Motores

Os motores são atuadores de grande importância em projetos das mais diversas áreas e em IOT não é diferente. É possível o uso de motores de correntes altas através de relés, como vimos anteriormente, porém são os motores de baixa corrente os mais utilizados e com o maior custo benefício, equilibrando valores baixos de aquisição e flexibilidade de aplicação nas mais diversas situações. Falaremos aqui especificamente de três deles, cada um com suas peculiaridades que os tornam mais ou menos adequados para cada tipo de projeto: motor DC, motor de passo e servo motor.

Motor DC + Ponte H

Os motores DC são os motores mais simples e baratos entre os utilizados em projetos de prototipagem e o seu uso consiste em alimentá-los para que os mesmos girem. Quanto maior a tensão aplicada, maior a velocidade e a força desenvolvidas. Geralmente (salvo em motores DC especiais, como os utilizados em drones), não possuímos a possibilidade de realizar ajustes finos e nem possuímos muitos parâmetros passíveis de personalização em seu movimento. Isso é uma desvantagem pela pouca flexibilidade, mas pode ser também uma vantagem por conta da simplicidade. Motores DC são amplamente utilizados na movimentação de robôs que não necessitem de um ajuste muito fino e nem da garantia de movimentos milimétricos. Também é bastante comum o seu uso aliado com correias, polias e esteiras, sendo um gerador de

movimentos posteriormente controlados com transferidores mecânicos. Neste motor, dois são os parâmetros que nos interessam controlar:

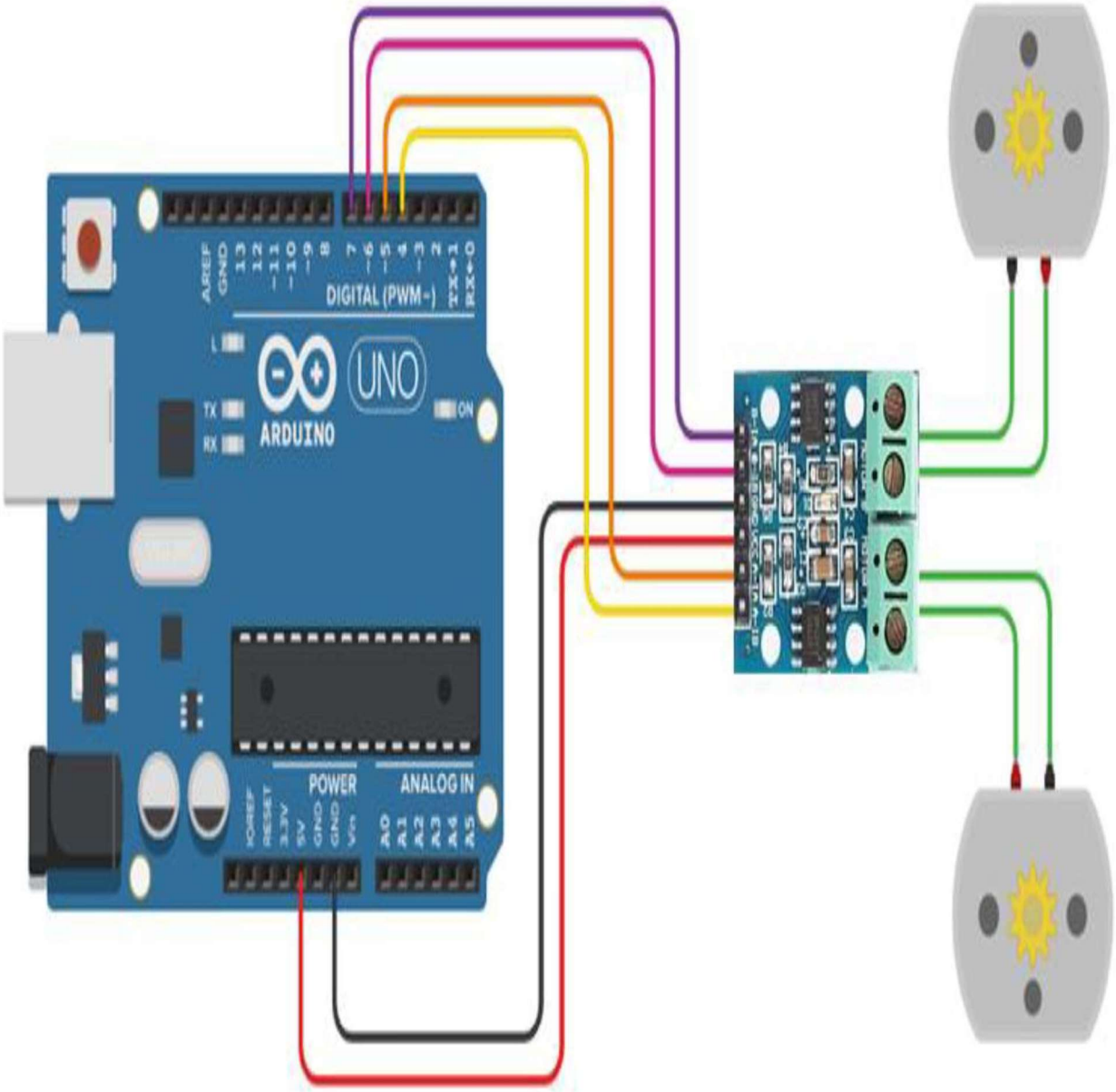
Velocidade: a velocidade do motor é diretamente proporcional à tensão aplicada a ele. Desde modo, se o motor for alimentado com até 5V, uma boa alternativa é ligar qualquer um dos seus polos (ele não é polarizado) ao GND e o outro à uma porta digital PWM qualquer, pois nela, através do controle do Duty Cycle, regulamos a tensão de saída. Já para sabermos a velocidade, uma das possibilidades mais simples é anexar um disco perfurado ao eixo do motor e com ele contarmos os giros por segundo através de um encoder óptico, transformando estes giros em metros por segundo e após em quilômetros por hora. Caso não lembre como fazer isso, consulte o capítulo 6.1.12 (Encoder óptico e Encoder magnético).

Direção do giro: a direção do giro dos motores DC (horário ou anti-horário), depende diretamente da polarização de sua ligação à fonte de alimentação. Se escolhermos um dos seus polos e ligarmos ele ao GND e o outro à uma alimentação (fonte, bateria, porta digital do Arduino, etc) ele irá girar em um sentido e se invertermos ele irá girar em um sentido inverso. Se para nosso uso o giro em um só sentido é suficiente, basta descobrir qual a polarização correta. Mas caso tenhamos a necessidade de que esse giro mude de sentido ao longo do uso (para provocar movimentos para frente ou para trás, para auxiliar em uma freada ou diminuição brusca de velocidade, etc), obviamente que não seria adequado que realizássemos essa inversão manualmente. É aí que entra a Ponte H. Este componente tem um princípio muito simples: em um dos seus lados, temos dois terminais onde conectamos um motor e no outro temos dois terminais onde conectamos a fonte de alimentação do mesmo. Porém, existem dois pinos adicionais binários que devem ser ligados à duas portas digitais e, de acordo com a combinação de estados dessas portas (LOW ou HIGH) ele inverte ou desinverte a polaridade, fazendo com que o motor mude o sentido de sua rotação. Uma das grandes vantagens da ponte H é que além da simplicidade nessa inversão, ela permite a alimentação com fontes externas para o controle de motores de maior potência. Quanto ao giro, os dois pinos ligados às portas digitais apresentam as seguintes combinações lógicas:

Pino A	Pino B	Consequência
LOW	LOW	motor não energizado
LOW	HIGH	motor girando em sentido horário
HIGH	LOW	motor girando em sentido antihorário
HIGH	HIGH	motor travado (freio)

As pontes H geralmente são encontrados em módulos duplos, permitindo o controle

de dois motores de forma simultânea e possuindo, desse modo, além de um VCC e um GND, mais 4 pinos de controle (2 para cada motor). Abaixo, um esquema eletrônico de uso de uma ponte H dupla.



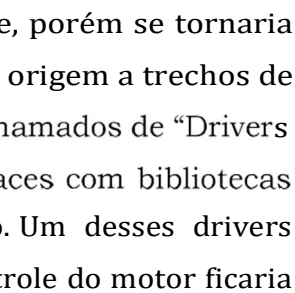
Neste esquema, o GND e o VCC estão conectados no Arduino, utilizando como alimentação dos motores os 5V fornecidos por ele. Porém se fossem motores mais potentes e que necessitassem de uma tensão maior, bastaria ligar o VCC e GND da ponte H em uma fonte externa (12V por exemplo) e não ao Arduino. Realizadas as ligações, o controle do sentido do giro fica bastante simples, baseado na mudança de estados das portas (6 e 7 no motor, 1 e 4 e 5 no motor 2). Abaixo um exemplo onde os motores iniciam desligados,

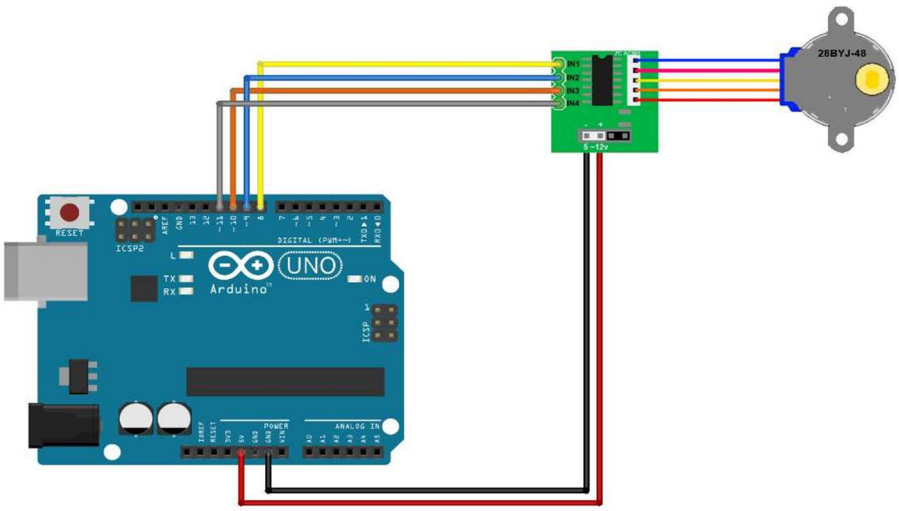
após movimentam-se juntos no mesmo sentido freando após 2 segundos e por último movimentam-se em sentidos opostos por mais 2 segundos.

```
#define A_M1 6
#define B_M1 7
#define A_M2 4
#define B_M2 5

void setup(){
  pinMode(A_M1, OUTPUT);
  pinMode(B_M1, OUTPUT);
  pinMode(A_M2, OUTPUT);
  pinMode(B_M2, OUTPUT);
}

void loop(){
  //iniciando com os motores desligados
  digitalWrite(A_M1, LOW);
  digitalWrite(B_M1, LOW);
  digitalWrite(A_M2, LOW);
  digitalWrite(B_M2,  LOW);
  delay(2000);

  //movimentando os dois para frente (mesmo sentido)
  digitalWrite(A_M1, LOW);
  digitalWrite(B_M1, HIGH);
  digitalWrite(A_M2, LOW);
  digitalWrite(B_M2, HIGH);
  delay(2000);

  //freando os dois motores
  digitalWrite(A_M1, HIGH);
  digitalWrite(B_M1, HIGH);
  digitalWrite(A_M2, HIGH);
  digitalWrite(B_M2, HIGH);
  delay(2000);

  //movimentando os dois em sentido inverso
  digitalWrite(A_M1, LOW);
  digitalWrite(B_M1, HIGH);
  digitalWrite(A_M2, HIGH);
  digitalWrite(B_M2, LOW);
  delay(2000);

}
```

Motor de passo

Os motores de passo são motores especiais com três características bastante interessantes: uma grande força (torque) proporcionada pelo uso de uma caixa de redução interna, a possibilidade do controle de sua velocidade e a possibilidade do controle da amplitude do seu movimento (ou seja, o quanto irá se movimentar a cada comando). Por estas caraterísticas, eles não são motores para uso de velocidade pois são bem mais lentos que os DC, porém contam com uma precisão incomparavelmente maior. Assim, são muito mais adequados para ações onde a precisão dos movimentos é mais importante que a velocidade de realização da ação. Essa precisão é possível pois os motores de passo trabalham internamente com múltiplas bobinas dispostas ao longo do perímetro do seu corpo circular. Os múltiplos canais (que alimentam estas bobinas) são ativados alternadamente em um movimento que force o deslocamento linear do rotor interno, girando o pino do motor e travando quando o movimento é interrompido. Essa sequência de energização das bobinas e o consequente movimento do rotor, pode ser implementado manualmente através da mudança de estados das portas de controle, porém se tornaria complexa de acordo com a angulação necessária a cada passo e daria origem a trechos de código extensos. Para facilitar o uso, existem adaptadores especiais chamados de "Drivers para Motor de Passo" que simplificam o processo, oferecendo interfaces com bibliotecas que contém métodos que possibilitam um controle mais lógico e direto. Um desses drivers (e talvez o mais famoso) é o ULN 2003 e o circuito eletrônico para controle do motor ficaria como o exemplo retirado do site FelipeFlop e desenvolvido por *Adilson Thomsen*:





Em sua grande maioria, os motores de passo (como o do exemplo) possuem 5 fios para o controle das bobinas e das sequências de energização, sendo que estes fios terminam em um plug que facilita a conexão do motor com um driver. O ULN2003, além de contar com o encaixe para o plug do motor, possui uma porta GND (identificada com o sinal de menos) e uma porta VCC (identificada com o sinal de mais) com uma tensão de entrada suportada entre 5V e 12V. Assim, caso o motor de passo utilizado seja de até 5V, estes dois pinos poderiam ser ligados diretamente ao VCC e GND do Arduino. Caso o motor de passo necessitasse de maior corrente (6V, 9V, 12V) estes pinos poderiam ser ligados diretamente à uma fonte que forneça essa tensão. Além disso, os 4 pinos de controle do driver (nomeados de IN1, IN2, IN3 e IN4), devem ser ligados à 4 portas digitais do Arduino para que através delas os movimentos e a velocidade sejam controlados.

O Arduino conta com uma biblioteca nativa para esse fim, chamada de Stepper. Ela nos traz um objeto do tipo Stepper que para ser instanciado necessita de 5 parâmetros: o número de passos por evolução (o padrão é 500 e esse será o valor que vamos usar para basear os outros movimentos) e as portas do Arduino onde foram ligados os pinos de controle, EXATAMENTE nesta ordem: IN1, IN3, IN2 e IN4. Instanciado o objeto, ele nos oferece dois métodos importantes: *setSpeed* para controle da velocidade e *step* para controle do movimento gerado pelo motor.

```
setSpeed( <velocidade> );

<velocidade>
 rotações por minuto do motor
```

```
step( <passos> );

<passos>
 total de passos de deslocamento (giro)
```

Na configuração padrão de um motor de passo, um giro completo necessita de 2048 passos (sequências possíveis na combinação das bobinas). Isso significa que cada grau é equivalente a 5 passos (2048 / 360 = 5). Deste modo, caso se precise ordenar uma rotação em graus ao motor, podemos facilmente calcular, multiplicando o ângulo por 5 para a conversão em passos. Além disso, o valor positivo gera um movimento em sentido horário e um valor negativo gera um movimento em sentido anti-horário. Quanto a velocidade, na configuração default (com o número de passos por evolução em 500), os motores mais simples suportam velocidade entre 0 e 70 RPM.

Para ilustrar, veja o código da próxima página. Nele é instanciado um objeto Stepper, respeitando a sequência de parâmetros estabelecida. Dentro do loop, o motor tem sua velocidade configurada para 30 RPM e recebe uma ordem para girar 360º em sentido

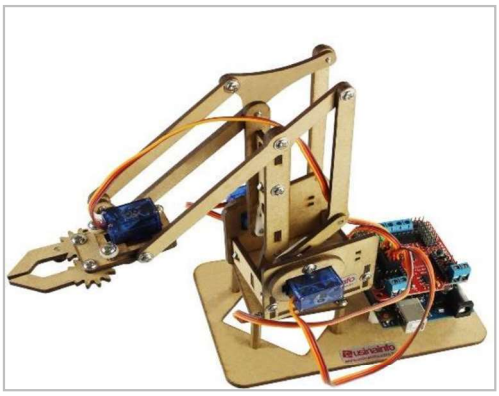
horário (valor positivo e ângulo multiplicado por 5 para transformação em passos) e após em sentido anti-horário (valor negativo e ângulo multiplicado por 5 para transformação em passos). Para finalizar, o motor tem sua velocidade reduzida para 10 RPM e recebe uma ordem para girar 180º em sentido horário e depois 180º em sentido anti-horário.

```
#include <Stepper.h>

Stepper motor(500, 8, 10, 9, 11);

void setup(){

}

void loop(){

  //360° em sentido horário e depois 360 em sentido anti-horário à 30 RPM
  motor.setSpeed(30);
  motor.step(360*5);
  delay(1000);
  motor.step(-360*5);
  delay(1000);

  //180° em sentido horário e depois 90° em sentido anti-horário à 10 RPM
  motor.setSpeed(10);
  motor.step(180*5);
  delay(1000);
  motor.step(-90*5);
  delay(1000);

}
```

Servomotor

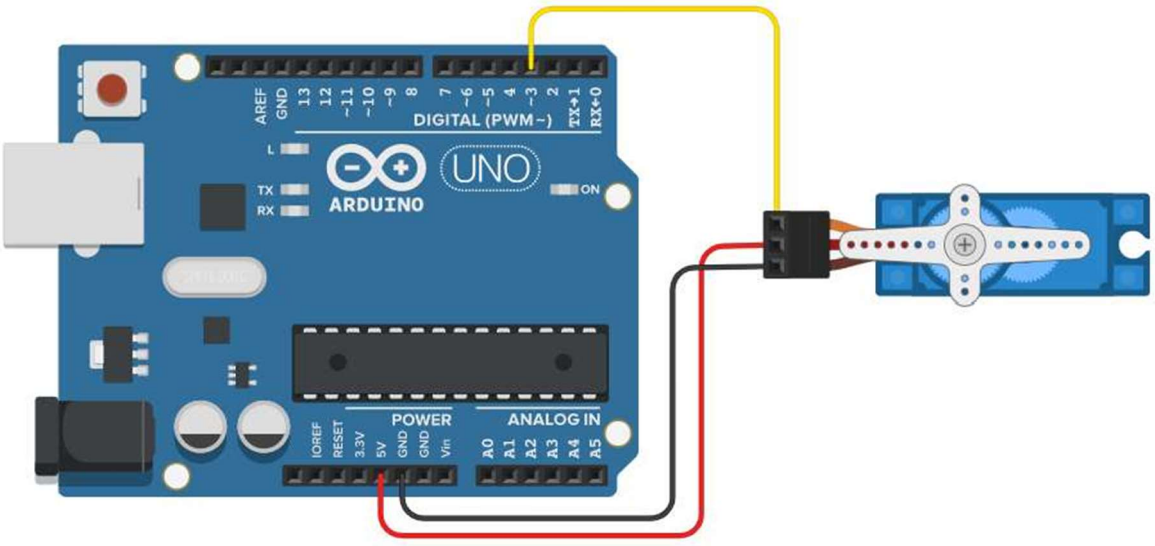
Os servomotores são motores similares aos de passo, com força e precisão, porém com uma facilidade de uso ainda maior, visto que seus movimentos se baseiam no deslocamento do eixo para um ângulo entre 0º e 180º. Assim, podemos perceber que a desvantagem é a limitação do movimento, não sendo possível o giro completo (salvo em servomotores especiais e muito mais caros, ou modificados internamento, o que não é recomendado). Mas mesmo assim, este giro de 180º geralmente é suficiente para a realização de movimentos que não sejam o de deslocamento, com a grande vantagem de os servos serem mais baratos do que os motores de passo. Para se ter ideia das possibilidades que os servos nos oferecem, mesmo com a limitação do giro, é disponibilizado um link da UsinaInfo onde *Matheus Gebert Straub* apresenta o funcionamento de um kit de peças de madeira que possibilitam a montagem de um braço

robótico, controlado através de 2 joysticks, com uso de 4 servos para rotação em 3 eixos além do movimento de abre e fecha da garra. Neste exemplo está sendo utilizada uma Shield para a conexão dos servomotores, mas a mesma é desnecessária já que eles podem ser conectados diretamente ao Arduino, usando, cada um, uma porta digital para o controle (além do VCC e do GND da placa).



https://www.usinainfo.com.br/blog/braco-robotico-arduino-com-servo-motor-e-joystick/

A ligação de um servomotor ao Arduino é bastante simples e similar ao esquema mostrado abaixo (onde o pino de controle foi ligado à porta digital 3):



Para o controle efetivo do movimento do servomotor, utilizamos a biblioteca Servo, nativa do Arduino, que nos oferece um objeto do tipo Servo que não necessita de nenhum parâmetro ao ser instanciado. Após, podemos acessar com este objeto dois métodos importantes: *attach* (que nos permite indicar à qual do Arduino o pino de controle do servo foi conectado) e write (que nos permite dizer ao servo o ângulo para o qual ele deve deslocar o seu eixo).

```
attach( <porta> );
```

```
<porta>
   porta do Arduino onde o pino de controle
   do servo foi ligado
```

```
write( <angulo> );
```

```
<angulo>
   ângulo entre 0° e 180° para o qual o
   Servo deve deslocar o seu eixo
```

Para um melhor entendimento, veja o código do exemplo abaixo onde iniciamos o servomotor em 0º, deslocamos para 180º, após voltamos para 90º, depois 45º até que o loop reinicie e o ângulo volte a ser de 0º, iniciando um novo ciclo de movimento. Perceba que diferentemente do motor de passo, não controlamos a velocidade do servo. Outra característica interessante é que como o ponto máximo à direita é de 0º e o ponto máximo à esquerda é de 180º, o deslocamento em sentido horário e anti-horário é natural, apenas informando a posição desejada através do ângulo, sem a necessidade do uso de valores positivos e negativos para informar o sentido.

```
#include <Servo.h>

Servo motor;

void setup(){
  motor.attach(3);
}

void loop(){

  //deslocando o eixo do servo para 0° → 180° → 90° → 45°
  motor.setSpeed(0);
  delay(1000);

  motor.setSpeed(180);
  delay(1000);

  motor.setSpeed(90);
  delay(1000);

  motor.setSpeed(45);
  delay(1000);
}
```