Disciplina:	ME575 - Sistemas Digitais 1
Professor:	José Rodrigues de Oliveira Neto

Universidade Federal de Pernambuco
Centro de Tecnologia e Geociências
Departamento de Engenharia Mecânica

Prática 4 - 2019.2

O objetivo desta prática é testar as entradas analógicas do Arduino, saídas PWM e a comunicação serial no padrão RS232. Essa "prática" será bem mais rápida se compararmos com a Prática 3. O intuito dela é apresentarmos como utilizar periféricos que serão necessários na próxima prática.

1 Comunicação serial - O Padrão RS232

A comunicação serial no padrão RS232 ainda é muito utilizada em sistemas embarcados, por ser simples de ser implementada. O Arduino se comunica com o computador a partir desse protocolo de comunicação, embora ele seja convertido num sinal padrão USB por um chip CH340DS1[1].

RS232 é um método **assíncrono** de **comunicação serial**. Para entender melhor o que significa uma **comunicação serial**, imagine um túnel que interliga dois dispositivos. Por este túnel, deve ser enviada uma mensagem do emissor para o receptor e para facilitar o nosso exemplo, imagine que esta mensagem seja composta por 10 palavras que formam uma frase. Casa palavra é composta de letras e essas letras por usa vez são codificadas em bits. O RS232 utiliza o sistema binário (1 e 0) para transmitir dados em formato ASCII (American Standard Code for Information Interchange), por exemplo. Este conjunto de bits que formam essas 10 palavras são então enviados por esse túnel bit a bit. Nesse caso, esse túnel é um fio e esses bits são enviados ao mudar a tensão entre dois valores pré-estabelecidos.

A comunicação serial pode ser feita por apenas um par de fios: TX - o fio que transmite o sinal; RX - o fio que recebe um sinal. Podem existir outros fios de controle, no entanto, nos foquemos nesses dois.

O protocolo RS-232 é um padrão de comunicação serial criado pela EIA (*Eletronic Industries Association*) para a comunicação entre um DTE (terminal de dados) e um DCE (um comunicador de dados), também conhecida com EIA-232. Normalmente, o pacote enviado é constituído de 10 ou 11bits, dos quais 8 bits constituem a mensagem (caracteres codificados em ASCII), 1bit de início (Start bit), 1/1,5/2bit(s) de parada (Stop bit) e 1bit de paridade (Parity bit) para o controle de erro, como mostrado na Figura 1.

1.1 A comunicação serial no Arduino

Como dito anteriormente, o Arduino se comunica com o computador por protocolo de comunicação serial assíncrono no padrão RS232, embora ele seja convertido num sinal padrão USB por um chip CH340DS1. Para utilizar esse meio de comunicação, é necessário utilizar as funções que já vem prontas no Arduino. Elas estão agrupadas no objeto `Serial`. Assim como qualquer periférico, é necessário primeiramente configurar a serial, para isso utiliza-se a função `Serial.begin(speed)`, em que *speed* é a velocidade de comunicação em bits por segundo (bps).

[1] <https://cdn.sparkfun.com/datasheets/Dev/Arduino/Other/CH340DS1.PDF>

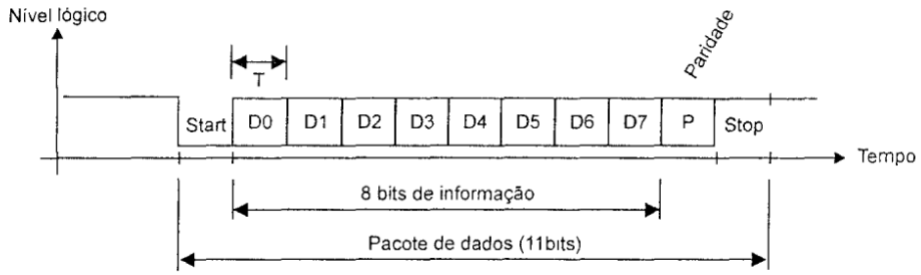

Figura 10.2: Transmissão RS-232 assíncrona.

Figura 1: Pacote de dados da comunicação serial no padrão RS232.

As taxas de transmissão comumente utilizadas são 300bps, 600bps, 1200bps, 2400bps, 4800bps, 9600bps, 19200bps, etc. Utilizaremos a taxa de 9600bps (ver código exemplo).

Para enviar um dado pela serial, utiliza-se as funções `Serial.print()` ou `Serial.println()`. A diferença entre elas é que a última imprime dados na porta serial como texto ASCII, assim como a `Serial.print()`, mas após isso imprime os caracteres de retorno de carruagem (ASCII 13, ou '\r') e de nova linha (ASCII 10, ou '\n').

1.2 Sintaxe no Arduino

- `Serial.print(val)`
- `Serial.print(val, formato)`
- `Serial.println(val)`
- `Serial.println(val, formato)`

1.3 Parâmetros

- `Serial`: objeto porta serial.
- `val`: o valor a ser impresso - qualquer tipo de dados
- `formato`: especifica a base do numeral (para tipos de dados int) ou número de casas decimais (para tipos de dados float)

1.4 Retorna

retorna o número de bytes escritos, porém a leitura desse número é opcional.

1.5 Código de Exemplo

O código abaixo imprime um valor na porta serial em vários formatos.

```
/*
  Usa um loop for para dados e imprime cada numero em varios formatos.
*/
int x = 0;  // variable

void setup() {
  Serial.begin(9600); // abre a porta serial a 9600 bps:
}
```

```
void loop() {
  // imprime rotulos para cada base
  Serial.print("NUMERO"); // imprime um rotulo
  Serial.print("\t"); // imprime uma tabulacao (TAB)

  Serial.print("DEC");
  Serial.print("\t");

  Serial.print("HEX");
  Serial.print("\t");

  Serial.print("OCT");
  Serial.print("\t");

  Serial.print("BIN");
  Serial.println("\t");

  for (x = 0; x < 64; x++) {  // apenas uma parte da tabela ASCII, edite para
   mais ou menos valores

    // imprime cada numero em varios formatos:
    Serial.print(x);        // imprime como decimal codificado em ASCII- o mesmo
   que "DEC"
    Serial.print("\t");     // imprime uma tabulacao

    Serial.print(x, DEC);   // imprime como decimal codificado em ASCII
    Serial.print("\t");     // imprime uma tabulacao

    Serial.print(x, HEX);   // imprime como hexadecimal codificado em ASCII
    Serial.print("\t");     // imprime uma tabulacao

    Serial.print(x, OCT);   // imprime como octal codificado em ASCII
    Serial.print("\t");     // imprime uma tabulacao

    Serial.println(x, BIN);// imprime como binario codificado em ASCII
    //entao adiciona o retorno (enter) com "println"

    delay(200);             // delay de 200 milissegundos
  }
  Serial.println();         // imprime outro retorno
}
```

1.6 O que fazer

1. **Teste o código exemplo:** utilize o terminal que já vem na IDE do Arduino para ver os dados. O terminal se encontra no menu: **Ferramentas** $\Rightarrow$ **Monitor Serial**.

2. Faço um código para que, ao invés de imprimir em diferentes bases numéricas, o código imprima o valor de número `float` (por exemplo, use para imprimir uma variável `valor = float(millis())/10000)`) com diferentes número de casas decimais $(1, 2, \cdots, 10)$.

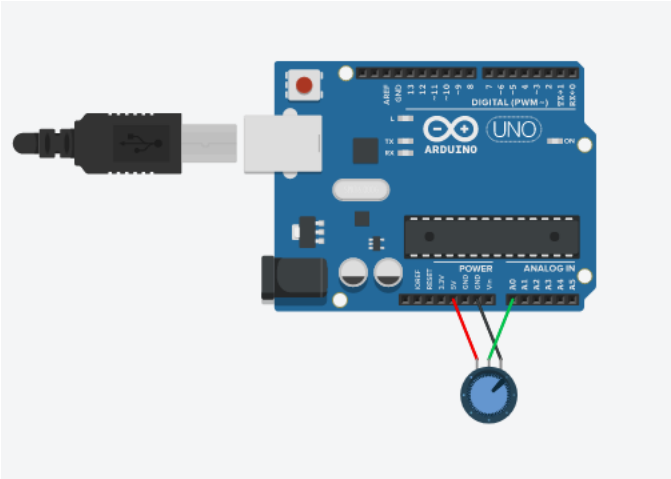

Figura 2: Montagem do circuito com potenciômetro e o Arduino.

2 Entradas analógicas

Para ler uma entrada de tensão analógica, utilizamos a função `analogRead(pino)`, que, ao contrário das equivalentes digitais, são restritas a alguns pinos específicos. As entradas analógicas são identificadas pelos pinos `ANALOG IN` (`A0` a `A5` no arduino) e são ligadas a um conversor analógico/digital (conversor AD) do microcontrolador, que transforma estes sinais numa palavra binária de 10 bits.

Como $2^{10} = 1024$, isto significa que a função `analogRead()` retorna um valor entre 0 (para uma entrada de 0 V) e 1023 (para uma entrada de 5 V).

2.1 O que fazer

1. Monte o circuito mostrado na Figura 2, utilizando uma *protoboard*.

2. Monte um código para ficar lendo a cada 200 ms o valor do potenciômetro e imprimindo ele no Monitor Serial.

3 Sinal PWM

Embora o Arduino possua um conversor AD, ele não possui um conversor DA (digital/analógico). Essa carência é suprida pela capacidade de muitos sistemas digitais de gerarem sinais *pulse width modulation* (PWM), ou sinais modulados por largura de pulso.

PWM é uma técnica utilizada por sistemas digitais para variação do valor médio de uma forma de onda periódica. A técnica consiste em manter a frequência de uma onda quadrada fixa e variar o tempo que o sinal fica em nível lógico alto. Esse tempo é chamado de *duty cycle*, ou seja, o ciclo ativo da forma de onda. Na Figura 3 são exibidas algumas modulações PWM.

Analisando as formas de onda mostradas na Figura 3, nota-se que a frequência da forma de onda tem o mesmo valor e varia-se o *duty cycle* da forma de onda. Quando o duty cicle está em 0% o valor médio da saída encontra-se em 0 V e consequentemente para um *duty cycle* de 100% a saída assume seu valor máximo, que no caso é 5V. Para um *duty cycle* de 50% a saída assumirá 50% do valor da tensão, 2,5 V e assim sucessivamente para cada variação no *duty cycle*. Portanto, para calcular o valor médio da tensão de saída de um sinal PWM pode-se

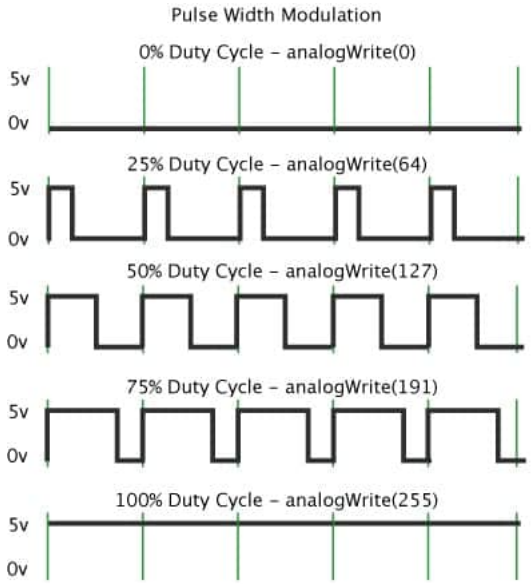

Figura 3: Exemplo de sinal PWM com diferentes *duty cycles*.

utilizar a seguinte equação:

$$V_{out} = \frac{duty\,cycle}{100} \times V_{cc} \tag{1}$$

em que,

- V_{out} - tensão de saída em V;
- $duty\,cycle$ - valor do ciclo ativo do PWM em %;
- V_{cc} - tensão de alimentação em V.

PWM pode ser usada para diversas aplicações, como por exemplo:

- controle de velocidade de motores;
- variação da luminosidade de leds;
- geração de sinais analógicos;
- geração de sinais de áudio.

3.1 PWM no Arduino

A placa Arduino Uno possui pinos específicos para saídas PWM, pinos 3, 5, 6, 9, 10 e 11. Para auxiliar na manipulação desses pinos a plataforma possui uma função que auxilia na escrita de valores de *duty cycle* para esses pinos. Essa função é a `analogWrite()`, cujo funcionamento nada tem a ver com a função `analogRead()`. Uma importante observação é que o pino que será utilizado para gerar o sinal PWM precisa ser inicializado como saída durante o *setup* do sistema.

3.1.1 Sintaxe

```
analogWrite(pino, valor)
```

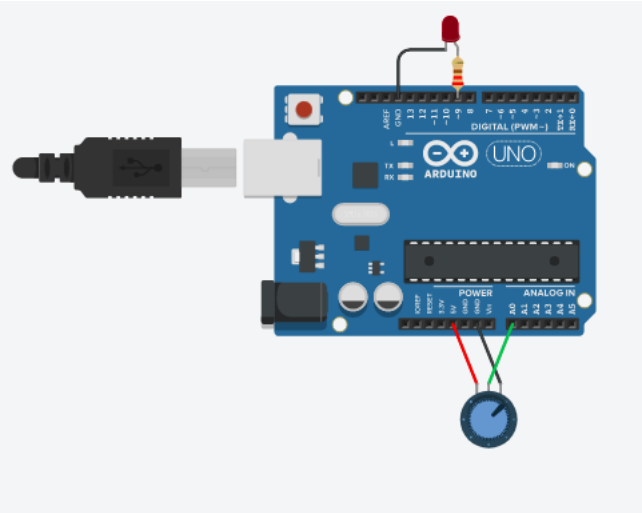

Figura 4: Montagem do circuito com potenciômetro, LED e o Arduino.

3.1.2 Parâmetros

- `pino`: o pino escolhido.
- `valor`: o *duty cycle*: entre 0 (sempre desligado) and 255 (sempre ligado).

3.2 O que fazer

1. Monte o circuito mostrado na Figura 4, utilizando uma *protoboard*.
2. Utilizando o código anterior, pegue a tensão lida no potenciômetro, através da função `analogRead()`, e use esse valor para variar o *duty cicle* do PWM que alimentará o LED. Para isso, leve em consideração que o valor lido pelo conversor AD é uma palavra de 10 bits e o *duty cicle* recebe valores de 8 bits (0 a 255). Para converter esse valor, o Arduino já possui a função `map(val, min_entrada, max_entrada, min_saida, max_saida)`, em que `val` é o valor que você quer converter. Para o nosso exemplo, queremos transformar um número de 0 a 1023 e um número de 0 a 255, logo a função fica `map(val,0,1023,0,255)`.

4 Controle de um Servomotor

O Servomotor é uma máquina, eletromecânica, que apresenta movimento proporcional a um comando, como dispositivos de malha fechada, ou seja: recebem um sinal de controle; que verifica a posição atual para controlar o seu movimento indo para a posição desejada com velocidade monitorada externamente sob *feedback* de um dispositivo sensor que pode ser um tacogerador ou sensor de efeito Hall ou encoder, dependendo do tipo de servomotor e aplicação.

Em contraste com os motores contínuos que giram indefinidamente, o eixo dos servo motores possui a liberdade de apenas cerca de 180^o graus (360^o em alguns modelos) mas são precisos quanto à sua posição.

Servos possuem três fios de interface, dois para alimentação e um para o sinal de controle. O sinal de controle utiliza o protocolo PWM, que possui três características básicas: largura mínima, largura máxima e taxa de repetição (frequência).

A largura do pulso de controle determinará a posição do eixo:

Figura 5: Servomotor que será usado na prática.

- largura máxima equivale ao deslocamento do eixo em $+90^o$ da posição central;
- largura mínima equivale ao deslocamento do eixo em -90^o;
- demais larguras determinam a posição proporcionalmente.

Em geral, a taxa de repetição é 50Hz (frequência base do PWM) e a largura do pulso do sinal de controle varia de 1 a 2ms. Porém um servo motor pode funcionar a 60Hz também.

4.1 Servomotores no Arduino

As características do sinal PWM que controla um servomotor são bem diferente das características do sinal PWM gerado pela função `analogWrite()`. Nesta função, a frequência base é 100 kHz, e o pulso varia entre 0 a 100 porcento do sinal. Para poder utilizar o Arduino para controlar um servomotor (o servo motor 9g SG90 mostrado na Figura 5) é necessário utilizar a biblioteca `Servo.h`. Para adiciona-la, basta utilizar o comando `#include <Servo.h>` no começo do arquivo do programa. Nessa biblioteca traz uma classe chamada `Servo` e funções para manipular o servomotor:

- `Servo nome_servo;`: cria um objeto *servo* com o nome que você colocar em `nome_servo` (crie esse objeto como global);
- `nome_servo.attach(pino)`: relaciona o objeto `nome_servo` ao pino digital passado em `pino`, lembre de escolher um pino que pode ser usado como saída PWM (coloque essa variável no `setup()`).
- `nome_servo.write(val)`: move o servomotor para a posição passada como parâmetro `val`: `val` = 0, servomotor na posição 0^o; `val` = 180, servomotor na posição 180^o;

4.2 O que fazer

1. Utilizando a montagem do circuito anterior, faça um código que movimenta o servomotor de 0 a 180 graus a partir do valor de tensão medido no potenciômetro.

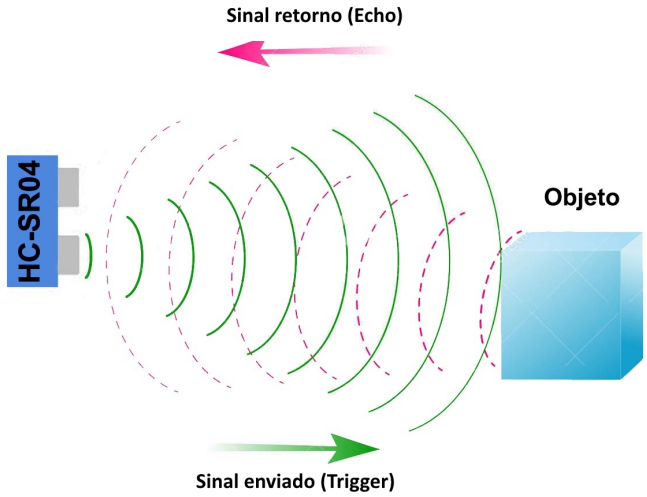

Figura 6: Esquema de como funciona um sensor ultrasom.

2. Primeiramente, utilize a interface serial (para ver o valor convertido de 0 a 1023 para 0 a 180) e o LED (para alimentar o sinal PWM, agora gerado pelo objeto `Servo`), para testar se o comportamento do circuito está como pedido.

3. Após o teste anterior, ligar o servomotor: Fio Marrom com GND, Fio Vermelho com 5v e Fio Laranja na Porta Digital 9 (porta usada pelo LED); e testar o circuito.

5 Sensor Ultrassônico

O Sensor Ultrassônico (usaremos o sensor HC-SR04) é um componente muito comum em projetos com sistemas embarcados, e permite que você faça leituras de distâncias entre 2 cm e 4 metros, com precisão de 3 mm. Pode ser utilizado simplesmente para medir a distância entre o sensor e um objeto, como para acionar portas, desviar um robô de obstáculos, acionar alarmes, etc.

O funcionamento do HC-SR04 se baseia no envio de sinais ultrassônicos pelo sensor, que aguarda o retorno (echo) do sinal, e com base no tempo entre envio e retorno, calcula a distância entre o sensor e o objeto detectado, ver Figura 6.

Primeiramente é enviado um pulso de 10μs, indicando o início da transmissão de dados, como mostrado na Figura 7. Depois disso, são enviado 8 pulsos de 40 KHz e o sensor então aguarda o retorno (em nível alto/high), para determinar a distância entre o sensor e o objeto, utilizando a equação

$$Distancia = \frac{(\text{Tempo echo em nível alto} \times \text{velocidade do som})}{2}.$$

Para ligação do sensor ao microcontrolador, são utilizados 4 pinos: **Vcc**, **Trigger**, **ECHO** e **GND**, como mostrado na Figura 8.

Poderíamos escrever nossa própria biblioteca para comunicarmos com o sensor ultrasom, mas já existem várias bibliotecas prontas na Internet. Usaremos a disponível em <https://github.com/filipeflop/Ultrasonic> e disponibilizada por pendrive (pedir ao professor). Para instalar essa biblioteca externa, basta ir em **Sketch** $\Rightarrow$ **incluir biblioteca** $\Rightarrow$ **Adicionar biblioteca .ZIP**, irá abrir uma janela para você procurar a pasta em que você deixou o

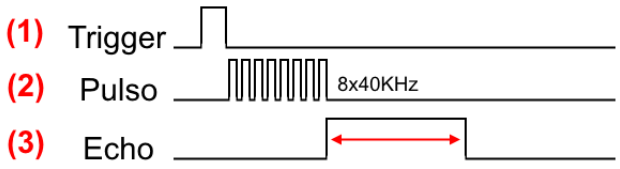

Figura 7: Sinal enviado pelo sensor ultrasom.

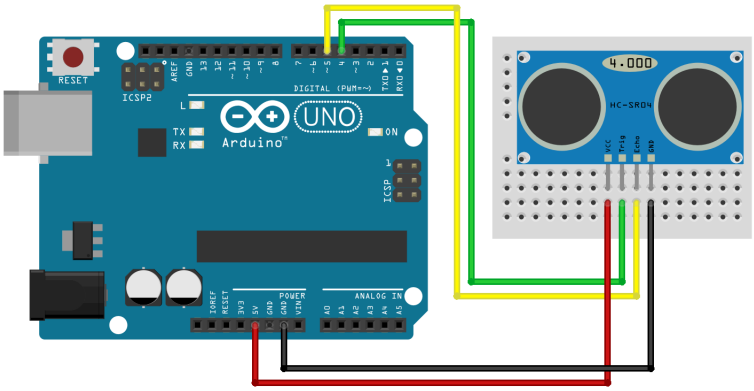

Figura 8: Ligação do sensor ultrassom no Arduino.

arquivo .ZIP com a biblioteca. Uma vez instalada, você pode incluir ela no seu código `#include <Ultrasonic.h>`

5.1 Como usar o Ultrasonic.h no Arduino

A biblioteca *Ultrasonic* é escrita em **C++**, assim como a *Servo* é escrita, por isso, vamos trabalhar com a **classe** `Ultrasonic` implementada na biblioteca que acabamos de instalar. Uma classe, em palavras bem simplistas, é uma estrutura que encapsula atributos (variáveis) e métodos (funções). Quando **instaciamos** uma classe, criamos um **objeto** desta classe. Cada objeto terá atributos e métodos próprios. Usando o exemplo da biblioteca que estamos usando, para criar um objeto:

- `Ultrasonic meuUltrassom(trig, echo);`: cria um objeto *Ultrasonic* com o nome que você colocar em `meuUltrassom`, e com os pinos passados em `trig` - o pino de *trigger* - `echo` - o pino que receberá o sinal de eco - crie esse objeto como global;

Para usar o objeto basca chamar algum método (função) que ele possua. No nosso caso, estamos interessados na medição de distância, logo, usaremos:

- `meuUltrassom.Ranging(CM);`: esse método retorna a distância para um aparato em centímetro (`CM`).

5.2 O que fazer

1. Utilizando uma *protoboard*, monte o circuito mostrado na Figura 8.
2. Utilizando a biblioteca *Ultrasonic*, crie um código que mede a distância a cada 200 ms e envia pela serial. Lembre-se de criar um objeto `Ultrasonic`

6 Ponte H

Utilizaremos o módulo Ponte H L298N Arduino. Esse módulo é projetado para controlar cargas indutivas como relés, solenóides, motores DC e motores de passo, permitindo o controle não só do sentido de rotação do motor, como também da sua velocidade, utilizando os pinos PWM do Arduino.

6.1 Especificações da Ponte H L298N

- Tensão de Operação: 4 35v;
- Chip: ST L298;
- Controle de 2 motores DC ou 1 motor de passo
- Corrente de Operação máxima: 2A por canal ou 4A max
- Tensão lógica: 5v
- Corrente lógica: 0 36mA
- Limites de Temperatura: -20 a +135°C
- Potência Máxima: 25W
- Dimensões: 43 x 43 x 27mm
- Peso: 30g

6.2 Funcionamento Ponte H L298N

Na Figura 9, é mostrado o módulo Ponte H L298n31 e usar partes constituintes:

- **(Motor A) e (Motor B)**: se referem aos conectores para ligação de 2 motores DC ou 1 motor de passo;
- **(Ativa MA) e (Ativa MB)**: são os pinos responsáveis pelo controle PWM dos motores A e B. Se estiver com *jumper*, não haverá controle de velocidade, pois os pinos estarão ligados aos 5v. Esses pinos podem ser utilizados em conjunto com os pinos PWM do Arduino.
- **(Ativa 5v) e (5v)**: este Driver Ponte H L298N possui um regulador de tensão integrado. Quando o driver está operando entre 6-35V, este regulador disponibiliza uma saída regulada de +5v no pino (5v) para um uso externo (com *jumper*), podendo alimentar por exemplo outro componente eletrônico. Portanto não alimente este pino (5v) com +5v do Arduino se estiver controlando um motor de 6-35v e jumper conectado, isto danificará a placa. O pino (5v) somente se tornará uma entrada caso esteja controlando um motor de 4-5,5v (sem *jumper*), assim poderá usar a saída +5v do Arduino.
- **(6-35v) e (GND)**: aqui será conectado a fonte de alimentação externa quando o driver estiver controlando um motor que opere entre 6-35v. Por exemplo, se estiver usando um motor DC 12v, basta conectar a fonte externa de 12v neste pino e (GND).
- **(Entrada)**: este barramento é composto por IN1, IN2, IN3 e IN4. Sendo estes pinos responsáveis pela rotação do Motor A (IN1 e IN2) e Motor B (IN3 e IN4).

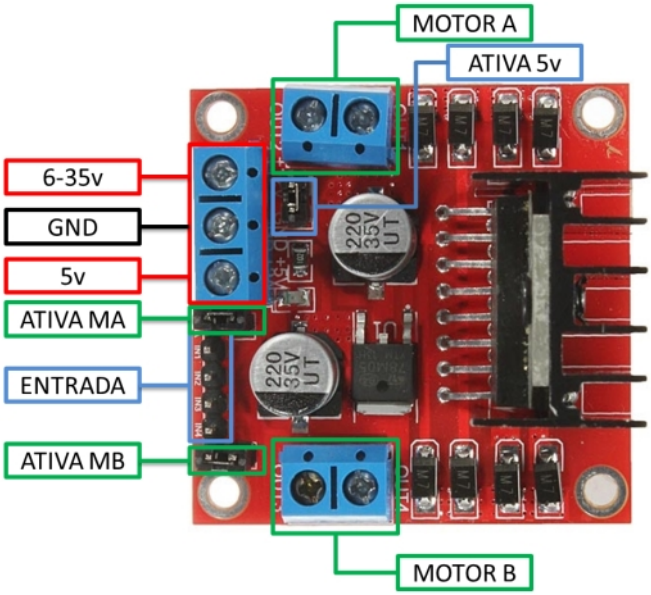

Figura 9: Módulo Ponte H L298n31 com suas partes constituintes em destaque.

A Tabela 2 mostra a ordem de ativação do Motor A através dos pinos IN1 e IN2. O mesmo esquema pode ser aplicado aos pinos IN3 e IN4, que controlam o Motor B. Os valores 5V e GND podem ser controlados por saídas digitais do Arduino em nível lógico 1, 5V, ou nível lógico 0, GND.

Tabela 1: Ordem de ativação do Motor A através dos pinos IN1 e IN2.

Sentido	IN1	IN2
Horário	5V	GND
Anti-Horário	GND	5V
Ponto-Morto	GND	GND
Freio	5V	5V

6.3 Ligação com o Arduino

Utilizaremos o módulo Ponte H sem o *jumpers* **Ativa MA** e **MB**, isso significa que controlaremos a velocidade dos motores individualmente a partir desses sinais: **Ativa MA** será um sinal PWM que controla a velocidade do **Motor A** e **Ativa MB** será um sinal PWM que controla a velocidade do **Motor MB**. A ligação dos sinais está feita como mostrado na Tabela 2.

6.4 O que fazer

Vamos controlar os dois motores ao mesmo tempo a partir de um potenciometro. Para isso, siga os passos listados a seguir:

1. Faça um código que lê o valor de tenção de um potenciômetro a cada 100ms e realiza o seguinte mapeamento:

Tabela 2: Ligação entre o módulo Ponte H e o Arduino.

Sinal	Pino
IN1	2
IN2	4
Ativa MA	3
IN3	6
IN4	7
Ativa MB	5

- Se o valor for maior ou igual a 512, os motores girarão em sentido horário e o valor do *duty cicle* será: 0% para `AnalogRead(PinoPotenciometro)`=512, e 100% para `AnalogRead(PinoPotenciometro)`=1023[2].
- Se o valor for menor que 512, os motores girarão no sentido anti-horário e o valor do *duty cicle* será: 0% para `AnalogRead(PinoPotenciometro)`=511, e 100% para `AnalogRead(PinoPotenciometro)`=0.

2. Utilize a serial para testar esse código: imprima na serial o sentido que os motores estão girando, os valores de IN1, IN2, IN3 e IN4, o valor do *duty cicle* em porcentagem e os valores de **Ativa MA** e **Ativa MB**.

3. Uma vez que esse código esta funcionando, teste na plataforma do carrinho com os motores reais.

7 Fontes

De onde eu tirei a maior parte desse texto:

- Funções já implementadas no Arduino: <https://www.arduino.cc/reference/pt/#functions>
- `Serial` no Arduino: <https://www.arduino.cc/reference/pt/language/functions/communication/serial/println/>
- `Serial` no Arduino: <https://www.arduino.cc/reference/pt/language/functions/communication/serial/print/>
- PWM: <https://www.embarcados.com.br/pwm-do-arduino/>
- Servomotor: <https://www.arduino.cc/en/Reference/Servo>
- Ultrassom: <https://www.filipeflop.com/blog/sensor-ultrassonico-hc-sr04-ao-arduino/>
- Ponte H: <https://www.filipeflop.com/blog/motor-dc-arduino-ponte-h-l298n/>

[2]OBS: Lembre de usar a função `map` para transformar o valor do *duty cicle* em um valor entre 0 e 255.