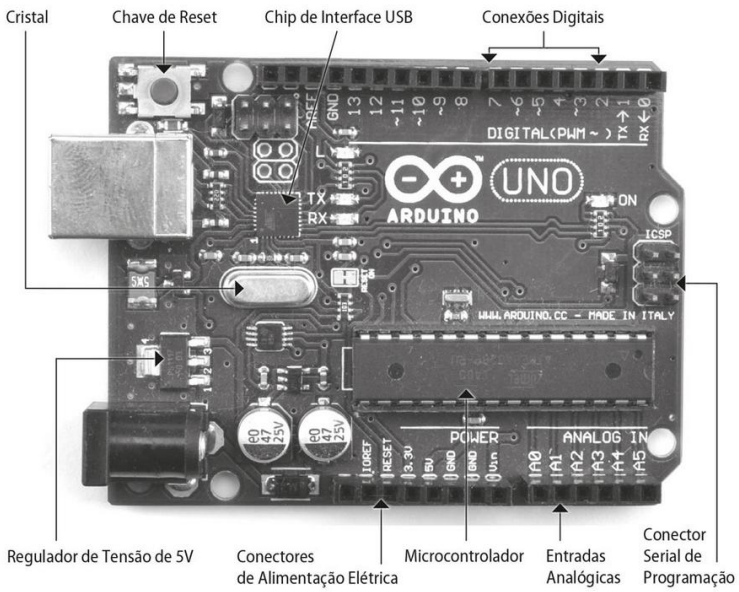
Oficina de Arduino

Professores: Thiago José da Luz

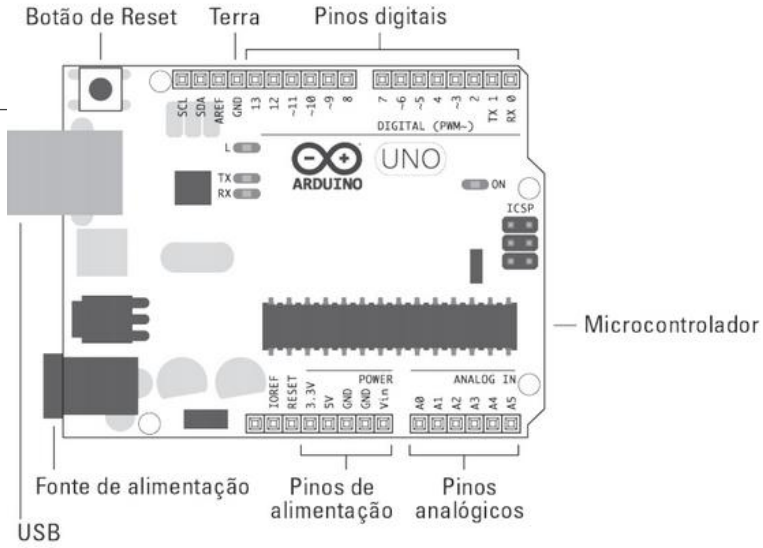
15/05

thiagojluz@gmail.com

Placa do Arduino



Placa do Arduino



Arduino

- A placa Arduino Uno apresenta 14 pinos que podem ser utilizados como entradas ou saídas digitais (pinos 0 a 13), e os pinos 3, 5, 6, 9, 10 e 11 podem ser utilizados para gerar um conjunto de valores inteiros entre 0 e 1.023 pela técnica de Pulse Width Modulation (PWM)

Arduino

- Os pinos A0 a A5 correspondem às entradas analógicas, enquanto os 3,3 V, 5 V e GND (Terra) permitem alimentar os componentes dos circuitos conectados ao Arduino.
- Possui um microprocessador ATmega328, com uma memória RAM de 2 KB, memória Flash de 32 KB e velocidade de clock de 16 MHz.

Fonte de alimentação

- Diretamente abaixo do conector USB, está o regulador de tensão de 5 volts (5V). Ele recebe qualquer tensão (entre 7V e 12V) fornecida pelo conector de alimentação CC e a converte em uma tensão constante de 5V.

Fonte de alimentação

- A alimentação elétrica do Arduino feita pelo conector CC (2,1 mm) é útil quando o Arduino trabalha com baterias ou uma fonte de alimentação CC.
- O Arduino Uno também pode ser alimentado por meio da porta USB, que também é usada para programar o Arduino.

Fonte de alimentação

- A Vin, que significa entrada da tensão, pode ser usada para fornecer uma tensão (V) igual à fornecida pelo conector jack externo (por exemplo, 12V). Você também pode usar esse pino para alimentar o Arduino através de outra fonte.

Fonte de alimentação

- Caso conecte uma fonte de alimentação oposta (centro negativo), ela é conhecida por apresentar uma polaridade invertida. Os componentes do Arduino Uno resistirão às suas tentativas de enviar a tensão do jeito errado ao redor da placa. Porém, eles podem derreter no processo de proteger sua placa, dependendo da quantidade de energia que está enviando e quanto tempo leva para que você sinta o cheiro de queimado!

Fonte de alimentação

- Se inverter a polaridade ao usar os pinos Vin, 5V ou 3,3V, você burlará essa proteção e destruirá quase instantaneamente várias partes da sua placa e do seu chip ATmega328P.

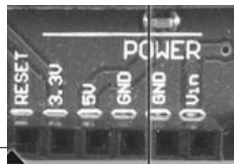
Fonte de alimentação

- A tensão recomendada para a placa Uno é de 7 a 12 volts. Caso forneça pouca energia, talvez sua placa não funcione adequadamente. Ou, se você fornecer muita energia, sua placa poderá superaquecer e ficar danificada.

Reset

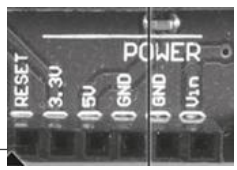
- O primeiro pino (Reset), faz a mesma coisa que o botão de Reset do Arduino. De forma semelhante ao que ocorre quando reiniciamos um computador PC, se ativarmos o pino de Reset do Arduino, o microcontrolador será inicializado começando a executar seus programas desde o ponto de partida inicial. Para inicializar o microcontrolador por meio do pino de Reset, você deve manter esse pino momentaneamente em nível baixo (conectando-o a 0V ou GND).

Pinos de tensão



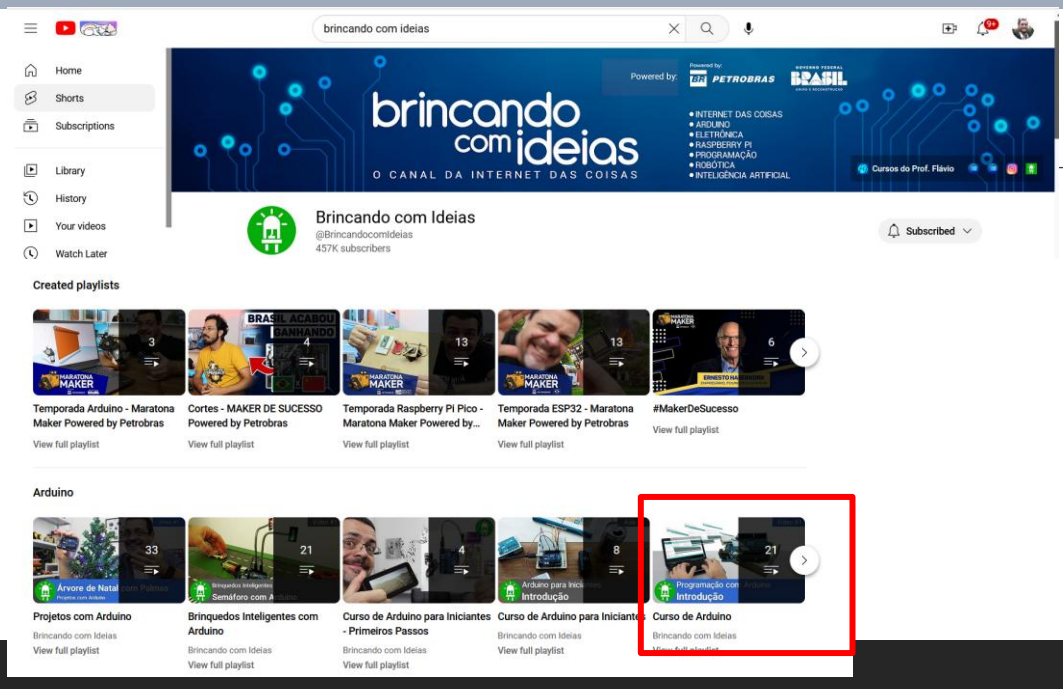
- Os demais pinos desta seção simplesmente fornecem diversas tensões (3,3V, 5V, GND e Vin), conforme estão indicadas na placa.

Pinos de tensão

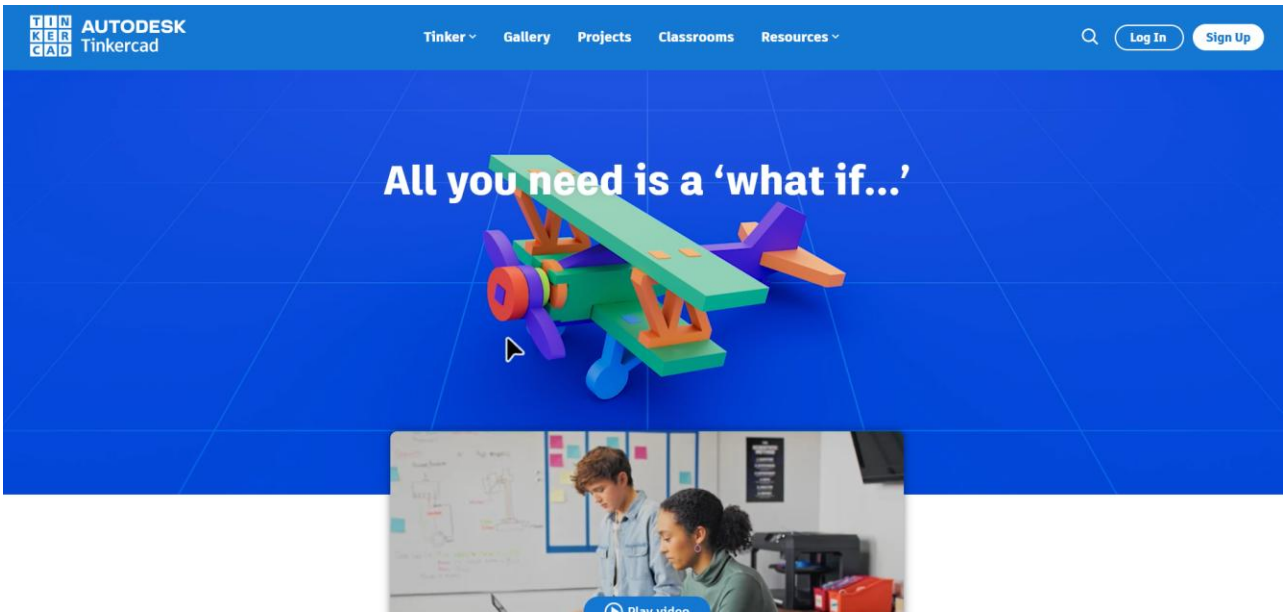


- O termo GND (ground ou terra) significa simplesmente zero volts. É a tensão que serve de referência a todas as demais tensões da placa.
- O GNDs são os pinos terra (ou de referência), essenciais para completar os circuitos. Há também um terceiro terra, ao lado do pino 13. Todos esses pinos estão conectados e compartilham o mesmo terra.

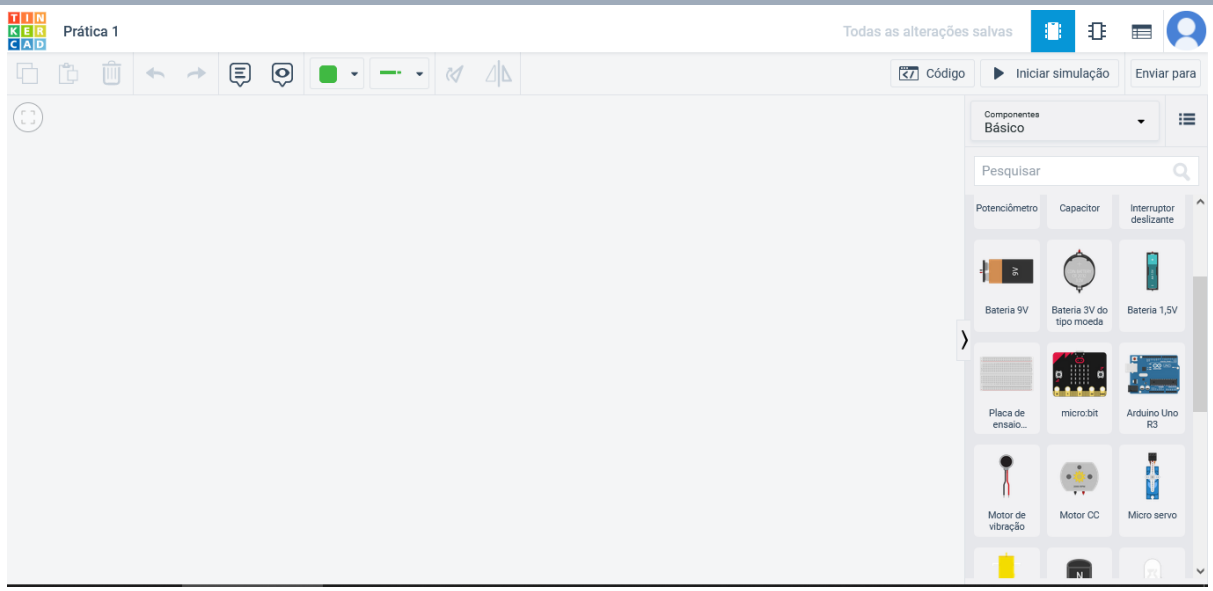
Oficina Arduino 15/



Oficina Arduino 16/



Oficina Arduino 17/



Oficina Arduino 18/

18

Cuidados

- Corrente máxima por pino: 20 mA, mas suporta 40 mA
- Total da placa: 200 mA
- Pino 3.3V: 50 mA
- Pino 5V: 500 mA (limitado pela USB)

Cuidados (Tensão Reversa)

O principal perigo da tensão reversa (também conhecida como surto de tensão, Contra-Eletromotriz) ao acionar bobinas com Arduino é a **queima imediata ou gradual dos componentes eletrônicos**.

Quando você corta a energia de um componente indutivo (bobina), como um **relé, solenoide ou motor**, a energia magnética armazenada colapsa e gera uma **tensão muito alta e invertida** na bobina, que pode ser até 10 vezes maior que a tensão de operação

sketch

- Um sketch é o nome que o Arduíno usa para um programa. É a unidade de código que é carregada e executada em uma placa Arduíno.

setup()

Um sketch do Arduino deve ter uma função setup() e uma função loop(), do contrário, não funcionará.

A função setup() é executada somente uma vez no início do programa, e é nela que você emitirá instruções gerais para preparar o programa antes que o loop principal seja executado, como a **definição dos modos dos pinos**, das **taxas de transmissão serial etc**.

setup()

```
void setup()
```

Isso diz ao compilador que sua função é chamada setup, que ela não retorna nenhum dado (void) e que você não passa nenhum parâmetro a ela (parênteses vazios).

setup()

Todo o código dentro da função está contido entre chaves. Um símbolo { inicia o bloco de código, e um símbolo } termina o bloco. Tudo que existir entre esses dois símbolos, no código, fará parte da função.

loop()

A função loop() é a função principal do programa e executa continuamente enquanto o Arduino estiver ligado. Todas as declarações dentro da função loop() (dentro das chaves) são executadas uma de cada vez, passo a passo, até que se alcance o fim da função; nesse ponto, o loop reinicia desde o princípio e assim infinitamente, ou até que o Arduino seja desligado ou o botão Reset pressionado.

pinMode(pino, modo)

pinMode diz ao Arduino que você deseja definir o modo de um de seus pinos como Saída (Output), e não Entrada (Input).

Dentro dos parênteses, você coloca o número do pino e o modo (OUTPUT ou INPUT)

pinMode(pino, modo)

O número de seu pino pode ser definido dentro de uma variável, por exemplo ledPin.

pinMode(pino, modo)

- **pino:** o número do pino do Arduino no qual se quer configurar o modo
- **modo:** o modo do pino. Este pode ser **INPUT**, **OUTPUT** ou **INPUT_PULLUP**; que correspondem respectivamente a entrada, saída e entrada com pull-up ativado.

digitalWrite(pino, HIGH)

Ele escreve um valor HIGH ou LOW para o pino dentro da instrução. Quando você define um pino como HIGH, está enviando 5 volts para ele. Quando define como LOW, o pino se torna 0 volt, ou terra.

digitalWrite(pino, HIGH)

Sintaxe

```
digitalWrite(pino, valor)
```

- pino: o número do pino do Arduino
- valor: HIGH ou LOW

delay(tempo)

Essa instrução simplesmente diz ao Arduino para esperar um tempo em milissegundos antes de executar a instrução seguinte

Variáveis

- Variáveis inteiras: int (integer ou inteiros) são o tipo de dados primário para armazenamento de números.

Variáveis

- **Sintaxe**

int var. = val.;

Nome: var.;

Valor: val.;

Tipo: int

Variáveis

- Você pode alterar o valor de uma variável usando uma atribuição (indicada por um sinal de igual). Por exemplo:

```
pin = 12;
```

- Isso mudará o valor da variável para 12. Observe que não especificamos o tipo da variável: ela não é alterada pela atribuição. Ou seja, o nome da variável está permanentemente associado a um tipo; apenas seu valor muda.

Variáveis

- Também podemos criar uma variável sem atribuir um valor:

```
int pin;
```

Revisando

void setup()

void loop()

pinMode(pino, modo); OUTPUT ou INPUT

digitalWrite(pino, HIGH ou LOW)

delay()

Prática 1 (Olá Mundo)

- Criar o código “olá mundo”, ou seja, ligar e apagar o LED da placa do Arduino que está conectado internamente no pino 13

Prática 1 (Olá Mundo)

- Agora dimensione o resistor para ligar um LED vermelho na saída digital 13
- Dados
- Tensão: 5V
- Tensão do LED: 2 V
- Corrente máxima: 20 mA

Entrada digital

- A entrada digital é utilizada para enviarmos ao Arduino um valor digital 0 (LOW) ou 1 (HIGH). Qualquer um dos 13 pinos digitais pode ser programado como entrada digital.
- Se a tensão da entrada for menor que 2,5 volts (metade de 5 volts), então ela será 0 (desligada) e, se for maior que 2,5 volts, ela será 1 (ligada).

digitalRead()

- Lê o valor de um pino digital especificado, que pode ser **HIGH** ou **LOW**.

Sintaxe

```
digitalRead(pino)
```

- **pino**: o número do pino digital do Arduino que você quiser verificar

digitalRead()

- **Retorna**: HIGH ou LOW

Código de Exemplo

Aciona o pino 13 para o mesmo valor que o pino 7, declarado como entrada.

```
int ledPin = 13;  // LED conectado ao pino digital 13
int inPin = 7;    // botão conectado ao pino digital 7
int val = 0;      // variável para guardar o valor lido

void setup() {
  pinMode(ledPin, OUTPUT);    // configura o pino digital 13 como saída
  pinMode(inPin, INPUT);      // configura o pino digital 7 como entrada
}

void loop() {
  val = digitalRead(inPin);   // lê o pino de entrada
  digitalWrite(ledPin, val);  // aciona o LED com o valor lido do botão
}
```


digitalRead()

Se o pino não está conectado a nada, digitalRead() pode retornar tanto HIGH como LOW (e isso pode mudar aleatoriamente).

Os pinos de entrada analógica podem ser também usados como pinos digitais, referidos como A0, A1, etc. As exceções são os pinos A6 e A7 das placas Arduino Nano, Pro Mini, e Mini, que podem ser usadas apenas como entradas analógicas.

INPUT

Por padrão, todos os pinos do Arduino já são declarados como input sem ser necessário pinMode()

Os pinos configurados dessa maneira estão em um estado de alta impedância.

Os pinos INPUT precisam de uma corrente muito pequena, tendo um entrada com resistência de 100 MΩ

INPUT

Os pinos configurados como pinMode (pin, INPUT) sem nada conectado a eles, ou com fios conectados a eles que não estão conectados a outros circuitos, relatarão mudanças aparentemente aleatórias no estado do pino, captando ruído elétrico do ambiente, ou acoplando capacitivamente o estado de um pino próximo.

Monitor Serial

- Serial é um método de comunicação entre um periférico e um computador.
- Neste caso, é a comunicação serial via porta USB (Porta Serial Universal). Os dados são enviados um byte de cada vez na ordem em que estão escritos. Ao ler sensores com um Arduino, os valores são enviados por meio desta conexão e podem ser monitorados ou interpretados em seu computador.

Serial.begin()

- Define a taxa de dados em bits por segundo para transmissão serial de dados.
- Para se comunicar com o Serial Monitor, certifique-se de usar uma das taxas de transmissão listadas no menu no canto inferior direito da tela.

Monitor Serial

Serial.begin()

Syntax

```
Serial.begin(speed)
Serial.begin(speed, config)
```

speed: em bits por segundo. Tipos de dados permitidos: long.

config: define dados, paridade e bits de parada.

Exemplo

```
void setup() {
    Serial.begin(9600); // opens serial port, sets data rate to 9600 bps
}

void loop() {}
```


Serial.print()

- `Serial.print(78)` gives "78"
- `Serial.print(1.23456)` gives "1.23"
- `Serial.print('N')` gives "N"
- `Serial.print("Hello world.")` gives "Hello world."

Serial.print()

Syntax

```
Serial.print(val)
Serial.print(val, format)
```

Serial.print("\t"); Espaço
Serial.print("\n"); nova linha

Serial.println()

- Imprime dados na porta serial como texto ASCII, seguido de uma nova linha (ASCII 10 ou '\n').
- Este comando assume as mesmas formas de Serial.print().

Serial.println()

Syntax

```
Serial.println(val)
Serial.println(val, format)
```

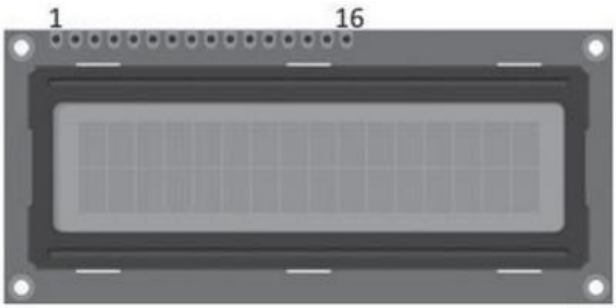

Exemplo

```
void setup()
{
   Serial.begin(9600);
   int a = 2;
   int b = 2;
   int c = a + b;
   Serial.println(c);
}
void loop()
{}
```

Displays LCD alfanuméricos

Displays LCD alfanuméricos

Inicialmente, é importante identificar a função de cada um dos pinos do display.



Displays LCD alfanuméricos

Na tabela a seguir temos a função de cada um dos pinos do display de LCD Modelo ACM 1602K de 16 linhas por duas colunas ou modelos similares.

Pino	Nome	Função
1	Gnd	Alimentação (polo negativo)
2	Vcc	Alimentação 5 V (polo positivo)
3	V0	Ajuste do contraste
4	RS	Habilita ou desabilita a seleção de registrador
5	R/W	Leitura / Escrita
6	E	Habilita a escrita no LCD
7	DB0	Dado

Displays LCD alfanuméricos

8	DB1	Dado
9	DB2	Dado
10	DB3	Dado
11	DB4	Dado
12	DB5	Dado
13	DB6	Dado
14	DB7	Dado
15	BL+	Alimentação de 5 V da Backlight (polo positivo)
16	BL-	Alimentação Gnd da Backlight (polo negativo)

Displays LCD alfanuméricos

1 | **Vss:** GND 0V (GND)

2 | **Vdd:** - +5V (Vcc)

3 | **V0:** Ajusta o brilho da tela do LCD (Ligar no potenciomentro)

4 | **RS:** Register Select (Seleção de Registrador). Esse pino é ajustado para 1 ou 0, dependendo se o Arduino está enviando dados de caractere ou de instrução. Uma instrução, por exemplo, pode fazer o cursor piscar. (Arduino)

Displays LCD alfanuméricos

5 | **R/W:** Leitura/Escrita (GND)

6 | **E:** Enable (Habilita). A saída desse pino apresenta um pulso para dizer que os dados dos quatro pinos seguintes estão prontos para serem lidos. (Arduino)

7 – 14 | **DB(0-7):** Dados. Esses quatro pinos são usados para transferir dados. O chip controlador do LCD usado pelo shield pode operar com dados de oito ou quatro bits. (Arduino)

Displays LCD alfanuméricos

15 | BL+: Alimentação BL+ Resistor de 1 k para VCC

16 | BL-: Alimentação BL- GND

Displays LCD alfanuméricos

O IDE do Arduino vem com uma biblioteca para LCD. Isso simplifica bastante o processo de usar um display LCD.

LiquidCrystal.h

Displays LCD alfanuméricos

A biblioteca fornece funções úteis que podem ser usadas por você:

- **clear:** limpa o display, deixando-o sem nenhum texto.
- **setCursor:** define a posição onde aparecerá o próximo caractere que você quer mostrar. A posição é dada por uma coluna e uma linha. A contagem de linhas e colunas começa em 0.
- **print:** escreve uma string nessa posição.

Displays LCD alfanuméricos

home: é o mesmo que setCursor(0,0): ela move o cursor para a posição mais à esquerda da linha de cima.

cursor: exibe o cursor.

noCursor: especifica que o cursor não deve ser exibido.

blink: faz o cursor piscar.

noBlink: faz o cursor parar de piscar.

Displays LCD alfanuméricos

noDisplay: desliga o display sem remover o conteúdo.

display: volta a ligar o display após um noDisplay.

scrollDisplayleft: todo o texto do display é deslocado (scroll) de uma posição de caractere para a esquerda (left).

scrollDisplayright: todo o texto do display é deslocado (scroll) de uma posição de caractere para a direita (right).

Displays LCD alfanuméricos

Autoscroll: ativa um modo no qual, à medida que novos caracteres são acrescentados na posição do cursor, o texto existente é deslocado (scroll) no sentido determinado pela função **leftToRight** (esquerda para direita) ou **rightToLeft** (direita para esquerda).

noAutoscroll: desliga o modo autoscroll.

LiquidCrystal - LiquidCrystal()

Syntax

```
LiquidCrystal(rs, enable, d4, d5, d6, d7)
LiquidCrystal(rs, rw, enable, d4, d5, d6, d7)
LiquidCrystal(rs, enable, d0, d1, d2, d3, d4, d5, d6, d7)
LiquidCrystal(rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7)
```

rs: o número do pino Arduino que está conectado ao pino RS no LCD

rw: o número do pino Arduino que está conectado ao pino RW no LCD (opcional)

LiquidCrystal - LiquidCrystal()

Syntax

```
LiquidCrystal(rs, enable, d4, d5, d6, d7)
LiquidCrystal(rs, rw, enable, d4, d5, d6, d7)
LiquidCrystal(rs, enable, d0, d1, d2, d3, d4, d5, d6, d7)
LiquidCrystal(rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7)
```

enable: o número do pino do Arduino que está conectado ao pino de habilitação no LCD

LiquidCrystal - LiquidCrystal()

Syntax

```
LiquidCrystal(rs, enable, d4, d5, d6, d7)
LiquidCrystal(rs, rw, enable, d4, d5, d6, d7)
LiquidCrystal(rs, enable, d0, d1, d2, d3, d4, d5, d6, d7)
LiquidCrystal(rs, rw, enable, d0, d1, d2, d3, d4, d5, d6, d7)
```

d0, d1, d2, d3, d4, d5, d6, d7: os números dos pinos do Arduino que estão conectados aos pinos de dados correspondentes no LCD. DB0, DB1, DB2 e DB3 são opcionais; se omitido, o LCD será controlado usando apenas as quatro linhas de dados (DB4, DB5, DB6, DB7).

LiquidCrystal - LiquidCrystal()

Example

```
#include <LiquidCrystal.h>

LiquidCrystal lcd(12, 11, 10, 5, 4, 3, 2);

void setup()
{
  lcd.begin(16,1);
  lcd.print("hello, world!");
}

void loop() {}
```

lcd.begin(16,2);

LiquidCrystal - print()

Imprime o texto no LCD

Syntax

```
lcd.print(data)
lcd.print(data, BASE)
```

lcd: uma variável do tipo LiquidCrystal

data: os dados a serem impressos (char, byte, int, long ou string)

BASE (opcional): a base na qual imprimir números: BIN para binário (base 2), DEC para decimal (base 10), OCT para octal (base 8), HEX para hexadecimal (base 16).

LiquidCrystal - setCursor()

Posicione o cursor do LCD; ou seja, defina o local no qual o texto subsequente gravado no LCD será exibido.

Syntax

```
lcd.setCursor(col, row)
```

lcd: uma variável do tipo LiquidCrystal

col: a coluna na qual posicionar o cursor (com 0 sendo a primeira coluna)

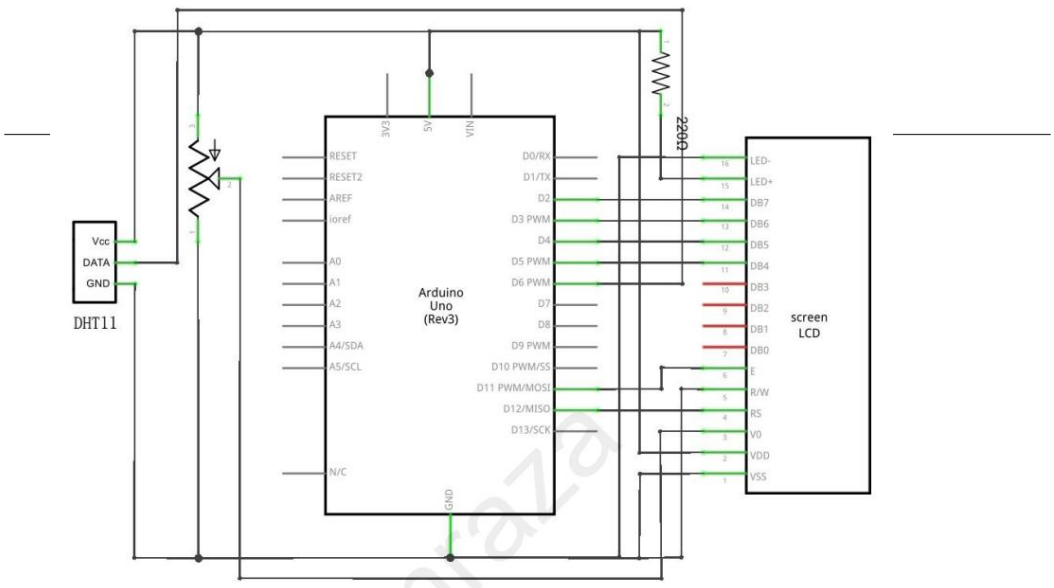
row: a linha na qual posicionar o cursor (com 0 sendo a primeira linha)

LiquidCrystal - clear()

Limpa a tela LCD e posiciona o cursor no canto superior esquerdo.

Syntax

```
lcd.clear()
```

Módulo I2C

Vimos que para conectar um display LCD 16×2 ou 20×4 ao Arduino, precisamos de pelo menos 6 fios (ou jumpers) para conexão.

Em placas com um número menor de portas, como o Arduino Uno, isso significa sacrificar algumas portas que poderiam ser utilizadas para ligação de outros componentes, como sensores ou motores.

Módulo I2C

O módulo que pode ser utilizado para contornar esse problema é o módulo I2C para display LCD

O protocolo I2C é largamente utilizado no desenvolvimento embarcado para realizar comunicações entre plataformas iguais ou diferentes utilizando apenas duas linhas seriais, como a de dados (SDA) e a linha de clock (SCL).

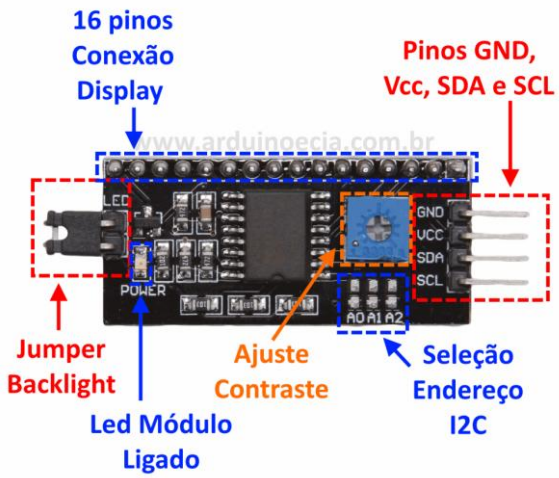
Módulo I2C

Com esse módulo, conseguimos controlar um display LCD, seja ele 16×2 ou 20×4, utilizando apenas dois pinos do Arduino: o **pino analógico 4 (SDA)** e o **pino analógico 5 (SCL)**, que formam a interface de comunicação I2C.

Módulo I2C

Estrutura do módulo I2C

Na lateral esquerda do módulo temos 4 pinos, sendo que dois são para alimentação (**Vcc** e **GND**), e os outros dois são da interface **I2C (SDA e SCL)**.



Módulo I2C

Jumper do Backlight: Jumper responsável por permanecer com o Backlight aceso quando conectado ao módulo ou apaga-lo quando desconectado.

Pinos para Conexão junto ao Display: Pinos que devem ser soldados na parte de trás do Display para garantir a comunicação e interpretação dos sinais.

Módulo I2C

Trimpot para ajuste do Backlight: Trimpot responsável por regular o contraste dos caracteres do display, tornando-os mais amostra ou menos.

Pinos para Conexão junto ao Arduino: Os pinos que constituem o Módulo Adaptador I2C são: GND (Negativo), VCC (Positivo), SDA (Pino de Comunicação I2C), SCL (Pino de Comunicação I2C).

Módulo I2C

Seletor para Código I2C: Através de uma combinação certa, possibilita diferentes endereços para comunicação I2C, possibilitando a conexão de mais de um equipamento nos mesmos pinos.

Módulo I2C

O potenciômetro da placa serve para ajuste do contraste do display. O jumper na lateral oposta permite que a luz de fundo (backlight) seja controlada pelo programa ou permaneça apagada.

Módulo I2C

Conforme a tabela abaixo, podemos verificar os sequenciais que definem cada um dos endereços de comunicação, ou seja, por possuir 8 possibilidades de endereço, podem ser conectados 8 diferentes display nas mesmas duas portas SDA e SCL.

Módulo I2C

Endereço	A0	A1	A2
0x20	0	0	0
0x21	1	0	0
0x22	0	1	0
0x23	1	1	0
0x24	0	0	1
0x25	1	0	1
0x26	0	1	1
0x27	1	1	1

Módulo I2C

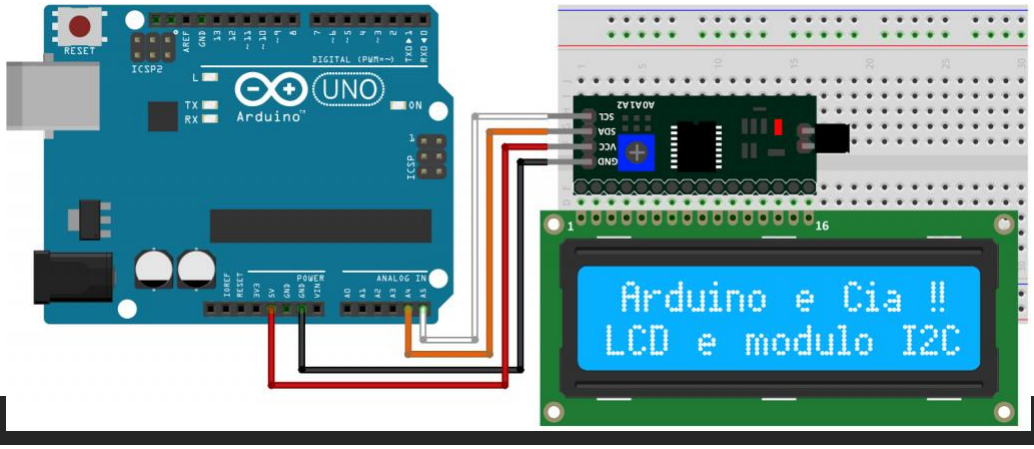
Por padrão, comumente o Módulo Adaptador I2C vem configurado com o endereço 0x27, mas você pode alterá-los sempre que necessário utilizando os pinos A0, A1 e A2

Módulo I2C

Para configurar o endereço de comunicação do módulo, basta utilizar a sequência de bits representada através da tabela vista anteriormente como base para identificar quais os terminais A0, A1 e A2 devem ser interligados

Módulo I2C

Ligação do módulo I2C com Arduino e display LCD 16×2



Módulo I2C

Programa e biblioteca LiquidCrystal_I2C

utilizar a biblioteca **LiquidCrystal_I2C**

Os comandos para controle do display são praticamente os mesmos da biblioteca *LiquidCrystal* que utilizamos normalmente, com comandos como **lcd.print()** e **lcd.setCursor()**.

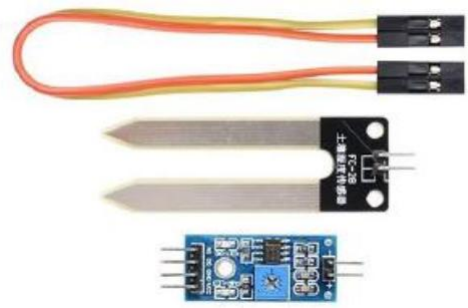
```
//Inicializa o display no endereco 0x27
LiquidCrystal_I2C lcd(0x27,16,2);
```

É preciso descobrir o endereço de cada tipo de módulo

Grupo 1: Irrigação

- Sensor de umidade de solo
- Módulo Relé
- Mini bomba de água
- Fonte de alimentação 12V

Grupo 1: Irrigação



Garden soil moisture	
A0	A0
D0	Null
GND	GND
VCC	+5V

```
val=analogRead(hum);
count=100-(val-435)/5.9;
```

Grupo 1: Irrigação

UNO R3		Relay module
5VC	->	VCC
GND	->	GND
D8	->	IN

https://www.tinkercad.com/things/j6156MqWZvB-irrigacao-automatica?sharecode=m-l_zqlu_zEG7xkPdqKcG129Pn0SO3CoeF4Lw60lQSY

Grupo 2: Esteira e controle de velocidade

- Driver de motor (Ponte H)
- Motores DC
- Potenciômetros

https://www.tinkercad.com/things/hYXrl3cKyIe-controle-esteira?sharecode=BBd5krdThTBbnScqKzAGc8MWfTMn_d8aQ3NmXOIikYw

Como escolher um motor

Corrente

Esta é a especificação elétrica mais importante, pelo menos no que diz respeito à escolha de um driver de motor.

A corrente média é a quantidade de corrente que o motor consumirá ao operar sob uma carga normal.

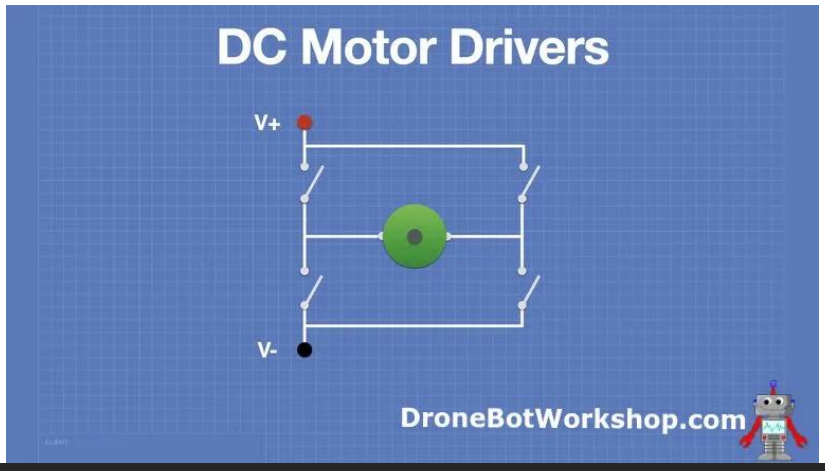
Idealmente, você selecionará um driver de motor com uma classificação de corrente acima da corrente média dos motores, de 30 a 50% é perfeito.

H-Bridge Motor Drivers

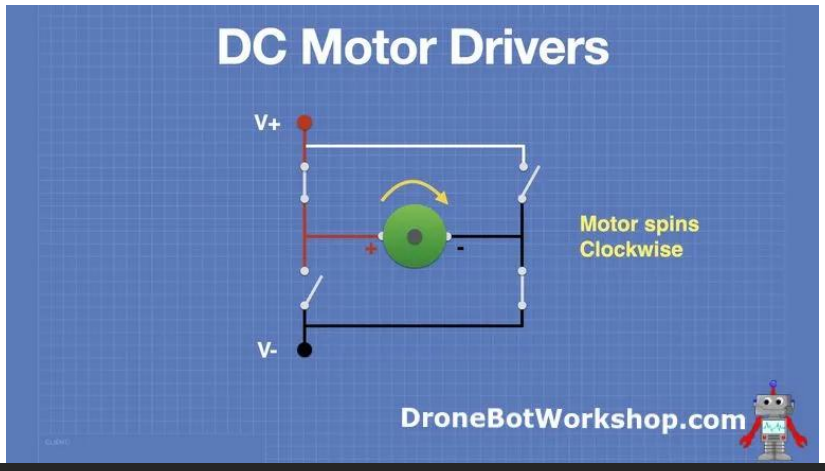
A inversão de um motor CC é simplesmente uma questão de inverter a polaridade da tensão aplicada a ele. O método mais comum de fazer isso é usar um projeto chamado “Ponte-H”.

A operação de uma Ponte-H pode ser ilustrada usando quatro chaves na seguinte configuração:

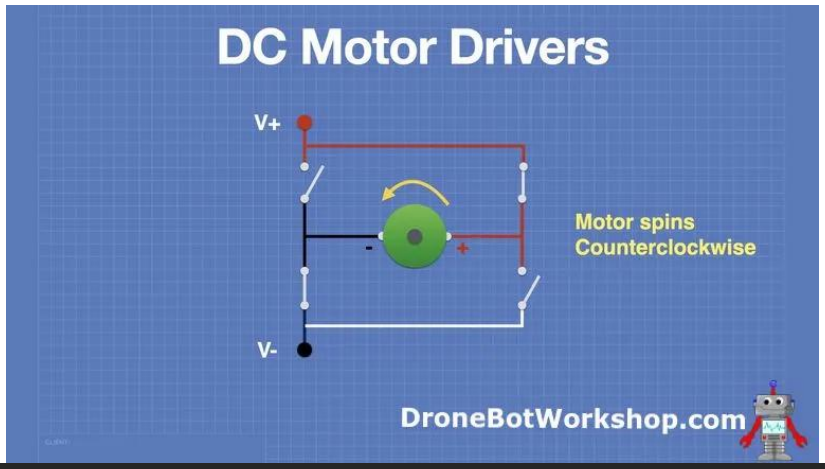
H-Bridge Motor Drivers



H-Bridge Motor Drivers



H-Bridge Motor Drivers



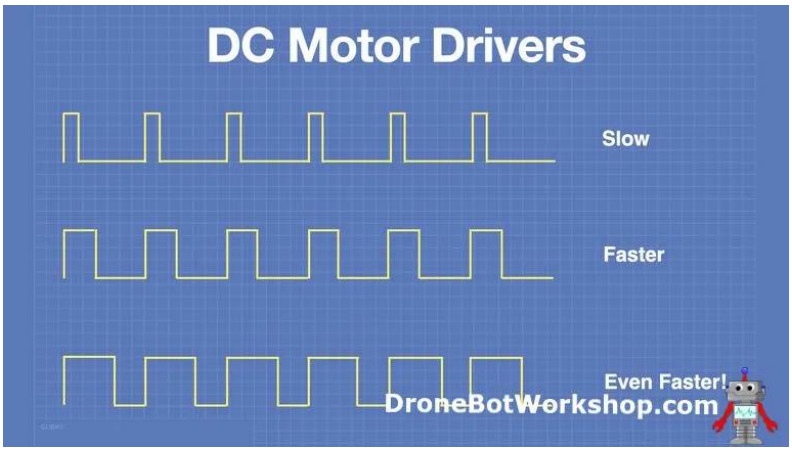
Controle de velocidade com PWM

Quando a tensão nominal é aplicada ao motor, ele girará em sua velocidade máxima.

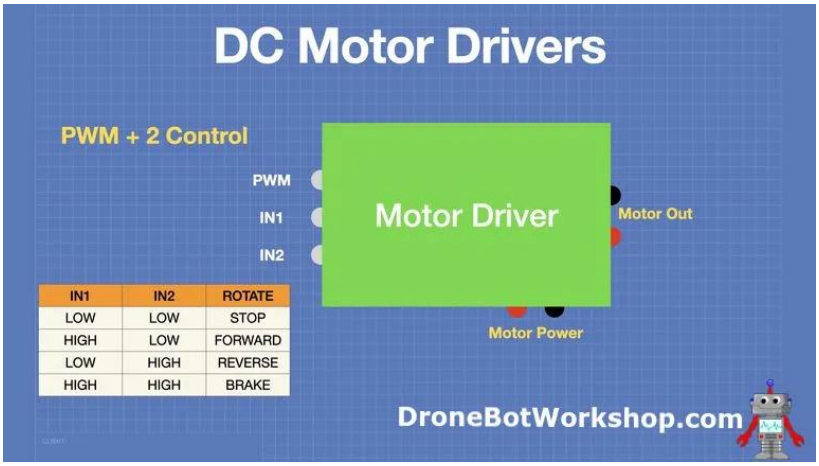
Pode parecer que apenas reduzir a tensão reduzirá a velocidade, mas esse não é realmente um bom método de controle do motor. Embora a velocidade realmente caia, o torque também cairá proporcionalmente.

Em vez disso, um método melhor é usar modulação por largura de pulso ou PWM.

Controle de velocidade com PWM



Controlando velocidade e direção com um microcontrolador



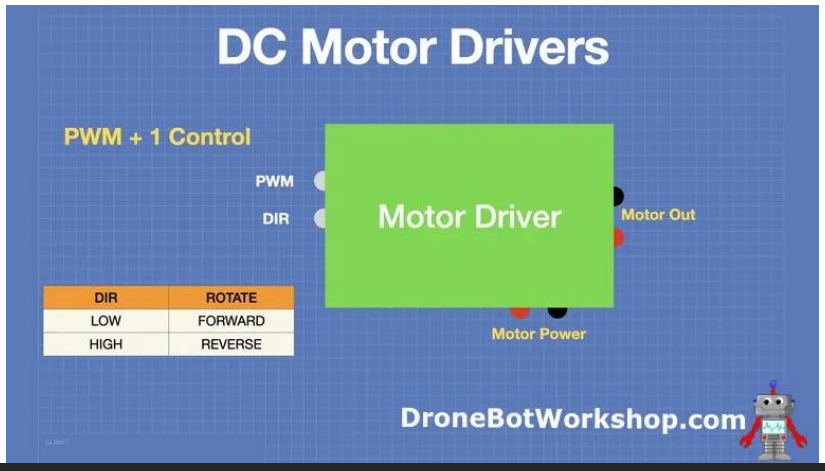
Controlando velocidade e direção com um microcontrolador

O método de controle PWM + 2 usa uma entrada para PWM mais duas entradas para controlar a direção.

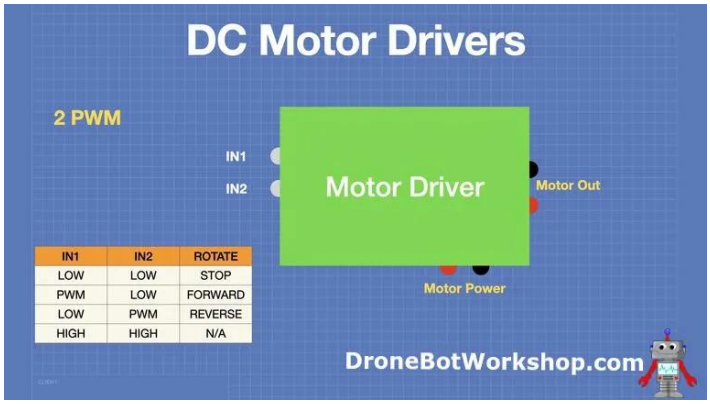
Na tabela verdade, podemos ver que o estado dos dois pinos de controle determina em que direção o eixo do motor gira.

Também posso parar o motor ou aplicar um “freio”. A função “freio” aplica corrente para manter o eixo do motor no lugar. Pode ser útil para projetos sem engrenagens, mas você deve ter cuidado, pois muitos motores podem suportar apenas alguns minutos de frenagem

PWM + 1 Controle



2 PWM



2 PWM

Com esse arranjo, você precisará de dois pinos de E/S que sejam compatíveis com PWM.

Os drivers de motor que usam o método de controle 2 PWM têm duas entradas, geralmente chamadas de IN1 e IN2.

2 PWM

Com esse arranjo, você precisará de dois pinos de E/S que sejam compatíveis com PWM.

Os drivers de motor que usam o método de controle 2 PWM têm duas entradas, geralmente chamadas de IN1 e IN2.

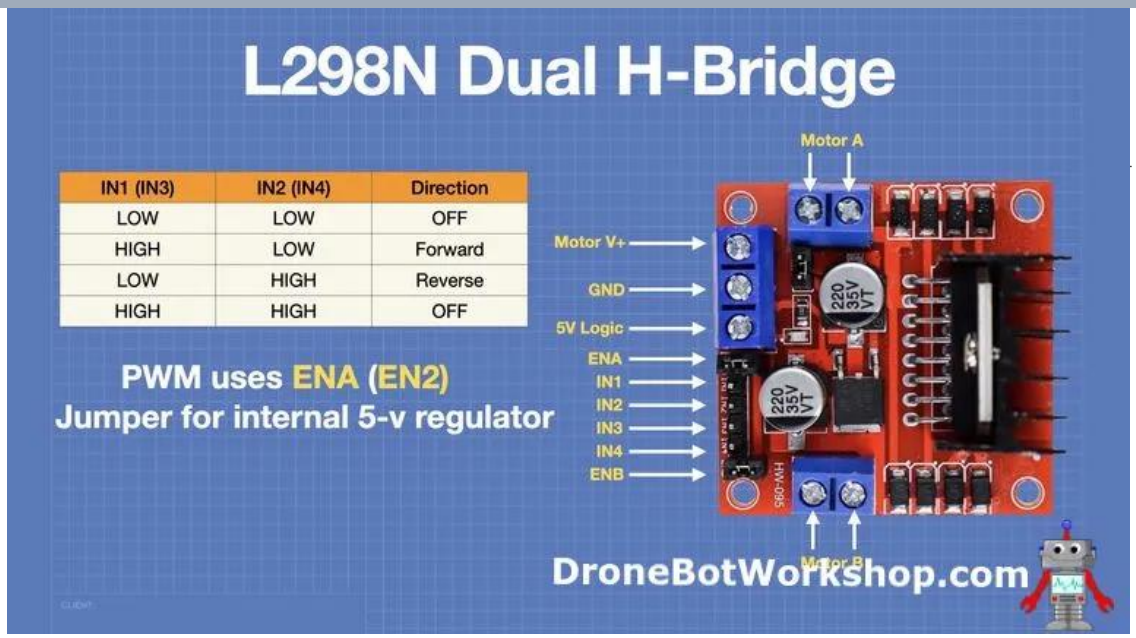
L298N Dual H-Bridge

Este é um projeto mais antigo baseado em transistores bipolares e, por ser produzido em um volume tão alto, é muito acessível.

L298N Dual H-Bridge

Olhando para o módulo, temos alguns terminais de parafuso, conectores DuPont e alguns locais para jumpers.

Existem dois conjuntos de terminais para as saídas do motor, além de outro bloco de 3 terminais para terra, tensão do motor e tensão lógica de 5 volts.



L298N Dual H-Bridge

Se você não quiser fornecer os 5 volts lógico, o módulo L298N pode usar um regulador de tensão interno para fornecer sua própria energia lógica.

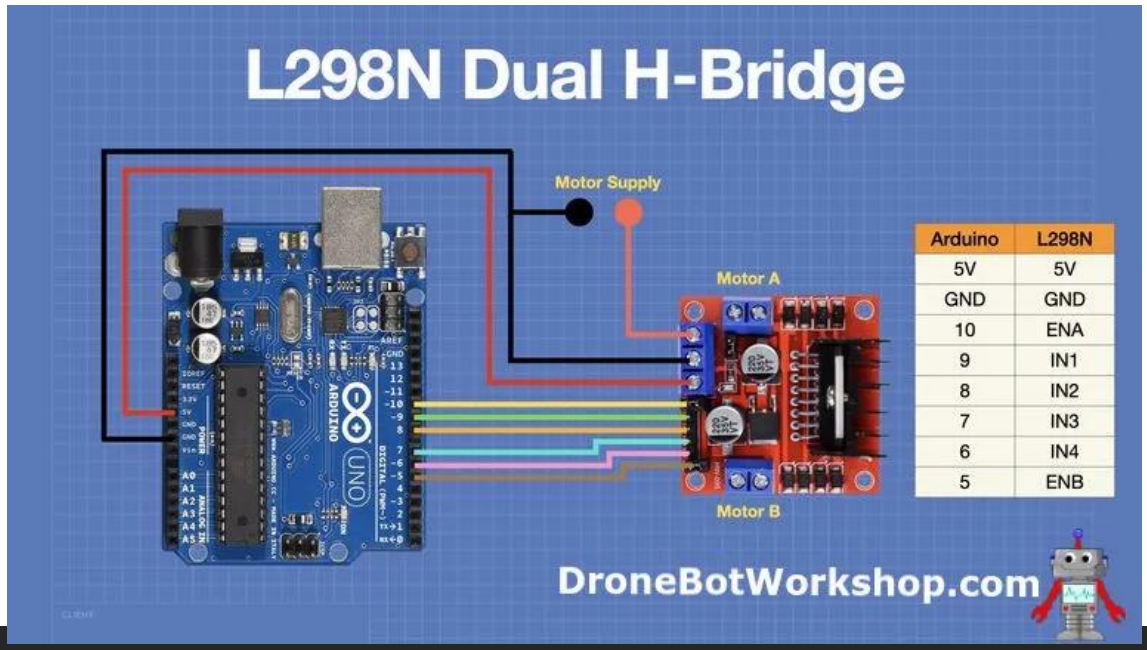
Porém, a tensão do motor precisa ser de no mínimo 7,5 volts.

L298N Dual H-Bridge

Os pinos ENA e ENB são pinos habilitadas, que podem ser alternadas por PWM para controlar a velocidade do motor.

Se você deseja apenas usar o módulo para controlar a direção do motor, pode jumpear ENA e ENB em HIGH.

IN1 a IN4 controlam a direção do motor, conforme mostrado na tabela verdade.



Arduino	L298N
5V	5V
GND	GND
10	ENA
9	IN1
8	IN2
7	IN3
6	IN4
5	ENB

L298N Arduino

Note que estamos fornecendo 5 volts do Arduino para o circuito lógico L298N.

Podemos eliminar esta conexão instalando o jumper no módulo para usar o regulador de tensão interno. Lembre-se de que, para fazer isso, a tensão de alimentação do motor precisará ser de pelo menos 7,5 volts.

Portas Analógicas

- O Arduino tem seis entradas/saídas analógicas com um conversor analógico-para-digital de 10 bits
- Isso significa que o pino analógico pode ler tensões, entre 0 e 5 volts, usando valores inteiros entre 0 (0 V) e 1.023 (5 V)

Portas Analógicas

- Isso representa uma resolução de 5V/1024 unidades, ou 0,0049 V (4,9 mV) por unidade.

Entradas analógicas

- Uma entrada digital fornece apenas dois valores, ligado ou desligado, dependendo do que está sendo aplicado à entrada de um dado pino da placa do Arduino.
- Por outro lado, uma entrada analógica fornece valores entre 0 e 1023 dependendo da tensão presente no pino de entrada analógica. No Arduino, as entradas analógicas correspondem aos pinos A0 a A5.

analogRead()

- Lê o valor do pino analógico especificado.

Syntax

```
analogRead(pin)
```

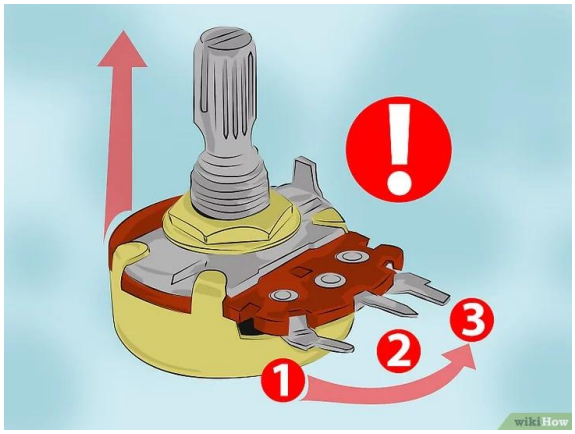
Notas e avisos

- Se o pino de entrada analógica não estiver conectado a nada, o valor retornado por **analogRead()** flutuará com base em vários fatores (por exemplo, os valores das outras entradas analógicas, quão perto sua mão está da placa, etc.).

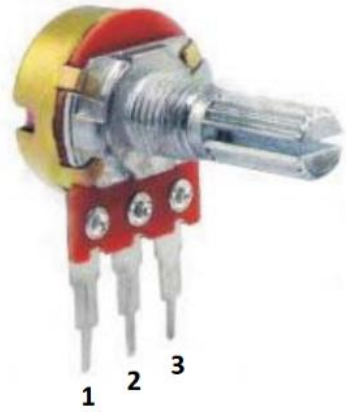
Potenciômetro

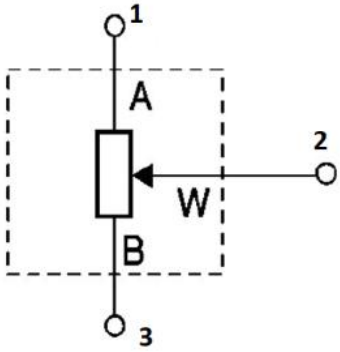
- Um potenciômetro é um resistor variável. Através do seu botão rotacional de ajuste é possível ajustar a resistência que vai de uma resistência residual (por exemplo 10 Ω), até seu valor nominal (por exemplo 10 kΩ). Ele pode ser em escala linear ou logarítmica (por exemplo ajuste de volume).

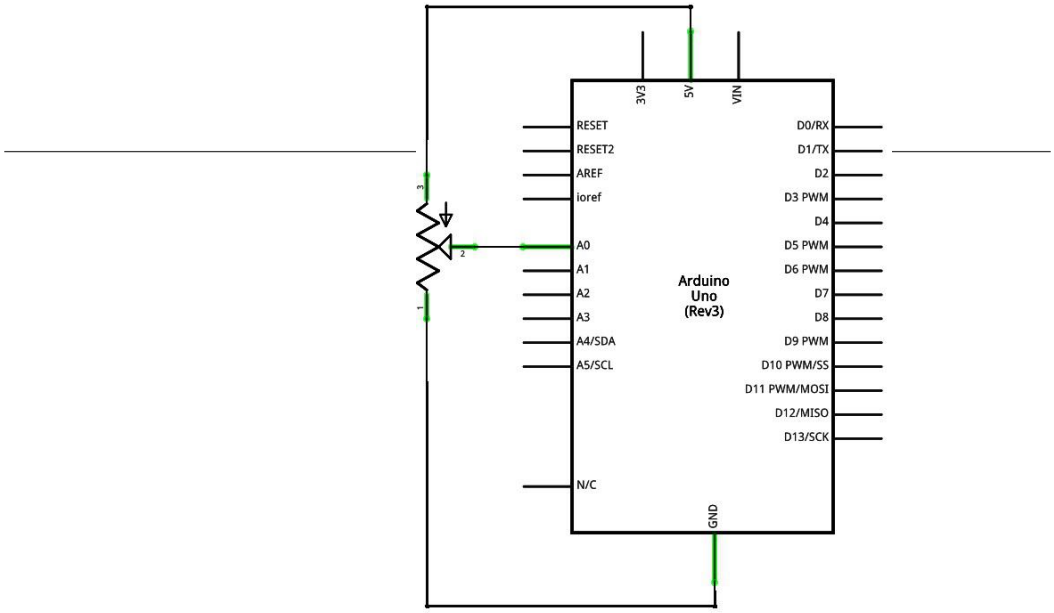
Potenciômetro



Potenciômetro







map()

- Re-mapeia um número de um intervalo para outro. Ou seja, um valor de **fromLow** seria mapeado para **toLow**, um valor de **fromHigh** para **toHigh**, valores intermediários para valores intermediários, etc.

map()

- Observe que os "limites inferiores" de qualquer intervalo podem ser maiores ou menores que os "limites superiores", portanto a função map() pode ser usada para inverter um intervalo de números, por exemplo:

```
y = map(x, 1, 50, 50, 1);
```


map()

Syntax

```
map(value, fromLow, fromHigh, toLow, toHigh)
```

- **value:** o número a ser mapeado.
- **fromLow**: o limite inferior do intervalo atual do valor.
- **fromHigh**: o limite superior do intervalo atual do valor.

map()

Syntax

```
map(value, fromLow, fromHigh, toLow, toHigh)
```

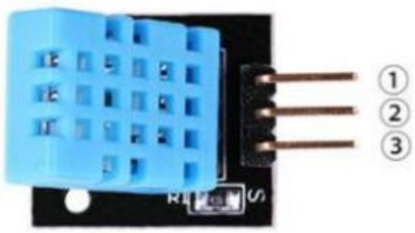
- **toLow**: o limite inferior do intervalo de destino do valor.
- **toHigh**: o limite superior do intervalo de destino do valor.

Grupo 3: Clima- Sensor de temperatura e umidade

- DHT11 ou DHT22
- LCD 16x2 I2C
- Lâmpada automotiva 12V
- Fonte de alimentação 12V

- https://wokwi.com/projects/463942511376560129

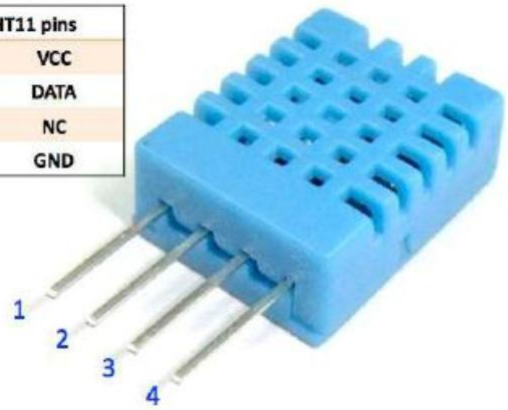
Grupo 3: Clima- Sensor de temperatura e umidade



1.GND:ground
2.VCC: 3.3V-5V DC
3.OUTPUT

Grupo 3: Clima- Sensor de temperatura e umidade

DHT11 pins	
1	VCC
2	DATA
3	NC
4	GND



Product parameters
Relative humidity:
Resolution: 16Bit
Repeatability: ±1% RH
Accuracy: At 25℃ ±5% RH
Interchangeability: fully interchangeable
Response time: 1 / e (63%) of 25℃ 6s
1m / s air 6s
Hysteresis: <± 0.3% RH
Long-term stability: <± 0.5% RH / yr in
Temperature:
Resolution: 16Bit
Repeatability: ±0.2℃
Range: At 25℃ ±2℃
Response time: 1 / e (63%) 10S
Electrical Characteristics
Power supply: DC 3.5 ~ 5.5V
Supply Current: measurement 0.3mA standby 60μA

Grupo 4: Seleção de madeiras por cor

- Sensores de luz (LDR)
- LEDs de alto brilho
- Resistores variados
- Micro Servo Motor
- https://www.tinkercad.com/things/2oEOtxY2ugn-selecao-madeira?sharecode=_UCMmQPJJABkilL3BZKlRwAZ7RjjqHKbvPOHpeHJu2U

Buzzer

O buzzer é um dispositivo que permite a geração de sinais sonoros (bipes) como aqueles encontrados em computadores. Para produzir o som, um buzzer vibra por meio de um oscilador. Essa oscilação é determinada por uma frequência que, por sua vez, define um som específico.

Buzzer

As campainhas podem ser categorizadas em ativas e passivas.

Vire os pinos do buzzer para cima, e aquele com placa de circuito verde é um Buzzer passivo, enquanto o outro fechado com uma fita preta é o ativo.

Buzzer

A diferença entre um Buzzer ativo e um Buzzer passivo é: um Buzzer ativo possui uma fonte oscilante integrada, portanto emitirá sons quando eletrificada.

Buzzer

um Buzzer passivo não possui tal fonte, portanto não emitirá tweets se sinais DC forem usados; em vez disso, você precisa usar ondas quadradas cuja frequência esteja entre 2K e 5K para acioná-lo.

Buzzer

O Buzzer ativo costuma ser mais cara que o passivo devido aos vários circuitos oscilantes integrados.

Active Buzzer

Os Buzzers ativos são chamadas de “ativos” porque podem produzir som diretamente quando conectadas a uma bateria. Se conectarmos os terminais positivo e negativo corretamente à bateria, ele poderá gerar som por si só porque possui um oscilador embutido. As campainhas ativas são as mais simples de usar. Eles normalmente estão disponíveis em faixas de tensão de 1,5V a 24V. eles podem produzir uma frequência sonora de cerca de 2kHz±300Hz. O consumo de corrente de uma campainha ativa é ≤ 25mA.

Active Buzzer

Specification

Voltage: DC 5V
Min Sound Output at 10cm: 85dB;
Total Size (Pin Not Included): 12 x 9mm/0.47" x 0.35"(D*H)

Passive Buzzer

No Buzzer passivo é preciso usar PWM para gerar o sinal de áudio.

É possível gerar diferentes tons, mudando a frequência do PWM

Passive Buzzer

Por exemplo, o envio de uma onda PWM na frequência de 523 Hz, produzirá um Dó

Um sinal de 587 Hz pode produzir Ré médio

Um sinal de 659 Hz pode produzir Mi.

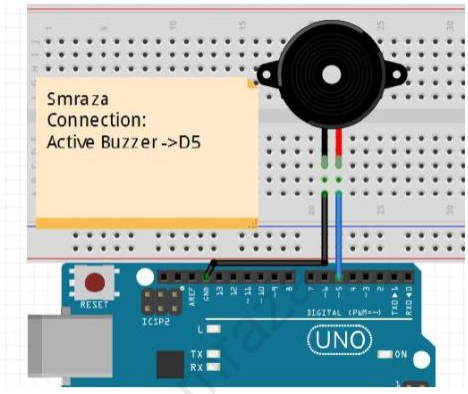
Passive Buzzer

Specification

Working Voltage: 3V/5V
Resistance: 16Ohm
Resonance Frequency: 2KHZ

Active Buzzer (Exemplo)

Connection diagram

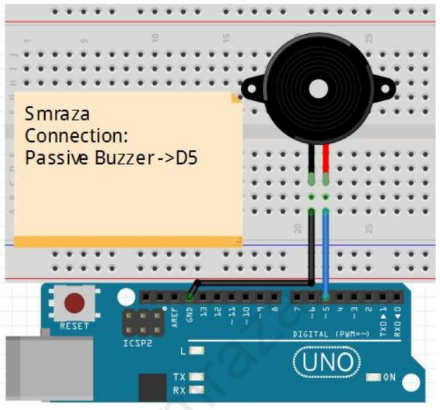


Active Buzzer (Exemplo)

```
int buzzer=5;
void setup()
{
      pinMode(buzzer,OUTPUT);
}
void loop()
{
      digitalWrite(buzzer, HIGH); // produce sound
}
```

Passive Buzzer (Exemplo)

Connection diagram



Passive Buzzer (Exemplo)

```
#define buzzer 5
void setup()
{
    // generates a 400Hz tone in output pin 8 with 2000ms of duration
    tone(buzzer, 400, 2000);
}
void loop()
{

}
```

tone()

Gera uma onda quadrada na frequência especificada (e duty cycle 50%) em um pino.

A duração pode ser especificada, do contrário a onda continua até uma chamada de noTone().

O pino pode ser conectado a um buzzer piezo ou outro speaker para tocar tons.

tone()

Apenas um tom pode ser gerado de cada vez.

Se um tom já está tocando em um pino diferente, a chamada de tone() não terá efeito.

Se o tom está tocando no mesmo pino, a chamada irá mudar sua frequência para a nova especificada.

tone()

Sintaxe

```
tone(pino, frequência)
```

```
tone(pino, frequência, duração)
```


tone()

Parâmetros

`pino`: o pino do Arduino no qual gerar o tom

`frequência`: a frequência do tom em Hertz - `unsigned int`

`duração`: a duração do tom em milissegundos (opcional) - `unsigned long`

Servomotor

Servomotores ou servomecanismos

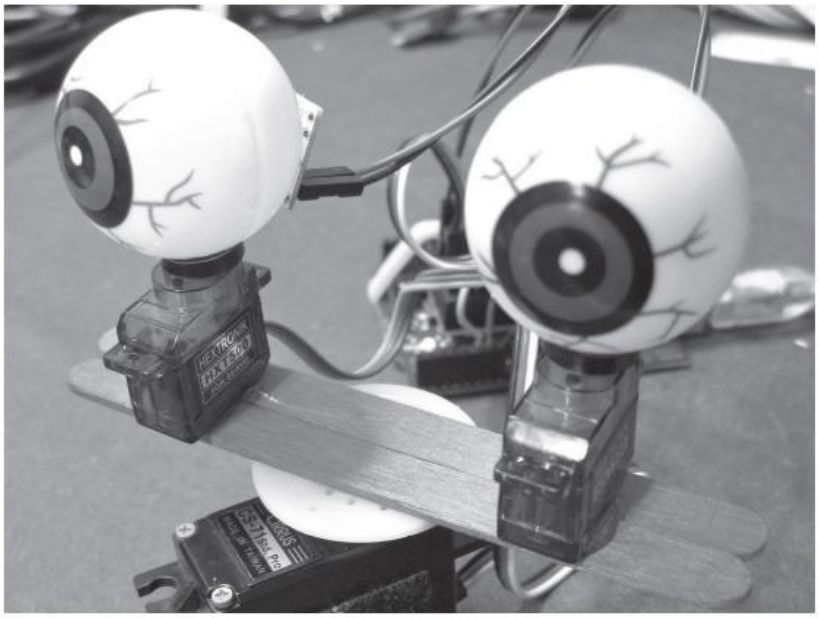
Um servo é um motor com um sistema de feedback, que auxilia no controle da posição do motor.

Servos tipicamente giram 180 graus, ainda que você também possa adquirir servos de rotação contínua, ou até mesmo modificar um servo padrão para obter esse efeito.

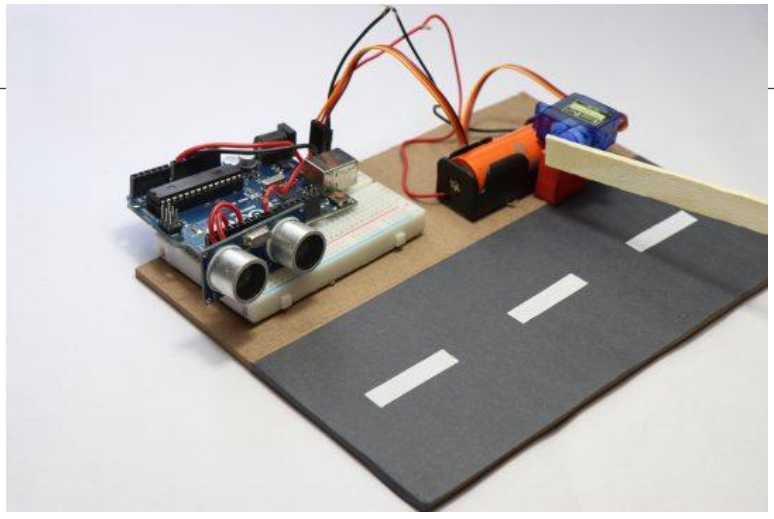
Servomotor

Caso você já tenha tido um aviãozinho de controle remoto, já se deparou com servos; eles são utilizados para controlar as superfícies de voo.

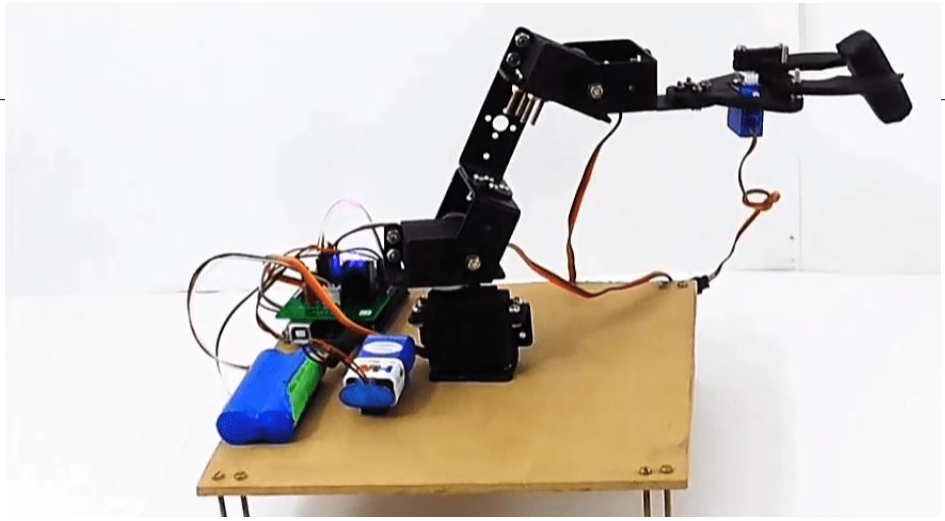
Carros de controle remoto utilizam servos no mecanismo de direção, e barcos de controle remoto, para controlar o leme.



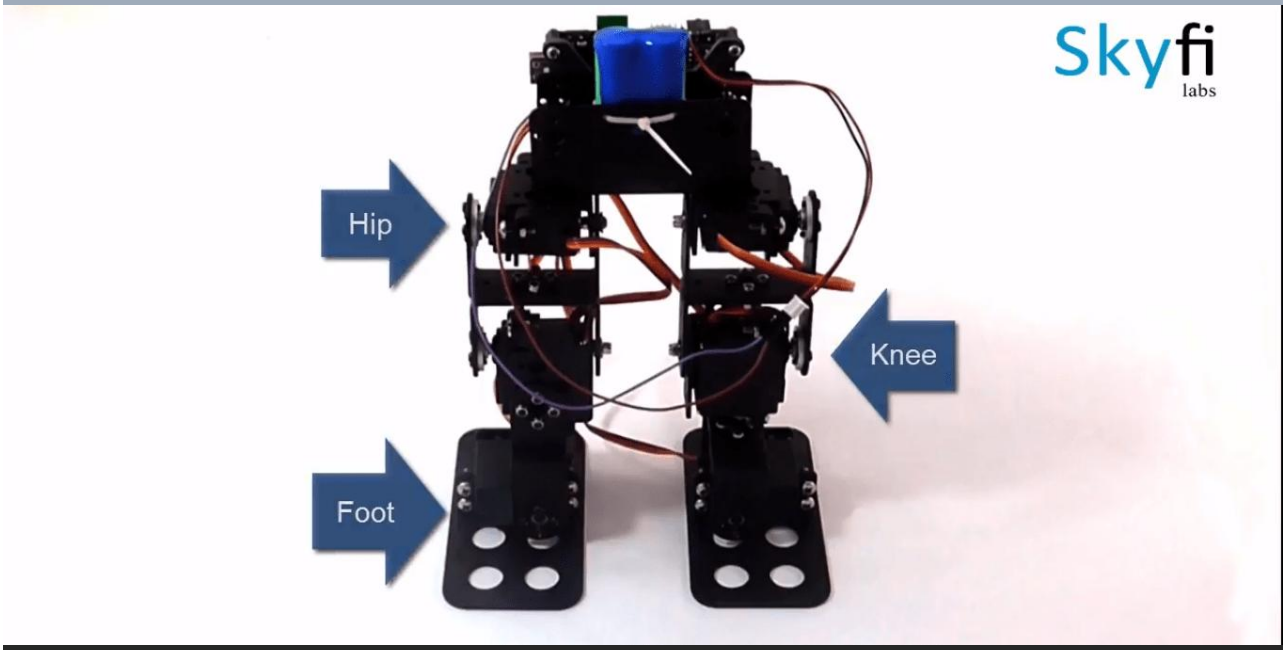
– *Três servos controlando a cabeça e os olhos de um robô (imagem por Tod E. Kurt).*



Oficina Arduino 155/



Oficina Arduino 156/



Servo Motor

Minúsculo e leve com alta potência de saída. O servo pode girar aproximadamente 180 graus (90 em cada direção) e funciona exatamente como os tipos padrão, mas menor

Você pode usar qualquer código servo, hardware ou biblioteca para controlar esses servos

Servo Motor

Bom para iniciantes que desejam fazer as coisas se moverem sem construir um controlador de motor com feedback e caixa de engrenagens, especialmente porque cabe em locais pequenos.

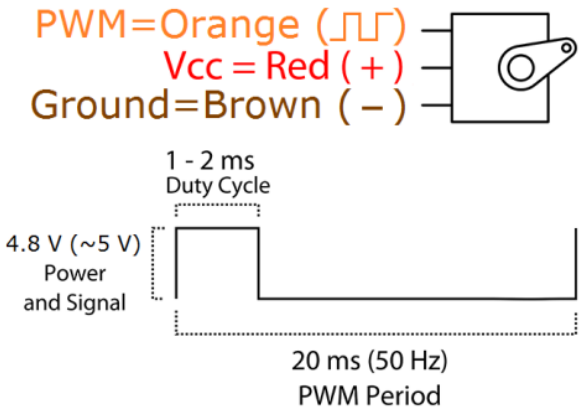
Ele vem com 3 chifres (braços) e hardware

Servo Motor

Specifications

- Weight: 9 g
- Dimension: 22.2 x 11.8 x 31 mm approx.
- Stall torque: 2.0 kgf·cm
- Operating speed: 0.09 s/60 degree
- Operating voltage: 4.8 V (~5V)
- Dead band width: 7 µsec
- Temperature range: -10 ºC – 55 ºC

Servo Motor



Servo Motor

A posição "0" (pulso de 1,5 ms) está no meio, 90° (pulso de ~2 ms) está totalmente à direita.

-90° (pulso de ~1 ms) está totalmente à esquerda.

Servo Motor

A biblioteca “Servo.h” está disponível por padrão no ambiente de desenvolvimento do Arduino, para a manipulação de servomotores.

Tal biblioteca apresenta um conjunto de métodos, tais como o attach, para definir qual pino está conectado e o método write que permite enviar o valor do ângulo no qual o eixo do motor deve ser posicionado.

Servo - attach()

Syntax

```
servo.attach(pin)
servo.attach(pin, min, max)
```

Servo - attach()

Parameters

- *servo*: a variable of type `Servo`
- *pin*: the number of the pin that the servo is attached to
- *min* (optional): the pulse width, in microseconds, corresponding to the minimum (0 degree) angle on the servo (defaults to 544)
- *max* (optional): the pulse width, in microseconds, corresponding to the maximum (180 degree) angle on the servo (defaults to 2400)

Servo - attach()

Example

```
#include <Servo.h>

Servo myservo;

void setup()
{
  myservo.attach(9);
}

void loop() {}
```

Servomotor

```
#include <Servo.h>
Servo myservo; // create servo object to control a servo
int potpin = 0; // analog pin used to connect the potentiometer
int val;
void setup() {
    myservo.attach(9); // attaches the servo on pin 9 to the servo object
}
void loop() {
    val = analogRead(potpin);
    val = map(val, 0, 1023, 0, 180);
    myservo.write(val);
    delay(15);
}
```