motor, the power supply and the load.



class Adafruit_StepperMotor

The Adafruit_StepperMotor class represents a stepper motor attached to the shield. You must declare an Adafruit_StepperMotor for each stepper motor in your system.

Adafruit_StepperMotor(void);

The constructor takes no arguments. The stepper motor is typically initialized by assigning a stepper object retrieved from the shield as below:

```
// Create the motor shield object with the default I2C address
Adafruit_MotorShield AFMS = Adafruit_MotorShield();

// Connect a stepper motor with 200 steps per revolution (1.8 degree)
// to motor port #2 (M3 and M4)
Adafruit_StepperMotor *myMotor = AFMS.getStepper(200, 2);
```

void step(uint16_t steps, uint8_t dir, uint8_t style = SINGLE);

The step() function controls stepper motion.

- The first parameter specifies how many steps to move.
- The second parameter specifies the direction: FORWARD or BACKWARD
- The last parameter specifies the stepping style: SINGLE, DOUBLE, INTERLEAVED or MICROSTEP

The `ste()` function is synchronous and does not return until all steps are complete. When complete the motor remains powered to apply "holding torque" to maintain position.

`void setSpeed(uint16_t);`

The `setSpeed()` function controls the speed of the stepper motor rotation. Speed is specified in RPM.

`uint8_t onestep(uint8_t dir, uint8_t style);`

The `oneStep()` function is a low-level internal function called by `step()`. But it can be useful to call on its own to implement more advanced functions such as acceleration or coordinating simultaneous movement of multiple stepper motors. The direction and style parameters are the same as for `step()`, but `onestep()` steps exactly once.

Note: Calling `step()` with a step count of 1 is not the same as calling `onestep()`. The `step` function has a delay based on the speed set in `setSpeed()`. `onestep()` has no delay.

`void release(void);`

The `release()` function removes all power from the motor. Call this function to reduce power requirements if holding torque is not required to maintain position.