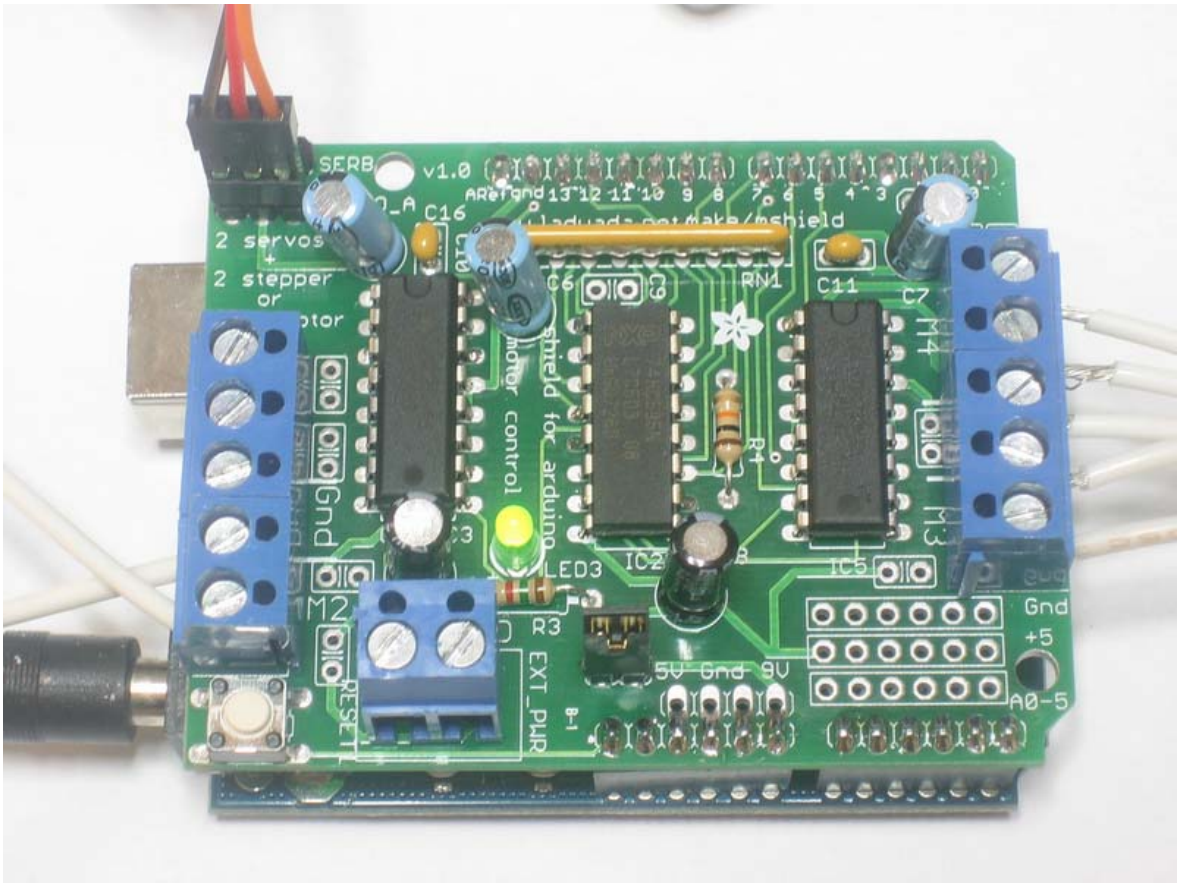
CNARDUINO

L293D Stepper Motor Controller

COMPANY: CNARDUINO-SHENZEN

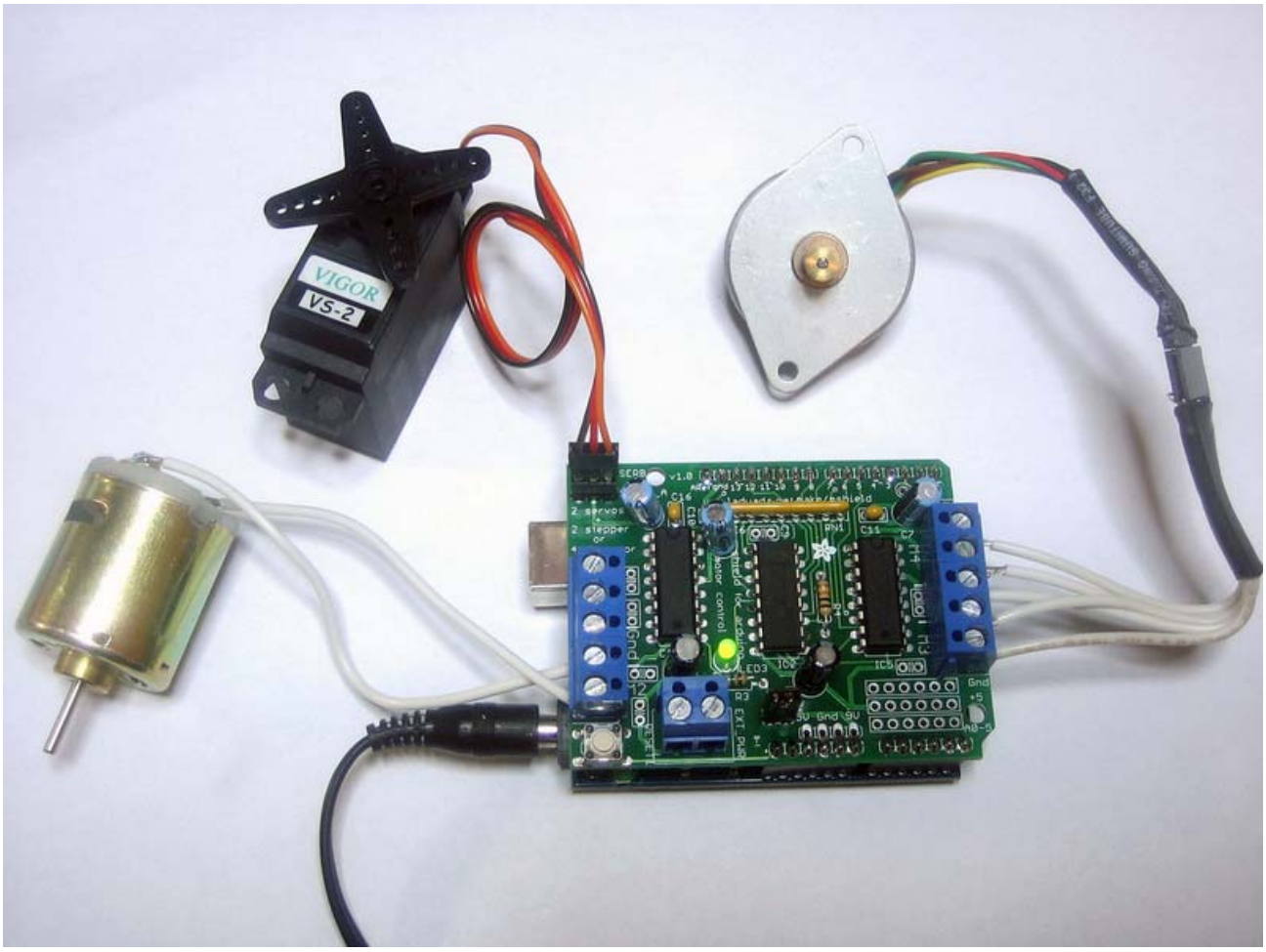
L293D STEPPER MOTOR CONTROLLER

11.01 OVERVIEW



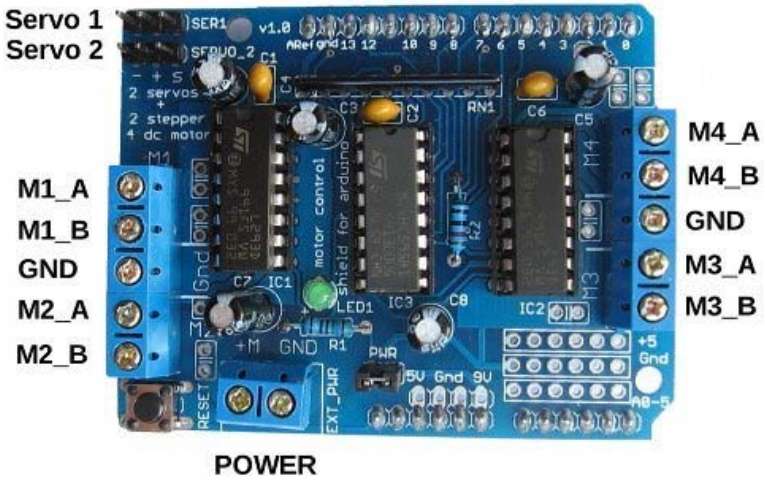
Arduino is a great starting point for electronics, and with a motor shield it can also be a nice tidy platform for robotics and mechatronics. Here is a design for a full-featured motor shield that will be able to power many simple to medium-complexity projects.

- **connections for 5V 'hobby' servos** connected to the Arduino's high-resolution dedicated timer - no jitter!
- **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.
- H-Bridges: L293D chipset provides 0.6A per bridge (1.2A peak) with thermal shutdown protection, 4.5V to 25V
- Pull down resistors keep motors disabled during power-up
- Big terminal block connectors to easily hook up wires (10-22AWG) and power
- Arduino reset button brought up top
- 2-pin terminal block to connect external power, for seperate logic/motor supplies
- Tested compatible with Mega, Diecimila, & Duemilanove



11.02 PROCEDURE TO USE

This Motor Shield kit is a great motor controller for Arduino, but it does a little care to make sure it's used correctly. Please read through all the User manual sections at left, especially the section about library installation and power requirements!



LIBRARY INSTALL

First Install the Arduino Library

Before you can use the Motor shield, you **must** install the **AF_Motor** Arduino library - this will instruct the Arduino how to talk to the this Motor shield, and it isn't optional!

- First, grab the library from github
- Uncompress the ZIP file onto your desktop
- Rename the uncompressed folder **AFMotor**
- Check that inside **AFMotor** is **AFMotor.cpp** and **AFMotor.h** files. If not, check the steps above.
- Place the **AFMotor** folder into your *arduinosketchfolder*/**libraries** folder. For Windows, this will probably be something like **MY Documents/Arduino/libraries** for Mac it will be something like**Documents/arduino/libraries**. If this is the first time you are installing a library, you'll need to create the **libraries** folder. Make sure to call it **libraries** exactly, no caps, no other name.
- Check that inside the **libraries** folder there is the **AFMotor** folder, and inside **AFMotor** is**AFMotor.cppAFMotor.h** and some other files
- Quit and restart the IDE. You should now have a submenu called **File->Examples->AFMotor->MotorParty**

11.03 POWER USAGE

POWERING YOUR DC MOTORS, VOLTAGE AND CURRENT REQUIREMENTS

Motors need a lot of energy, especially cheap motors since they're less efficient. 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 **4.5V to 25V**.

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 600 mA per motor, with 1.2A peak current. Note that once you head towards 1A you'll probably want to put a heatsink on the motor driver, otherwise you will get thermal failure, possibly burning out the chip.

On using the SN754410: Some people use the SN754410 motor driver chip because it is pin-compatible, has output diodes and can provide 1A per motor, 2A peak. After careful reading of the datasheet and discussion with TI tech support and power engineers it appears that **the output diodes were designed for**

ESD protection only and that using them as kickback-protection is a hack and not guaranteed for performance. For that reason the kit does not come with the SN754410 and instead uses the L293D with integrated kickback-protection diodes. If you're willing to risk it, and need the extra currrent, feel free to buy SN754410's and replace the provided chips.

Need more power? Buy another set of L293D drivers and solder them right on top of the ones on the board (piggyback). Voila, double the current capability! You can solder 2 more chips on top before it probably isnt going to get you much benefit

You can't run motors off of a 9V battery so don't even 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!

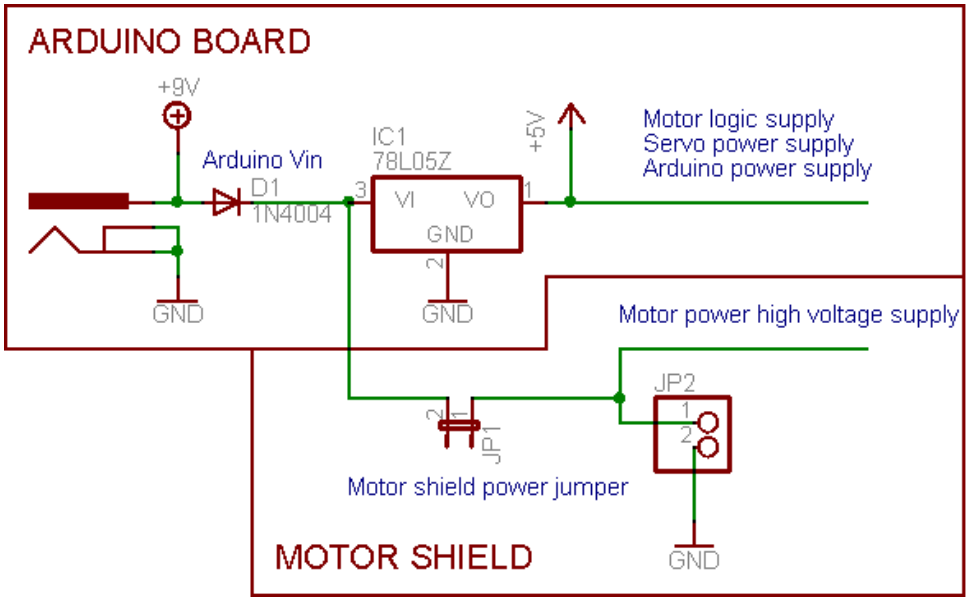
HOW TO SET UP THE ARDUINO + SHIELD FOR POWERING MOTORS

Servos are powered off of the same regulated 5V that the Arduino uses. This is OK for the small hobby servos suggested. If you want something beefier, cut the trace going to + on the servo connectors and wire up your own 5-6V supply!

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 5V line**. This is a very very very bad idea unless you are sure you know what you're doing!

There are two places you can get your motor 'high voltage supply' from. One is the DC jack on the Arduino board and the other is the 2-terminal block on the shield that is labeled **EXT_PWR**. 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. However the **EXT_PWR terminals on the shield do not have a protection diode** (for a fairly good reason). **Be utterly careful not to plug it in backwards** or you will destroy the motor shield and/or your Arduino!

Here's how it works:



If you would like to have a **single DC power supply for the Arduino and motors**, simply plug it into the DC jack on the Arduino or the 2-pin PWR_EXT block on the shield. Place the power jumper on the motor shield.

If you have a Diecimila Arduino, set the Arduino power source jumper to EXT. Note that you may have problems with Arduino resets if the battery supply is not able to provide constant power, and it is not a suggested way of powering your motor project

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 PWR_EXT block on the shield. Do not place the jumper on the shield. This is a suggested method of powering your motor project (If you have a Diecimila Arduino, don't forget to set the Arduino power jumper to USB. If you have a Diecimila, you can alternately do the following: plug the DC power supply into the Arduino, and place the jumper on the motor shield.)

If you would like to have **2 seperate 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 PWR_EXT block. Make sure the jumper is removed from the motor shield.

If you have a Diecimila Arduino, set the Arduino jumper to EXT. This is a suggested method of powering your motor project

Either way, if you want to use the DC motor/Stepper system the motor shield LED should be lit indicating good motor power.

USE OF 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 16bit PWM output lines 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

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 trace right below the servo pins (on v1.2 boards) and connect a 5V or 6V DC supply directly. Using an external supply is for advanced users as you can accidentally destroy the servos by connecting a power supply incorrectly!

When using the external supply header for servos, take care that the bottom of the header pins do not contact the metal USB port housing on the Arduino. A piece of electrical tape on the housing will protect against shorts.

11.04 KNOB WITH RC SERVO WITHOUT MOTOR CONTROLLER

Control the position of a RC (hobby) servo motor with your Arduino and a potentiometer.
This example makes use of the Arduino servo library.

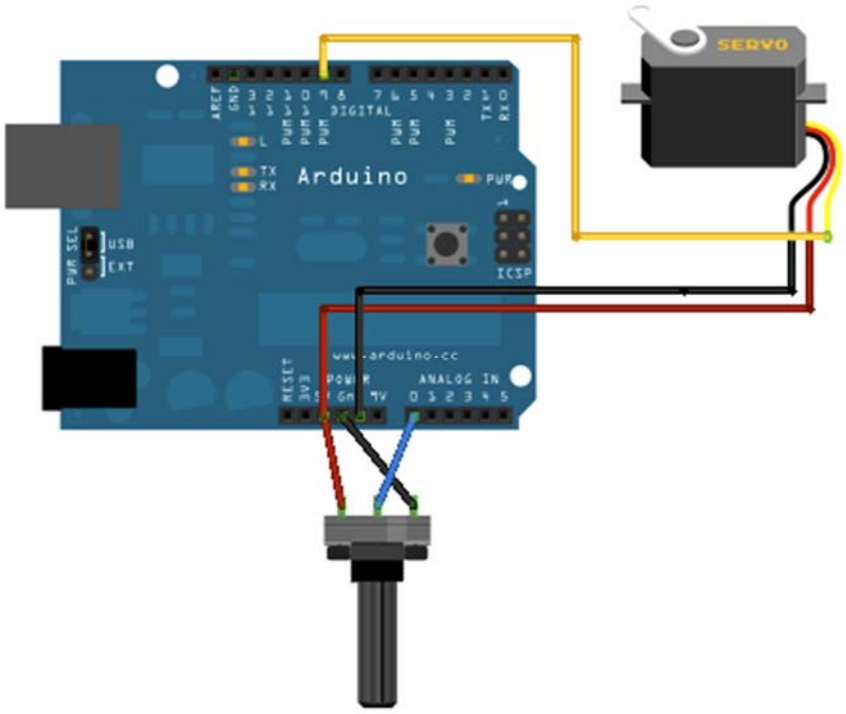
HARDWARE REQUIRED

- Arduino Board
- Servo Motor
- Potentiometer
- hook-up wire

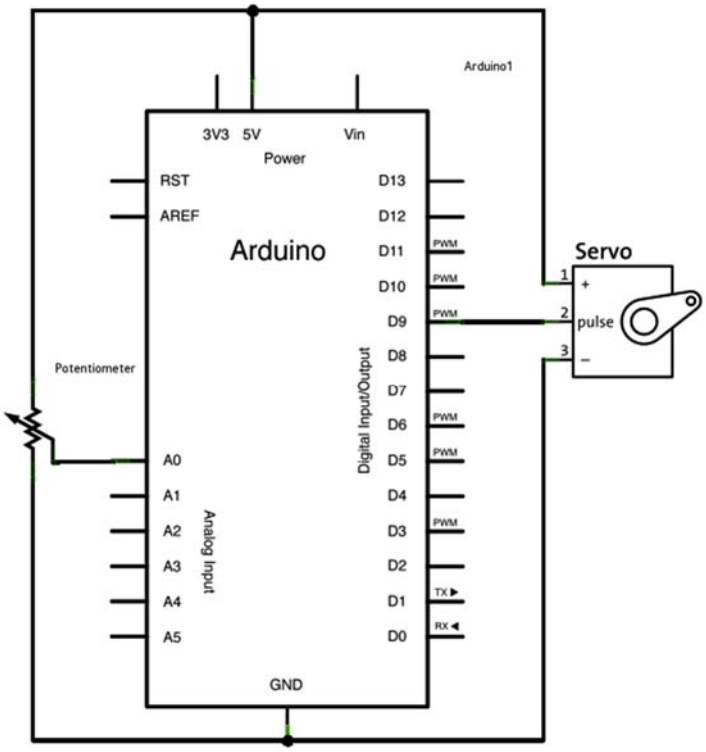
CIRCUIT

Servo motors have three wires: power, ground, and signal. The power wire is typically red, and should be connected to the 5V pin on the Arduino board. The ground wire is typically black or brown and should be connected to a ground pin on the Arduino board. The signal pin is typically yellow or orange and should be connected to pin 9 on the Arduino board.

The potentiometer should be wired so that its two outer pins are connected to power (+5V) and ground, and its middle pin is connected to analog input 0 on the Arduino.



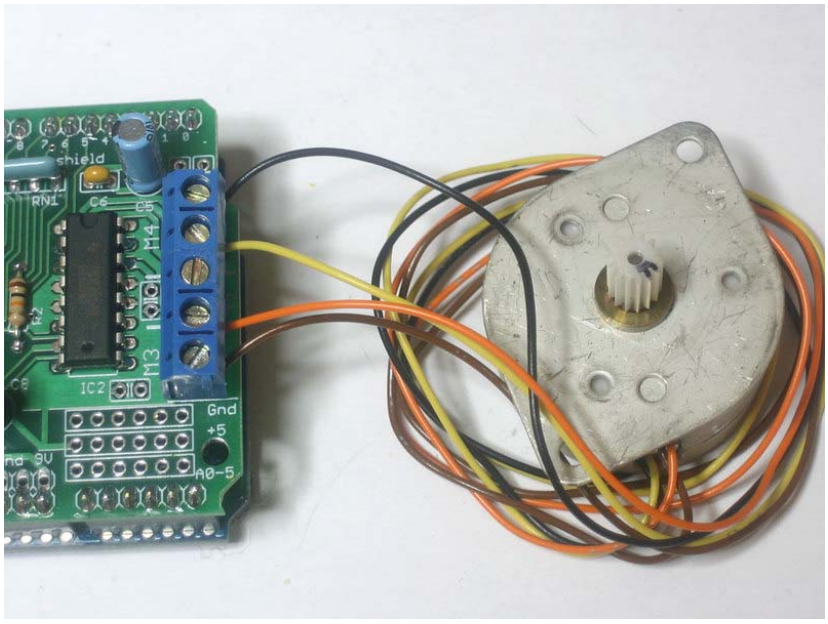
SCHEMATIC



CODE

```
// Controlling a servo position using a potentiometer (variable
//resistor)
// by Michal Rinott <http://people.interaction-ivrea.it/m.rinott>

#include <Servo.h>

Servo myservo;  // create servo object to control a servo

int potpin = 0;  // analog pin used to connect the potentiometer
int val;    // variable to read the value from the analog pin

void setup()
{
  myservo.attach(9);  // attaches the servo on pin 9 to the servo
//object
}

void loop()
{
  val = analogRead(potpin);            // reads the value of the
//potentiometer (value between 0 and 1023)
  val = map(val, 0, 1023, 0, 179);     // scale it to use it with the
//servo (value between 0 and 180)
  myservo.write(val);                  // sets the servo position
//according to the scaled value
  delay(15);                           // waits for the servo to get
//there
}
```

11.05 USING STEPPER MOTORS WITH MOTOR CONTROLLER



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 it's a 5-wire motor then there will be 1 that is the center tap for both coils. 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: it's just like unipolar motors except there's 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

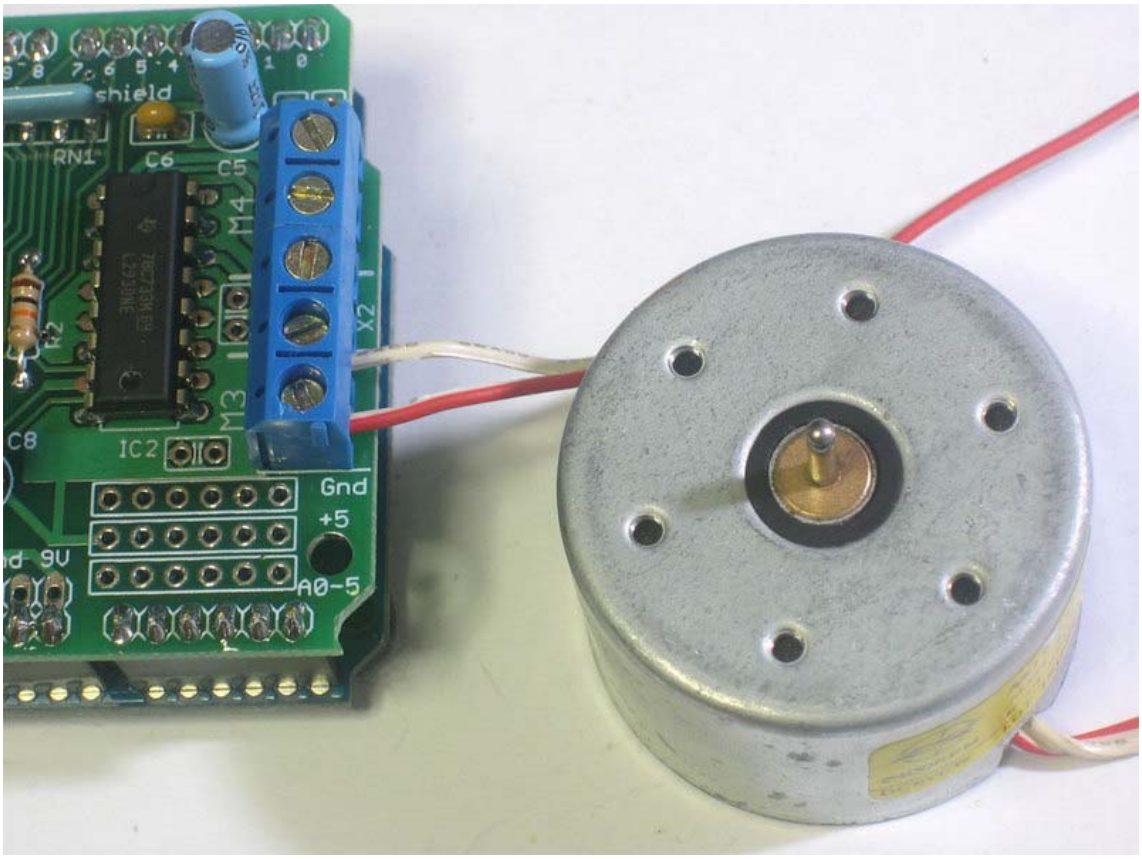
- Make sure you #include <AFMotor.h>
- Create the stepper motor object with **AF_Stepper(***steps,* **stepper#)** to setup the motor H-bridge and latches. **Steps** indicates how many steps per revolution the motor has. a 7.5degree/step motor has 360/7.5 = 48 steps. **Stepper#** is which port it is connected to. If you're using M1 and M2, its port 1. If you're using M3 and M4 it's port 2
- Set the speed of the motor using **setSpeed(***rpm***)** where **rpm** is how many revolutions per minute you want the stepper to turn.

- 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) and "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 (install similarly to how you did with AFMotor) which has some examples for controlling two steppers simultaneously with varying acceleration, here is guide to Arduino libraries.

```
#include <AFMotor.h>
AF_Stepper motor(48, 2);
void setup() {
  Serial.begin(9600);           // set up Serial library at 9600 bps
  Serial.println("Stepper test!");
  motor.setSpeed(10);  // 10 rpm

  motor.step(100, FORWARD, SINGLE);
  motor.release();
  delay(1000);
}

void loop() {
  motor.step(100, FORWARD, SINGLE);
  motor.step(100, BACKWARD, SINGLE);

  motor.step(100, FORWARD, DOUBLE);
  motor.step(100, BACKWARD, DOUBLE);

  motor.step(100, FORWARD, INTERLEAVE);
  motor.step(100, BACKWARD, INTERLEAVE);

  motor.step(100, FORWARD, MICROSTEP);
  motor.step(100, BACKWARD, MICROSTEP);
}
```

11.06 USING DC MOTORS WITH MOTOR CONTROLLER



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 loads over 0.6A or that peak over 1.2A so this is for *small*motors. Check the datasheet for information about the motor to verify its OK.

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

- Make sure you #include <AFMotor.h>
- Create the AF_DCMotor object with **AF_DCMotor(***motor#, frequency***)**, to setup the motor H-bridge and latches. The constructor takes two arguments. The first is which port the motor is connected to, **1, 2, 3** or **4**. *Frequency* is how fast the speed controlling signal is For motors 1 and 2 you can choose **MOTOR12_64KHZ**, **MOTOR12_8KHZ**, **MOTOR12_2KHZ**, or **MOTOR12_1KHZ**. A

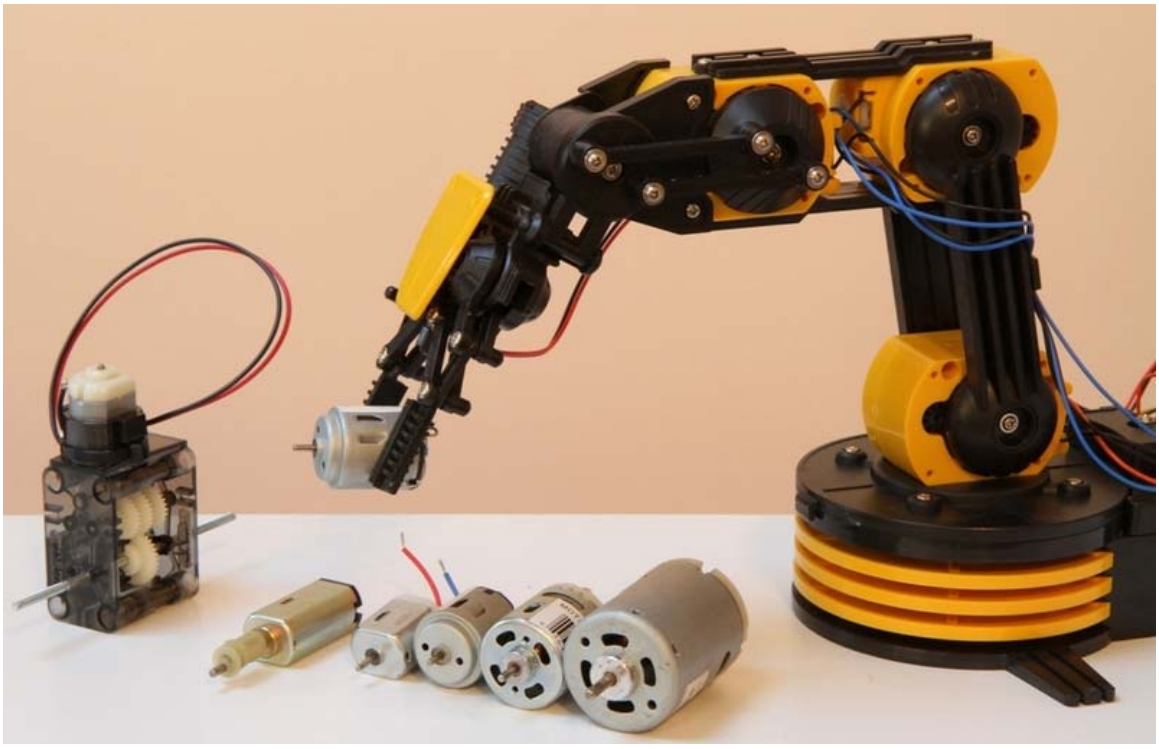
high speed like 64KHz wont be audible but a low speed like 1KHz will use less power. Motors 3 & 4 are only possible to run at 1KHz and will ignore any setting given

- Then you can 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.
- 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.

CODE

```
#include <AFMotor.h>

AF_DCMotor motor(2, MOTOR12_64KHZ); // create motor #2, 64KHz pwm

void setup() {
  Serial.begin(9600);           // set up Serial library at 9600 bps
  Serial.println("Motor test!");

  motor.setSpeed(200);     // set the speed to 200/255
}

void loop() {
  Serial.print("tick");

  motor.run(FORWARD);      // turn it on going forward
  delay(1000);

  Serial.print("tock");
  motor.run(BACKWARD);     // the other way
  delay(1000);

  Serial.print("tack");
  motor.run(RELEASE);      // stopped
  delay(1000);
}
```

11.07 AF_DCMOTOR CLASS



The AF_DCMotor class provides speed and direction control for up to four DC motors when used with this Motor Shield. To use this in a sketch you must first add the following line at the beginning of your sketch:

```
1. #include <AFMotor.h>
```

AF_DCMotor motorname(portnum, freq)

This is the constructor for a DC motor. Call this constructor once for each motor in your sketch. Each motor instance must have a different name as in the example below.

Parameters:

- **port num** - selects which channel (1-4) of the motor controller the motor will be connected to
- **freq** - selects the PWM frequency. If no frequency is specified, 1KHz is used by default.

Frequencies for channel 1 & 2 are:

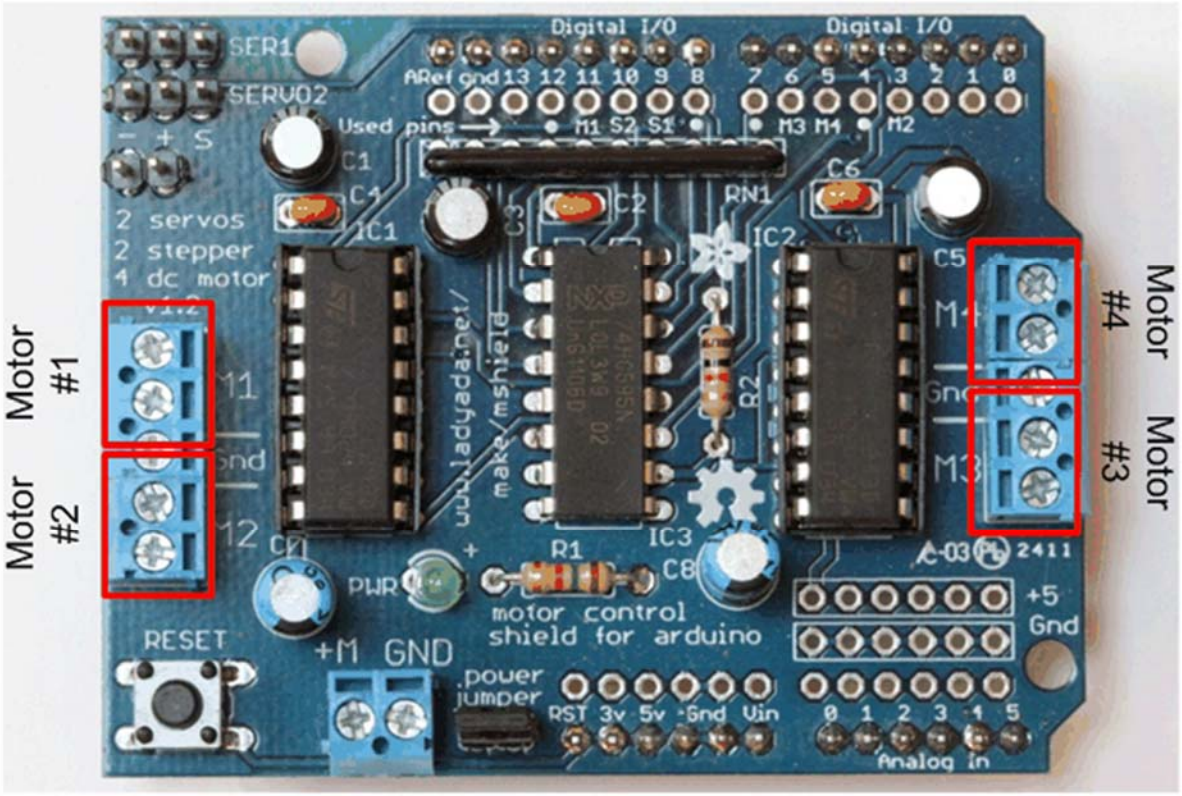
- MOTOR12_64KHZ
- MOTOR12_8KHZ
- MOTOR12_2KHZ
- MOTOR12_1KHZ

Frequencies for channel 3 & 4 are:

- MOTOR34_64KHZ
- MOTOR34_8KHZ
- MOTOR34_1KHZ

Example:

```
1. AF_DCMotor motor4(4); // define motor on channel 4 with 1KHz default PWM
2. AF_DCMotor left_motor(1, MOTOR12_64KHZ);  // define motor on channel 1 with 64KHz PWM
```



Note: Higher frequencies will produce less audible hum in operation, but may result in lower torque with some motors.

setSpeed(speed)

Sets the speed of the motor.

Parameters:

- **speed**- Valid values for 'speed' are between 0 and 255 with 0 being off and 255 as full throttle.

Note: DC Motor response is not typically linear, and so the actual RPM will not necessarily be proportional to the programmed speed.

run(cmd)

Sets the run-mode of the motor.

Parameters:

- **cmd** - the desired run mode for the motor

Valid values for cmd are:

- **FORWARD** - run forward (actual direction of rotation will depend on motor wiring)
- **BACKWARD** - run backwards (rotation will be in the opposite direction from FORWARD)
- **RELEASE** - Stop the motor. This removes power from the motor and is equivalent to setSpeed(0). The motor shield does not implement dynamic breaking, so the motor may take some time to spin down

Example:

```
motor.run(FORWARD);
delay(1000);  // run forward for 1 second
motor.run(RELEASE);
delay(100);  // 'coast' for 1/10 second
motor.run(BACKWARDS);  // run in reverse
```

11.08 AF_STEPPER CLASS

The AF_Stepper class provides single and multi-step control for up to 2 stepper motors when used with the Adafruit Motor Shield. To use this in a sketch you must first add the following line at the beginning of your sketch:

```
1. #include <AFMotor.h>
```

AF_Stepper steppername(steps, portnumber)

The AF_Stepper constructor defines a stepper motor. Call this once for each stepper motor in your sketch. Each stepper motor instance must have a unique name as in the example below.

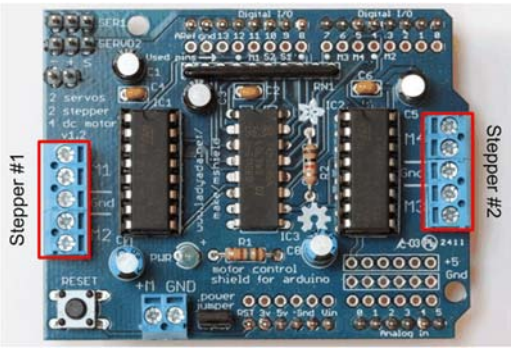
Parameters:

- **steps** - declare the number of steps per revolution for your motor.
- **num** - declare how the motor will be wired to the shield.

Valid values for 'num' are 1 (channels 1 & 2) and 2 (channels 3 & 4).

Example:

```
AF_Stepper Stepper1(48, 1);  // A 48-step-per-revolution motor on channels 1 & 2
AF_Stepper Stepper2(200, 2);   // A 200-step-per-revolution motor on channels 3 & 4
```



step(steps, direction, style)

Step the motor.

Parameters:

- **steps** - the number of steps to turn
- **direction** - the direction of rotation (**FORWARD** or **BACKWARD**)
- **style** - the style of stepping:

Valid values for 'style' are:

- **SINGLE** - One coil is energized at a time.
- **DOUBLE** - Two coils are energized at a time for more torque.
- **INTERLEAVE** - Alternate between single and double to create a half-step in between. This can result in smoother operation, but because of the extra half-step, the speed is reduced by half too.
- **MICROSTEP** - Adjacent coils are ramped up and down to create a number of 'micro-steps' between each full step. This results in finer resolution and smoother rotation, but with a loss in torque.

Note: Step is a synchronous command and will not return until all steps have completed. For concurrent motion of two motors, you must handle the step timing for both motors and use the "onestep()" function below.

Example:

```
Stepper1.step(100, FORWARD, DOUBLE); // 100 steps forward using double coil stepping
Stepper2.step(100, BACKWARD, MICROSTEP);   // 100 steps backward using double microstepping
```

setSpeed(RPMspeed)

set the speed of the motor

Parameters:

- **Speed** - the speed in RPM

Note: The resulting step speed is based on the 'steps' parameter in the constructor. If this does not match the number of steps for your motor, you actual speed will be off as well.

Example:

```
Stepper1.setSpeed(10);  // Set motor 1 speed to 10 rpm
Stepper2.setSpeed(30);  // Set motor 2 speed to 30 rpm
```

onestep(direction, stepstyle)

Single step the motor.

Parameters:

- **direction** - the direction of rotation (**FORWARD** or **BACKWARD**)
- **stepstyle** - the style of stepping:

Valid values for 'style' are:

- **SINGLE** - One coil is energized at a time.
- **DOUBLE** - Two coils are energized at a time for more torque.
- **INTERLEAVE** - Alternate between single and double to create a half-step in between. This can result in smoother operation, but because of the extra half-step, the speed is reduced by half too.
- **MICROSTEP** - Adjacent coils are ramped up and down to create a number of 'micro-steps' between each full step. This results in finer resolution and smoother rotation, but with a loss in torque.

Example:

```
Stepper1.onestep(FORWARD, DOUBLE);  // take one step forward using double coil stepping
```

release()

Release the holding torque on the motor. This reduces heating and current demand, but the motor will not actively resist rotation.

Example:

```
Stepper1.release(); // stop rotation and turn off holding torque.
```

11.09 ONLINE STORE LINK TO BUY

- http://www.aliexpress.com/store/product/Stepper-Motor-Expension-Board-L293D-Motor-Control-Shield-for-Arduino/1036551_1578135709.html