UNIVERSIDADE FEDERAL DE UBERLÂNDIA - UFU

FACULDADE DE ENGENHARIA ELÉTRICA – FEELT

SINAIS E SISTEMAS EM ENGENHARIA BIOMÉDICA

**Acionamento de Motor de CC**

**Alunos:**

1. Ítalo Gustavo Sampaio Fernandes - 11511EBI004
2. Nathalia Rodrigues - 11411EBI018
3. Paulo Camargos Silva - 11611EBI023

**Prof.** Sérgio Ricardo de Jesus Oliveira

Uberlândia, **18** de **outubro** de 2017

**Acionamento de Motor de CC com o L298**

**1 – Introdução**

O Arduino DUE possui 54 pinos digitais de entrada ou saída, onde 12 desses podem ser utilizados para controle PWM aumentando a capacidade de controle PWM se comparada a Arduino UNO que possui apenas 6 saídas PWM. Os conversores analógicos-digital (ADC) dessa placa são rápidos e podem operar em até 1 MHz de Frequência de amostragem.

Essa nova placa possui 12 entradas de ADC, com 12 bits de resolução. Além disso, a placa possui dois conversores digital-analógicos (DAC) de 12 bits, o que permite mais versatilidade a diferentes aplicações. É possível então, ter uma saída com um valor de tensão entre 0 e 3,3V, com 4096 diferentes valores, nesse intervalo. Pela existência destes conversores D/A, é possível gerar sinais senoidais, triangulares, quadradas, dente-de-serra e etc., com diversas frequências, de acordo com o período da onda analógica em questão.

Um [motor cc](http://www.citisystems.com.br/motor-cc/) nada mais é do que um motor alimentado por corrente contínua (CA), sendo esta alimentação proveniente de uma bateria ou qualquer outra de alimentação CC. A sua comutação (troca de energia entre rotor e estator) pode ser através de escovas (escovado) ou sem escovas (brushless) e com relação a velocidade, o [motor cc](http://www.citisystems.com.br/motor-cc/) pode ser controlado apenas variando a sua tensão, diferentemente de um [motor elétrico](http://www.citisystems.com.br/motor-eletrico/) de corrente alternada (CA) cuja a velocidade é variada pela frequência. Vejamos mais a frente como funciona este tipo de motor.

O princípio básico de funcionamento do motor CC é o seguinte: “Sempre que um condutor conduzindo uma corrente elétrica (em vermelho) é colocado em um campo magnético (em azul), este condutor experimenta uma força mecânica (em verde)” gerando o torque e o giro do eixo do motor.

As pontes H em possuem este nome devido ao formato que é montado o circuito, semelhante a letra H. O circuito utiliza quatro chaves (S1, S2, S3 e S4) que são acionadas de forma alternada, ou seja, (S1-S3) ou (S2-S4). Dependendo da configuração entre as chaves teremos a corrente percorrendo o motor ora por um sentido, ora por outro. O CI L298N é muito utilizado para o propósito de controle de motores, ele nada mais é que uma ponte H em um componente integrado. Uma das vantagens do uso desse CI é o menor espaço ocupado, a baixa complexidade do circuito e o fato de ele já possuir dois circuitos H, podendo assim, controlar dois motores. Uma das vantagens do L289 e que se no lugar de usar sinais lógicos TTL for usado sinais de PWM, é possível regular a tensão de saída, e dessa forma regular a velocidade dos motores.

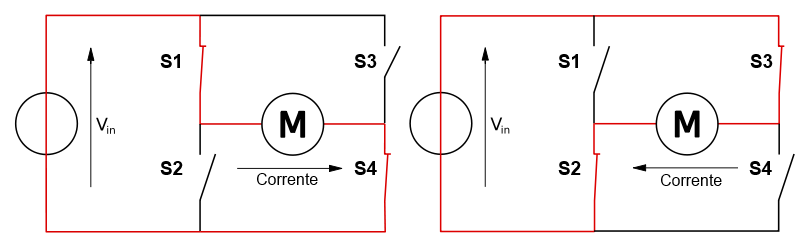


Figura nº 1: Explicação do funcionamento de uma ponte H.

**2 – Materiais e Métodos**

Os materiais utilizados neste experimento foram:

* Arduino;
* Módulo L298;
* Osciloscópio e Multímetro;
* 1 Motor CC;
* 1 Potenciômetro;
* Fonte de Alimentação 5v.

**2.1 – Esquema do circuito**

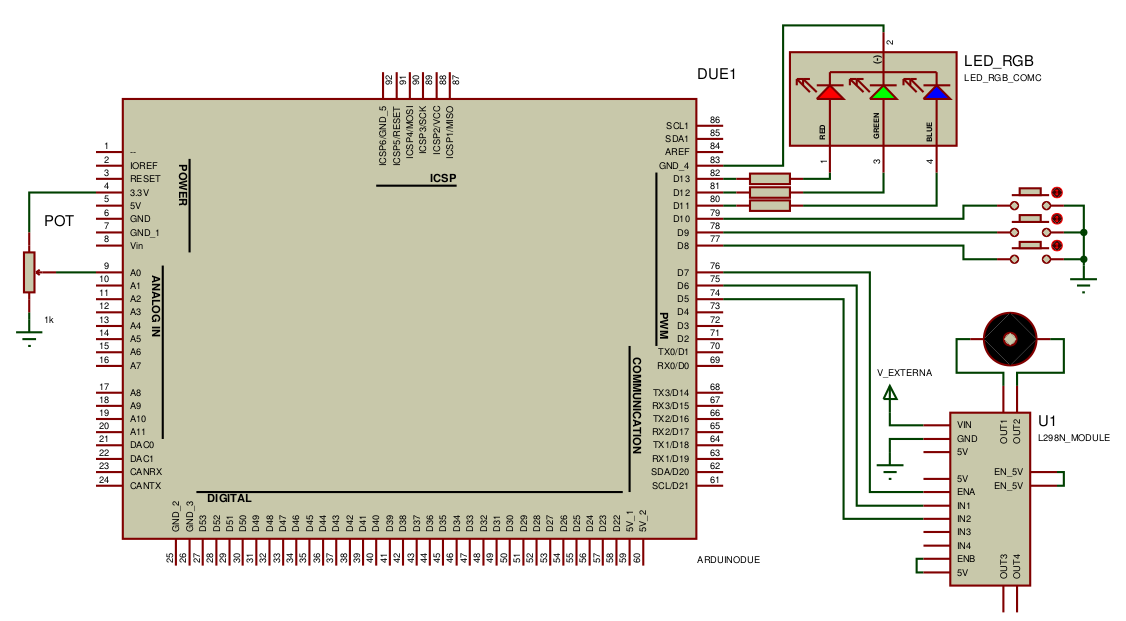


Figura 3: Esquema do circuito montado para o controle do motor CC com o módulo L298N.

**2.2 – Código**

O código que se encontra nos anexos I e II, foi dividido em dois programas. Primeiramente, foi escrita uma biblioteca (classe Motor.h, Anexo II) para gerenciamento do módulo L298N, e a partir das funções implementadas nesta biblioteca, o programa .ino (Anexo I) foi escrito de forma que fosse reduzido. A biblioteca Motor.h realiza as funções descritas no roteiro desta aula prática e o programa. ino coordena quais métodos serão executados.

No programa. ino, as variáveis de leitura dos botões liga e desliga, freio magnético e potenciômetro são declaradas e configuradas como saída e entrada. Também ocorre a criação de um objeto da classe Motor.h e seus pinos IN1, IN3 e ENABLE são configurados como OUTPUT. No interior da função *void loop()* ocorre a chamada da função para leitura da tensão no pino do potenciômetro. Posteriormente, é verificado o estado botões de liga-desliga e freio magnético.

Caso o freio seja pressionado, o método da classe Motor.h responsável pela ativação é chamado e executado. Caso contrário, o botão liga-desliga é verificado. Se estiver ligado, o motor funciona normalmente, chamando o método *rodaMotor(float tensao)* que irá gerenciar o sentido de rotação de acordo com o parâmetro passado *tensão*, caso contrário, o motor é desligado. O diagrama abaixo exibe a sequência de passos de funcionamento do código.

Setup:

Pinos de Saída:

IN1, IN2 e ENA

Pinos de Entrada:

BTN1, BTN2, BTN3

Inicia a Porta Serial.

Loop:

Lê Valor do Conversor AD:

*analogRead(pin)*

Converte para Tensão:

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

Lê Botões: *digitalRead(pin)*

Ligado?

False

Freio?

True

Escreve HIGH em IN1 e IN2:

*digitalWrite(pin)*

Deixa a velocidade em Máximo:

*analogWrite(pin)*

True

False

Converte tensão para velocidade e direção.

Seta a velocidade: *analogWrite(pin)*

Seta a Direção (IN1 sendo o contrário de IN2): *digitalWrite(pin)*

Figura 4 - Diagrama do Código

**3 – Resultados e discussões**

Foi possível observar o funcionamento adequado do motor, tendo como resultado um sistema capaz de alterar a velocidade desde tanto quanto alterar sua direção e freá-lo.

Notou-se que em baixas velocidades o motor não apresentou torque suficiente para romper a inércia, ele começava a se movimentar somente após um valor mínimo de tensão média vinda do PWM.

Também foi possível notar que quando o potenciômetro estava próximo ao seu valor central o sinal de direção do movimento se alternava, entrando isso não movimentava o motor devido ao duty cycle próximo a 0% no PWM. Essa alternância na direção se dá a características do conversor AD e do sistema físico, fazendo com que a tensão lida varie em torno de 1.65 V com erros de 3.23 mV (no caso da resolução de 10 bits e fundo de escala de 3.3 V) para mais ou para menos.

Um terceiro tópico importante a ser discutido foi a velocidade de parada do freio magnético, após medidas rápidas com um cronômetro foi possível notar um tempo ligeiramente menor para a parada com o freio magnético e um som característico de parada, mas ambos foram muito rápidos comparados ao tempo de reação do observador que manipulava o cronômetro. Em trabalhos futuros é sugerido o uso de um *encoder* para verificar a melhora do tempo de parada em termos quantitativos.

**4 – Conclusão**

Com a variação da tensão aplicada na porta analógica A0 do Arduino, que a partir da mudança na chave do potenciômetro, podemos gerar valores em unidade respectivos ao valor de tensão analógica. Estes valores podem ser utilizados como controle da velocidade e sentido de rotação do motor usando sinal PWM. Foi possível também criar uma biblioteca com as funções exigidas no roteiro e observar o funcionamento juntamente com o código. ino.

**5 – Referências**

1. https://www.arduino.cc/en/Tutorial/Debouncexdf
2. https://portal.vidadesilicio.com.br/modulo-ponte-h-l298n-arduino/
3. https://www.citisystems.com.br/motor-cc/

**ANEXO I - CÓDIGO. ino PARA MOTOR**

1. #include "Motor.h" // Import classe Motor.h, que deve estar na mesma pasta deste arquivo .ino
3. Motor \*motor;
5. #define DANCE\_BUTTON 8    // Ignorar. Apenas funcao extra
6. #define LD\_BUTTON 7       // Botão liga-desliga do sistema
7. #define FR\_BUTTON 6       // Botão frei magnético do sistema
8. #define POT\_V A0          // Pino de leitura da tensão no potenciometro
10. //Variaveis para debouncing do LD\_BUTTON
11. bool chButtonState, lastChButtonState = 0;
12. unsigned long lastDeboucingTime = 0, debounceDelay = 50;
13. bool estadoSistema = 1;
14. bool teste;
16. void setup() {
17. /\* Informações da classe motor\*/
18. // Cria um objeto 'motor' da classe 'Motor' e inicializacao dos pinos Motor(IN1, IN2, ENABLE);
19. motor = new Motor(13, 12, 10);
20. pinMode(motor->getIN1(), OUTPUT);     // Configuracao do pino IN1 como OUTPUT
21. pinMode(motor->getIN2(), OUTPUT);     // Configuracao do pino IN2 como OUTPUT
22. pinMode(motor->getEnable(), OUTPUT);  // Configuracao do pino ENABLE como OUTPUT
24. //Variáveis internas do programa (botoes)
25. pinMode(DANCE\_BUTTON, INPUT\_PULLUP);  // Configuracao do pino DANCE\_BUTTON como INPUT\_PULLUP
26. pinMode(LD\_BUTTON, INPUT\_PULLUP);     // Configuracao do pino LD\_BUTTON como INPUT\_PULLUP
27. pinMode(FR\_BUTTON, INPUT\_PULLUP);     // Configuracao do pino FR\_BUTTON como INPUT\_PULLUP
29. Serial.begin(9600);                   // Inicializacao da porta serial
30. Serial.println(" -------- PROGRAMA EM FUNCIONAMENTO -------- ");
31. }
33. void loop() {
34. float f\_tensao = calculaTensao();       // Chamada da funcao para calculo da tensao no potenc.
35. //Ativa o freio mag. se o motor estiver funcionando e a chave FR\_BUTTON pressionada
36. if (!digitalRead(FR\_BUTTON) && ligaDesliga()) {
37. motor->stopMotor(f\_tensao);          // Metodo do objeto 'motor' para freio magnetico
38. } else      // Operador ternario
39. // Se o sistema estiver ligado, faz o motor funcionar. Caso contrario, desliga motor.
40. ligaDesliga() ? motor->rodaMotor(f\_tensao) : motor->desligaMotor();
41. }
43. float calculaTensao() {  // funcao para calculo da tensao no potenc.
44. return analogRead(POT\_V) \* 3.3 / 1023;
45. }
47. bool ligaDesliga() { // Debounce do botao liga desliga. Retorna 1 para ligado, 0 para desligado.
48. // Ver referencia em https://www.arduino.cc/en/Tutorial/Debounce para entendimento da funcao
49. int chReading = !digitalRead(LD\_BUTTON);
50. if (chReading != lastChButtonState) {
51. lastDeboucingTime = millis();
52. }
54. if ((millis() - lastDeboucingTime) > debounceDelay) {
55. if (chReading != chButtonState) {
56. chButtonState = chReading;
57. if (chButtonState) {
58. estadoSistema = !estadoSistema;
59. }
60. }
61. }
62. lastChButtonState = chReading;
63. return estadoSistema;

**ANEXO II - BIBLIOTECA Motor.h ESCRITA PARA MOTOR E MÓDULO L298N**

1. /\* DOCUMENTACAO CLASSE MOTOR
2. Descricao: Classe para controle de motores utilizando o modulo l298n.
3. Atributos:
4. \_in1 - Pino IN1 conectado do modulo.
5. \_in2 - Pino IN2 conectado do modulo.
6. \_enable - Pino ENABLE do m�dulo.
7. \_sentido  - variavel do tipo enum para armazenamento o sentido de rotaacao
8. do motor.
9. \_last\_time - Variavel auxiliar para funcao millis();
10. Metodos:
11. getters() e setters();
12. void rodaMotor(float f\_tensao) - Inicia a rotacao de acordo com a tensao f\_tensao.
13. - Para f\_tensao = 0, motor parado.
14. - Para f\_tensao >= 3.3/2, sentido horario.
15. - Para f\_tensao <= 3.3/2, sentido anti-horario.
16. void dancaMotor(int tempo) - Gira motor com sentido variado e intervalo definido (tempo).
17. void giraHorario(float tensao) - Gira motor girar em sentido horario.
18. void giraAntiHorario(float tensao) - Gira motor girar em sentido anti-horario.
19. void stopMotor(float tensao) - Para o motor.
20. void desligaMotor() - Desativa (0) os pinos conectados ao madulo.
21. \*/
22. enum sentido {ANTI\_HORARIO, HORARIO, PARADO};
24. class Motor {
25. private:
26. int \_in1;
27. int \_in2;
28. int \_enable;
29. sentido \_sentido;
30. int \_last\_time; //Variavel auxiliar para a fun��o millis(), ignorar.
31. public:
32. Motor(int in1, int in2, int enable);
33. int getIN1(); void setIN1(int in1);
34. int getIN2(); void setIN2(int in2);
35. int getEnable(); void setEnable(int veloc);
36. sentido getSentido(); void setSentido(sentido s);
37. void rodaMotor(float f\_tensao);
38. void dancaMotor(float f\_tensao, int tempo);
39. void giraHorario(float tensao);
40. void giraAntiHorario(float tensao);
41. void stopMotor(float tensao);
42. void desligaMotor();
43. };
45. Motor::Motor(int in1, int in2, int enable) {
46. this->\_in1 = in1;
47. this->\_in2 = in2;
48. this->\_enable = enable;
49. this->\_last\_time = 0;
50. }
52. int Motor::getIN1() {
53. return this->\_in1;
54. }
56. int Motor::getIN2() {
57. return this->\_in2;
58. }
60. int Motor::getEnable() {
61. return this->\_enable;
62. }
64. sentido Motor::getSentido() {
65. return this->\_sentido;
66. }
68. void Motor::setIN1(int in1) {
69. this->\_in1 = in1;
70. }
72. void Motor::setIN2(int in2) {
73. this->\_in2 = in2;
74. }
76. void Motor::setEnable(int enable) {
77. this->\_enable = enable;
78. }
79. void Motor::setSentido(sentido s) {
80. this->\_sentido = s;
81. }
83. void Motor::giraHorario(float f\_tensao) {
84. int cicloPWM = abs(f\_tensao - 1.65) \* 255.0 / 1.65;
85. //int cicloPWM = map(tensao, 330 / 2, 330, 0, 255);
86. setSentido(HORARIO);
87. digitalWrite(getIN2(), 0);
88. digitalWrite(getIN1(), 1);
89. analogWrite(getEnable(), cicloPWM);
90. int t\_mv = 1000\*f\_tensao;
91. Serial.print("-- PWM: " + String (cicloPWM)+"\t");
92. Serial.println("-- SENTIDO HORARIO | TENSAO: " + (String) f\_tensao + " V");
93. }
95. void Motor::giraAntiHorario(float f\_tensao) {
96. int   cicloPWM = abs(f\_tensao - 1.65) \* 255.0 / 1.65;
97. //int tensao = f\_tensao \* 100;
98. //int cicloPWM = map(tensao, 0, 330 / 2, 0, 255);
99. setSentido(ANTI\_HORARIO);
100. digitalWrite(getIN1(), 0);
101. digitalWrite(getIN2(), 1);
102. analogWrite(getEnable(), cicloPWM);
103. int t\_mv = 1000\*f\_tensao;
104. Serial.print("-- PWM: " + String (cicloPWM)+"\t");
105. Serial.println("-- SENTIDO ANTI-HORARIO | TENSAO: " + (String) f\_tensao + " V");
106. }
108. void Motor::stopMotor(float f\_tensao) { // Freio magn�tico do motor
109. setSentido(PARADO);
110. digitalWrite(getIN1(), 1);
111. digitalWrite(getIN2(), 1);
112. analogWrite(getEnable(), 255);
113. int t\_mv = 1000\*f\_tensao;
114. Serial.println("-- MOTOR PARADO | TENSAO: " + (String) f\_tensao + " V");
115. }
117. void Motor::rodaMotor(float tensao) {
118. if (tensao == 3.3 / 2)
119. desligaMotor();
120. else
121. tensao > 3.3 / 2 ? giraHorario(tensao) : giraAntiHorario(tensao);
122. }
124. void Motor::dancaMotor(float f\_tensao, int tempo) {
125. int tensao = f\_tensao;
126. if (millis() - \_last\_time > tempo) {
127. \_last\_time = millis();
128. \_sentido ? giraHorario(map(tensao, 0, 330, 165, 330)) : giraAntiHorario(map(tensao, 0, 330, 0, 165));
130. }
131. }
133. void Motor::desligaMotor() {
134. setSentido(PARADO);
135. digitalWrite(getIN1(), 0);
136. digitalWrite(getIN2(), 0);
137. analogWrite(getEnable(), 0);
138. Serial.println("-- MOTOR DESLIGADO ---------- ");
139. }
140. // --------------------------- FIM DA CLASSE Motor.h -----------------