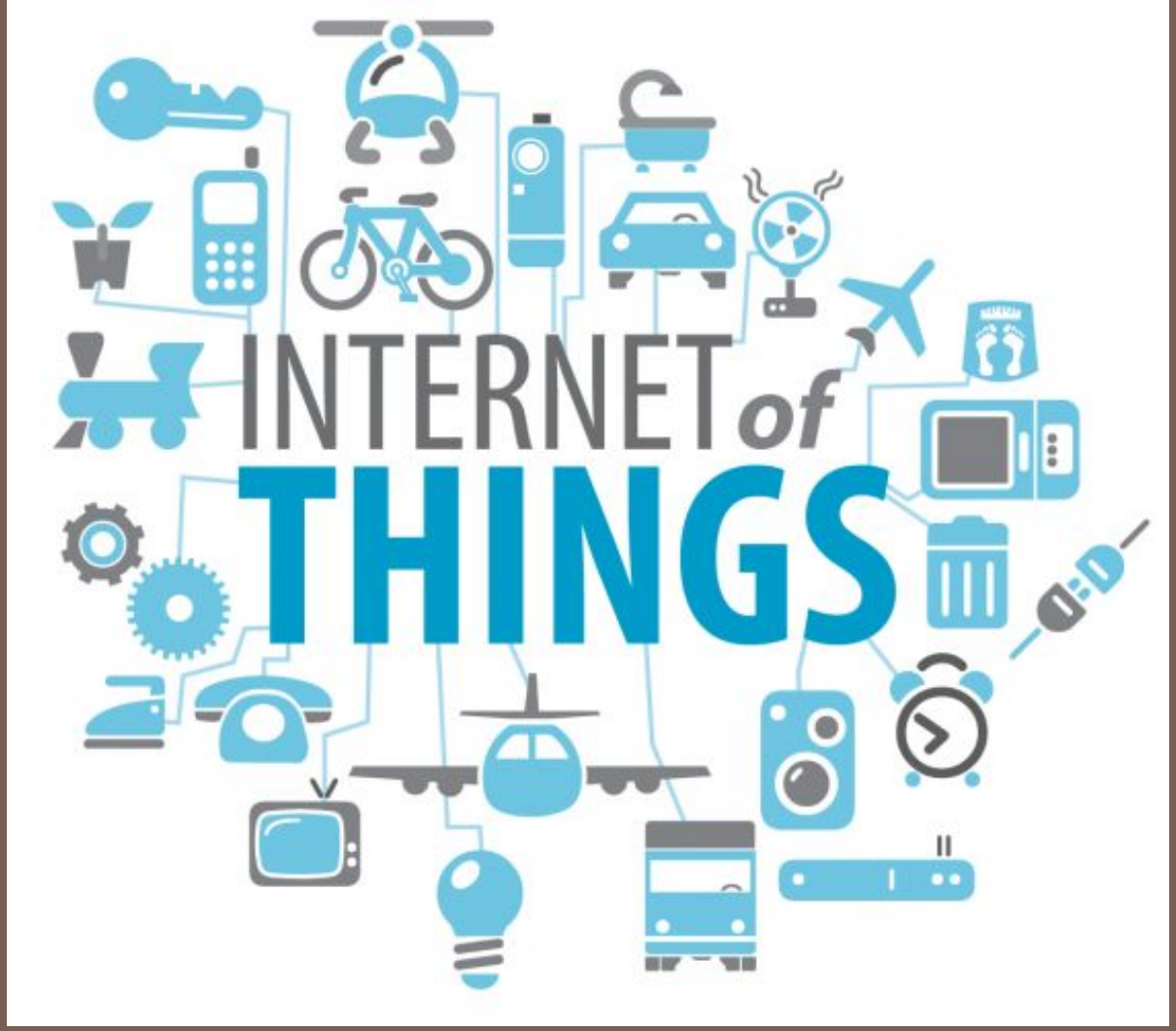


IMPLEMENTING IOE

Week 8

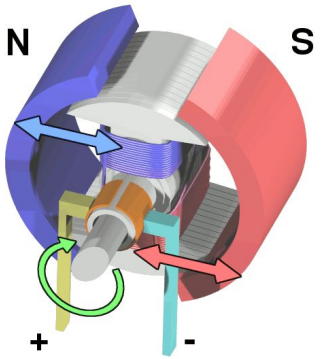
Assist. Prof. Rassim Suliyev - SDU 2017

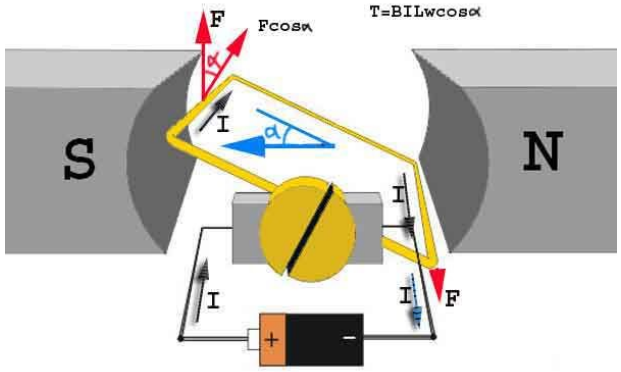
Physical Output

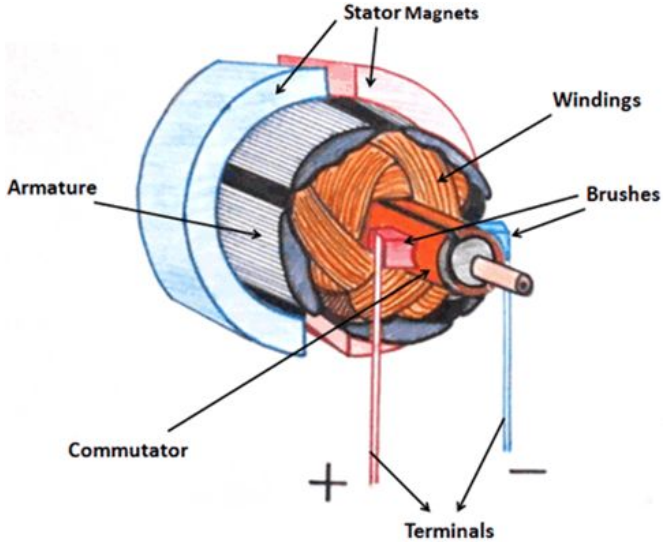
- Make things move by controlling motors with Arduino
- Servo-motors
  - Rotary actuator that allows for precise control of angular position
- DC-motors
  - Converts direct current electrical power into mechanical power
- Stepper-motors
  - Divides a full rotation into a number of equal steps

Brushed DC Motors

- Simple devices with two leads connected to brushes (contacts)
  - Control the magnetic field of the coils
  - Drives a metallic core (armature)
- Direction of rotation can be reversed by reversing the polarity
- Require a transistor to provide adequate current
- Primary characteristic in selecting a motor is torque
  - How much work the motor can do

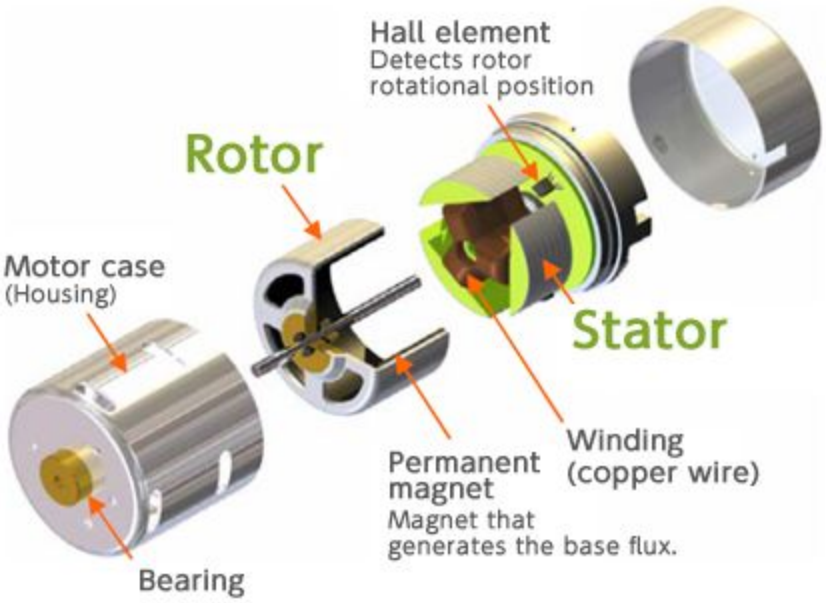


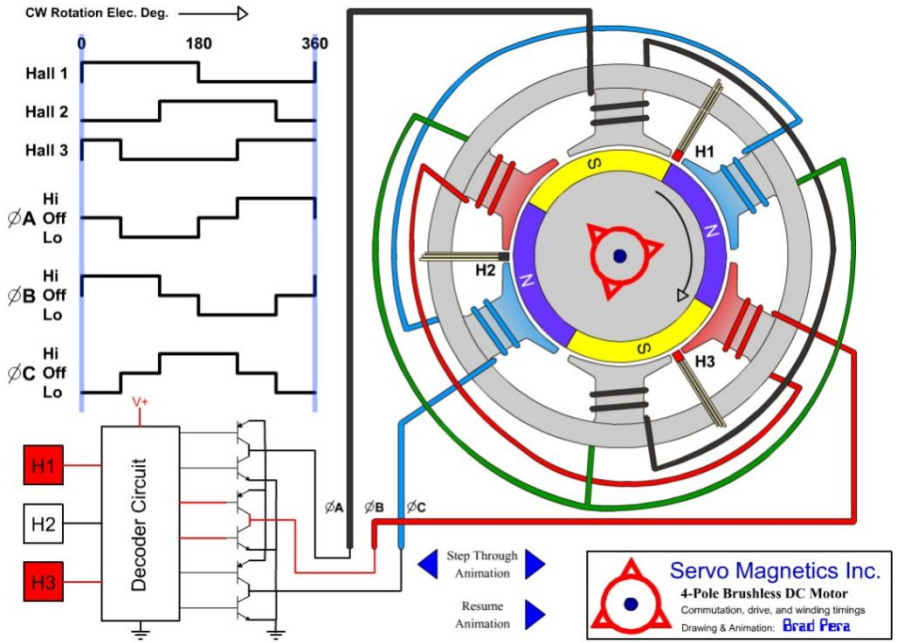




Brushless Motors

- More powerful and efficient for a given size
- Three phases of driving coils
- Require more complicated electronic control
  - Electronics speed controllers





DC Motor Parameters

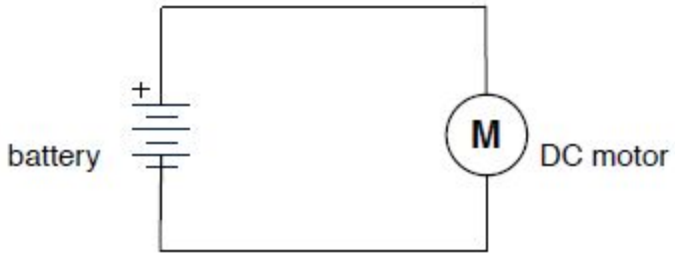
- Direct-drive vs. gearhead – built-in gears or not
- Voltage – what voltage it best operates at
- Current (efficiency) – how much current it needs to spin
- Speed – how fast it spins
- Torque – how strong it spins
- Size, shaft diameter, shaft length

DC Motor Characteristics

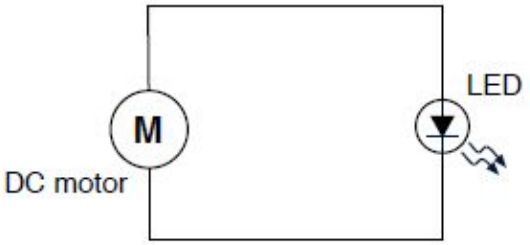
- When the first start up, they draw a lot more current, up to 10x more
- If you “stall” them (make it so they can’t turn), they also draw a lot of current
- They can operate in either direction, by switching voltage polarity
- Usually spin very fast: >1000 RPM
- To get slower spinning, need gearing

Driving DC Motor

- To drive them, apply a voltage
- The higher the voltage, the faster the spinning
- Polarity determines which way it rotates

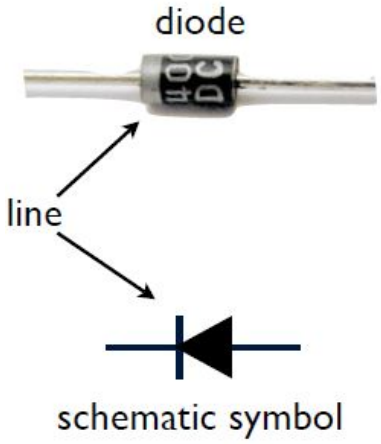


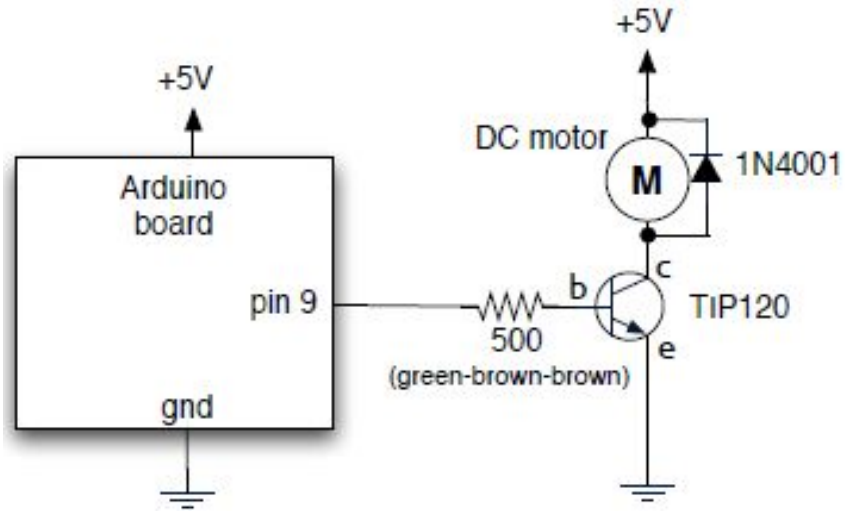
- Can be used as voltage generators

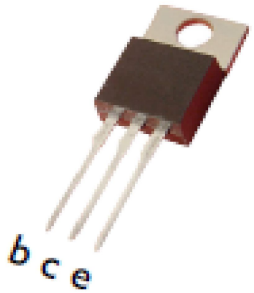


Switching Motors with Transistors

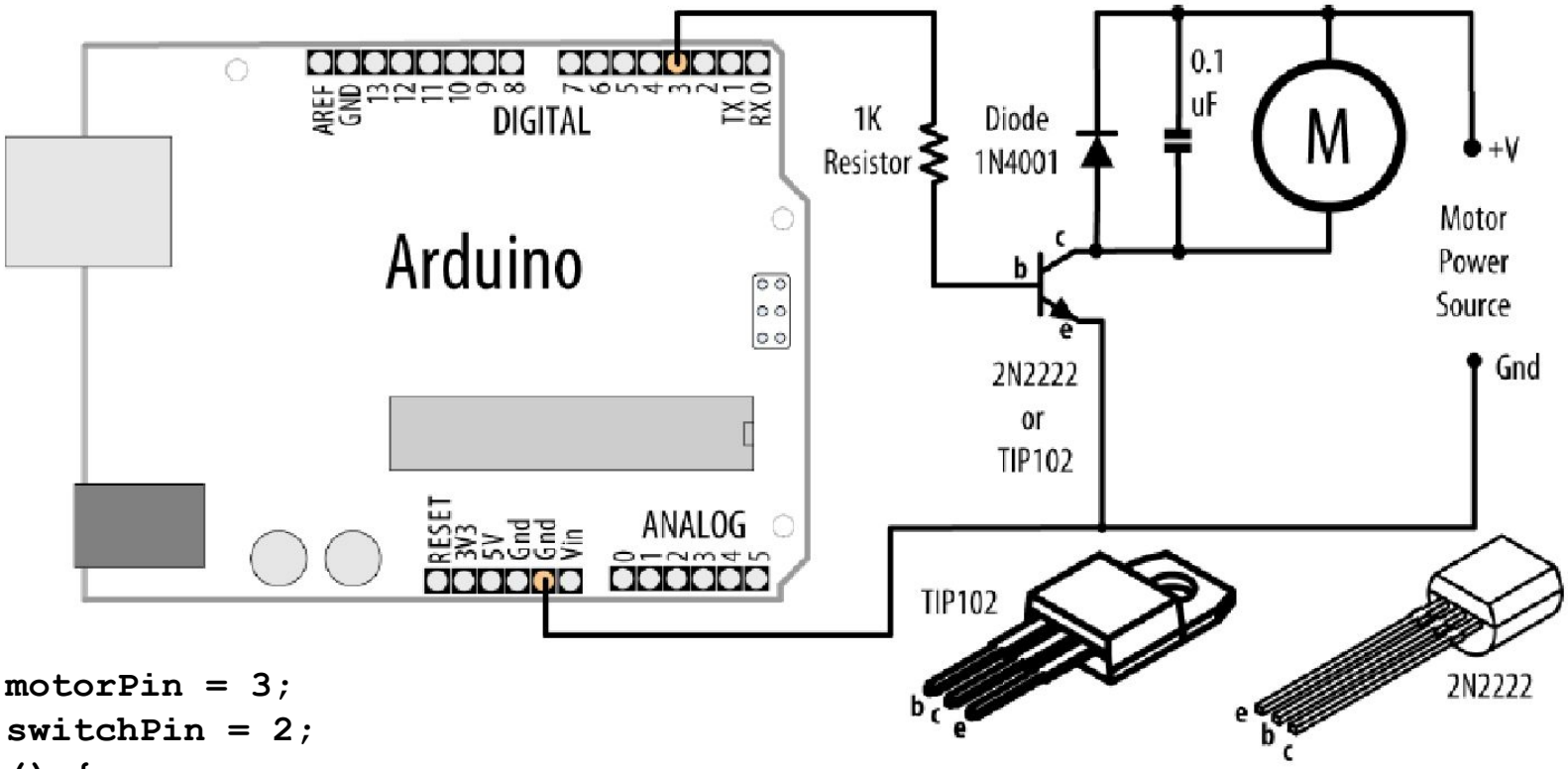
- Transistors switch big signals with little signals
- Since motors can act like generators,
- Need to prevent them from generating "kickback" into the circuit
- Can control speed of motor with analogWrite()





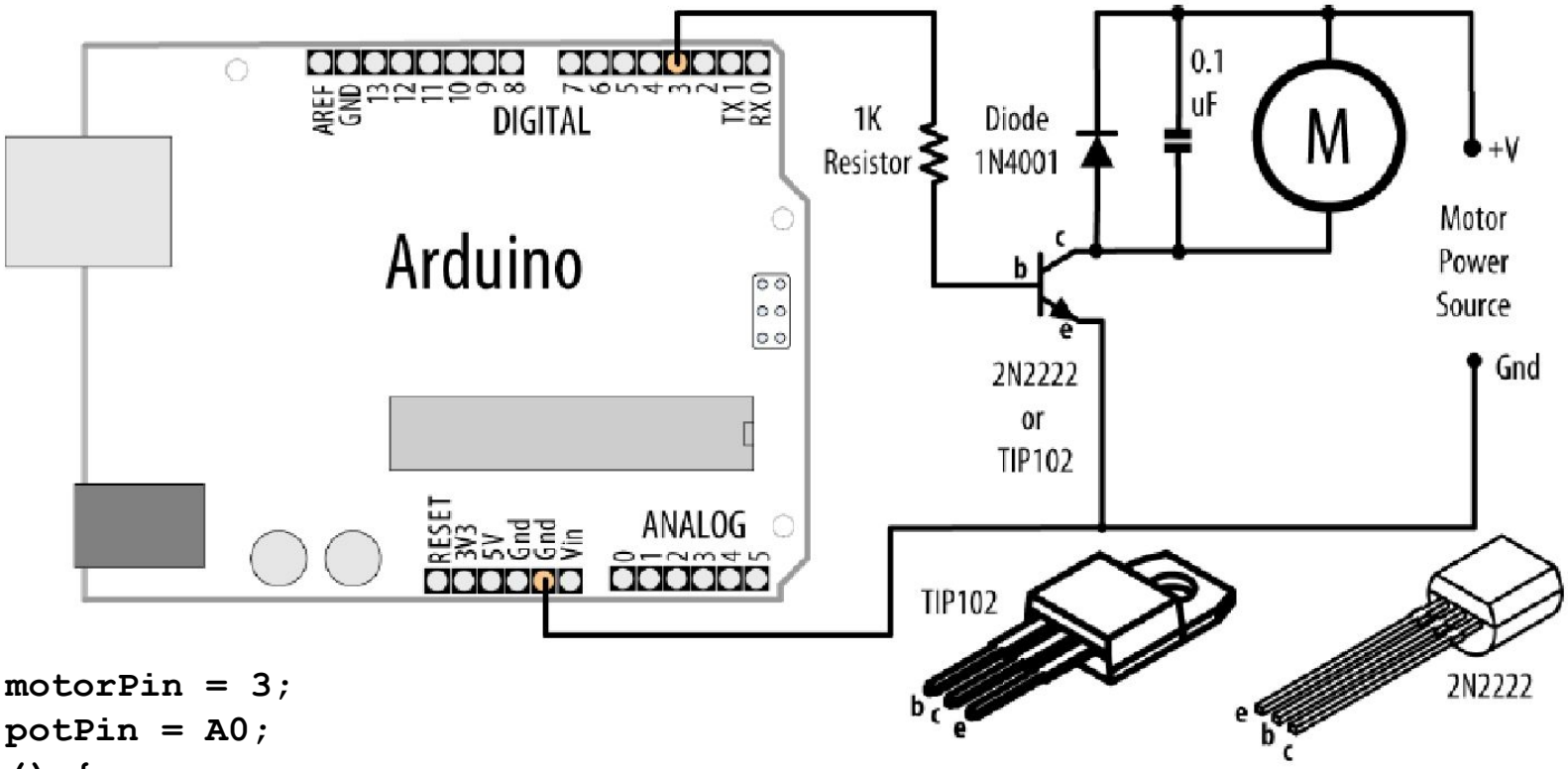


Driving a Brushed Motor




```
const int motorPin = 3;
const int switchPin = 2;
void setup() {
  pinMode(switchPin, INPUT);
  pinMode(motorPin, OUTPUT);
}
void loop() {
  digitalWrite(motorPin, digitalRead(switchPin));
}
```

Controlling Speed of DC-Motor

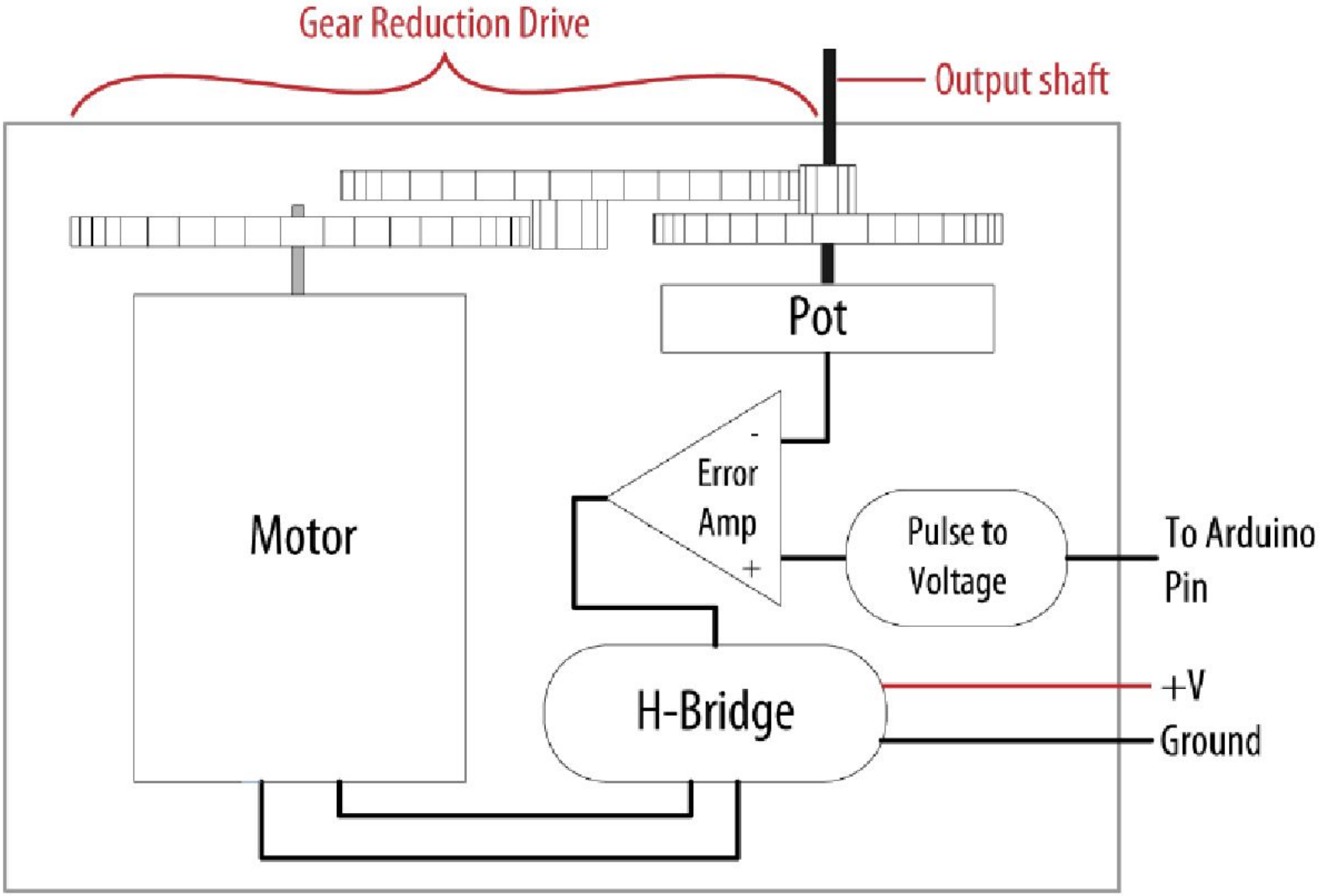



```
const int motorPin = 3;
const int potPin = A0;
void setup() {
}
void loop() {
  int spd = analogRead(potPin);
  spd = map(spd, 0, 1023, 0, 255);
  analogWrite(motorPin, spd);
}
```

Servo-Motors

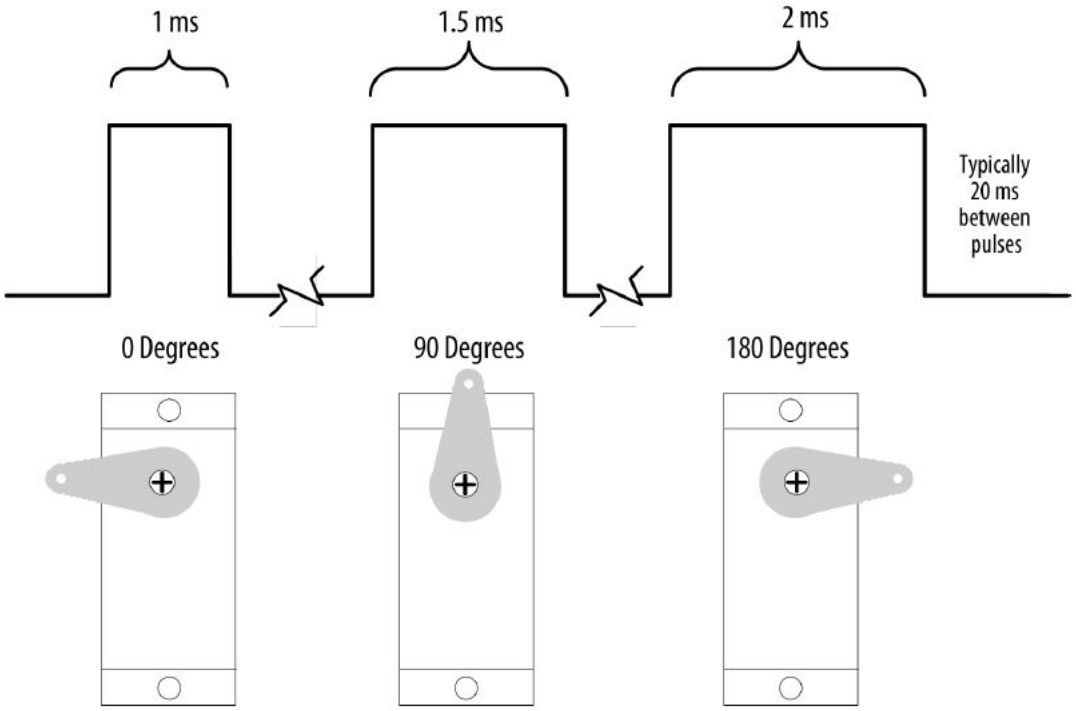
- Allow accurately control physical movement
- Move to a position instead of continuously rotating
- Rotate over a range of 0 to 180 degrees
- Motor driver is built into the servo
  - Small motor connected through gears
  - Output shaft drives a servo arm
  - Connected to a potentiometer to provide position feedback
- Continuous rotation servos
  - Positional feedback disconnected
  - Rotate continuously clockwise and counter clockwise with some control over the speed

Servo-Motors



Servo-Motors

- Respond to changes in the duration of a pulse
  - Short pulse of 1 ms will cause to rotate to one extreme
  - Pulse of 2 ms will rotate the servo to the other extreme



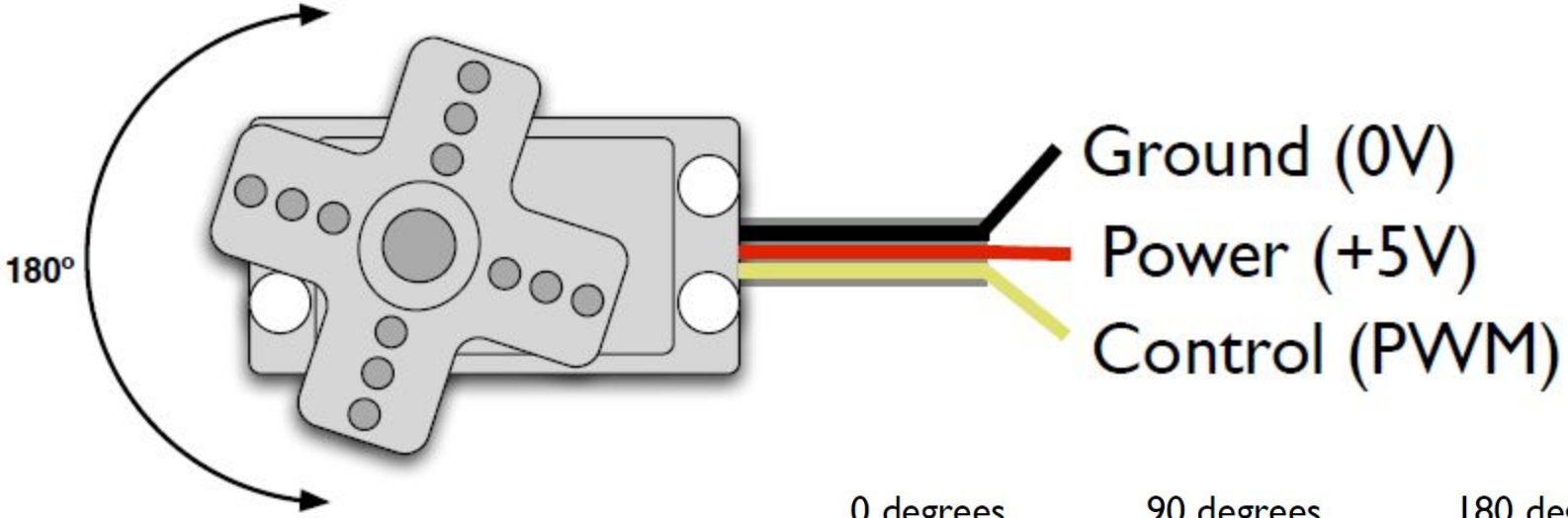
Servos require pulses different from the PWM output from analogWrite

Servo-Motors

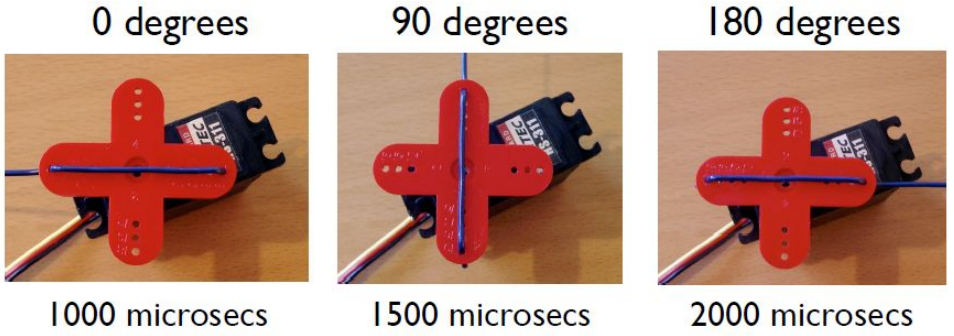
- Come in all sizes
  - from super-tiny
  - to drive-your-car
- All have same 3-wire interface
- Servos are spec'd by:
  - weight: 9g
  - speed: .12s/60deg @ 6V
  - torque: 22oz/1.5kg @ 6V
  - voltage: 4.6~6V
  - size: 21x11x28 mm

Servo Control

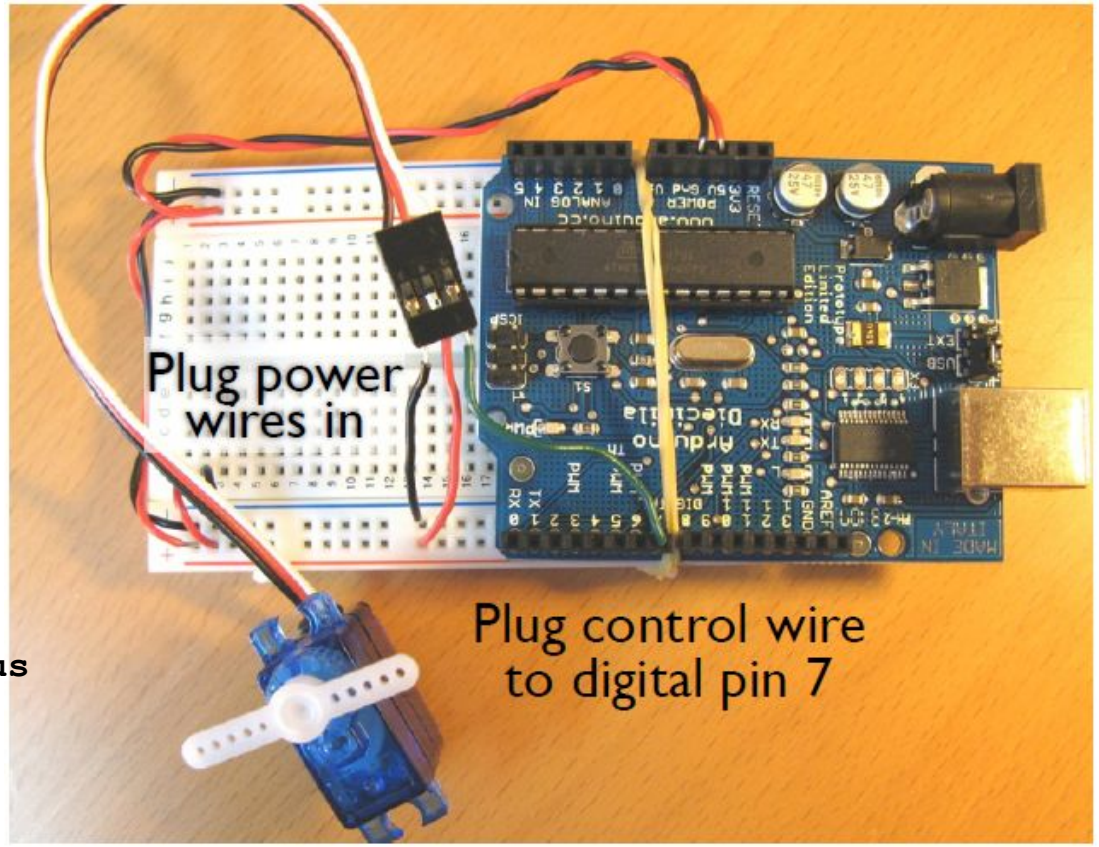
- PWM freq is 50 Hz (i.e. every 20 millisecs)
- Pulse width ranges from 1 to 2 millisecs



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



Servo and Arduino



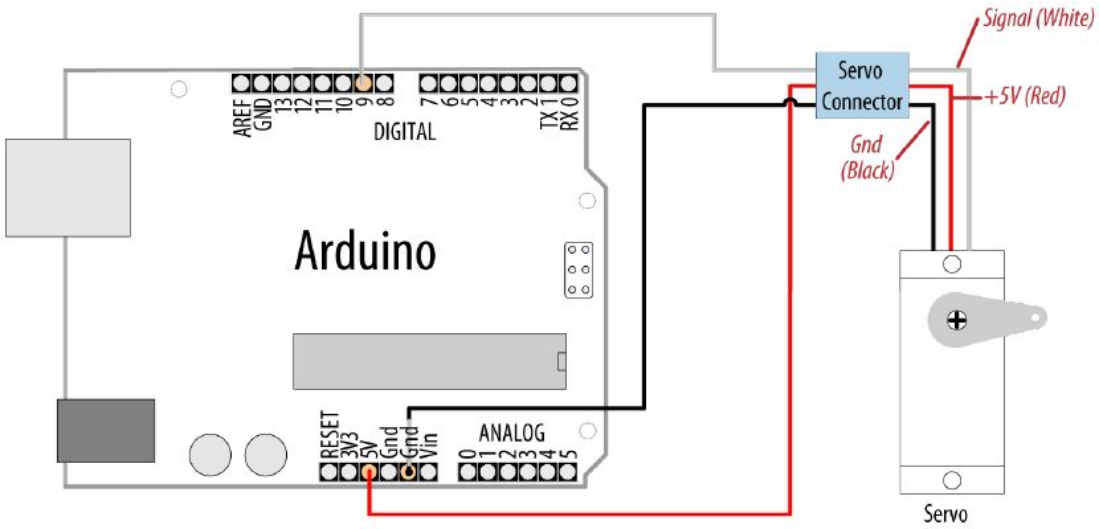

```
const int servoPin = 7;
const int potPin = A0;
const int pulsePeriod = 20000; //us
void setup() {
  pinMode(servoPin, OUTPUT);
}
void loop() {
  int hiTime = map(analogRead(potPin), 0, 1023, 600, 2500);
  int loTime = pulsePeriod - hiTime;
  digitalWrite(servoPin, HIGH);  delayMicroseconds(hiTime);
  digitalWrite(servoPin, LOW);  delayMicroseconds(loTime);
}
```

Use the Servo library

- servo.attach(pin[, min][, max]) – attach the servo
  - pin- the pin number that the servo is attached to
  - *min (optional) - t*he pulse width, in microseconds, corresponding to the minimum (0-degree) angle on the servo (defaults to 544)
  - *max (optional) - t*he pulse width, in microseconds, corresponding to the maximum (180-degree) angle on the servo (defaults to 2,400)
- servo.write(angle) – turn the servo arm
  - angle – the degree value to write to the servo (from 0 to 180)

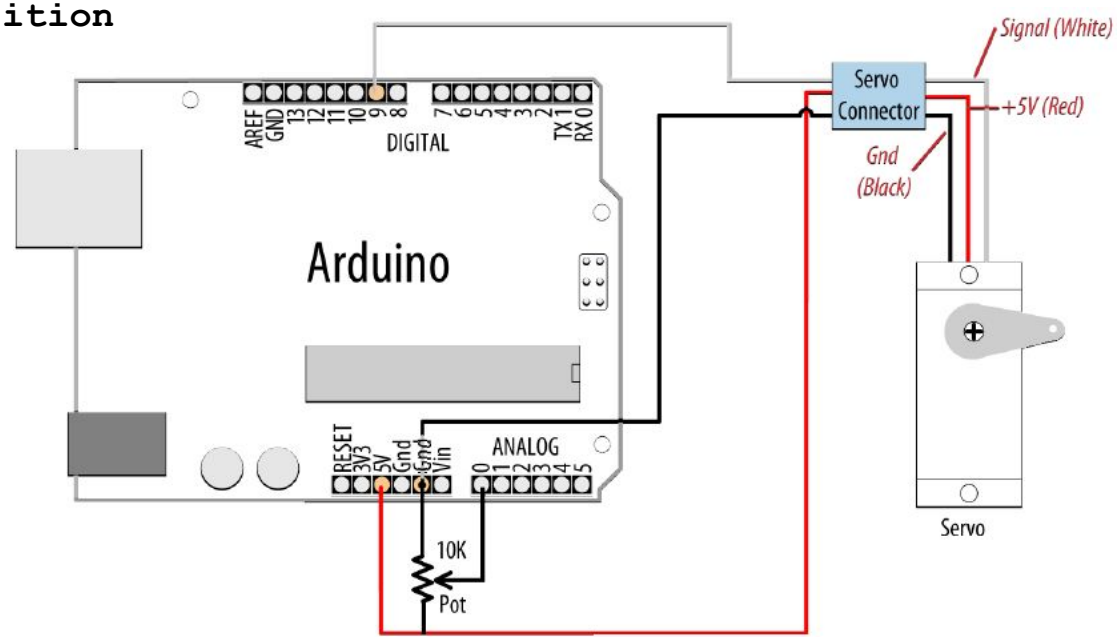
Servo sweeper

```
#include <Servo.h>
Servo myservo; // create servo object to control a servo
int angle = 0; // variable to store the servo position
void setup(){
  myservo.attach(9); // attaches the servo on pin 9 to the servo object
}
void loop(){
  for(angle = 0; angle < 180; angle += 1){ // goes from 0 degrees to 180
    myservo.write(angle); //tell servo to go to position in variable 'angle'
    delay(20); // waits 20ms between servo commands
  }
  for(angle = 180; angle >= 1; angle -= 1){ // goes from 180 degrees to 0
    myservo.write(angle);
    delay(20);
  }
}
```



Controlling angle with pot

```
#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
  val = map(val, 0, 1023, 0, 180); // scale it to use it with the servo
  myservo.write(val); // sets position
  delay(15);
}
```

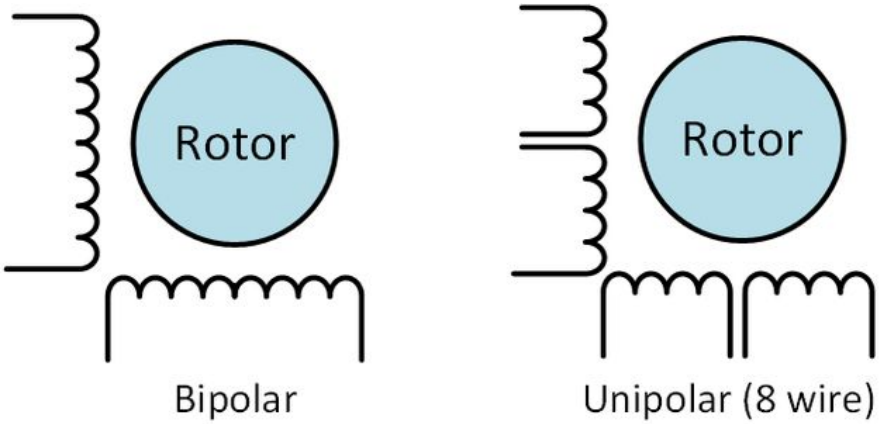


Stepper Motors

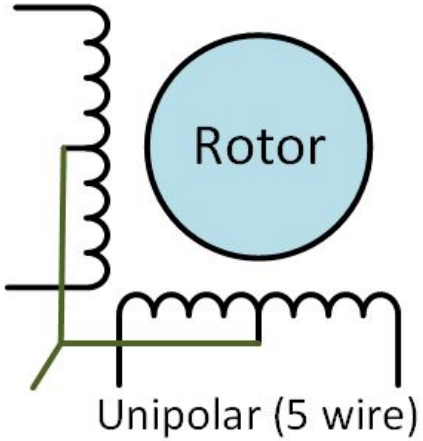
- Rotate a specific number of degrees in response to control pulses
- Number of degrees for a step is motor-dependent
  - Ranging from one or two degrees per step to 30 degrees or more
- Two types of steppers
  - Bipolar - typically with four leads attached to two coils
  - Unipolar - five or six leads attached to two coils
- Additional wires in a unipolar stepper are internally connected to the center of the coils

Stepper Motors

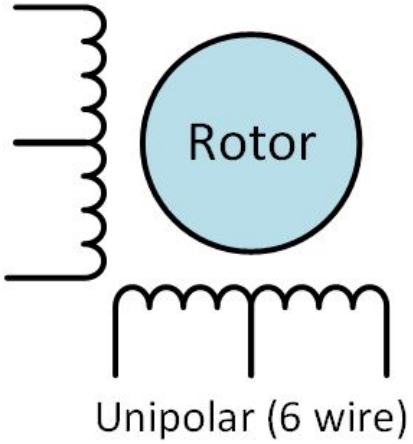
- Unipolar drivers always energize the phases in the same way
  - Single "common" lead, will always be negative.
  - The other lead will always be positive
  - Disadvantage - less available torque, because only half of the coils can be energized at a time
- Bipolar drivers work by alternating the polarity to phases
  - All the coils can be put to work



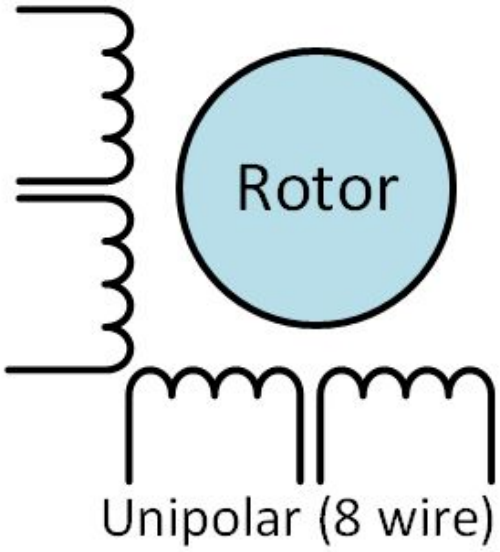
Stepper Motors



All of the common coil wires are tied together internally and brought out as a 5th wire. This motor can only be driven as a unipolar motor.



This motor only joins the common wires of 2 paired phases. These two wires can be joined to create a 5-wire unipolar motor. Or you just can ignore them and treat it like a bipolar motor!

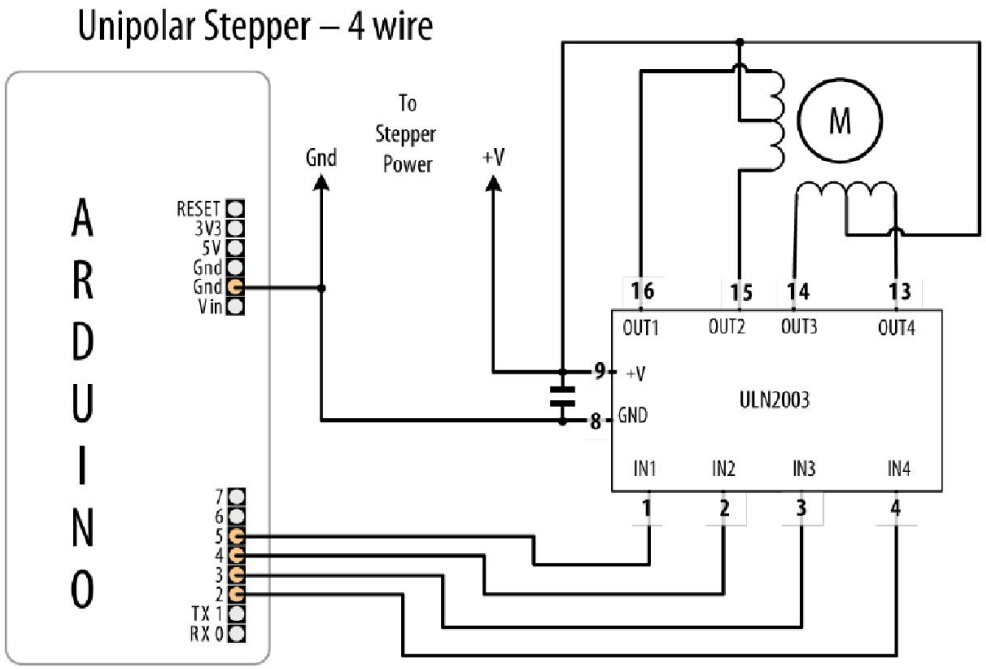


It can be driven in several ways:

- 4-phase unipolar - All the common wires are connected together - just like a 5-wire motor.
- 2-phase series bipolar - The phases are connected in series - just like a 6-wire motor.
- 2-phase parallel bipolar - The phases are connected in parallel. This results in half the resistance and inductance - but requires twice the current to drive. The advantage of this wiring is higher torque and top speed.

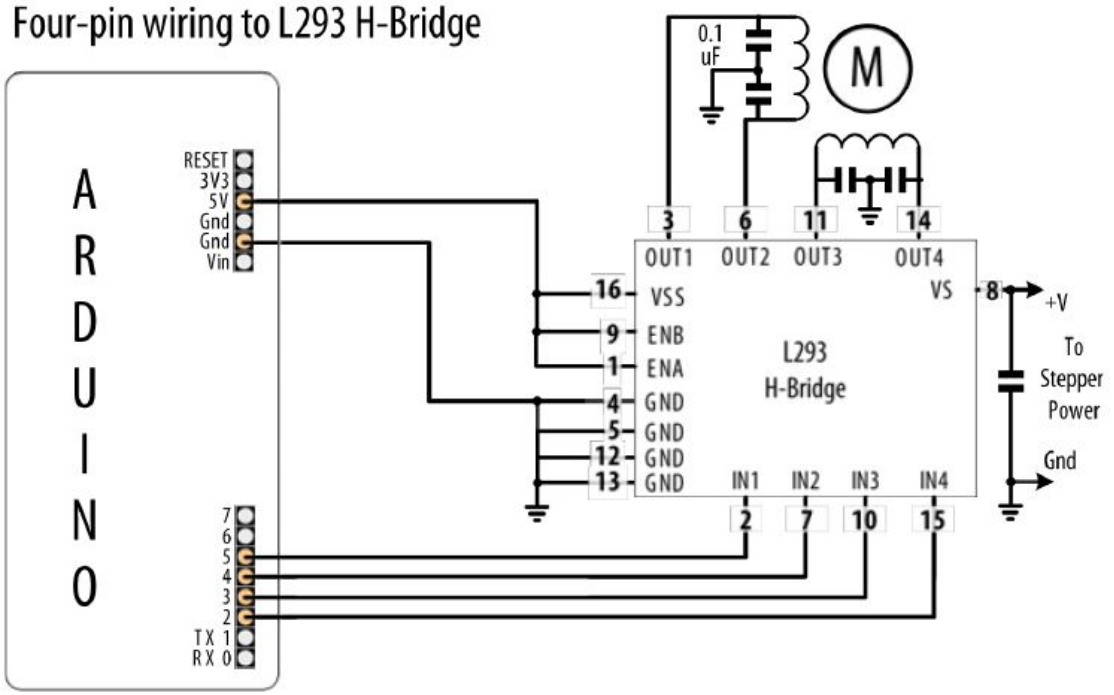
Driving a Unipolar Stepper Motor

```
const int stepperPins[4] = {2, 3, 4, 5};
int delayTime = 5;
void setup() {
  for(int i=0; i<4; i++)
    pinMode(stepperPins[i], OUTPUT);
}
```




```
void loop() {
  digitalWrite(stepperPins[0], HIGH);
  digitalWrite(stepperPins[1], LOW);
  digitalWrite(stepperPins[2], LOW);
  digitalWrite(stepperPins[3], LOW);
  delay(delayTime);
  digitalWrite(stepperPins[0], LOW);
  digitalWrite(stepperPins[1], HIGH);
  digitalWrite(stepperPins[2], LOW);
  digitalWrite(stepperPins[3], LOW);
  delay(delayTime);
  digitalWrite(stepperPins[0], LOW);
  digitalWrite(stepperPins[1], LOW);
  digitalWrite(stepperPins[2], HIGH);
  digitalWrite(stepperPins[3], LOW);
  delay(delayTime);
  digitalWrite(stepperPins[0], LOW);
  digitalWrite(stepperPins[1], LOW);
  digitalWrite(stepperPins[2], LOW);
  digitalWrite(stepperPins[3], HIGH);
  delay(delayTime);
}
```

Driving a Bipolar Stepper Motor



Step	wire 1	wire 2	wire 3	wire 4
1	High	low	high	low
2	low	high	high	low
3	low	high	low	high
4	high	low	low	high

```
const int stepperPins[4] = {2, 3, 4, 5};
int delayTime = 5;
void setup() {
  for(int i=0; i<4; i++)
    pinMode(stepperPins[i], OUTPUT);
}
void loop() {
  digitalWrite(stepperPins[0], LOW);
  digitalWrite(stepperPins[1], HIGH);
  digitalWrite(stepperPins[2], HIGH);
  digitalWrite(stepperPins[3], LOW);
  delay(delayTime);
  digitalWrite(stepperPins[0], LOW);
  digitalWrite(stepperPins[1], HIGH);
  digitalWrite(stepperPins[2], LOW);
  digitalWrite(stepperPins[3], HIGH);
  delay(delayTime);
  digitalWrite(stepperPins[0], HIGH);
  digitalWrite(stepperPins[1], LOW);
  digitalWrite(stepperPins[2], LOW);
  digitalWrite(stepperPins[3], HIGH);
  delay(delayTime);
  digitalWrite(stepperPins[0], HIGH);
  digitalWrite(stepperPins[1], LOW);
  digitalWrite(stepperPins[2], HIGH);
  digitalWrite(stepperPins[3], LOW);
  delay(delayTime);
}
```

Arduino Stepper Library

- Allows to control unipolar or bipolar stepper motors
- stepper(steps, pin1, pin2, pin3, pin4) – attach and initialize stepper
  - steps: number of steps in one revolution of motor
  - pin1, pin2, pin3, pin4: 4 pins attached to the motor
- setSpeed(rpms) - Sets the motor speed in rotations per minute (RPMs)
- step(steps) - Turns the motor a specific number of steps, positive to turn one direction, negative to turn the other
  - This function is blocking
  - wait until the motor has finished moving before passing control to the next line in sketch

Arduino Stepper Library

```
#include <Stepper.h>
const int stepsPerRevolution = 200;  // change this to fit the number of steps
Stepper myStepper(stepsPerRevolution, 2, 3, 4, 5);
void setup() {
  myStepper.setSpeed(60);
  Serial.begin(9600);
}
void loop() {
  Serial.println("clockwise");
  myStepper.step(stepsPerRevolution);
  delay(5);
  Serial.println("counterclockwise");
  myStepper.step(-stepsPerRevolution);
  delay(5);
}
```