

# Stepper Motor Control

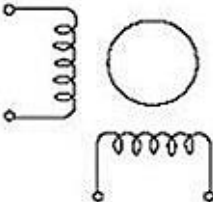
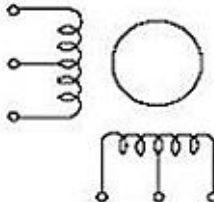
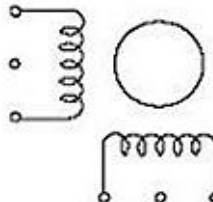
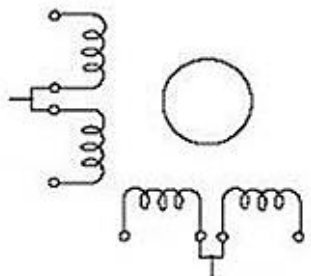
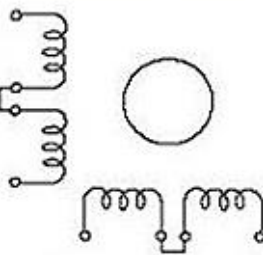
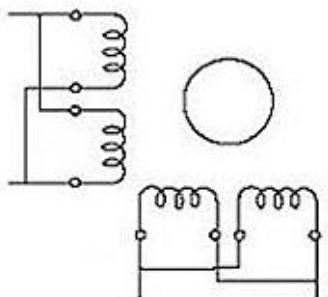
## Overview

There are two types of stepper motors:

**UNIPOLAR** Two windings per phase, allowing the magnetic field to be reversed without having to reverse the direction of current in a coil. Drawback is that only half of the phase is carrying current at any given time, which decreases the torque you can get out of the stepper motor.

**BIPOLAR** One coil per phase, requiring the direction of current to be reversed to reverse the magnetic field. Drawback is that they require more complicated control circuitry (typically an H-bridge for each phase).

### Wire Connection Diagrams

4 Lead Bipolar Connection	6 Lead Unipolar Connection	6 Lead Bipolar (Series) Connection
		
8 Lead Unipolar Connection	8 Lead Bipolar (Series) Connection	8 Lead Bipolar (Parallel) Connection
		

## Standardization

The US National Electrical Manufacturers Association (NEMA) standardized the dimensions, markings, and specifications for stepper motors. This ensures installing motors from different manufacturers all conform to the same hole placement and wiring. For instance, NEMA 17 is a **hybrid stepping motor with a 1.8° step angle (200 steps/revolution)**.

Unfortunately, the standard does not specify the current and voltage ratings. One model could indicate each phase draws 1.2 A at 4 V while another manufacturer specifies 1.7 A. Finding an adequately rated motor driver is crucial for ensuring the motor is meeting the desired performance.

You have to also take the voltage rating of a stepper with a grain of salt. They are not voltage-controlled devices. They are current controlled devices. What you want is a stepper driver that has current feedback so that it drives the coils with a constant current.

### RECOMMENDED MOTORS

MODEL NO.	MOTOR CONNECTION 1 = SERIES 2 = PARALLEL 3 = UNIPOLAR	MOTOR LENGTH mm (inch)	MAXIMUM HOLDING TORQUE <sup>2</sup> (oz-inch)	LEADS	STEP ANGLE (DEG)	VOLTS	AMPS	OHMS	MH	ROTOR INERTIA (oz-inch <sup>2</sup> / g-cm <sup>2</sup> )	MOTOR WEIGHT g (lb)			
OMHT17-075	1	47 (1.85)	62.8	8	1.8	5.7	0.85	6.6	12.0	0.37/68	331 (0.73)			
	2		44.4				2.8	1.70	1.7			3.0		
	3						4.0	1.20	3.3			3.0		
OMHT17-275	1	48 (1.90)	62.3	8	1.8	5.7	0.85	6.6	10.0	0.44/82	357 (0.79)			
	2		44.0				2.8	1.70	1.7			2.5		
	3						4.0	1.20	3.3			2.5		
OMHT17-278	1	63 (2.47)	113.0	8	1.8	6.4	1.0	6.4	12.0	0.66/121	357 (1.32)			
	2		79.0				3.2	2.0	1.6			3.0		
	3						4.5	1.4	3.2			3.0		

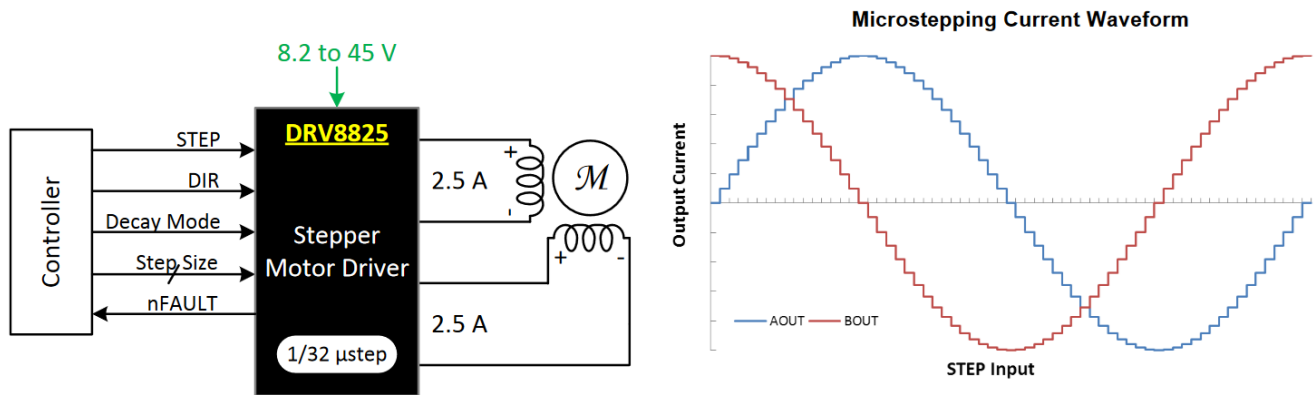
## Microstepping

Microstepping-enabled motor driver will adjust the current in the stator coils to position the permanent magnet rotor in an intermediate position between two subsequent full-steps. A full-step is then divided into a number of microsteps, and each microstep is achieved by the two coil currents.

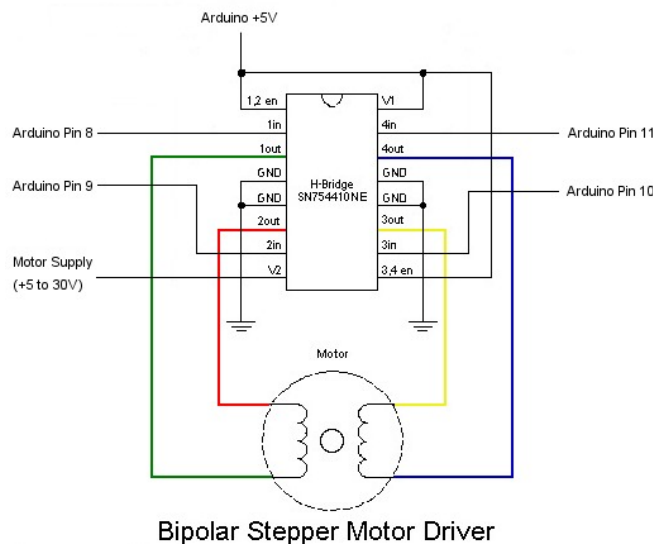
One thing usually not mentioned is that a stepper motor driver can only hold a micro step when powered. Since the microstepping works by current limiting the stators to something like  $\sin(\theta)$  and  $\cos(\theta)$ , if you cut power and disable the driver, the shaft will snap to the nearest full step. So if you want a stepper due to low power draw when not in motion, make sure you can park it on a full step. Keep in mind that even after the motor snaps to the nearest step on power-down, the motor retains just a tiny amount of the full torque capable to hold the shaft. So, once power is removed, they are just a generator and will freely spin. This is important to note for two reasons, you lose position of the shaft and you run the risk of generate reverse current. Be sure your driver has reverse current protection.

# Driving Bipolar Stepper Motors

Although there are two types of stepper motor constructions and over 6 different wiring configurations, this document only focuses on the most common configuration as demonstrated by the DRV8825 datasheet.

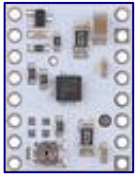


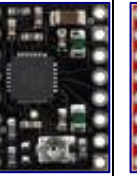
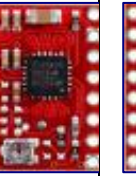



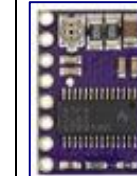
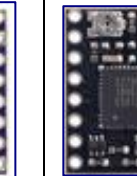
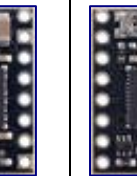
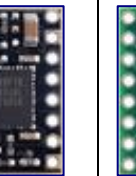
However, there are more than a few references in articles where the SN754410 Quadruple Half-H driver is used. The device is designed to drive inductive loads such as bipolar stepper motors. Note that these dedicated drivers do not contain integrated control circuitry. So microstepping, current sensing, and other useful features typical to a modern motor controller must now be implemented separately.



# Controllers

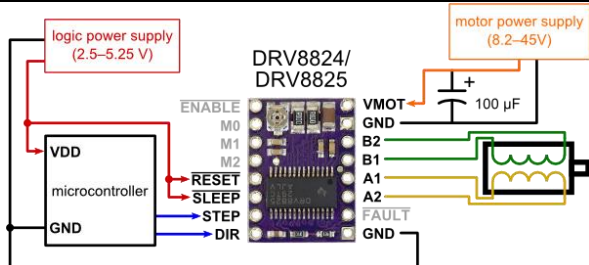
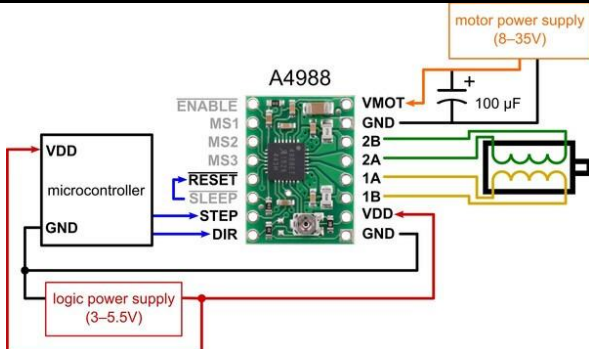
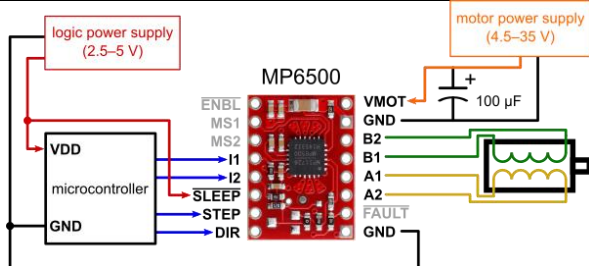
Many controllers on the market include an integrated H-bridge driver. Consequently, configuring a stepper motor loop is quite trivial. Most of them use just three wires, step, direction, and ground, between the Arduino and the driver.

					
<b>Driver chip:</b>	STSPIN220	DRV8834	A4988		MP6500
<b>Min operating voltage:</b>	1.8 V	2.5 V	8 V		4.5 V
<b>Max operating voltage:</b>	10 V	10.8 V	35 V		35 V
<b>Max continuous current per phase:</b>	1.1 A	1.5 A	1 A	1.2 A	1.5 A
<b>Peak current per phase:</b>	1.3 A	2 A	2 A		2.5 A 2 A
<b>Microstepping down to:</b>	1/256	1/32	1/16		1/8
<b>Board layer count:</b>	4	4	2	4	4
<b>Special features:</b>					digital current control
<b>1-piece price:</b>	\$7.95	\$9.95	\$14.45	\$14.45	\$12.95 \$12.95

					
<b>Driver chip:</b>	STSPIN820	DRV8825	TB67S279FTG	TB67S249FTG	DRV8434A
<b>Min operating voltage:</b>	7 V	8.2 V	10 V	10 V	4.5 V
<b>Max operating voltage:</b>	45 V	45 V	47 V	47 V	48 V
<b>Max continuous current per phase:</b>	0.9 A	1.5 A	1.1 A	1.6 A	1.2 A
<b>Peak current per phase:</b>	1.5 A	2.2 A	2 A	4.5 A	2 A
<b>Microstepping down to:</b>	1/256	1/32	1/32	1/32	1/256
<b>Board layer count:</b>	4	4	4	4	4
<b>Special features:</b>			Auto Gain Control, ADMD	Auto Gain Control, ADMD	Stall detect, smart tune ripple control decay
<b>1-piece price:</b>	\$14.95	\$18.95	\$10.75	\$21.95	\$12.95

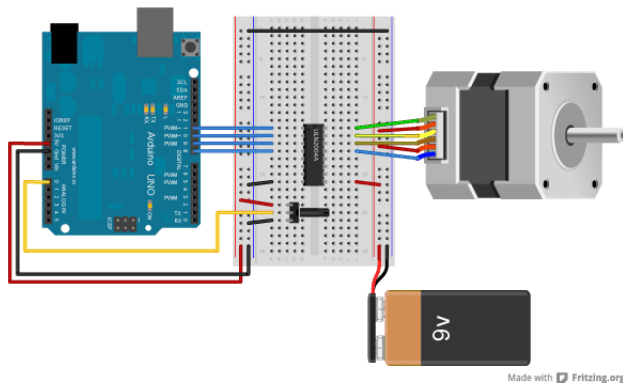
# Common Driver Control

The most popular drivers (DRV8825, A4988, and MP6500) all use a common set of controls for their input. Each pulse signal to the **STEP** input corresponds to one microstep of the stepper motor in the direction selected by the **DIR** pin. The MP6500 includes an additional pair of input pins referred to as I1 and I2, which set the current limit. They aren't entirely necessary, and can remain disconnected (High impedance or "Z").

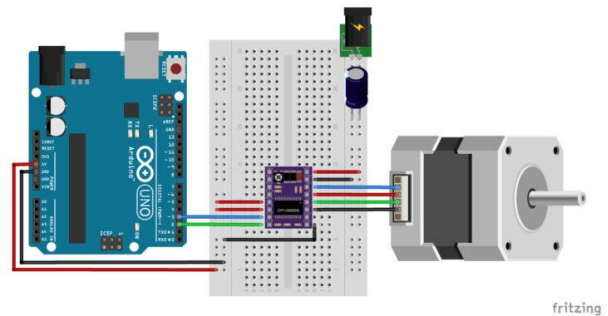
Driver	Pins	Images										
<a href="#">DRV8825</a>	<table> <tr> <th>Input</th><th>Output</th></tr> <tr> <td>Step</td><td>B2</td></tr> <tr> <td>Dir</td><td>B1</td></tr> <tr> <td></td><td>1A</td></tr> <tr> <td></td><td>1B</td></tr> </table>	Input	Output	Step	B2	Dir	B1		1A		1B	
Input	Output											
Step	B2											
Dir	B1											
	1A											
	1B											
<a href="#">A4988</a>	<table> <tr> <th>Input</th><th>Output</th></tr> <tr> <td>Step</td><td>2B</td></tr> <tr> <td>Dir</td><td>2A</td></tr> <tr> <td></td><td>1A</td></tr> <tr> <td></td><td>1B</td></tr> </table>	Input	Output	Step	2B	Dir	2A		1A		1B	
Input	Output											
Step	2B											
Dir	2A											
	1A											
	1B											
<a href="#">MP6500</a>	<table> <tr> <th>Input</th><th>Output</th></tr> <tr> <td>Step</td><td>B2</td></tr> <tr> <td>Dir</td><td>B1</td></tr> <tr> <td>I1 (current limiting)</td><td>A1</td></tr> <tr> <td>I2 (current limiting)</td><td>A2</td></tr> </table>	Input	Output	Step	B2	Dir	B1	I1 (current limiting)	A1	I2 (current limiting)	A2	
Input	Output											
Step	B2											
Dir	B1											
I1 (current limiting)	A1											
I2 (current limiting)	A2											

# Firmware

Arduino hosts several stepper motor libraries through its Library Manager. The two most popular are the `<Stepper.h>` and `<AccelStepper.h>` libraries. These libraries function fundamentally the same except the AccelStepper library host several improvements, such as acceleration and native support for controllers like the DRV8825 and A4988. Shown below, the SN754410 example on the left demonstrates control of a *motor driver* while on the right the DRV8825 example demonstrates control a *motor controller*.



SN754410



DRV8825

## DRV8825

### Public Types

```
enum MotorInterfaceType {  
    FUNCTION = 0, DRIVER = 1, FULL2WIRE = 2, FULL3WIRE = 3,  
    FULL4WIRE = 4, HALF3WIRE = 6, HALF4WIRE = 8  
}  
Symbolic names for number of pins. Use this in the pins argument the AccelStepper constructor to provide a symbolic name for the number of pins to use. More...
```

```
0 // Include the AccelStepper library:  
1 #include <AccelStepper.h>  
2  
3 // Define stepper motor connections and motor interface type  
4 #define dirPin 2 // pin used to set direction  
5 #define stepPin 3 // pin used to set step size  
6  
7 // Define stepper  
8 AccelStepper stepper(AccelStepper::DRIVER, stepPin, dirPin);  
9
```

## SN754410

```
0 // Include the Stepper library:  
1 #include <Stepper.h>  
2  
3 // steps per revolution for the specific motor  
4 const int stepsPerRevolution = 200;  
5  
6 // initialize the stepper library on pins 8 through 11  
7 Stepper myStepper(stepsPerRevolution, 8, 9, 10, 11);
```

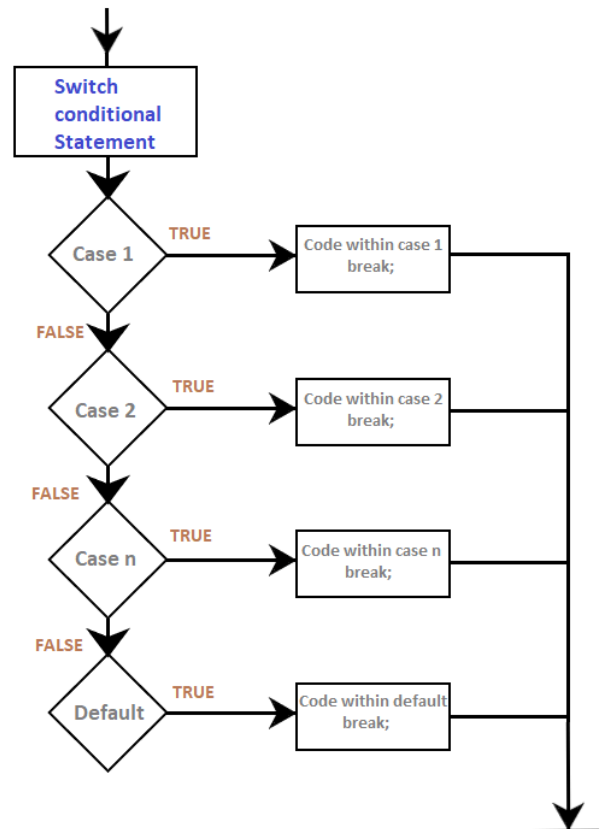
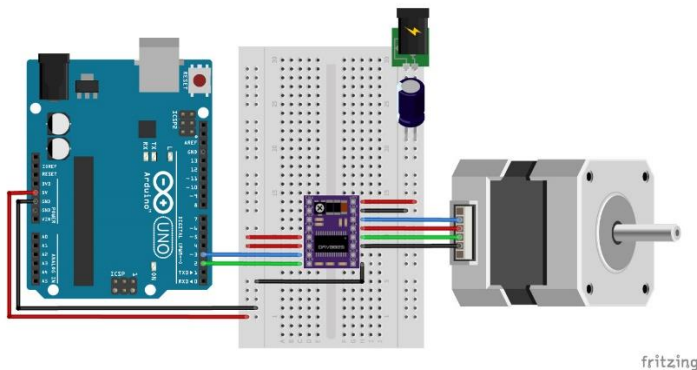
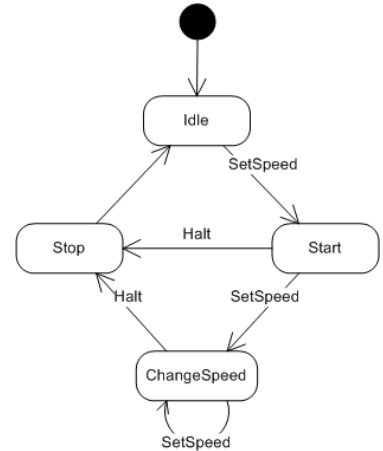


# State Machines

Arduino's, like all microprocessors, operate on the concept of a **finite-state machine**. A finite-state machine is a "machine" or process that has a finite number of states and rules dictating when the machine moves from one state to the other. This machine runs in a loop and when it reaches the last state that it can, it goes back up to the top and starts over again.

In the example of controlling a motor, the state machine is quite simple. The basic principle is an Arduino enters its main loop in an idle state. Nothing is happening. So, it moves onto the next code block of instructions. Perhaps we intend to start the motor. Well, first we set the speed, then run the motor. Great, let's go back up to the top of our loop. We are no longer in idle and the motor is moving. Should we stop the motor? Should we set a new speed for the motor? These are decision branches that we can traverse inside our loop and once that instruction set is complete, we just go back up to the top of our loop until a new process needs to start.

There are many ways to write state machines for Arduino. The best approach, however, is to build your code blocks around a switch case. Switch statements function somewhat similarly to the *if* statement. In fact, they are so similar that some languages, such as Python, do not even include a switch case statement! The premise of the switch is that there is some defining state that indicates where the switch case should go. Much like a train station. The train is your Arduino, and it wants to go somewhere. Each case in a switch is just a new set of tracks the train can travel.



The state machine is indexed by a global state enum {SURFACE, UNDERWATER, ASCEND, DESCEND}. Once each function is complete, we return back to the top and reenter the switch case. Each time we descend, we need to start the motor in a certain direction until a given set point. Every time we need to ascend, we need to start up the motor again, but in the reverse direction until a given set point. During the periods where we aren't using the motor, we are taking measurements.

