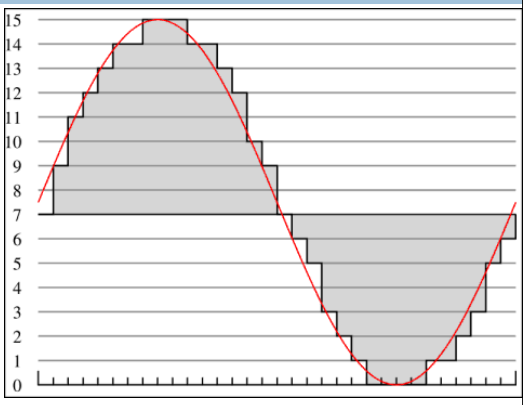
DC MOTORS 101

Servos, DC motors, Stepper motors, motor drivers

Agenda

- Start with PWM (Pulse Width Modulation)
  - "analog" output feature of Arduinos – equivalent to varying output voltage
  - Use for dimming LEDs or speed control of DC motors
- DC motors
  - Basic operation
  - Driving with transistors
  - Bi-directional drive with H-Bridge
- Stepper motors
  - Unipolar and Bipolar versions
  - Stepper driving circuits
  - Example of Arduino-based stepper object library

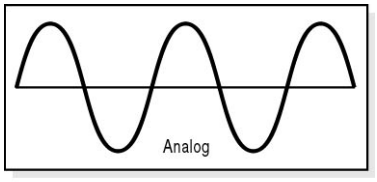
Analog vs. Digital *Input*

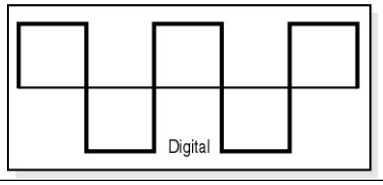


- digitalRead(pin);
  - Returns true/false, 1/0, off/on
  - Great for switches
- analogRead(pin);
  - Returns a range of values
  - Actual voltage on the input should be between 0-5v
  - Converted to 0-1023 discrete steps using ADC
  - ADC is why the analog pins are special…

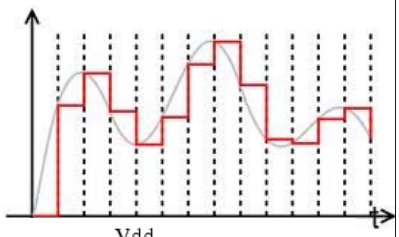
Analog vs. Digital *Output*

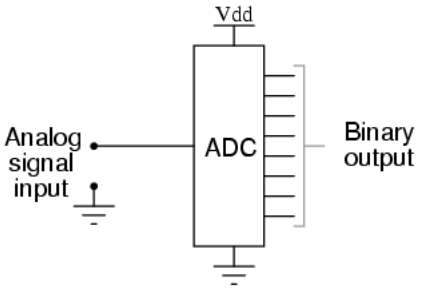
- Digital Out = on/off, up/down, left/right, black/white, etc
- Analog Out = how hot, how fast, how bright, how loud, how grey? etc.
- As with digital output, we have current considerations, this time, how can we generate enough energy to control analog devices?
- Can we generate an analog voltage between 0-5v on the output?

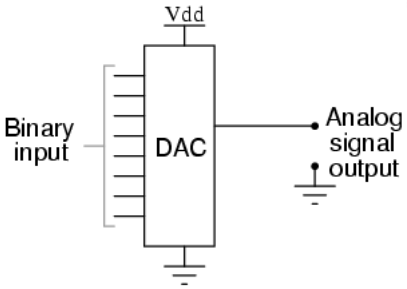




ADC and DAC







Arduino version is a 10-bit ADC

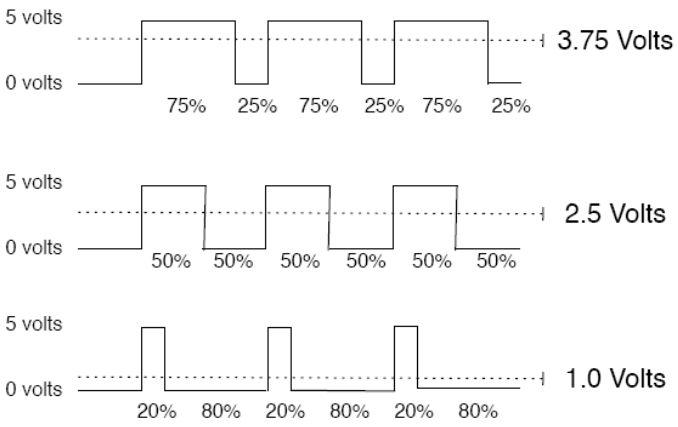
Arduino has no built-in DAC…

PWM

Pulse Width Modulation

Output voltage is averaged from on *vs.* off time

output_voltage = (on_time / off_time) * max_voltage



Arduino PWM

- The Arduino has 6 PWM pins (3, 5, 6, 9, 10, 11) that can receive an analogWrite() command.
  - analogWrite(pin, pulsewidth);
- 0 = 0 volts, 255 = 5V
  - a number in between will provide a specific PWM signal.
  - 128 will be seen as 2.5v, for example

Knob Fade

```
int knobPin = A0;                    // the analog input pin from the potentiometer
int ledPin = 9;                      // pin for LED (a PWM pin)
int val;                             // Variable to hold light sensor value

void setup () {
    pinMode(ledPin, OUTPUT);         // declare ledPin as output
   pinMode(knobPin, INPUT);          // knobPin is an (analog) input
}

void loop () {
    val = analogRead(knobPin);        //read the value from the pot
    val = map(val, 0, 1023, 100, 255); // map to reasonable values
    val = constrain(val, 0, 255);     // Make sure it doesn't go out of range
    analogWrite(ledPin, val);         // write it to the LED using PWM
}
```

Arduino PWM Alternative

- It is also possible to create a pwm signal by simply 'pulsing' a digital out pin very quickly:

```
digitalWrite(pin, HIGH);
delayMicroseconds(pulsewidth); // note: "Micro…!"
digitalWrite(pin, LOW);
delayMicroseconds(pulsewidth);
```

- Or, use a Motor Shield…

Dimming Lights

- Incandescent blubs
  - Vary the voltage, or use PWM
- LEDs
  - Use PWM
- Fluorescents (compact or regular)
  - In general you can't dim them…
  - There are a few "dimmable" compact fluorescents

Dimming Lights

- Fixed resistance and varying voltage = varying current
  - V = IR, V/R = I
- Pay attention to the lamp ratings
  - Remember P = IV
- For high current devices, you will need to electrically isolate the Arduino from the high current circuit.

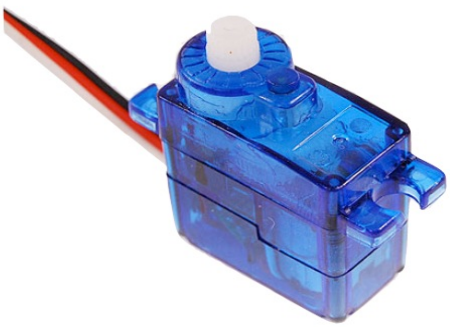


Activity: Arduino "flickering candle"

- Connect an LED to the Arduino on a PWM output pin
  - Always remember current-limiting resistor…
- Write a program to make that LED "flicker" like a candle
  - Chose between random brightness values
  - Choose random times to change the value

Servos Also Use PWM

- DC motor
- High-torque gearing
- Potentiometer to read position
- Feedback circuitry to read pot and control motor
- All built in, you just feed it a PWM signal



- From Tod Kurt, totbot.com

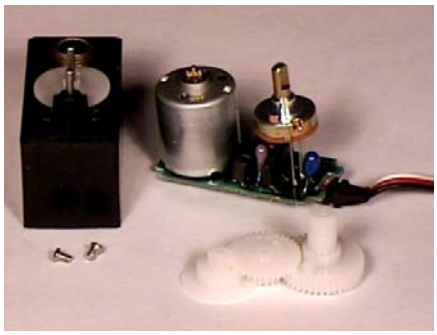
Servos Also Use PWM

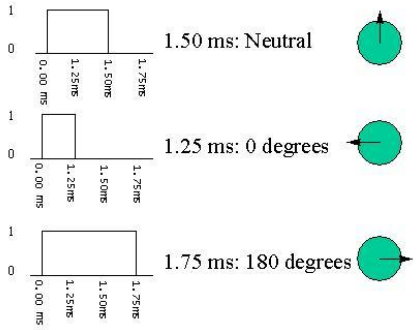
Servomotors

- Can be positioned from 0-180° (usually)
- Internal feedback circuitry & gearing takes care of the hard stuff
- Easy three-wire PWM 5V interface

- From Tod Kurt, totbot.com

Servo Innards





Servo Specs

- Come in all sizes
  - from super-tiny
  - to drive-your-car
- But all have the same 3-wire interface
- Servos are spec'd by:

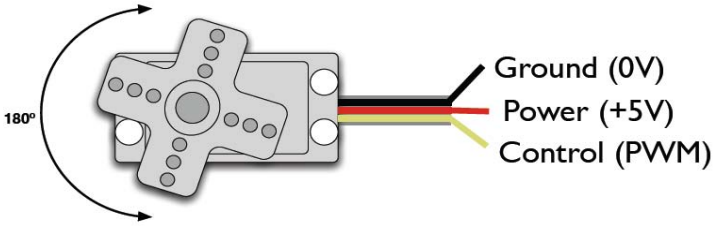
weight: 9g
speed: .12s/60deg @ 6V
torque: 22oz/1.5kg @ 6V
voltage: 4.6~6V
size: 21x11x28 mm

9g

157g

- From Tod Kurt, totbot.com

Servo Control

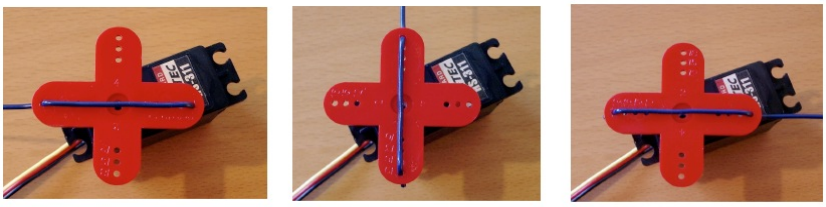


- PWM freq is 50 Hz (i.e. every 20 millisecs)
- Pulse width ranges from 1 to 2 millisecs
  - 1 millisec = full anti-clockwise position
  - 2 millisec = full clockwise position

- From Tod Kurt, totbot.com

Servo Control

0 degrees | 90 degrees | 180 degrees



1000 microsecs | 1500 microsecs | 2000 microsecs

In practice, pulse range can range from 500 to 2500 microsecs

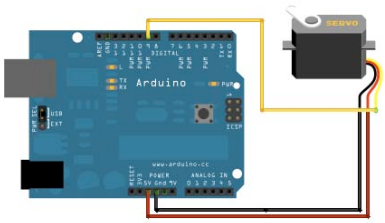
- From Tod Kurt, totbot.com

Programming Servo Motors

- The first thing to do when programming servos is work out the pulse range of the specific servo you are using.
  - With the arduino, this can be between 0.5 and 2.5 ms.
  - Servos also need 'refresh' time between pulses -- usually 20ms
- The frequency of the Arduino's built-in pwm pins is too high for servos, so we have to use the pseudo-pwm method instead:

```
digitalWrite(servoPin, HIGH);
delayMicroseconds(pulse);
digitalWrite(servoPin, LOW);
delayMicroseconds(refreshTime);
```

- Or alternatively, use the Arduino servo lib



- Black = gnd
- Red = +5v
- Yellow = signal

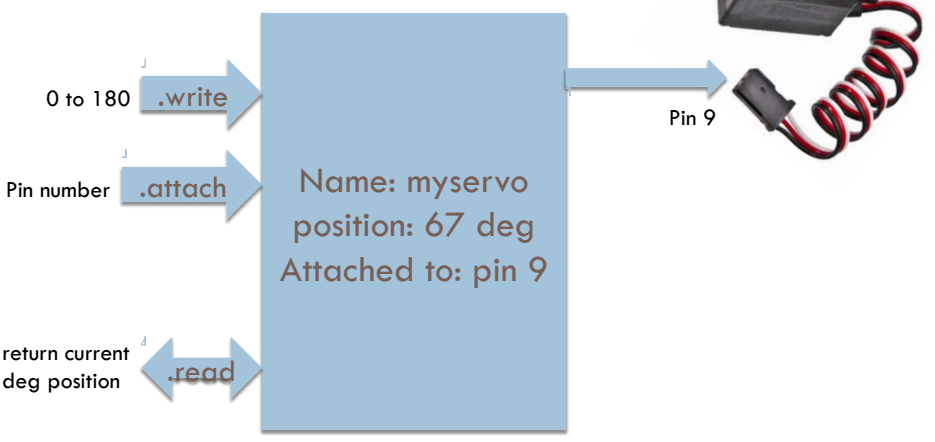
Servo Example Program

```
#include <Servo.h> // include the built-in servo library
Servo myservo;      // create a servo object to control the servo (one per servo)
int pos = 0;        // variable to store the servo position

void setup() {
    myservo.attach(9);          // attach servo control to pin 9
}
void loop() {
    for (pos = 0; pos < 180; pos++) {    // go from 0 to 180 degrees
            myservo.write(pos);           // move the servo
            delay(20);l                   // give it time to get there
    }
    for (pos = 180; pos>=1; pos--) {   // wave backwards
            myservo.write(pos);
            delay(20);
    }
}
```

Servo Functions

- Servo is a class
  - Servo myservo; // creates an instance of that class
- myservo.attach(pin);
  - attach to an output pin (doesn't need to be PWM pin!)
  - Servo library can control up to 12 servos on our boards
  - but a side effect is that it disables the PWM on pins 9 and 10
- myservo.write(pos);
  - moves servo – pos ranges from 0-180
- myservo.read();
  - returns the current position of the servo (0-180)

Servo Object (class instance)



Moving on…

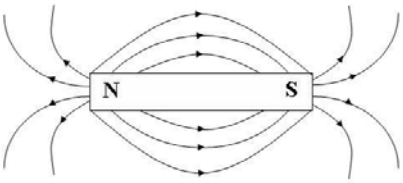
- Write a program to control the position of the servo from a pot, or from a photocell
  - remember analogRead(); values are from 0-1023
  - measure the range of values coming out of the photocell first?
  - use Serial.print(val); for example
  - use map(val, in1, in2, 0, 180); to map in1-in2 values to 0-180
  - Can also use constrain(val, 0, 180);

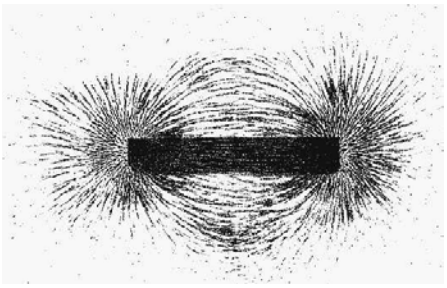
Side Note - Power

- Servos can consume a bit of power
  - We need to make sure that we don't draw so much power out of the Arduino that it fizzles
  - If you drive more than a couple servos, you probably should put the servo power pins on a separate power supply from the Arduino
  - Use a wall-wart 5v DC supply, for example

Moving on: Electromagnetism

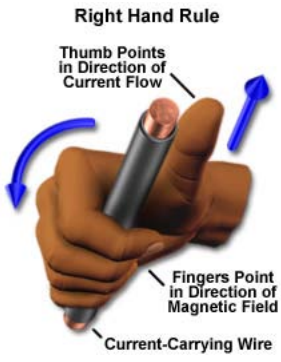
- Permanent magnets have magnetic fields that flow from North to South poles of the magnet

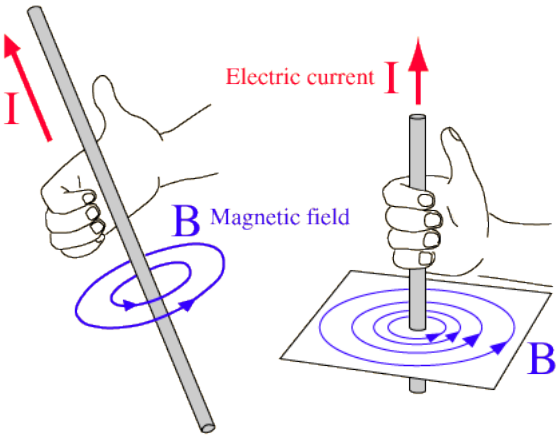




Electromagnets

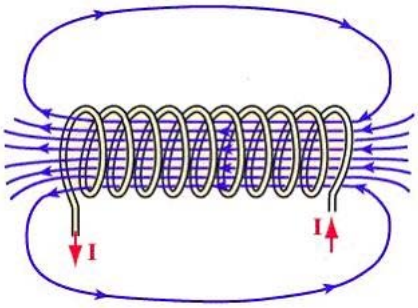
- Current flowing through a conductor creates a magnetic field
  - Right hand rule: Point your thumb in direction of current, and your fingers curl in the direction of the magnetic field

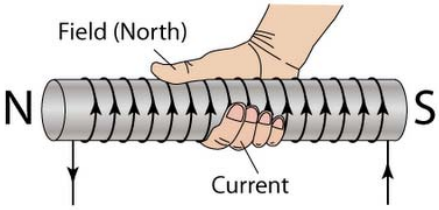




Electromagnets

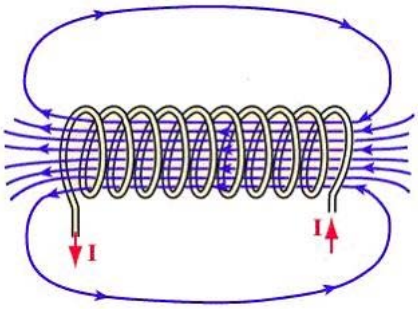
- Current flowing through a coiled conductor creates a magnetic field
  - Right hand rule: Point your thumb in direction of current, and your fingers curl in the direction of the magnetic field
  - Alternately, curl your fingers in the direction of the current, your thumb points to North magnetic pole

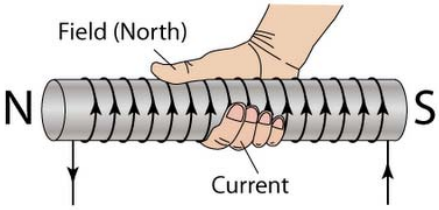




Electromagnets

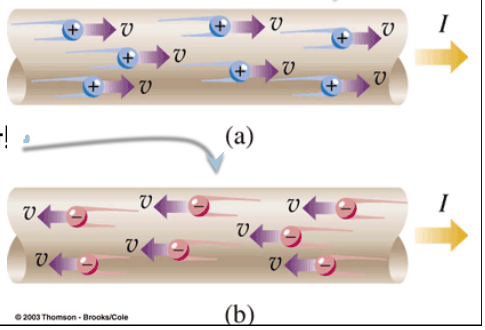
- Current flowing through a coiled conductor creates a magnetic field
  - Important implication: When the current it reversed, so is the magnetic field!





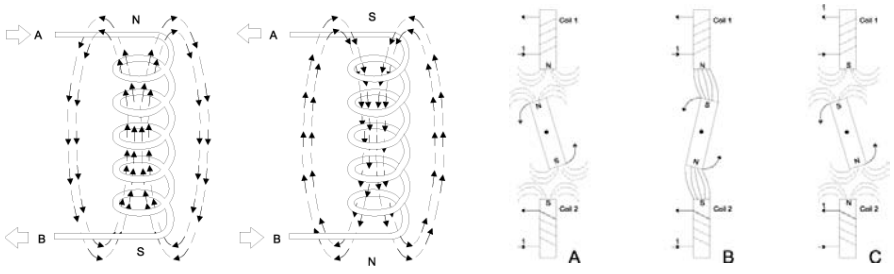
Aside: Current dir vs. Carrier dir

- Current (amps) flows from positive to negative
  - "positive current"
- BUT – the charge carriers are (usually) electrons
  - Electrons have negative charge
  - So – the electrons move from negative to positive
- Charge carriers move in opposite direction from current!
  - Yes, it's confusing…
  - We'll tackle this again talking about transistors!



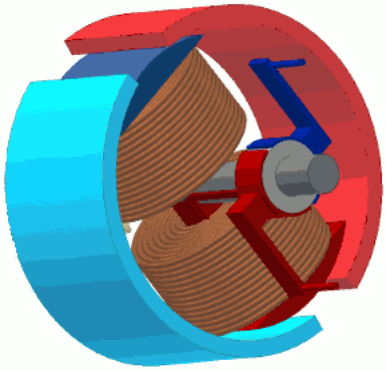
Electromagnets and Motors

- Motors use the reversing feature of an electromagnet, and magnetic attraction
  - Permanent magnets on one side, electromagnets that can switch polarity on the other



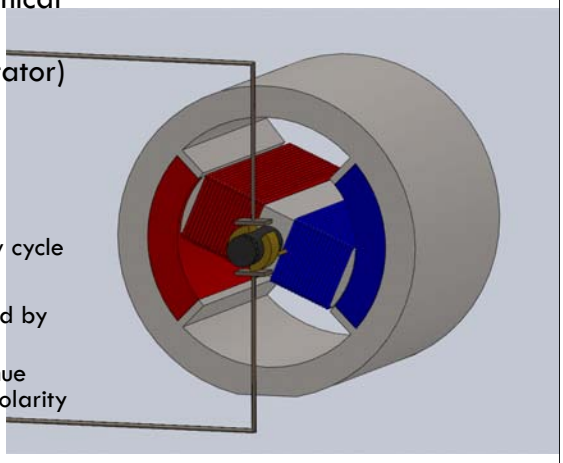
Basic DC Motor Behavior

- Electrical energy -> mechanical motion
- Motors have fixed parts (stator) and moving parts (rotor)
- A DC motor consists of:
  - Commutator
    - Rotary switch
    - Reverses current twice every cycle
  - Electromagnetic coils
    - Opposing polarities switched by commutator
    - Inertia causes them to continue rotating at the moment of polarity switching
  - Fixed Magnets
    - Opposing polarities



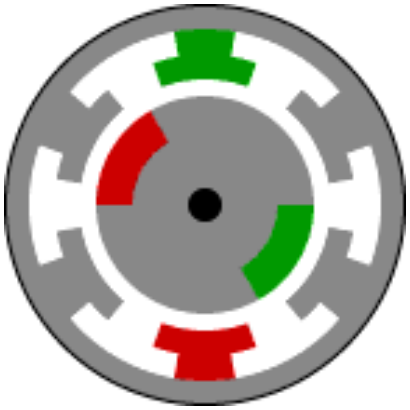
Basic DC Motor Behavior

- Electrical energy -> mechanical motion
- Motors have fixed parts (stator) and moving parts (rotor)
- A DC motor consists of:
  - Commutator
    - Rotary switch
    - Reverses current twice every cycle
  - Electromagnetic coils
    - Opposing polarities switched by commutator
    - Inertia causes them to continue rotating at the moment of polarity switching
  - Fixed Magnets
    - Opposing polarities



Basic DC Motor Behavior

- Electrical energy -> mechanical motion
- Motors have fixed parts (stator) and moving parts (rotor)
- A DC motor consists of:
  - Commutator
    - Rotary switch
    - Reverses current twice every cycle
  - Electromagnetic coils
    - Opposing polarities switched by commutator
    - Inertia causes them to continue rotating at the moment of polarity switching
  - Fixed Magnets
    - Opposing polarities



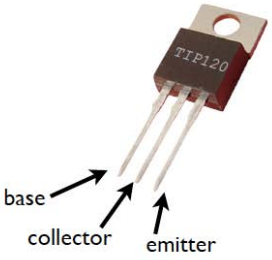
Motor Speed Control

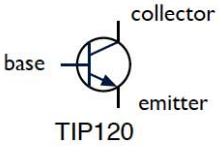
- For a simple DC motor, PWM works great
  - Motor "integrates" the pulsing waveform for an effective lowering of the drive voltage
  - Motors have inertia, and also minimum operating voltages. Therefore, often it will seem that the motor will only operate at a high duty cycle initially.
  - Connect motors up using external power supplies
    - Make sure to connect grounds together…

Transistors as Switches

Act like switches

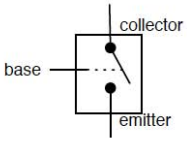
electricity flicks the switch instead of your finger





TIP120

schematic symbol

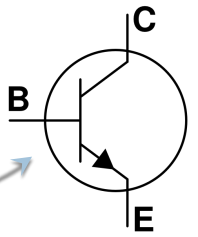


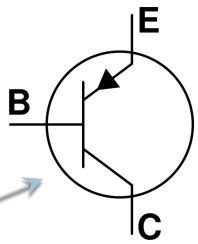
how it kind of works

- From Tod Kurt, totbot.com

Transistors (bipolar)

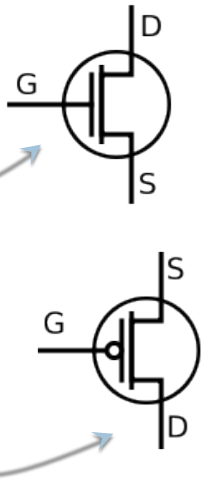
- 3 terminals (emitter, base, collector): different voltages at the input terminal controls the conductivity between the other two.
  - Faster and cheaper than relays
  - DC only
- Bipolar (BJT) are common
  - Polarity is PNP or NPN
  - NPN has small input current and positive voltage at Base to control a large Collector-to-Emitter current
  - PNP has small output current and negative voltage at Base to control a large Collector-to-Emitter current



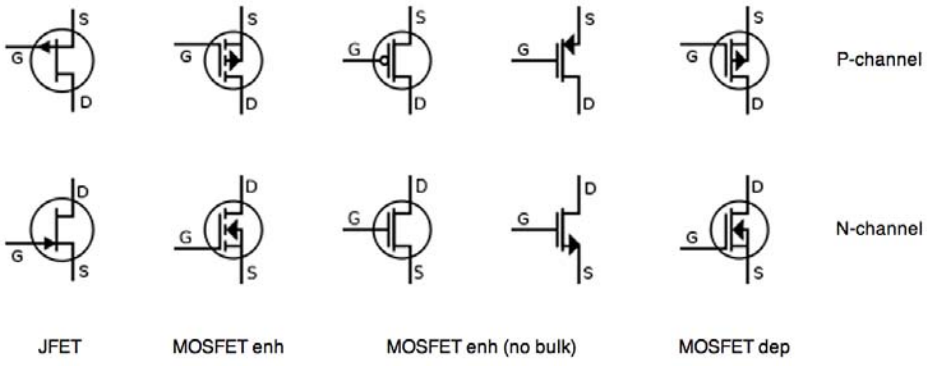


Transistors (mosfet)

- 3 terminals (source, gate, drain): different voltages at the input terminal controls the conductivity between the other two.
  - Faster and cheaper than relays
  - Possible to use with AC
- MOSFETs are also common
  - Polarity is Ntype or Ptype
  - Ntype has positive voltage at gate to control a large Source-Drain current
  - Ptype has negative voltage at Gate to control a large Source-Drain current

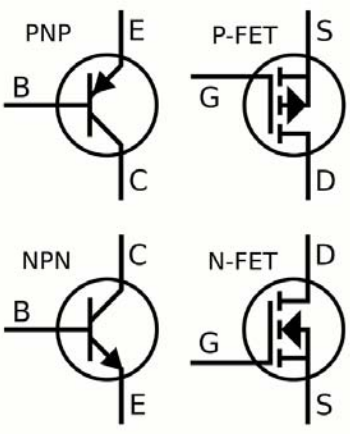


MOSFET symbols



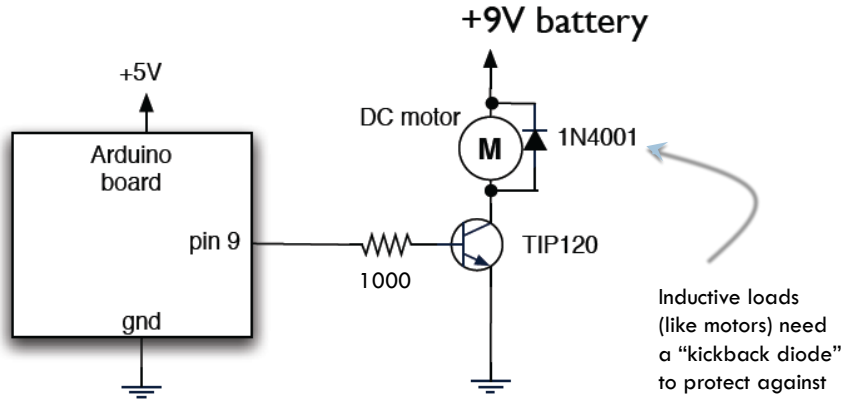
BJT vs. MOSFET

- BJT involves current in/out of the Base
  - Current gain is measure of how much more current flows from C-E than in B
  - Predictable gain response
- MOSFETs have high impedance inputs
  - No current in/out of gate
  - Can handle high power, but complex gain response
  - Can be more fragile – esp. to static charge



Transistors for Motors

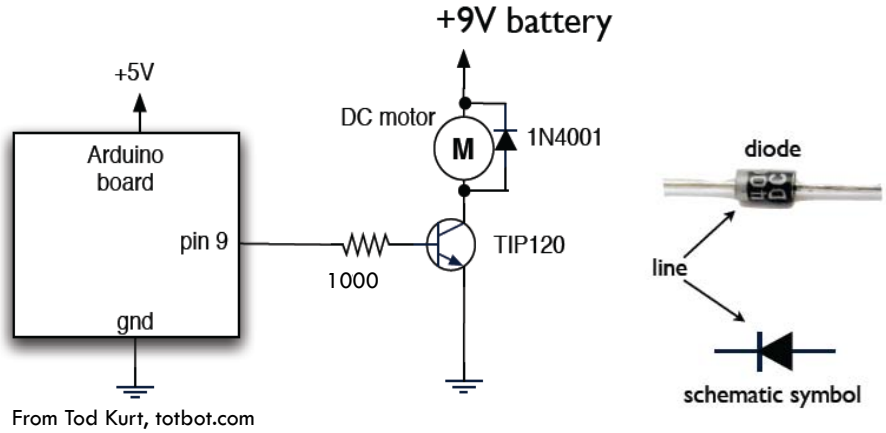
- Use a transistor to switch the motor's voltage source
  - Can be different from Arduino power



- From Tod Kurt, totbot.com

Transistors for Motors

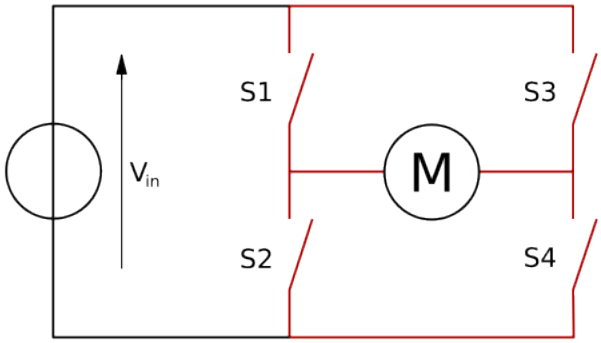
- Use a transistor to switch the motor's voltage source
  - Can be different from Arduino power



- From Tod Kurt, totbot.com

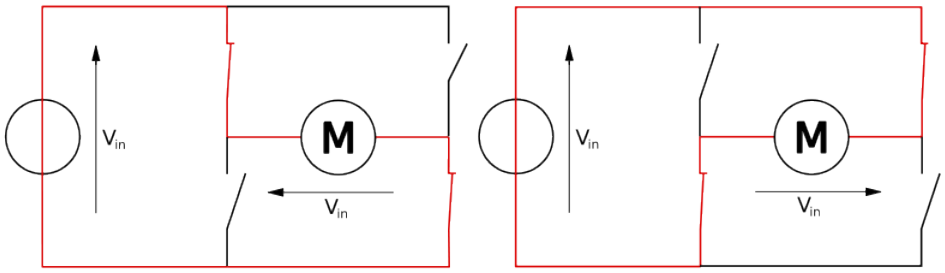
Bidirectional DC Motor

- By reversing the current, you can change motor direction
  - Clever circuit – H-Bridge
  - Can be switches or transistors…



Bidirectional DC Motor

- By reversing the current, you can change motor direction
  - Clever circuit – H-Bridge
  - Can be switches or transistors…



Bidirectional DC Motor

- By reversing the current, you can change motor direction
  - Clever circuit – H-Bridge
  - Can be switches or transistors…

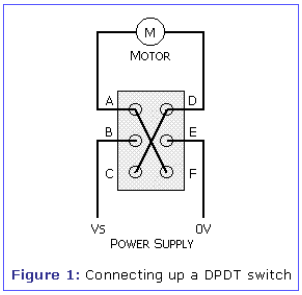
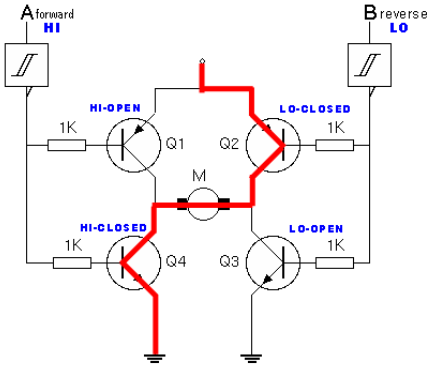
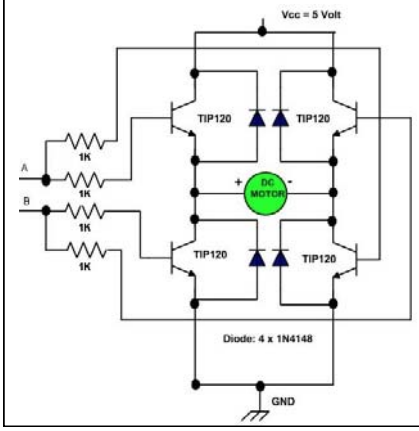


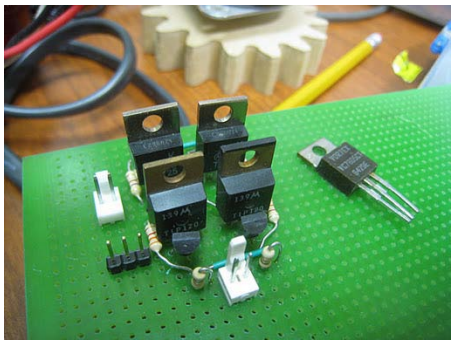
Figure 1: Connecting up a DPDT switch



H-Bridge

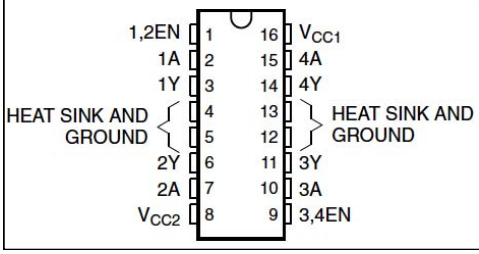
- Could build this from individual transistors (like TIP120)

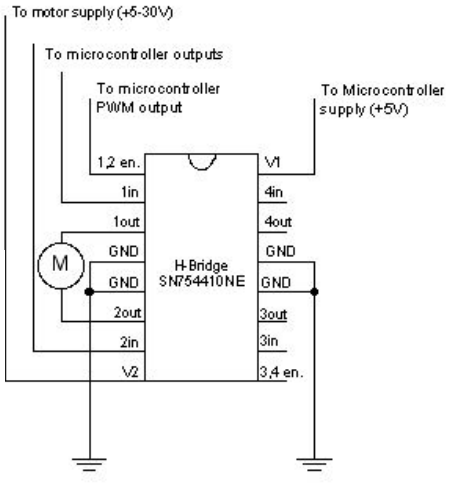




H-Bridge

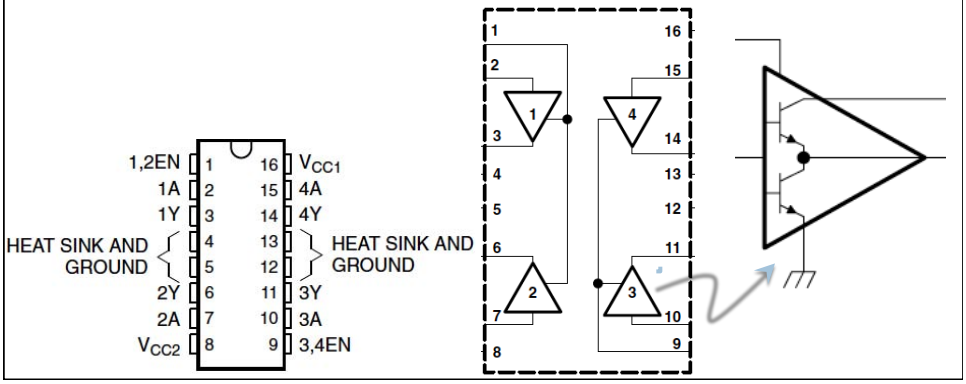
- OR, you could get this in a handy chip form
  - Quad Half H-Bridge
  - L293D or SN774410



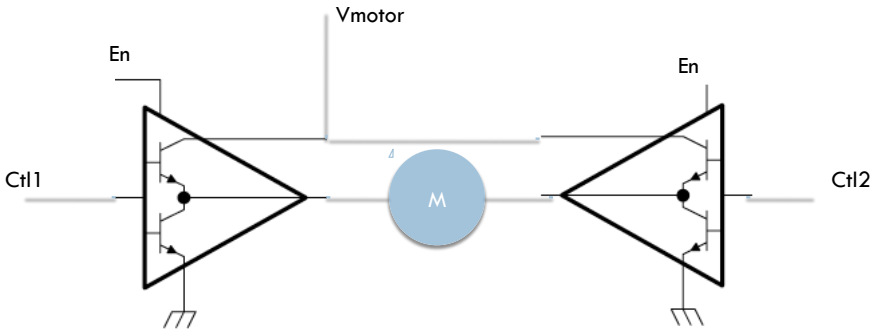


H-Bridge

- OR, you could get this in a handy chip form
  - Quad Half H-Bridge
  - L293D or SN774410



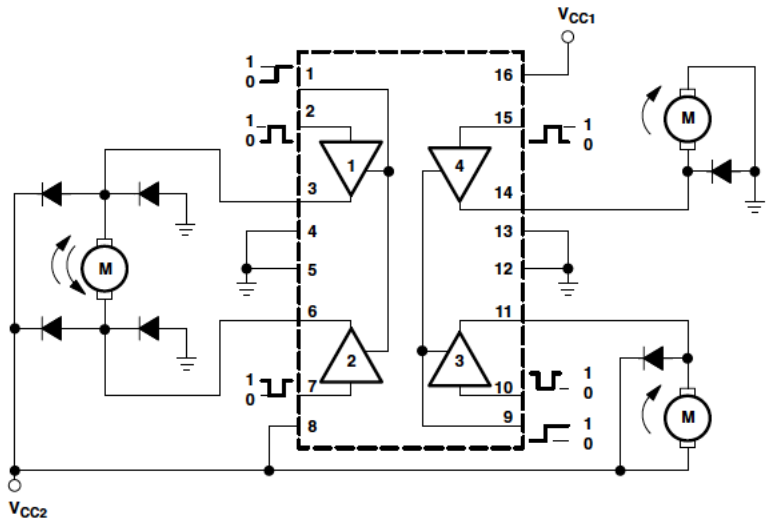
Quad Half H-Bridge



Not shown – built-in inversion at Ctl inputs - one switch is always on, other is off…

Can use PWM on the En signal to modulate speed

Quad Half H-Bridge

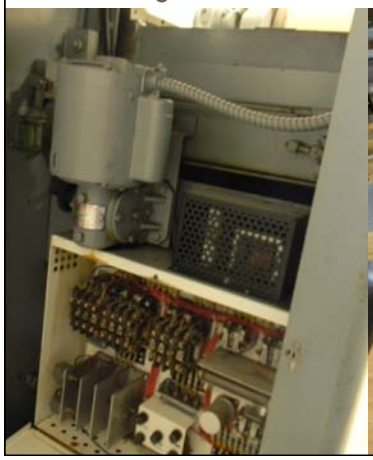


Real Life (big) example

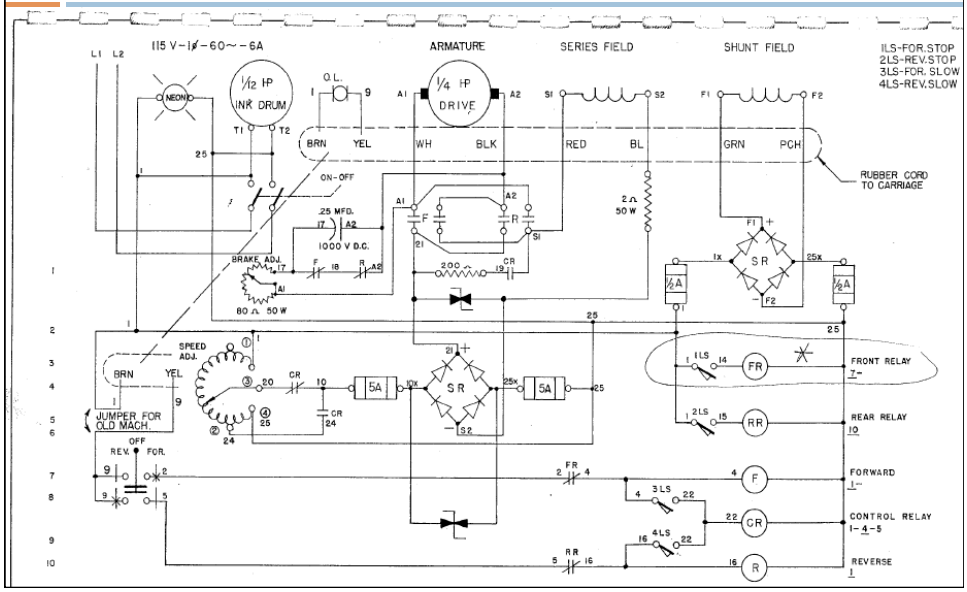
- Saltgrass Printmakers in Salt Lake City
  - www.SalrtgrassPrintmakers.org
- We have a Vandercook proof press with a powered carriage

Vandercook Electronics

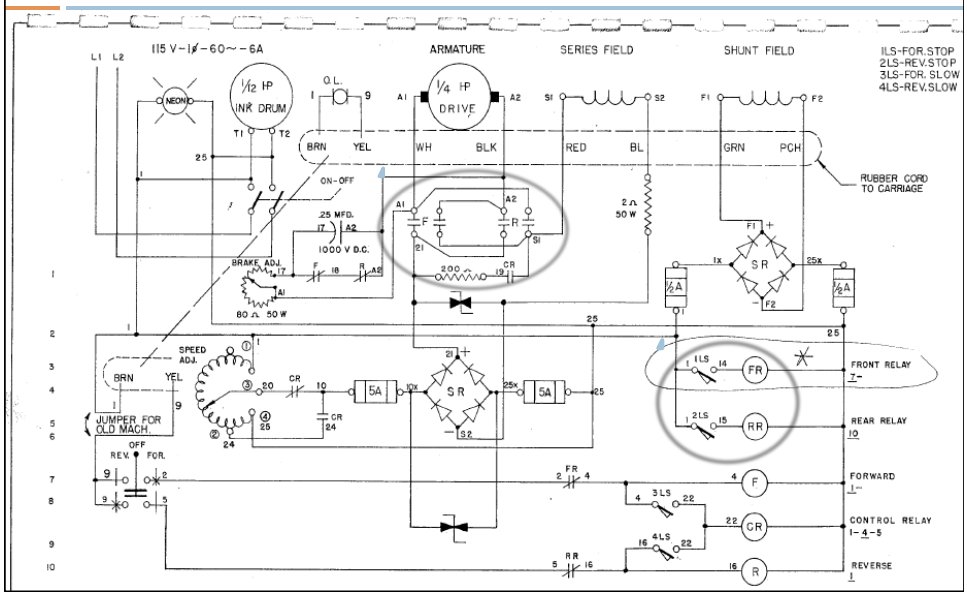
- ¼ HP DC motor moves the press bed back and forth
  - Large mechanical "reversing contactor" is the H-Bridge



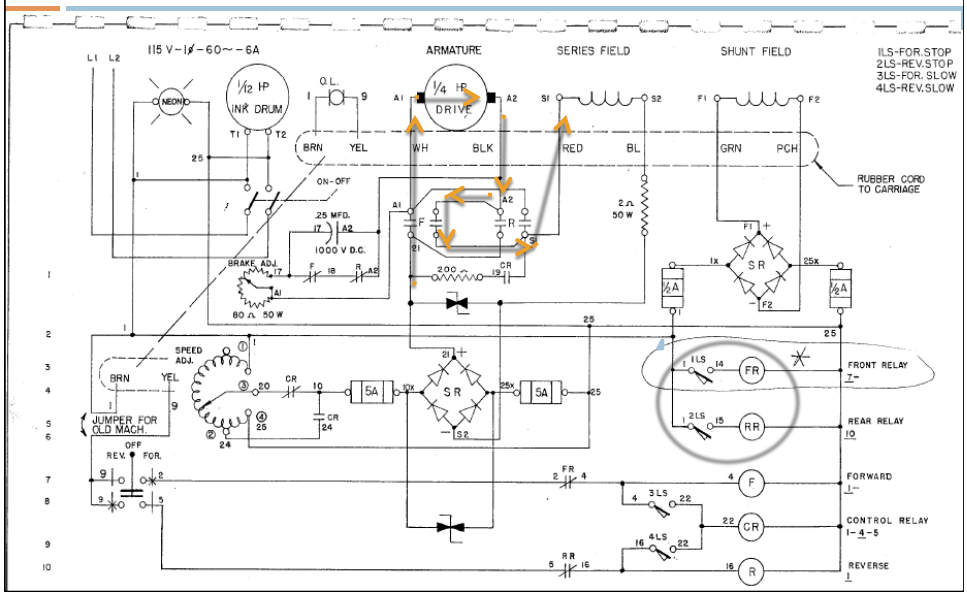
Vandercook Schematic



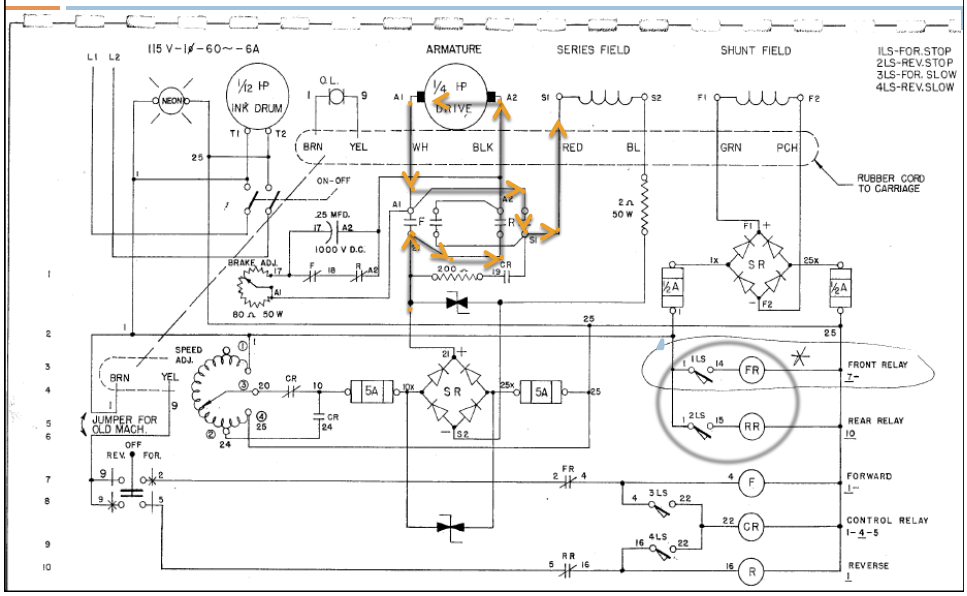
Vandercook Schematic



Vandercook Schematic



Vandercook Schematic

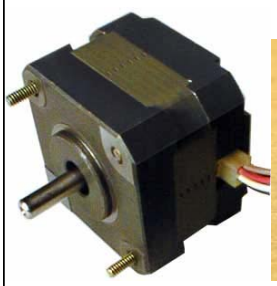
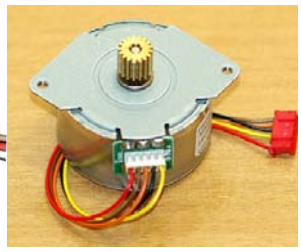


Activity: Bidirectional Motor

- Grab a SN754410 or L293D
  - Wire it up for a bidirectional motor
  - Connect a DC motor – use external power
  - Control directional switching and speed from Arduino
  - Look at DC_motor_hbridge on class page

Stepper Motors

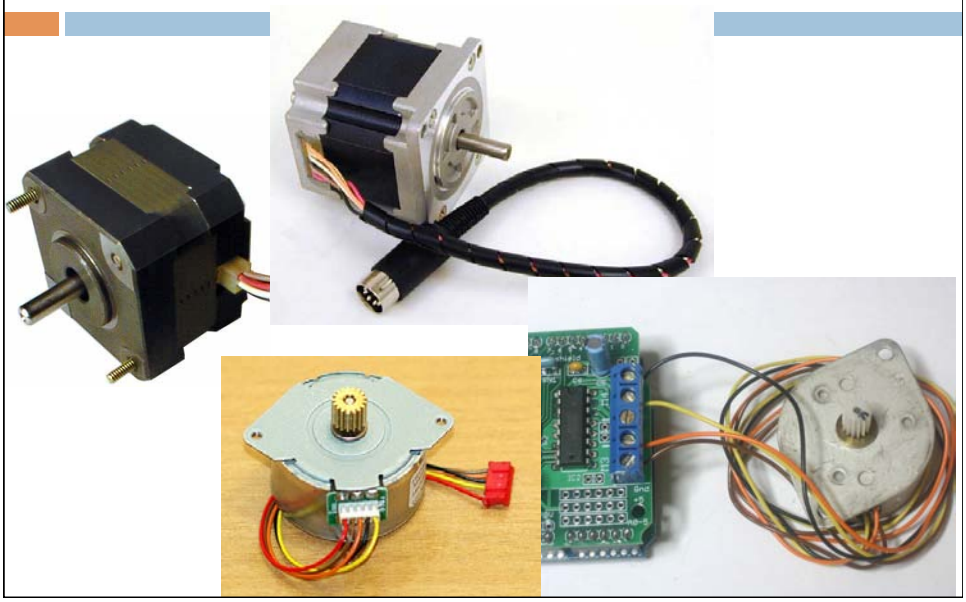
- DC motors with precise control of how far they spin
  - They have a fixed number of "steps" the take to turn one full revolution
  - You can control them one step at a time
  - Makes for very precise and repeatable positioning


Why use steppers?

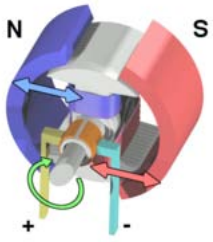
- Very precise
- Much stronger than servos
  - But, they use more current than Arduino can provide
  - So, you need some sort of external power source
- They're a little tricky to drive
  - It's handy to have some sort of code library, or external driver board
- I suggest to use both – a library, and an external board

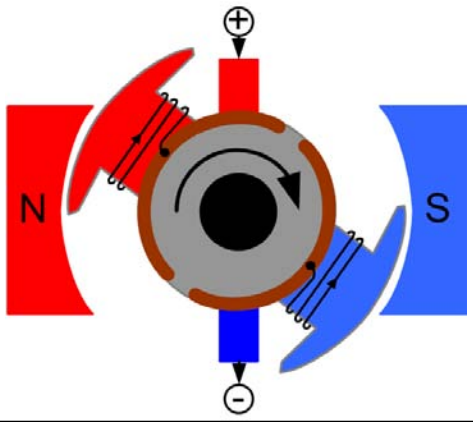
They always have multiple wires



How do they Work?

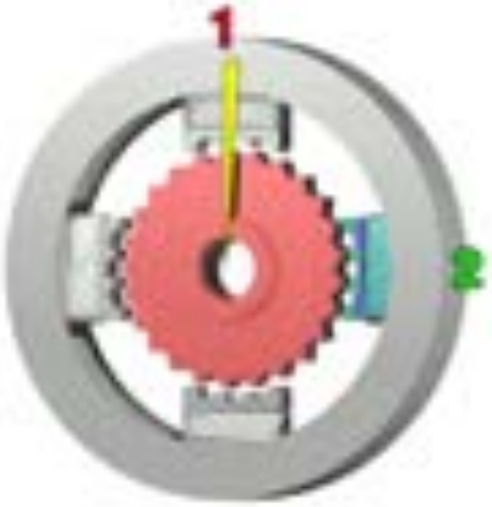
- Like all motors – electro-magnets get energized and push/pull the rotor around.



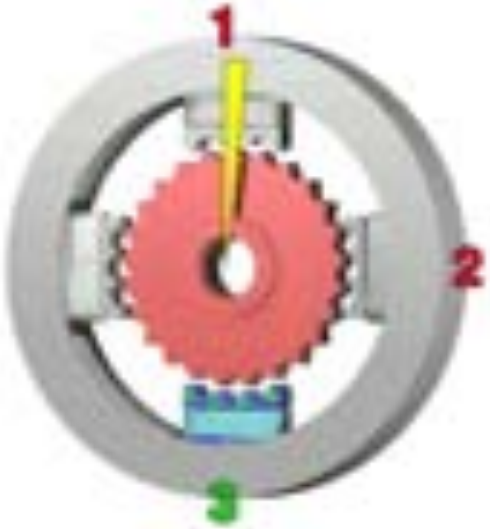


Steppers have precise internals

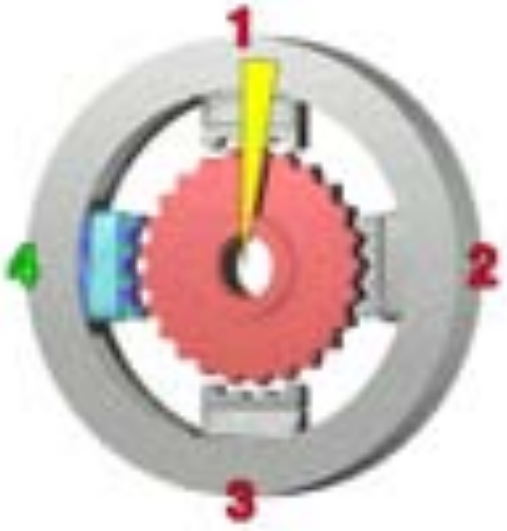
Steppers have precise internals



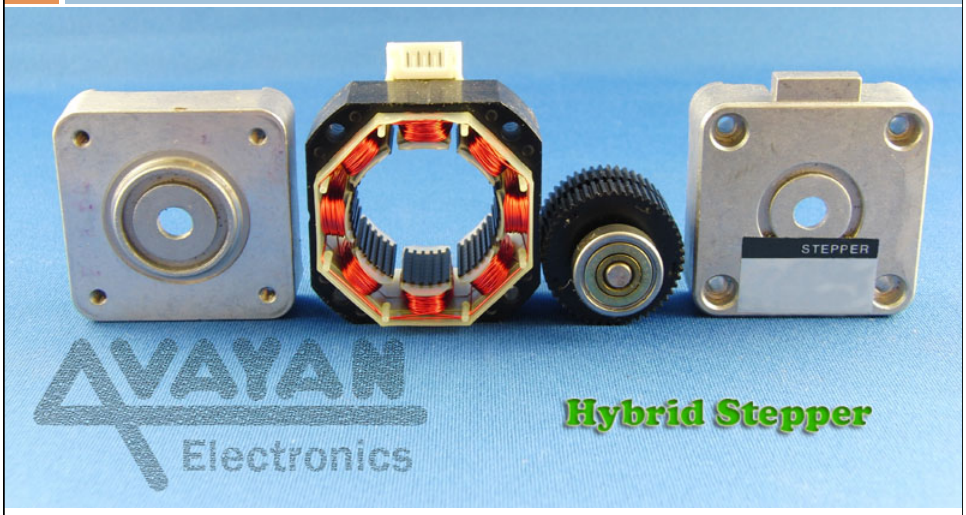
Steppers have precise internals



Steppers have precise internals

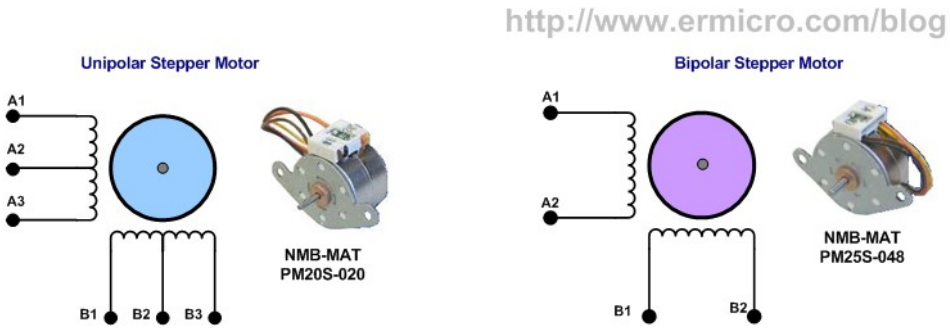


Stepper Internals



Different Flavors of Steppers

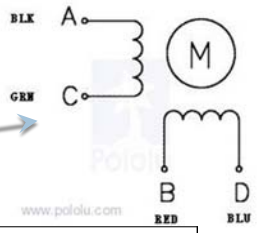
- Unipolar vs. Bipolar

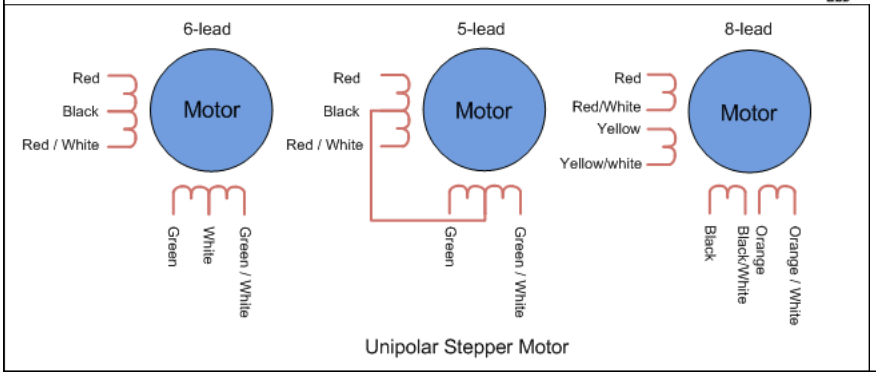


The Unipolar and Bipolar Stepper Motor Windings

Different Flavors of Steppers

- Unipolar vs. Bipolar

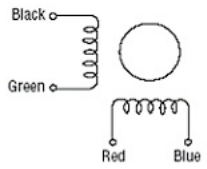
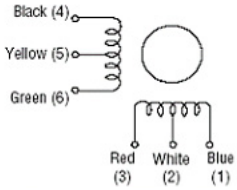
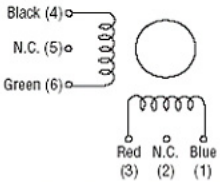
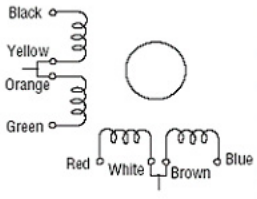
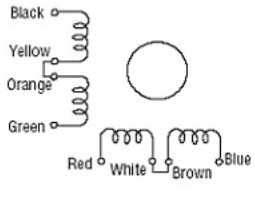
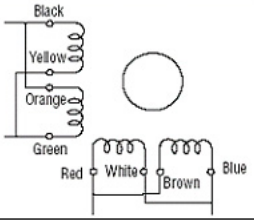




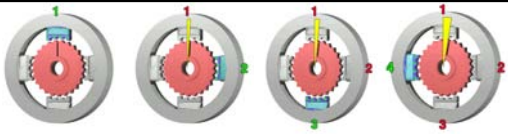
Unipolar Stepper Motor

Unipolar used as Bipolar

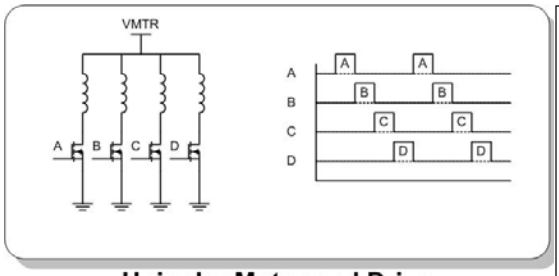
Wire Connection Diagrams

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**
		

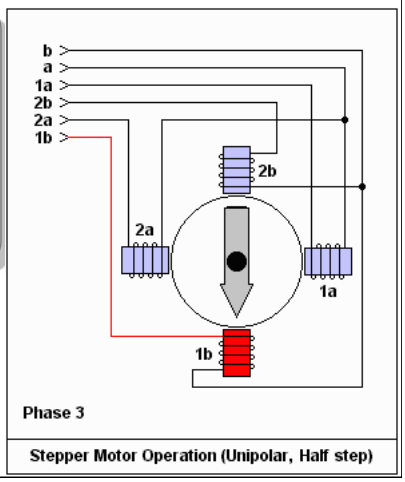
Make it Turn



- Energize the coils in a very specific sequence

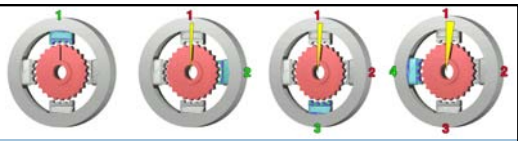


Unipolar Motor and Drive



Stepper Motor Operation (Unipolar, Half step)

Make it Turn



- Energize the coils in a very specific sequence

Index	1a	1b	2a	2b
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	1	0	0	0
6	0	1	0	0
7	0	0	1	0
8	0	0	0	1

Clockwise Rotation

Index	1a	1b	2a	2b
1	1	0	0	1
2	1	1	0	0
3	0	1	1	0
4	0	0	1	1
5	1	0	0	1
6	1	1	0	0
7	0	1	1	0
8	0	0	1	1

Clockwise Rotation

Alternate Full Step Sequence
(Provides more torque)

Index	1a	1b	2a	2b
1	1	0	0	0
2	1	1	0	0
3	0	1	0	0
4	0	1	1	0
5	0	0	1	0
6	0	0	1	1
7	0	0	0	1
8	1	0	0	1
9	1	0	0	0
10	1	1	0	0
11	0	1	0	0
12	0	1	1	0
13	0	0	1	0
14	0	0	1	1
15	0	0	0	1
16	1	0	0	1

Clockwise Rotation

Half Step Sequence

Use a Library

Functions

- Stepper(steps, pin1, pin2)
- Stepper(steps, pin1, pin2, pin3, pin4)
- setSpeed(rpm)
- step(steps)

Example

- Motor Knob

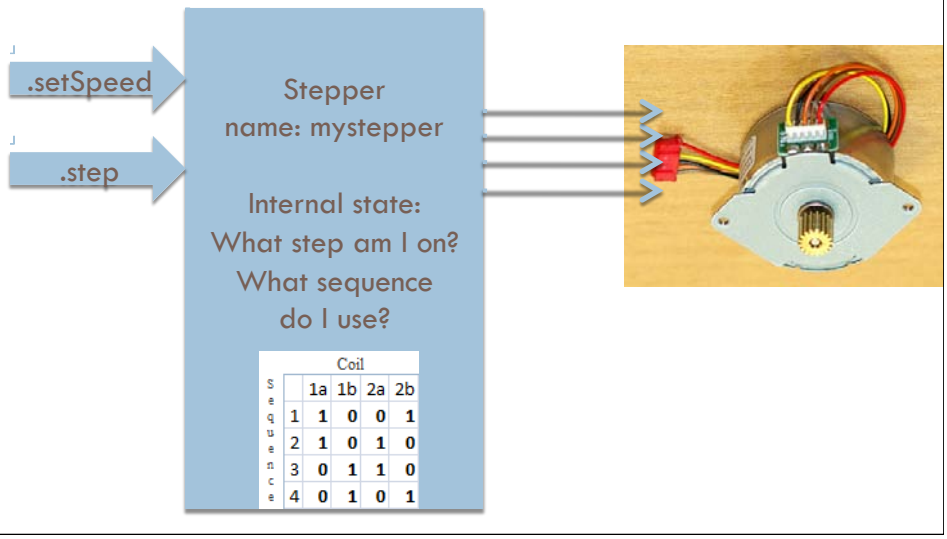
Simple Example

```
/* By Tom Igoe */
#include <Stepper.h>
const int steps = 200;  // change for steps/rev for your motor
Stepper myStepper(steps, 8,9,10,11);   // init and attach your stepper

void setup() {
  myStepper.setSpeed(60);     // set the speed at 60 rpm:
  Serial.begin(9600);         // initialize the serial port:
}

void loop() {
   Serial.println("clockwise");// step one revolution in one direction:
  myStepper.step(steps);
  delay(500);

  Serial.println("counterclockwise");  // step one revolution in other direction:
  myStepper.step(-steps);
  delay(500);
}
```

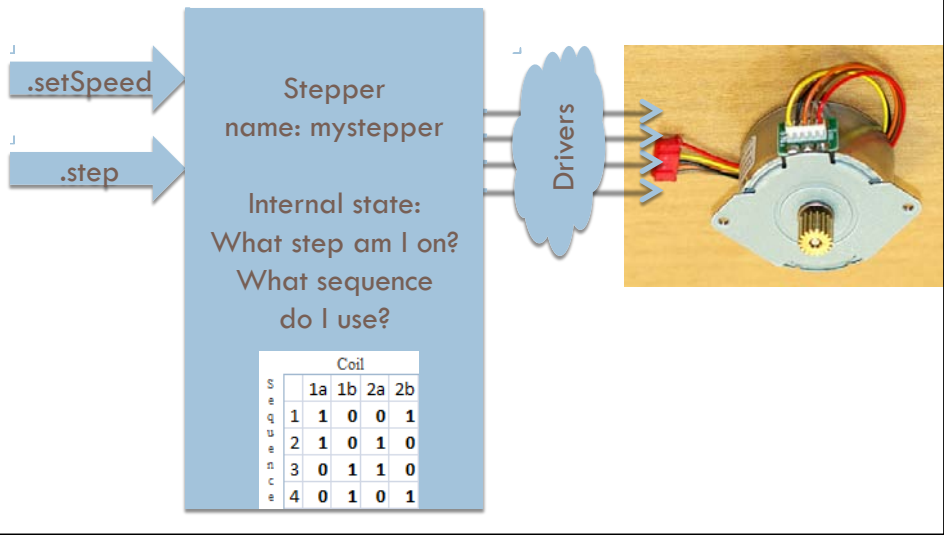
Knob Example

```
#include <Stepper.h>
#define STEPS 200    // Number of steps in one rev
Stepper stepper(STEPS, 8, 9, 10, 11); // Create and attach stepper
int previous = 0; // previous reading from analog in

void setup() {
  stepper.setSpeed(30);  // set the speed of the motor to 30 RPMs
}

void loop()  {
  int val = analogRead(0); // get the sensor value
  // move a number of steps equal to the change in the
  // sensor reading
  stepper.step(val - previous);
  previous = val;  // remember the previous value of the sensor
}
```

Stepper Object

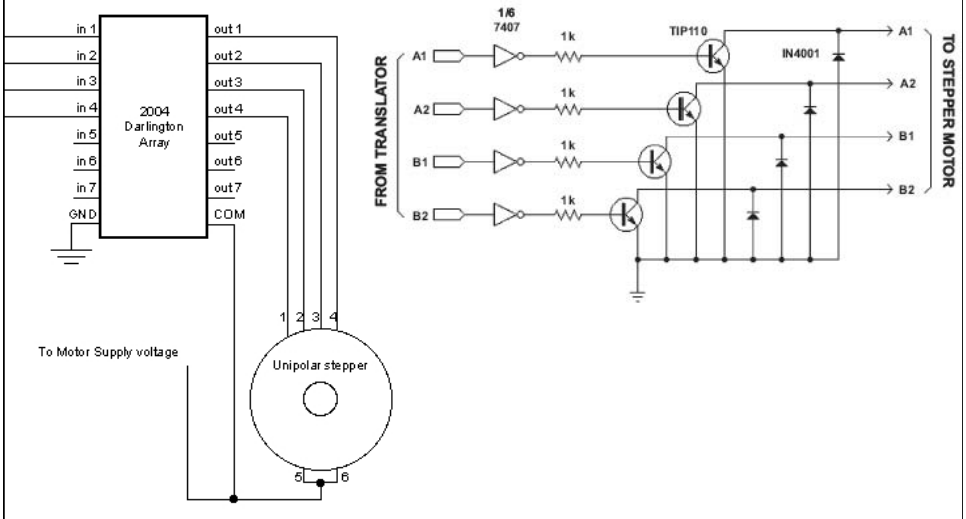


Stepper Object

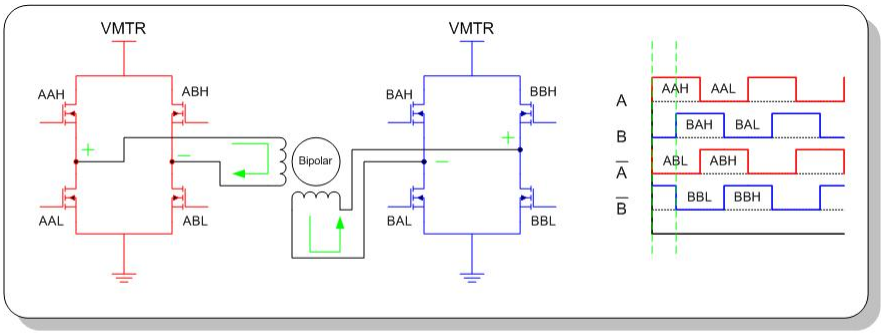


Sequence	Coil 1a	1b	2a	2b
1	1	0	0	1
2	1	0	1	0
3	0	1	1	0
4	0	1	0	1

Driver Circuits… (unipolar)

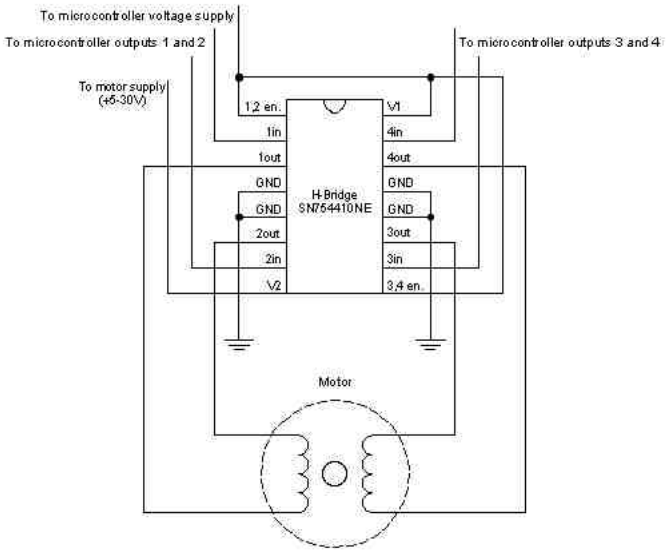


Driver Circuits… (bipolar)



Bipolar Motor and Drive

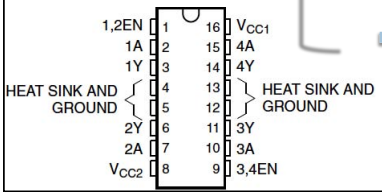
Driver Circuits… (chips)



Driver Circuits

Control inputs from Arduino

You can use the <Stepper.h> library for this



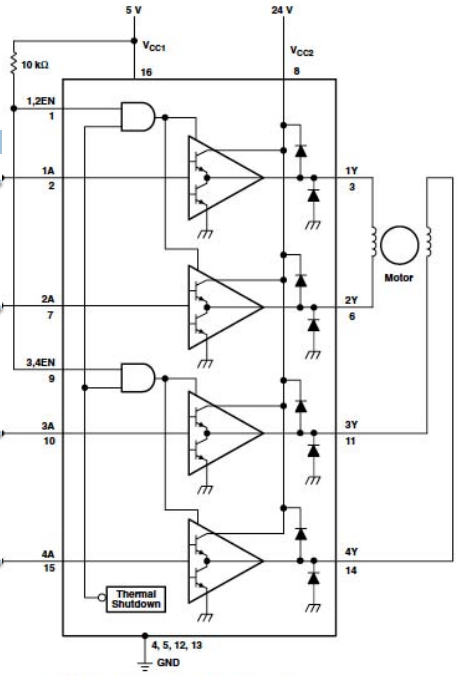
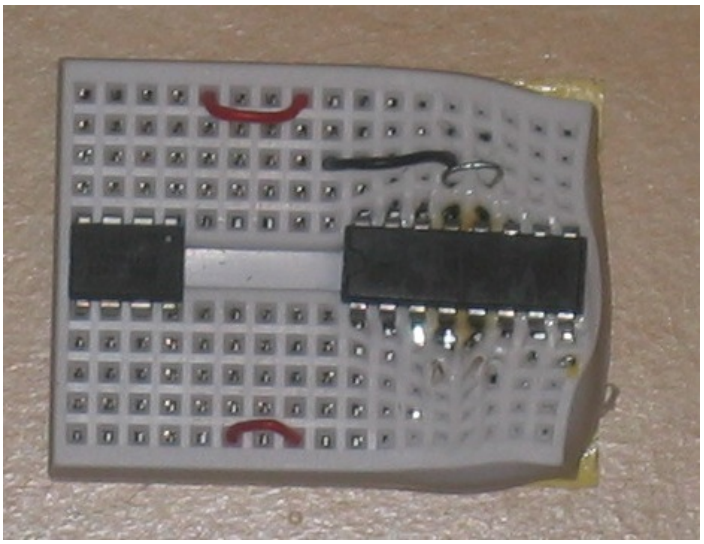


Figure 3. Two-Phase Motor Driver (L293D)

Make sure you consider power!



Stepper Specs

- Degrees/Step
  - Common values: 15, 7.5, 3.6, 1.8 deg/step
  - This is the same as 24, 48, 100, and 200 steps/full-rev
- Coil Resistance
  - Measured resistance of motor coils
- Volts/Amps
  - Rated values for running the motor
  - Amps is the important one!
  - Remember V=IR, so V/R = I
  - Example: 6VDC, 7.9Ω = .76A

So far…

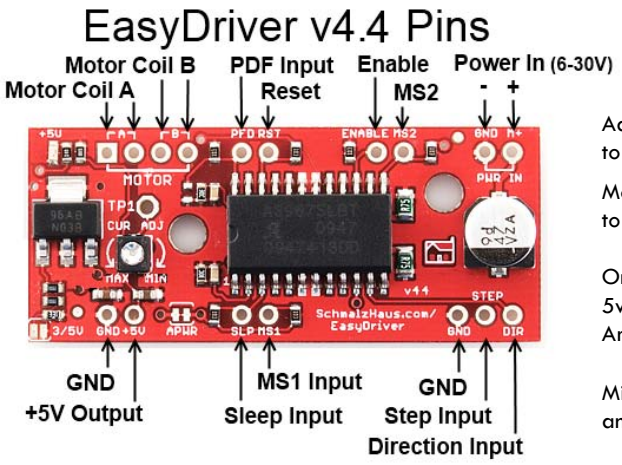
- Steppers move very precisely and are relatively powerful
- But are a bit of a pain to drive
  - Four wires from the Arduino
  - External driver circuits
  - Extra power supply to worry about
  - Use stepper library to make stepper "objects" for each one that you use
  - Your program needs to keep track of how far you've turned

Easier Motor Driving…

- There are chips specifically designed for driving steppers
  - They manage the sequence of signals
  - They manage the higher voltages of the motors
  - They have "chopper drives" to limit current
  - They can even do "microstepping"
    - This lets you do ½, ¼, 1/8, or 1/16 step
    - Increases resolution and smoothness, but might reduce power

Easy Driver

- Available from Sparkfun (among others)



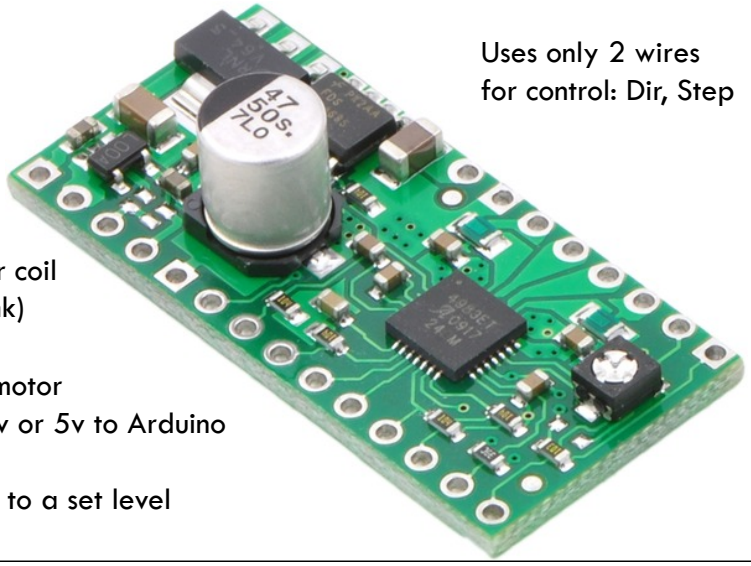
Adjustable from 150mA to 750mA per coil

Motor power from 6v to 30v

Onboard regulator for 5v or 3.3v power for Arduino

Microstepping full, ½, ¼, and 1/8th step

Pololu A4988 driver



Uses only 2 wires for control: Dir, Step

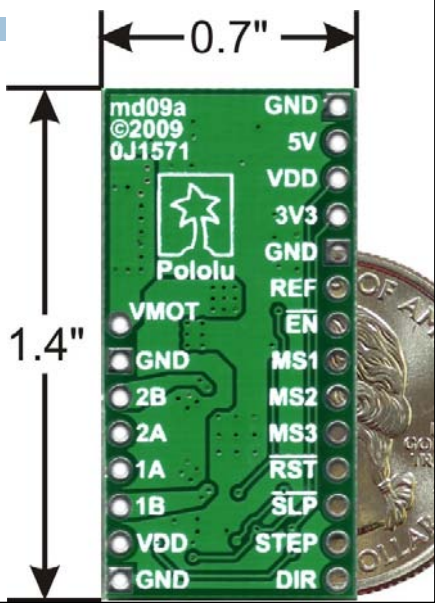
Up to 2A per coil (with heat sink)

8 – 35V on motor
Provides 3.3v or 5v to Arduino

Limits current to a set level

Pololu A4988 driver

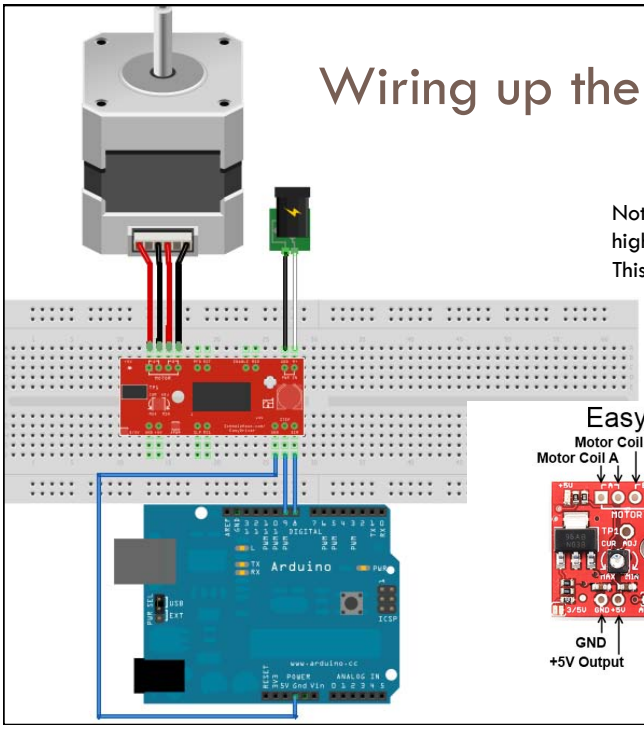
MS1	MS2	MS3	Microstep Resolution
Low	Low	Low	Full step
High	Low	Low	Half step
Low	High	Low	Quarter step
High	High	Low	Eighth step
High	High	High	Sixteenth step



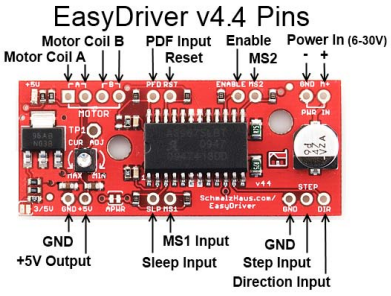
"Chopping" current driver…

- These boards have a little magic in them: Chopping Current Driver
  - Constant current source – you set limit
- This means you can use a higher voltage than your stepper is rated for
  - Chip will limit current to correct value
  - Higher voltage means more torque
  - Make SURE you set the current limit correctly though…

Wiring up the EasyDriver

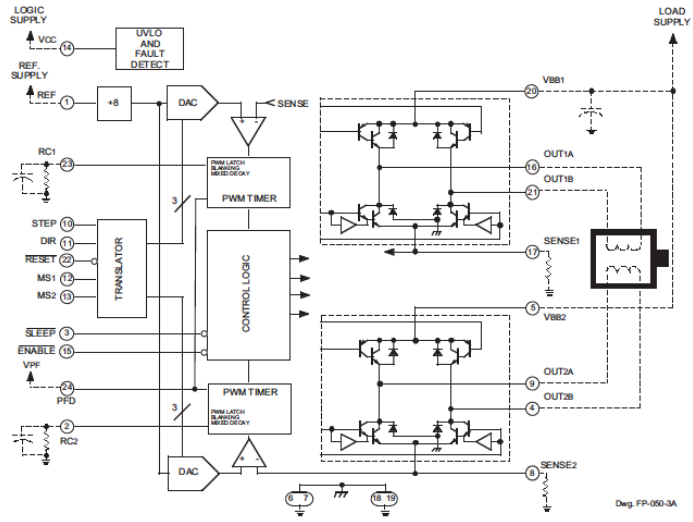


Note: MS1 and MS2 default high if you don't pull them low. This defaults to 1/8th microstepping



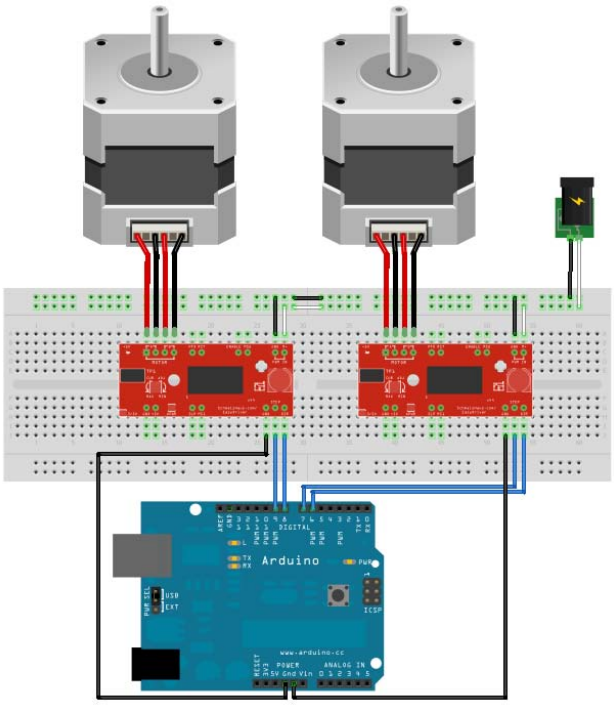
EasyDriver chip

MS1	MS2	Resolution
L	L	Full step (2 phase)
H	L	Half step
L	H	Quarter step
H	H	Eighth step



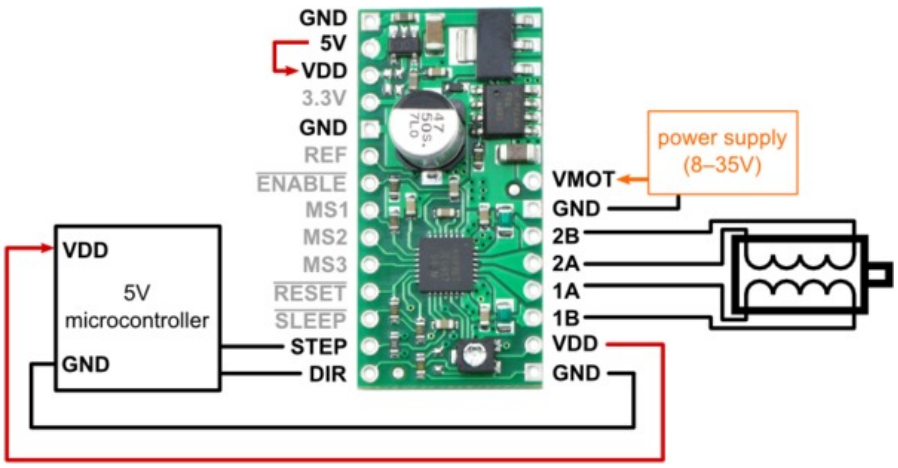
A3967

One EasyDriver per Stepper…



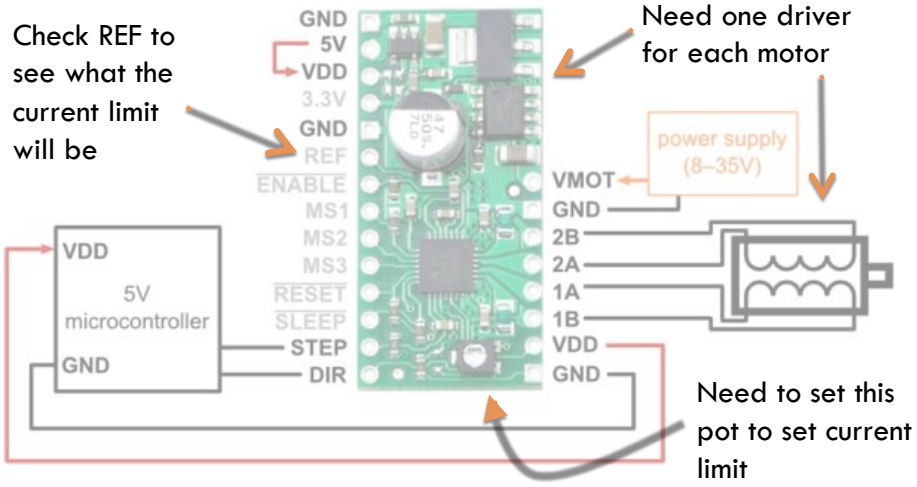
Wiring the Pololu A4988 driver

Minimal Connection



Using the Pololu A4988 driver

Minimal Connection



Current Limit on Pololu

- Turn pot (use a tiny screwdriver) and check REF

$$I_{TripMAX} = V_{REF} / (8 \times R_S)$$

  - Rs = 0.05 Ω

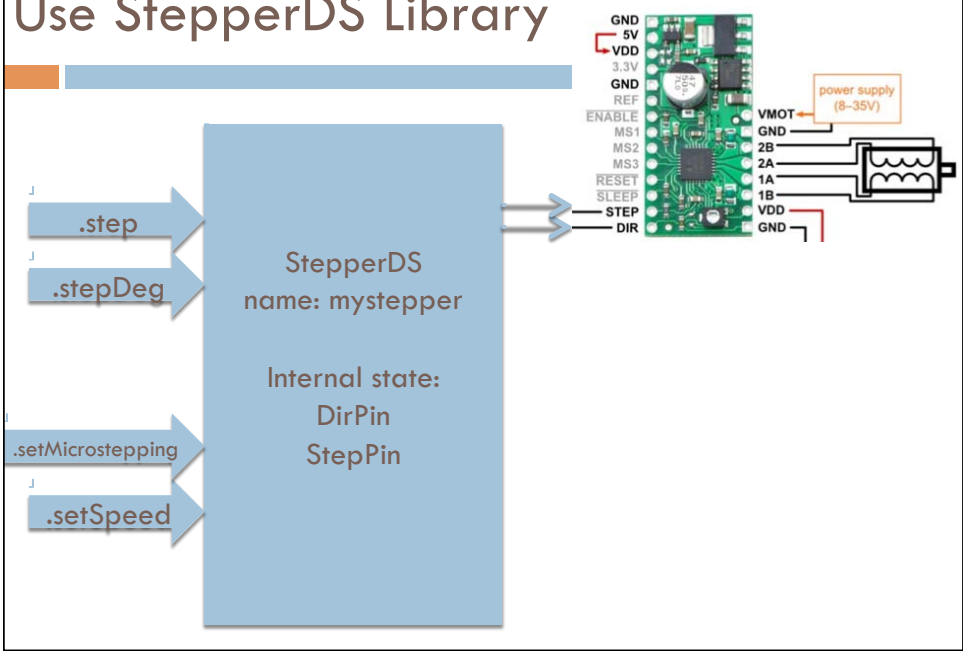
V REF	Current Limit
.1v	.250A
.15v	.375A
.2v	.500A
.25v	.625A
.3v	.750A
.35v	.875A
.4v	1.000A
.45v	1.125A

Using a Dir/Step driver

- Set the Dir pin – 0 is one dir, 1 is the other
- Toggle the Step pin up and down
  - You get one step per rising edge

```
for (int i=0; i < steps; i++) {
  digitalWrite(STEP_PIN, HIGH);
  delayMicroseconds(usDelay);
  digitalWrite(STEP_PIN, LOW);
  delayMicroseconds(usDelay);
}
```

Use StepperDS Library



Use the StepperDS Library

```
#include <StepperDS.h>
#define STEPS 200     // Steps per rev for your motor
#define DIR_PIN 8     // Dir pin
#define STEP_PIN 9    // step pin
#define knobPin A0    // potentiometer pin
StepperDS myStepper(STEPS, DIR_PIN, STEP_PIN); // make StepperDS object
int previous = 0; // the previous reading from the analog input

void setup()  {
  myStepper.setSpeed(60); // set speed to 60 RPMs
}

void loop() {
  int val = analogRead(knobPin);  // get the sensor value
  // move a number of steps equal to the change in the sensor reading
 myStepper.step(val - previous);
  previous = val; // remember the previous value of the sensor
}
```

Use the StepperDS Library

```
void loop()  {
  Serial.println("rotateDeg(360) - 60 RPM i.e. clockwise fast");
  stepper.stepDeg(360); //rotate a specific number of degrees
  delay(1000);

  Serial.println("rotateDeg(-360) - at speed 10 i.e. CCW slow");
  stepper.setSpeed(10);
  stepper.stepDeg(-360);  // degrees in reverse
  delay(1000);

  Serial.println("rotate(400) CW at half speed two times around");
  stepper.setSpeed(30);
  stepper.step(400); // rotate a specific number of steps - remember
  delay(1000);       // to take microstepping into account...

  Serial.println("rotate(-400) CCW at quarter speed, two times around");
  stepper.setSpeed(15);
  stepper.step(-(STEPS*2)); //steps in reverse
  delay(1000);
}
```

Installing StepperDS

- Grab the zip file from the class web site
- unpack – put the StepperDS folder in your Arduino/libraries folder
  - This is your "Sketchbook Location" , libraries sub-folder
  - Find out where this is using the Preferences dialog in Arduino
    - Mine is in Documents/Arduino/libraries on my Mac
- Restart Arduino and you should be good to go
  - #include <StepperDS.h>
  - Examples will show up in your "Examples" menu

Activity: Wire up a stepper

- Use a BigEasy stepper driver
- Use my StepperDS library if you like
- Or roll your own with your own code
- Make the stepper do something…
  - Follow a fixed pattern
  - Follow a knob
  - Follow a light sensor
  - React from Processing over the serial port

```
for (int i=0; i < steps; i++) {
  digitalWrite(STEP_PIN, HIGH);
  delayMicroseconds(usDelay);
  digitalWrite(STEP_PIN, LOW);
  delayMicroseconds(usDelay);
}
```

StepperDS

- Example of using classes/methods in Arduino
  - Also example of making a library that you can re-use and export to others
- Arduino language is really C/C++
  - Class definitions go in a .h file
  - Class code goes in a .cpp file
  - Both files go in a folder named for the library
  - Examples go in an "Examples" sub-folder
  - Each example gets its own sub-sub-folder
  - Examples are "regular" Arduino sketches (.ino files)

StepperDS.h

```
/*
StepperDS.h - - Stepper library for Arduino, using a Dir/Step (DS) driver
such as the EasyDriver or Pololu (A4988) driver

version (0.1) by Erik Brunvand

Loosly based on the "Stepper" library:
Original library     (0.1) by Tom Igoe.
Two-wire modifications   (0.2) by Sebastian Gassner
Combination version   (0.3) by Tom Igoe and David Mellis
Bug fix for four-wire   (0.4) by Tom Igoe, bug fix from Noah Shibley

This is based on the original Arduino Stepper library, but uses the
step/dir interface instead of driving the stepper phases directly.

These Dir/Step drivers use a two wire interface where one pin says
what direction, and the other causes the stepper to move one step
for each rising edge on the step pin.

These drivers also allow microstepping by driving the motors so
that they move a fraction of a full step on each "step" pulse. The
microstepping is set by some configuration pins on the driver boards.
This library doesn't manage the microstepping configuration pins. It
assumes that you've already done that. But, you can tell the StepperDS
objects what you've set things to.

If you do set the microstepping to something besides 1, then the
library scales the delay_per_step so that you're still operating
at the overall RPM that you specified. That is, it automatically
reducesthe delay_per_step to compensate for the larger number of
steps required to go 360 degrees. But, it's up to the user to know
how many steps it takes to go all the way around. The "step" method
always takes the raw number of (micro)steps to move.
```

Include LOTS of comments in the header to explain what's going on…

StepperDS.h

Some standard housekeeping stuff for Arduino libraries…

```
// ensure this library description is only included once
#ifndef StepperDS_h
#define StepperDS_h

// Hack to include the right .h file depending on which version of
// the Arduino IDE you're usi
#if defined(ARDUINO) && ARDUINO >= 100
#include "Arduino.h"
#else
#include "WProgram.h"
#endif
```

StepperDS.h Define StepperDS class

Start with the public stuff
Constructor functions, and method prototypes

```
// library interface description
class StepperDS {
  public:
    // constructors:
    StepperDS();
    StepperDS(int dirPin, int stepPin);
    StepperDS(int steps_per_rev, int dirPin, int stepPin);
    StepperDS(int steps_per_rev, int dirPin, int stepPin, int microStepping);
    StepperDS(int steps_per_rev, int dirPin, int stepPin, int microStepping, int whatSpeed );

    // Attach (or change) the pin assignment of a stepper
    void attach(int dirPin, int stepPin);

    // speed setter method (in rpm)
    void setSpeed(int whatSpeed);

    // set microstepping parameter (1, 2, 4, 8, or 16)
    void setMicroStepping(int microStepping);

    // move by number of (micro)steps method
    void step(int number_of_steps);
```

StepperDS.h: Define StepperDS class

More method prototypes, and private information

```
    // move by degrees method (both int degrees and float degrees)
    // I'm including all the possibilities, just for completeness
    void stepDeg(int number_of_degrees);
    void stepDeg(long number_of_degrees);
    void stepDeg(float number_of_degrees);
    void stepDeg(double number_of_degrees);

    // return the version number
    int version(void);

  private:
    void initStepperPins();     // initialize the stepper pins

    int RPM;                    // speed in RPM
    unsigned long step_delay;   // delay between steps, in ms, based on speed and microSte
    int steps_per_rev;          // steps in one full revolution at full-stepping
    long last_step_time;        // time stamp in ms of when last step was taken
    int microStepping;          // microStepping divisor (1, 2, 4, 8, or 16)

    // motor driver pin numbers:
    int dirPin;
    int stepPin;

};

#endif
```

Note "#endif" for "#ifndef StepperDS_h"

StepperDS.cpp: Fill in the methods

Start with comments again, of course!
Make sure to include StepperDS.h
Make some default definitions to use later…

```
#include "StepperDS.h"

// STEP_EDGE is the delay in microseconds between step-pin edges
// The A4988 driver used on the Pololu board has a 1us min
// delay between step edges.
#define STEP_EDGE 1             // delay between step-pin edges (us)
#define DEFAULT_RPM 60          // default RPM speed of the motor
#define DEFAULT_STEPS 200       // default steps/rev (200 = 1.8deg/step)
#define DEFAULT_MICRO 1         // default microstepping divisor (1, 2, 4, 8, or 16)
#define MS_PER_MIN 60L * 1000L  // number of microseconds in a second
```

StepperDS.cpp - constructors

Each constructor – with different number of arguments – needs to have its code filled in.
Zero-arg constructor is all defaults…
"this" refers to the object being constructed…

```
/*
 * zero-arg constructor.
 *
 */
StepperDS::StepperDS()
{
    this->RPM = DEFAULT_RPM;                // motor speed in RPM
    this->steps_per_rev = DEFAULT_STEPS;    // total number of steps for this motor
    this->microStepping = DEFAULT_MICRO;    // microstepping divisor
    // step_delay is ms per step.
    // 60000ms/min * 1min/60rev * 1rev/200steps  = ms/step
    this->step_delay = MS_PER_MIN / DEFAULT_RPM / DEFAULT_STEPS;

    // Arduino pins for the motor control connection. It's not valid
    // to have them both 0. If you use this constructor, you need to
    // use the name.attach(dir,step); function to assign pins
    this->dirPin = 0;
    this->stepPin = 0;

}
```

StepperDS.cpp - constructors

Three-arg constructor is probably the most common…

Note that it calls the private function initStepperPins()

```
/*
 * three-arg constructor.
 * Sets steps per rev, and which wires should control the motor
 */
StepperDS::StepperDS(int steps_per_rev, int dirPin, int stepPin)
{
    this->RPM = DEFAULT_RPM;                // motor speed in RPM
    this->steps_per_rev = steps_per_rev;    // total number of steps for this motor
    this->microStepping = DEFAULT_MICRO;    // microstepping divisor
    this->step_delay = MS_PER_MIN / DEFAULT_RPM / steps_per_rev;

    // Arduino pins for the motor control connection:
    this->dirPin = dirPin;
    this->stepPin = stepPin;

    initStepperPins(); // init the stepper pins

}
```

StepperDS.cpp: initStepperPins

This function is private to the class – it can't be called by user code

```
/*
 Initialize the step/dir pins as ouputs, and to a LOW value
 */
void StepperDS::initStepperPins()
{
    // setup the pins on the microcontroller as outputs
    pinMode(dirPin, OUTPUT);
    pinMode(stepPin, OUTPUT);
    // set the output values to some defaults
    digitalWrite(dirPin, LOW);
    digitalWrite(stepPin, LOW);
}
```

StepperDS.cpp: method code

Each method needs to be filled in – basically they set internal values in the stepper object…

```
/*
 * Set the dir and step pins for the stepper
 */
void StepperDS::attach(int dir, int step)
{
    this->dirPin = dir;
    this->stepPin = step;

    initStepperPins(); // init the stepper pins
}

/*
  Sets the speed in revs per minute
*/
void StepperDS::setSpeed(int whatSpeed)
{
    this->RPM = whatSpeed;
    this->step_delay = MS_PER_MIN / whatSpeed / steps_per_rev / microStepping;
}
```

StepperDS.cpp: method code

Remember to comment things!
Even you will have a hard time remembering what you did when you come back to your code later…

```
/*
 Sets the microStepping parameter - also sets speed as a byproduct
 The microStepping should be 1, 2, 4, 8, or 16 depending on what your
 driver board supports. If it's something else, the stepper will still
 operate, but the number of steps to make a full revoluation won't be
 so obvious. But, the function doesn't check to make sure the number is
 in range...
 */
void StepperDS::setMicroStepping(int microStepping)
{
    this->microStepping = microStepping;
    this->step_delay = MS_PER_MIN / RPM / steps_per_rev / microStepping;
}
```

```
/*
 Moves the motor steps_to_move steps.  If the number is negative,
 the motor moves in the reverse direction. This measures the time
 in ms since the last step so that other things can keep going in
 the background.

 Because we're targetting a step/dir stepper driver, we need
 to set the dir pin, and then make the step pin go both high
 and low to take a single step.
 */
void StepperDS::step(int steps_to_move)
{
    int steps_left = abs(steps_to_move);  // how many steps to take

    // determine direction based on whether steps_to_move is + or -:
    // set the direction pin to the correct value
    digitalWrite(dirPin, (steps_to_move > 0) ? HIGH:LOW);
    delayMicroseconds(STEP_EDGE); // tiny delay for dirPin to take effect
    // the stepPin will start out low because it's initialized that way, and
    // this step function leaves it LOW when it's done.

    // Decrement the number of steps, moving one step each time.
    // A full step needs both rising and falling edges on stepPin.
    while(steps_left > 0) {
        // move only if the appropriate delay has passed:
        if (millis() - last_step_time >= step_delay) { // time to take a step
            // get the timeStamp of when you stepped:
            this->last_step_time = millis();

            steps_left--;                   // decrement the number of steps left
            digitalWrite(stepPin, HIGH);    // Take a single step
            delayMicroseconds(STEP_EDGE);   // make sure to delay a little
            digitalWrite(stepPin, LOW);     // between stepPin edges
            delayMicroseconds(STEP_EDGE);

        }
    }
}
```

StepperDS.cpp: method code

```
void StepperDS::step(int steps_to_move)
{
    int steps_left = abs(steps_to_move);  // how many steps to take

    // determine direction based on whether steps_to_move is + or -:
    // set the direction pin to the correct value
    digitalWrite(dirPin, (steps_to_move > 0) ? HIGH:LOW);
    delayMicroseconds(STEP_EDGE); // tiny delay for dirPin to take effect
    // the stepPin will start out low because it's initialized that way, and
    // this step function leaves it LOW when it's done.

    // Decrement the number of steps, moving one step each time.
    // A full step needs both rising and falling edges on stepPin.
    while(steps_left > 0) {
        // move only if the appropriate delay has passed:
        if (millis() - last_step_time >= step_delay) { // time to take a step
            // get the timeStamp of when you stepped:
            this->last_step_time = millis();

            steps_left--;                   // decrement the number of steps left
            digitalWrite(stepPin, HIGH);    // Take a single step
            delayMicroseconds(STEP_EDGE);   // make sure to delay a little
            digitalWrite(stepPin, LOW);     // between stepPin edges
            delayMicroseconds(STEP_EDGE);

        }
    }
}
```

StepperDS.cpp: method code

Also versions with float and double as inputs…

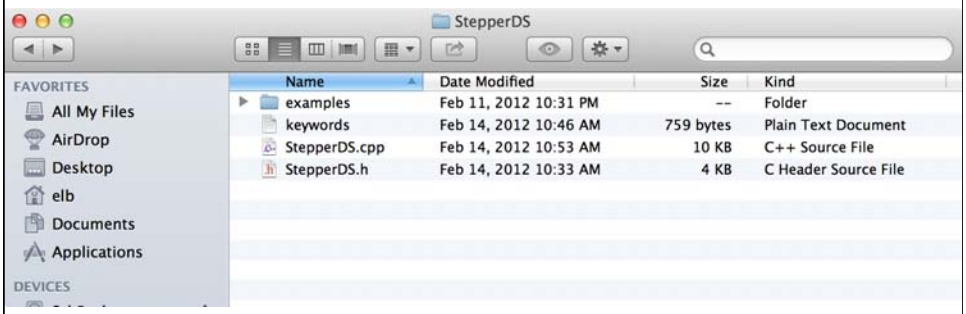
```
/*
 Moves the motor by deg degrees. This value can be negative to move
 in the other direction. This has both an int and float version. Especially
 with microstepping, you might wnat to specify degrees with a decimal point
 */
void StepperDS::stepDeg(int number_of_degrees)
{
    // Figure out how many steps are in that number of degrees
    // degrees * steps/rev * microsteps/step * 1rev/360deg = # of microsteps
    // Need a long here for the intermediate values in steps
    long steps = ((long)number_of_degrees * steps_per_rev * microStepping) / 360L;
    // Now, move that number of steps
    step((int)steps);

}
void StepperDS::stepDeg(long number_of_degrees)
{
    // Figure out how many steps are in that number of degrees
    // degrees * steps/rev * microsteps/step * 1rev/360deg = # of microsteps
    // Need a long here for the intermediate values in steps
    long steps = (number_of_degrees * steps_per_rev * microStepping) / 360L;
    // Now, move that number of steps
    step((int)steps);

}
```

StepperDS.cpp: method code

More standard housekeeping stuff
Return a version number so that later on
if there are multiple versions, the programmer
could do something different based on the
version being used

```
/*
  version() returns the version of the library:
*/
int StepperDS::version(void)
{
  return 1;
}
```

StepperDS library folder



Keywords is a text file that defines special words used in this library that should be colored differently in the Arduino IDE

StepperDS Keywords

keywords — Edited

```
#######################################
# Syntax Coloring Map For StepperDS
#######################################

#######################################
# Datatypes (KEYWORD1)
#######################################

StepperDS	KEYWORD1

#######################################
# Methods and Functions (KEYWORD2)
#######################################

step	KEYWORD2
stepDeg	KEYWORD2
setSpeed	KEYWORD2
setMicroStepping	KEYWORD2
version	KEYWORD2

#######################################
# Instances (KEYWORD2)
#######################################


#######################################
# Constants (LITERAL1)
#######################################
STEP_EDGE	LITERAL1
DEFAULT_RPM	LITERAL1
DEFAULT_STEPS	LITERAL1
DEFAULT_MICRO	LITERAL1
MS_PER_MIN	LITERAL1
```

Whew! Summary…

- PWM is a standard way to approximate analog output
  - Useful for dimming LEDs and speed control on DC motors
- Servos also use PWM
  - Very handy, fairly precise positioning on limited range (0-180 degrees)
- DC motors move bigger things
  - Use H-bridge to reverse direction
- Stepper motors enable precise angular control
  - Use stepper driver board for best results
  - Pay attention to electronics (volts, amps, ground, etc.)

What to use?

- Your go-to motors should probably be:
  - Servos – for anything that doesn't require a huge range of motion or lots of power
  - Steppers – for anything that requires full rotation and/or precise positioning
    - Use an EasyDriver or Pololu
    - You can often harvest steppers from broken printers and scanners…
  - DC motors for anything that just needs to spin without precise control
    - Could add a shaft encoder, but that's another story…