EP1000

Actuators

Actuators

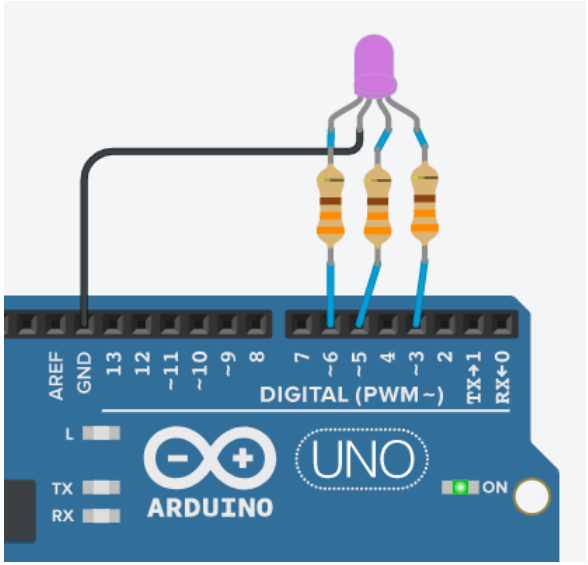
- A component of a machine that moves/controls a mechanism in a system.
- An “output” device.
- Examples:
  - Displays – LED, LCD, Neopixels
  - Motors – DC Motors, Stepper Motors, Servo Motors
  - Valves – Water, gas
  - Solenoids – controls heavier voltages, currents
  - Sounds – buzzers, alarms

Display devices

- The most common output display devices used in projcects are:
  - LEDs (indicators)
  - LCD (display panels, information) 16x2, 20x2
  - 7-segment tube displays
  - Dot-matrix displays (running messages)
  - Oled displays
  - NeoPixels (strip lights, multi RGB leds)

LEDs

- Types: single, RGB, 7-segment
- Identify the Anode (or Cathode)
- Ensure there is proper current limiting resistors
- You need a delay to maintain the display (persistence of vision)
- You can use PWM to obtain different intensity levels.




```
3   const int RED = 6;
4   const int BLUE = 5;
5   const int GREEN = 3;
6
7   void setup()
8   {
9   }
10
11  void loop()
12  {
13    analogWrite(RED,random(256));
14    analogWrite(GREEN,random(256));
15    analogWrite(BLUE,random(256));
16    delay(500);
17  }
18
```

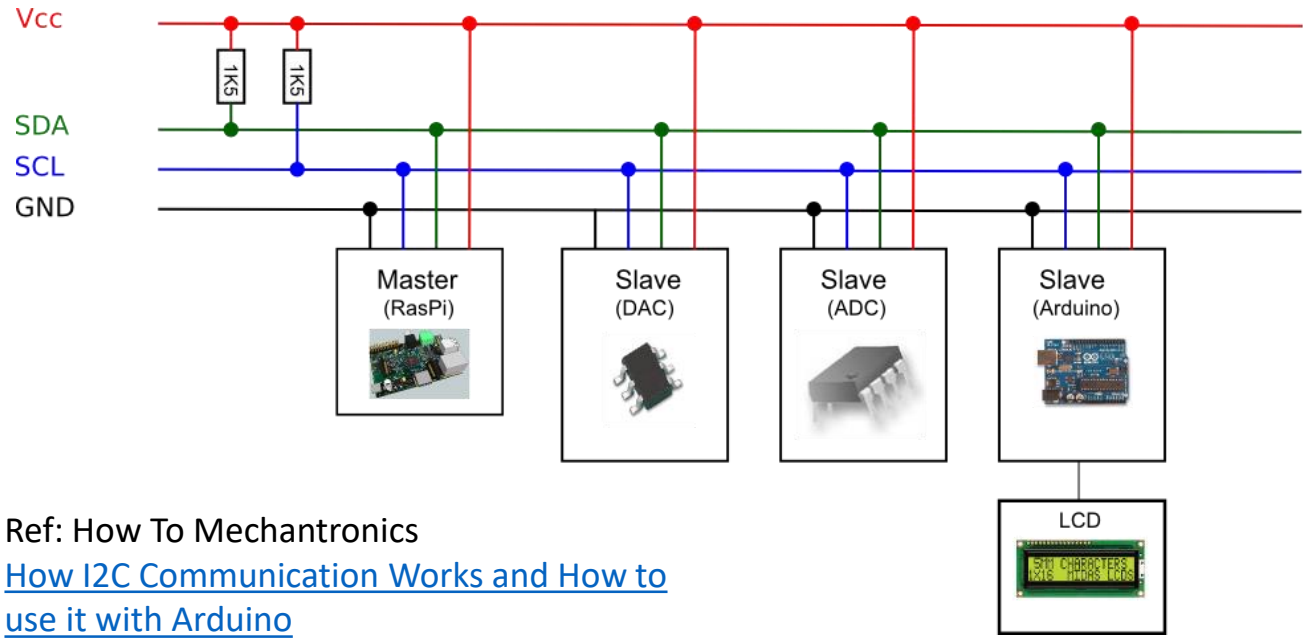
Uno Random RGB Led using PWM

LCD Displays

- Common method of displaying information
- Usually Hitachi HD44780 control requires 11 lines (8 data, RS, RW, En)
- Made simpler using an I2C 1602 controller for the LCD panel
- Library: LiquidCrystal I2C by Frank de Brabander.
- Reference: LastMinuteEngineers

I2C Interface

- A method of Serial communication.
- Uses 2 wires (SDA, SCL) to communicate between devices.
- Each device has a unique address which identifies it.
- Can have up to 1024 devices



Ref: How To Mechantronics
[How I2C Communication Works and How to use it with Arduino]

Identifying I2c devices

- Use the controller (master) to scan the bus for devices (slaves).
- The addresses will be printed out on the Serial Monitor.
- You can then use the address to write/read from the device.
- Library: Arduino Wire Library
- Example code is usually provided by the device library.

```
// I2C Library
#include <Wire.h>

void setup() {
  Serial.begin (9600);
  Serial.println ("I2C scanner. Scanning ...");
  // initialise I2C
  Wire.begin();
  for (byte i = 8; i < 120; i++)
  {
    Wire.beginTransmission (i);
    if (Wire.endTransmission () == 0)
      {
      Serial.print ("Found address: ");
      Serial.print (i, DEC);
      Serial.print (" (0x");
      Serial.print (i, HEX);
      Serial.println (")");
      delay (1);  // maybe unneeded?
      } // end of good response
  } // end of for loop
  Serial.println ("Done.");
}

void loop() {
  // do nothing here
}
```

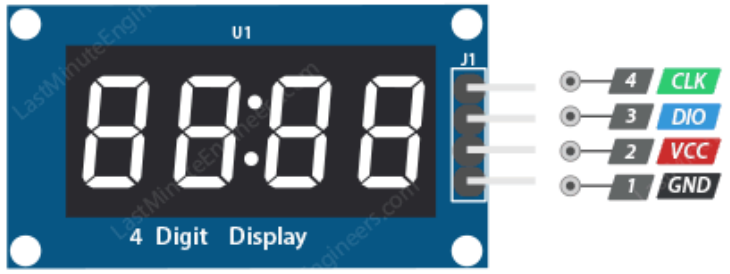
Writing to I2C LCD display

- Using the LiquidCrystal_I2C library.
- Create the object lcd
- Initial housekeeping
  - Initialise
  - Clear the screen
  - Turn on backlight
- To print messages
  - Position the cursor
  - Print the message

```
// LCD I2C library
#include <LiquidCrystal_I2C.h>
// set the LCD address to 0x3F, 16 chars
// and 2 line display
LiquidCrystal_I2C lcd(0x3F,16,2);

void setup() {
  lcd.init();
  lcd.clear();
  lcd.backlight();      // Make sure backlight is on

  // Print a message on both lines of the LCD.
  // Set cursor to character 2 on line 0
  lcd.setCursor(2,0);
  lcd.print("Hello world!");
  //Move cursor to character 2 on line 1
  lcd.setCursor(2,1);
  lcd.print("LCD Tutorial");
}

void loop() {
  // write your update code here
}
```

Tube 7-Segment Display TM1637

- 7-segment displays common in projects
- Having multiple 7-segments require conversion and multiplexing
- TM1637 provides 4-digit 7-segment display with only 2 digital I/O lines.
- Library: TM1637-master by Avishay Orpaz.
- Tutorial:
  - Last Minute Engineers
  - Makerguides



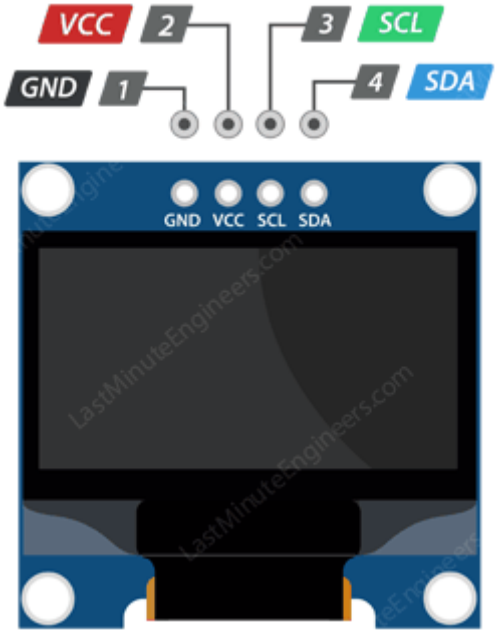
TM1637 Module Pinout

Last Minute ENGINEERS.com

Other versions have 6~8 digits
Simplifies circuitry

Oled Displays SSD1306

- Very small, graphic displays (30x20mm)
- Getting popular, for IOT projects for information display.
- Resolution: 128x128, 128x64, 128x32 pixels
- Uses I2C interface
- Library: SSD1306 by Adafruit
- Tutorials:
  - Last Minute Engineers
  - Adafruit



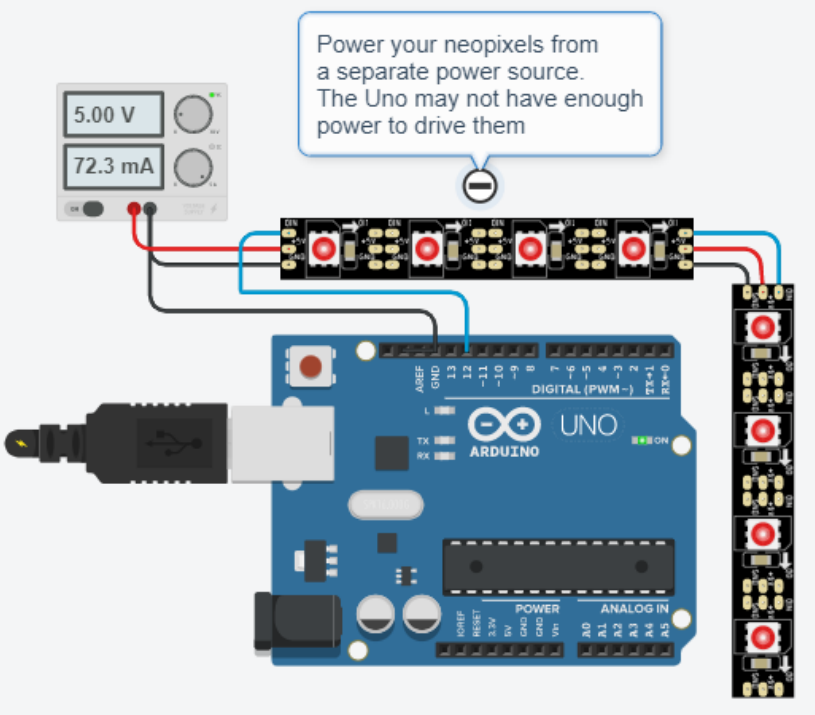
NeoPixels

- Individually addressable LEDs housed on a strip and controlled by a single digital I/O pin from the microcontroller.
- Can control
  - Which LED to turn ON/OFF
  - Intensity of light
  - Color (If RGB)
- Good for lighting, ambient light control, effects
- Library: Adafruit Neopixel
- Tutorial:
  - Adafruit

Neopixels require quite a lot of current. Always supply the neopixels from a separate power source, not from the microcontroller.

Neopixels - demo

- Always power up neopixels using a separate power source for the extra current required.
- You can join strips of neopixels together.
- Adafruit Library comes with a number of very useful methods:
  - Color(r, g, b) returns a 32bit color
  - fill() fills neopixels with same color
  - setColor() sets individual neopixels
  - setBrightness() controls intensity
  - clear() blanks out
  - show() must be called before the neoplxels can display



Uno Neopixel Strip

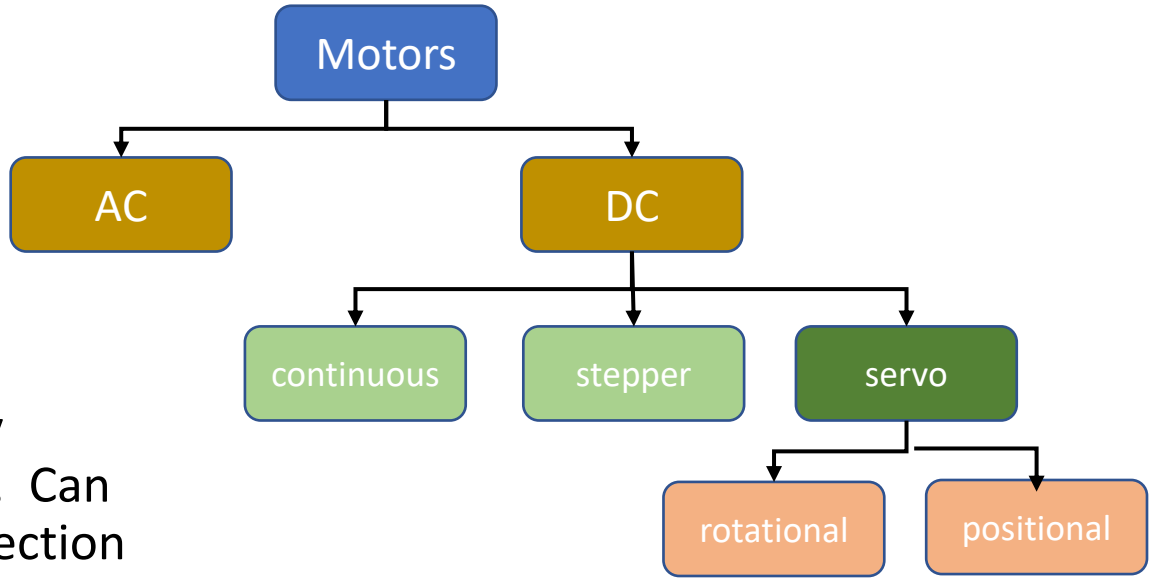
Neopixels – demo code

```ino
// using Adafruit Neopixel library
#include <Adafruit_NeoPixel.h>
const int PIN = 12;     // digitalIO pin
const int NUMPIX = 8;   // num of neopix
// create a neopixel object
Adafruit_NeoPixel np = Adafruit_NeoPixel(
        NUMPIX, PIN, NEO_GRB + NEO_KHZ800);

void setup()
{
  Serial.begin(9600);
  np.setBrightness(128);
}

void loop()
{
  // flash all, R->G->B
  unsigned long c = np.Color(255,0,0);
  flash(c);
  c = np.Color(0,255,0);
  flash(c);
  c = np.Color(0,0,255);
  flash(c);
  // scroll random lights
  for (int i=0; i<10; ++i)
    runLights();
}
```

```ino
void runLights()
{
  const int DLY = 100;
  int r, g, b;
  // obtain a random color
  r = random(256); g = random(256); b = random(256);
  unsigned long c = np.Color(r,g,b);
  for (int p=0; p < NUMPIX; ++p)
  {
    np.setPixelColor(p, c);
    np.show();
    delay(DLY);
  }
}

void flash(unsigned long c)
{
  np.fill(c, 0, NUMPIX);
  np.show();
  delay(500);
  // clear
  np.clear();
  np.show();
  delay(500);
}
```

uint32_t = unsigned long = 32bit
uint16_t = unsigned int = 16 bit

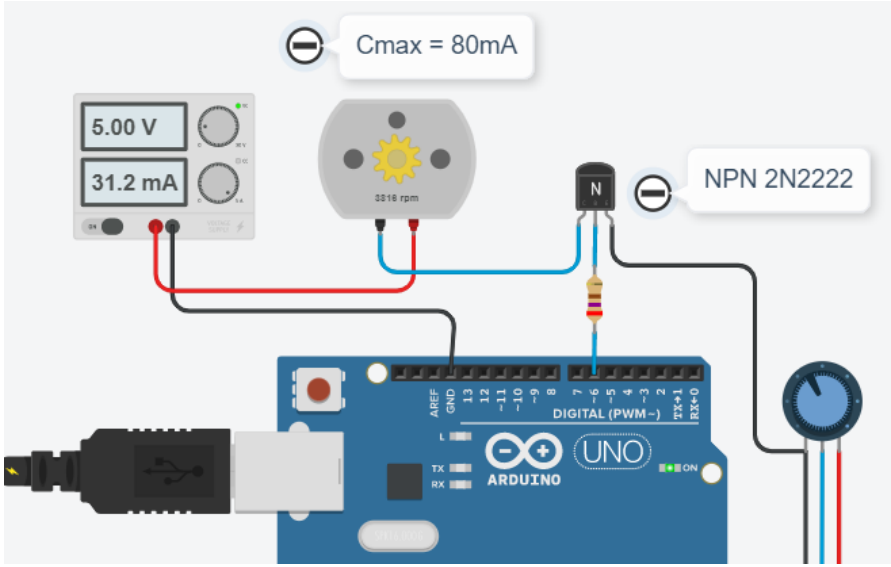
Motors



- DC Motors require a DC current to work.
- DC motors are usually continuously rotating. Can control speed and direction
- Stepper motors can move in ‘steps’. Usually done using a controller module, or by changing phase activation.
- Servo motors can be controlled to move, stop/hold, change direction using pulses.
- Motors consume a lot of current, hence requires an interface circuit. Should not be driven from a digital IO pin.

DC Motor control

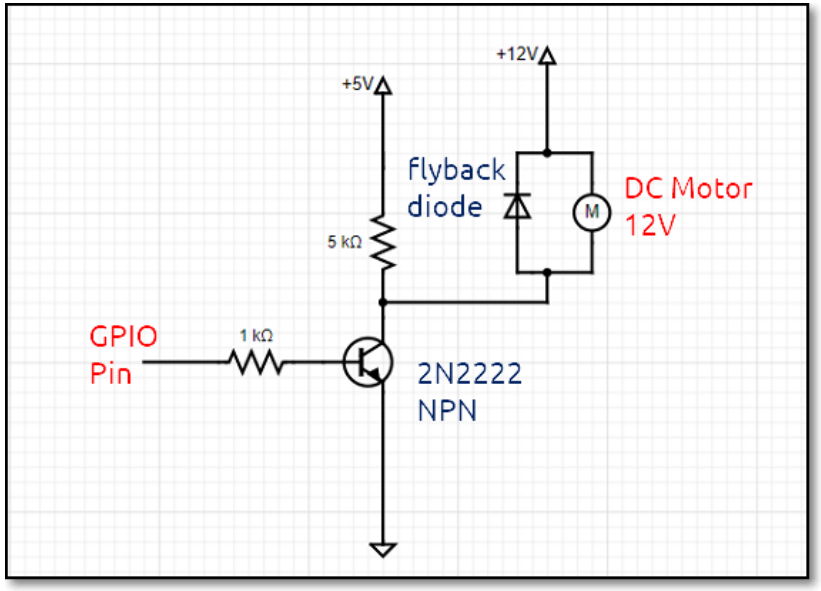
- Use a digital IO pin to control a transistor (Q). Transistor acts like a “tap”
- Qon allows current to flow, which makes motor move.
- Qoff turns off the current.
- Using PWM to control the On/OFF sequence allows control in motor speed.
- PWM at low speeds do not provide much torque.
- This method only allows one direction of motor.



PWM Motor Control Simulation

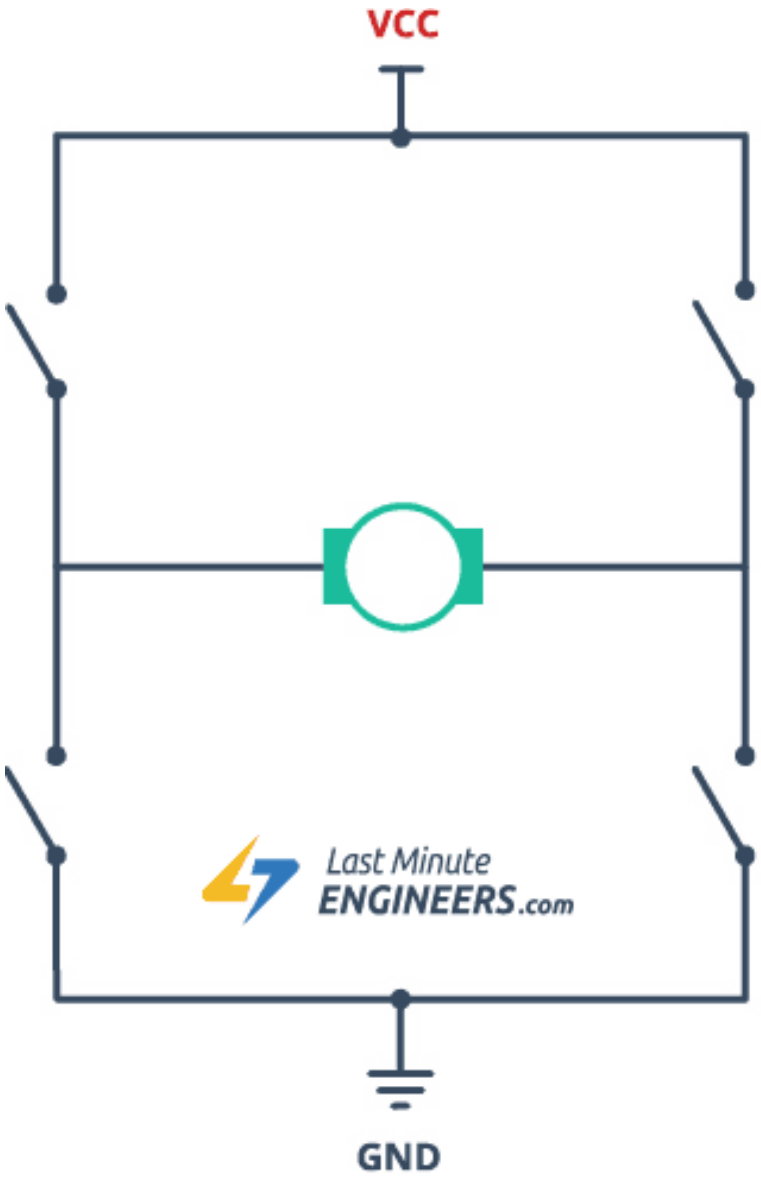
Motor Control with PWM

- Many circuits available
  - Transistor
  - MOSFET
- Transistor acts as a “control tap/valve” for current. Used in the saturation mode.
- Problems:
  - Single direction only
  - Need to worry about resistor values
  - Small duty cycles result in lower torque.



H-bridge

- Allows PWM motor control in both directions.
- H-bridge is an IC made up of transistors or MOSFETs
- Most common method of controlling DC motors
- References:
  - Dronebot Workshop
  - LastMinuteEngineers



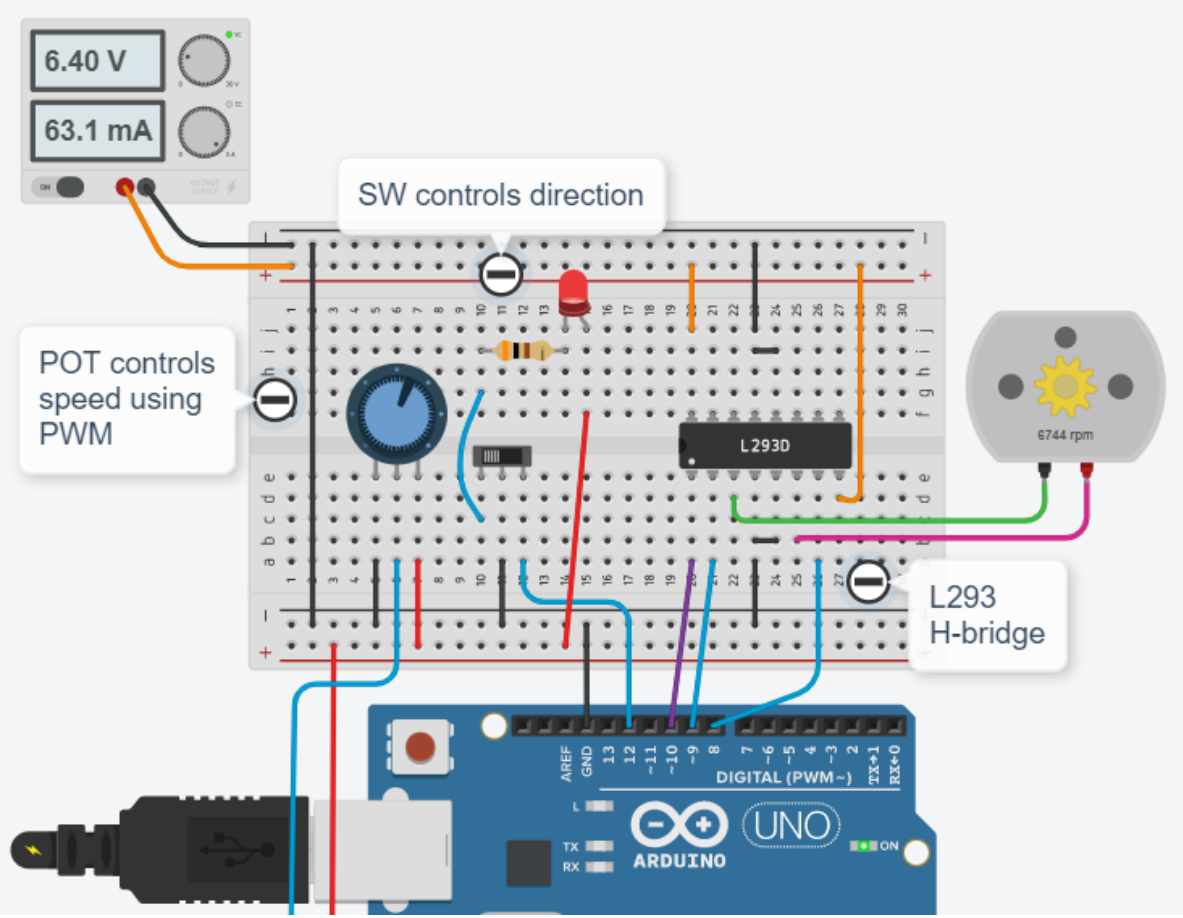
H-bridge connections

- Transistor driven, combined voltage drop is 1.4V.
- If motor is 9V, you will need minimum 9+1.4 = 10.4V.
- Can drive 2 DC motors
- Enable line controls speed of motors if PWM is applied.
- L298 can be supplied separately from the motor voltages

Control	Purpose
IN1	Input 1 for Motor A
IN2	Input 2 for Motor A
EN1	Enable line for Motor A

IN1	IN2	Direction
0	0	Motor OFF
5V	0	Forward
0	5V	Reverse
5V	5V	Not used

H-bridge Simulation



[H-bridge L293]

H-bridge (L293) control of a dc motor.
Use SW to control direction of motor (IN1, IN2)
Use POT to control speed of the motor (EN1)

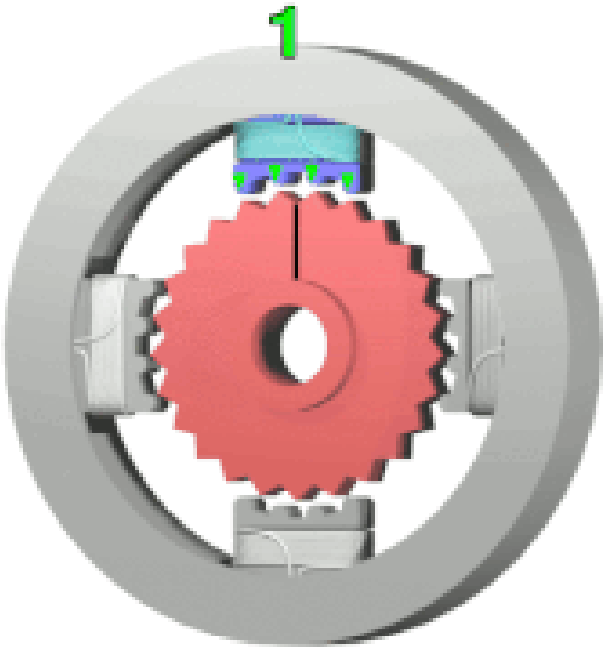
H-bridge code

```
5   const int IN1 = 9;
6   const int IN2 = 8;
7   const int EN1 = 10;
8   const int SW = 12;
9   const int POT = A0;
10  int speed = 0;
11
12  void setup()
13  {
14    pinMode(IN1, OUTPUT);
15    pinMode(IN2, OUTPUT);
16    // pinMode(EN1, OUTPUT);
17    pinMode(SW, INPUT_PULLUP);
18  }
19
20  void loop()
21  {
22    int ain = analogRead(POT);
23    int newspeed = map(ain, 0, 1023, 0, 255);
24    if (newspeed != speed){
25      // change the speed
26      speed = newspeed;
27    }
```

```
28    // set direction
29    if (digitalRead(SW) == HIGH)
30    {
31      // forward
32      digitalWrite(IN1, LOW);
33      digitalWrite(IN2, HIGH);
34    }
35    else
36    {
37      // reverse
38      digitalWrite(IN1, HIGH);
39      digitalWrite(IN2, LOW);
40    }
41    // set the speed using PWM
42    analogWrite(EN1, speed);
43  }
44
```

Stepper Motors

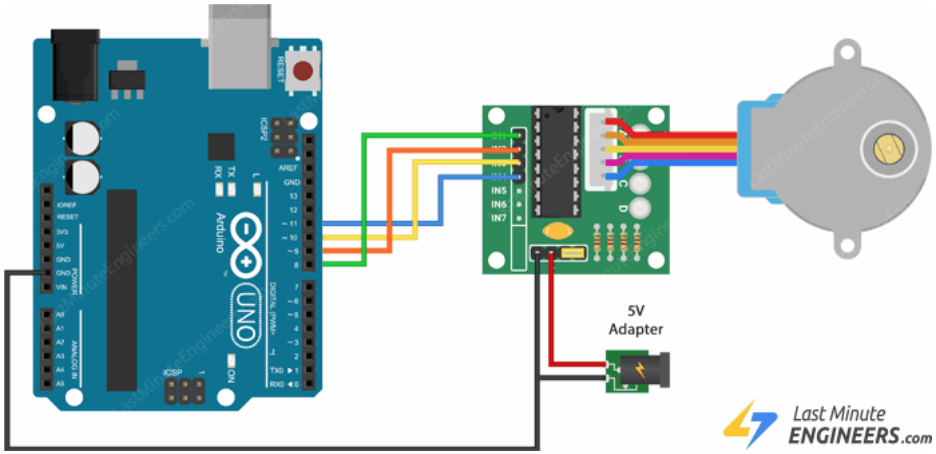
- A motor that can move in steps.
- Each step is a division (in degrees) of a full rotation.
- Done by activation motor coils.
- Stepper Motors are good
  - For exact positional control
  - Movement at specific speeds
  - High torque (even at rest)
- Have Stepper control modules

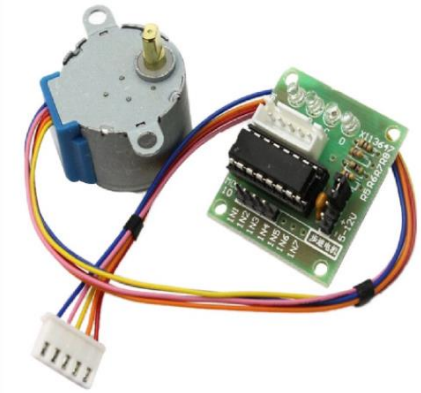


- References:
  - Wikipedia: Stepper Motor
  - Dronebot Workshop
  - LastMinuteEngineers

Simple Stepper Motor: 28BYJ-48

- Common hobby motor with controller board for many projects.
- Able to work with 5V supply (external)
- Can be controlled manually or with a library
- Library: Arduino Stepper
- References:
  - LastMinuteEngineers
  - Dronebot workshop (has manual code and library examples)





Stepper Motor Code

```
2
3   //Includes the Arduino Stepper Library
4   #include <Stepper.h>
5   // Defines the number of steps per rotation
6   const int stepsPerRevolution = 2038;
7   // Creates an instance of stepper class
8   // Pins entered in sequence
9   // IN1-IN3-IN2-IN4 for proper step sequence
10  Stepper myStepper = Stepper(stepsPerRevolution,
11                              8, 10, 9, 11);
12
13  void setup() {
14      // Nothing to do
15  }
16
17  void loop() {
18      // Rotate CW slowly
19      myStepper.setSpeed(100);
20      myStepper.step(stepsPerRevolution);
21      delay(1000);
22
23      // Rotate CCW quickly
24      myStepper.setSpeed(700);
25      myStepper.step(-stepsPerRevolution);
26      delay(1000);
27  }
28
```

- Uses stepper motor library
- There is a min and max speed the motor can achieve.

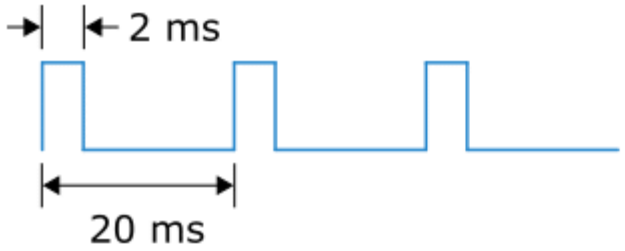
Servo Motors

- Has an internal control circuit with geared motors
- Controlled using PWM
- Pulse width determine
  - Position of servo
  - Direction of rotation
- Library: Arduino Servo
- Reference:
  - LastMinuteEngineers
  - Dronebot Workshop

- Positional Servo (FS90)
  - 3.0~5.0V, 120^{o} angle
  - Servo pulse 900~1200uS determines position
  - Maintains control position
- Continuous Rotation (FS90R)
  - 3.0~5.0V, 360^{o} rotation
  - Servo pulse 900~1200uS controls direction of rotation or speed
  - For small robots or moving objects

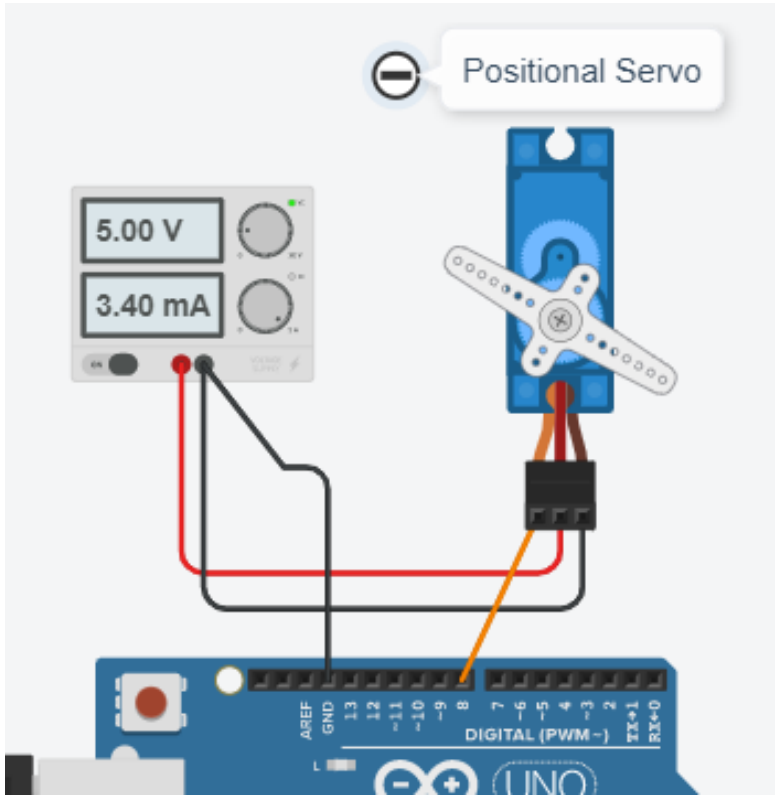
How Servos work

- Controlled by pulse width
  - 0 = 1ms (min)
  - 90 = 1.5ms (neutral)
  - 180 = 2ms (max)
- Varys slightly with motor
- Continuous motor
  - 0 = full speed direction 1
  - 90 = stop
  - 180 = full speed reverse direction 1
- Must always use external power.



Positional Servo

```
3  // Arduino's Servo Library
4  #include <Servo.h>
5
6  const int SERVO = 8;
7  const int MAXDEG = 120;
8  // create a servo object
9  Servo myServo;
10
11 void setup()
12 {
13   myServo.attach(SERVO);
14   Serial.begin(9600);
15   Serial.println("Starting");
16 }
17
18 void loop()
19 {
20   // swing servo full arc
21   for(int i=0; i<MAXDEG; i=i+10)
22   {
23     myServo.write(i);
24     Serial.println(i);
25     delay(1000);
26   }
27 }
28
```

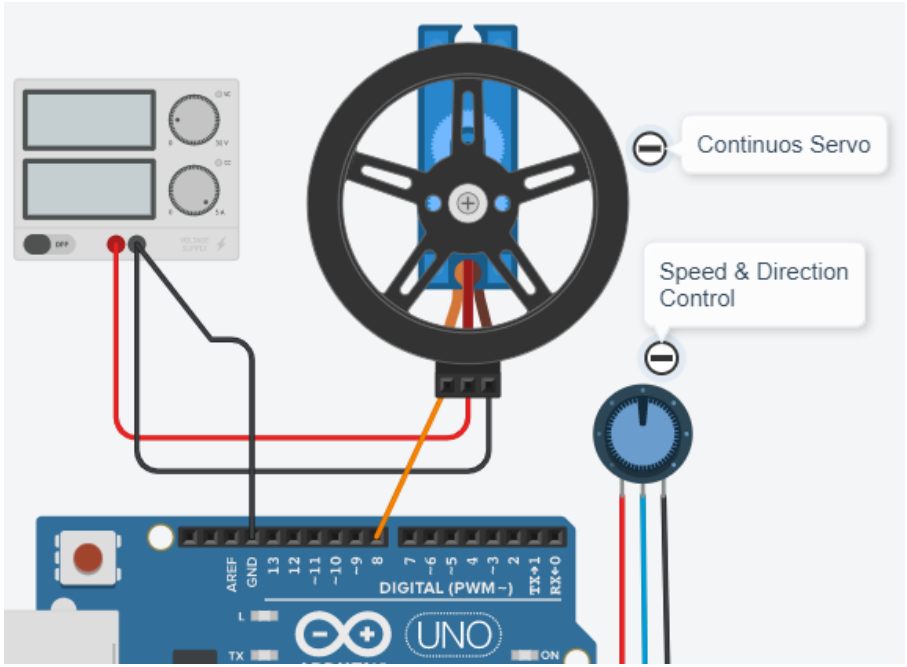


Positional Servo

Use any digital I/O pin

Assume max angle = 120°

Continuous Servo - code



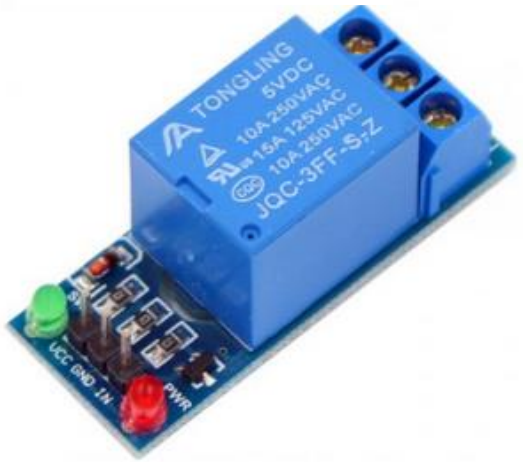
[Uno Continuous Rotational Servo]

```
Serial Monitor

Starting
Pot= 511        Deg= 89
Pot= 552        Deg= 97
Pot= 593        Deg= 104
Pot= 614        Deg= 108
Pot= 573        Deg= 100
Pot= 552        Deg= 97
Pot= 471        Deg= 82
```

```
// Arduino's Servo Library
#include <Servo.h>

const int POT = A0;
const int SERVO = 8;
// create a servo object
Servo myServo;

int oldspeed = 0;

void setup()
{
  myServo.attach(SERVO);
  Serial.begin(9600);
  Serial.println("Starting");
}

void loop()
{
  // read potentiometer
  int speed = analogRead(A0);
  if (speed != oldspeed)
  {
    // new speed
    int deg = map(speed, 0, 1023, 0,180);
    myServo.write(deg);
    Serial.print("Pot= "); Serial.print(speed);
    Serial.print("\tDeg= ");Serial.println(deg);
    oldspeed = speed;
  }
}
```

Controlling high powered applications

- Use transistors or Mosfets (Ref: Dronebot workshop – Transistors & MOSFETS)
- Use relays/relay modules (Ref: Dronebot workshop – Relays)
- Control = Turn ON/OFF high power devices.
- Be **very careful** when working with higher voltages!



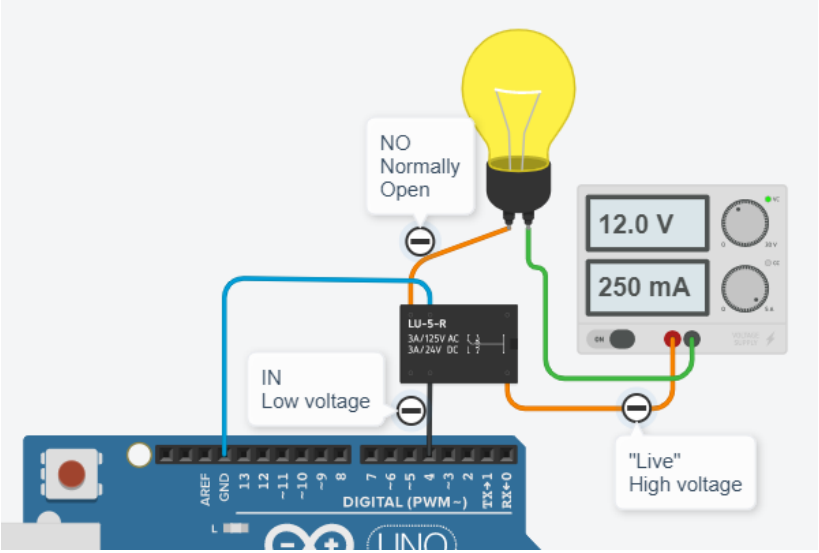
Typical Relay Module

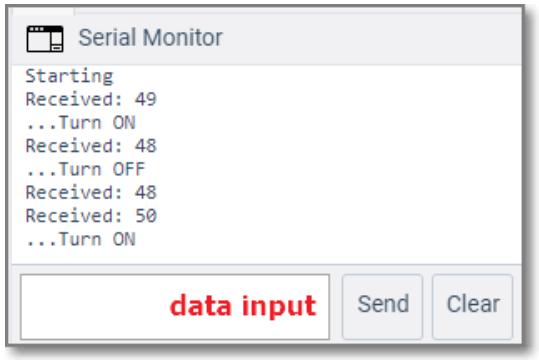
- Inputs 5V DC
- Outputs 10A 250VAC
- Relay module must be powered up on the input side.
- One digital I/O pin (IN) for control in switching
- Split the “Live” end on high voltage side for switch
- Ref:
  - LastminuteEngineers
  - Dronebotworkshop

IN	Outputs NO-COM
0 V	No connection
5 V	Conducts

NC (Normally Closed) works opposite to NO (Normally Open)

Relay connection





[Control High Power devices using a relay]

```
// relay IN pin
const int RIN = 4;
int state = 0;

void setup()
{
  pinMode(RIN, OUTPUT);
  digitalWrite(RIN, state);
  Serial.begin(9600);
  Serial.println("Starting");
}

void loop()
{
  if (Serial.available()){
    // data available
    int data = Serial.read();
    Serial.print("Received: ");
    Serial.println(data);
    if (data != '0'){
      // request to turn ON
      if (state == LOW){
        digitalWrite(RIN, HIGH);
        state = HIGH;
        Serial.println("...Turn ON");
      }
    }
    else {
      // request to turn ODD
      if (state == HIGH){
        digitalWrite(RIN, LOW);
        state = LOW;
        Serial.println("...Turn OFF");
      }
    }
  }
}
```

EP1000

Actuators

End