DEPARTAMENTO DE TECNOLOGIA EM ELETRO-ELETRÔNICA

COORDENAÇÃO DE ENGENHARIA INDUSTRIAL ELÉTRICA

LEANDRO DINIZ FERREIRA

CONTROLE PID APLICADO A UM ROBÔ MANIPULADOR ELETROMECÂNICO COM O AUXÍLIO DO MICROCONTROLADOR ARDUINO

SALVADOR, BA

SETEMBRO DE 2014

LEANDRO DINIZ FERREIRA

CONTROLE PID APLICADO A UM ROBÔ MANIPULADOR ELETROMECÂNICO COM O AUXÍLIO DO MICROCONTROLADOR ARDUINO

Trabalho apresentado ao professor Eduardo Telmo da disciplina microprocessamento e microcontrole como requisito parcial para obtenção da média semestral.

SALVADOR, BA

SETEMBRO DE 2014

SUMÁRIO

1 INTRODUÇÃO 1

1.1 OBJETIVOS 1

1.1.1 Objetivo Geral 1

1.1.1 Objetivos Específicos e Metodologia 2

2 ESTRUTURA DO RD5NT 2

3 ESTRATÉGIA DE CONTROLE PID 4

4 COMPONENTES DO CIRCUITO 5

4.1 O MICROCONTROLADOR – ARDUINO MEGA 6

4.2 PONTE H DUPLA L298 7

4.3 FONTE 9

5 CONTROLE DO POSICIONAMENTO DO PULSO 10

6 PROGRAMA DE ACIONAMENTO DO ROBÔ RD5NT 13

7 ANÁLISE DOS RESULTADOS E CONCLUSÕES 14

REFERÊNCIAS BIBLIOGRÁFICA 16

ANEXO - PROGRAMAS 17

1 INTRODUÇÃO

A aplicação de robôs na indústria evidencia numerosos benefícios, dos quais pode-se enumerar o aumento da produtividade, consistência e melhoria da qualidade do produto final, confiabilidade do processo, facilidade de programação e substituição da atividade de determinados robôs, operação em ambientes insalubres e de alta periculosidade, entre outros.

Dessa forma, a implementação de braços robóticos para execução de determinadas tarefas como pintura, soldagem, movimentação de cargas, inspeção de produtos e montagem é sinônimo de agilidade e precisão com segurança na indústria, sem expor a vida humana.

Em contrapartida, além dos ambientes hostis, a utilização de braços robóticos somente na linha de produção tem contrastado com aplicação em locais antes não imaginados. É o caso da companhia C & S Wholesale Grocers, a maior distribuidora de produtos alimentícios dos Estados Unidos, que está empregando robôs para carregar, organizar e armazenar seus estoques de forma mais eficiente do que a mão de obra humana[5].

O processo de automação industrial também tem levantado muitas discussões a respeito do desemprego de trabalhadores. No entanto, especialistas defendem que a aplicação de novas tecnologias apenas substitui a qualificação profissional, gerando empregos em outras áreas. De fato, a presença do trabalho humano é extremamente importante no processo de inovação e questionamento a respeito do processo produtivo, bem como na programação dos robôs. Sendo assim, a familiarização dos profissionais da área com os braços eletromecânicos deve fazer parte da sua qualificação.

1.1 OBJETIVOS

1.1.1 Objetivo Geral

O objetivo geral deste estudo é implementar um circuito capaz de acionar e controlar um braço robótico didático, o RD5NT fabricado pela Didacta Itália, colocando-o numa posição desejada.

1.1.1 Objetivos Específicos e Metodologia

I. Esquematizar e montar um circuito amplificador utilizando o circuito integrado L298N.
II. Utilizar o Arduino Mega como placa controladora para efetuar os ensaios de coleta dos valores do potenciômetro e em seguida convertê-los em graus.
III. Implementar e gravar no microcontrolador do Arduino Mega, ATmega1280, um programa que seja capaz de controlar o posicionamento do braço robótico através da estratégia de controle PID.

2 ESTRUTURA DO RD5NT

O robô RD5NT é um braço eletromecânico composto de cinco graus de liberdade fabricado pela empresa Didacta Itália. Trata-se de um robô com quatro eixos de rotação e uma garra, sendo que cada eixo e a garra possuem um motor DC e um potenciômetro rotativo linear que funciona como um *encoder*.

Os motores DC possuem velocidade de rotação alta sendo necessária relação de engrenagens capazes de reduzir a velocidade do eixo. Sendo assim, a transmissão de cada movimento é feita através de um bloco redutor que inclui o motor DC e as engrenagens.

Os potenciômetros acompanham o movimento do eixo e da garra, modificando sua resistência de forma linear. O aspecto linear assegura o posicionamento angular de cada eixo, possibilitando um mapeamento da queda de tensão sobre a resistência daquela posição angular. Por exemplo, o pulso representado na Figura 1 possui liberdade de rotação que vai de 0º a aproximadamente 360º. Como o valor do potenciômetro é ajustável de 0 a 5kΩ, é possível estabelecer uma relação linear entre a posição angular e o valor da resistência lida:

$$\theta = \frac{360}{5000} R_f \tag{1}$$

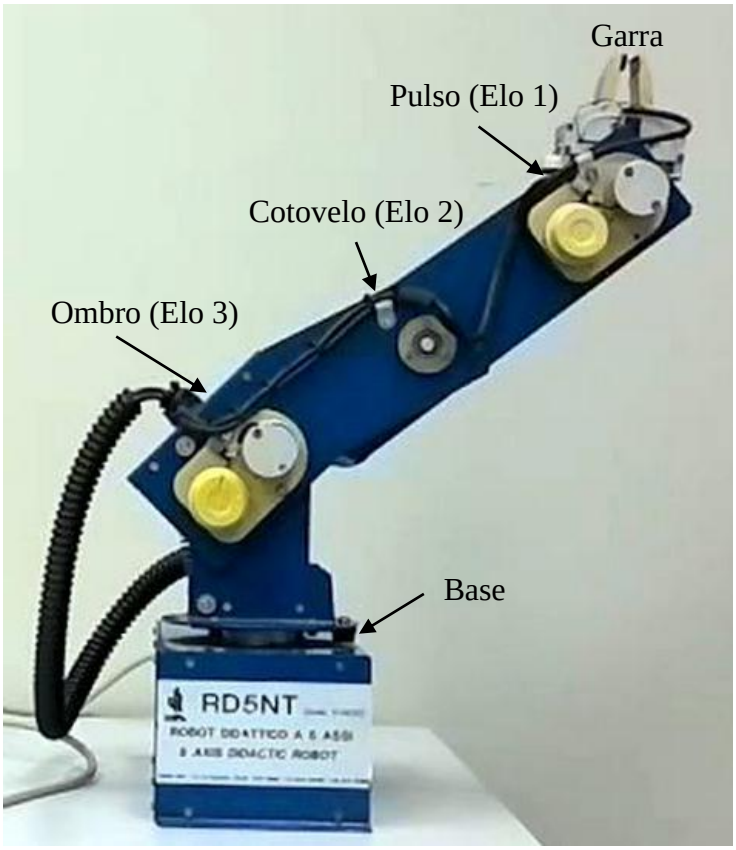


Figura 1 - Robô RD5NT

A tabela relaciona os arcos em graus de rotação de cada elo. Os valores da resistência foram lidos com o auxílio do Arduino em valores binários e posteriormente convertidos em graus utilizando através da Eq. (1).

Tabela 1 – Arcos permissíveis de cada eixo

Eixo	Arcos de rotação
Pulso	0 a 360º
Cotovelo	0 a 293º
Ombro	0 a 96º
Base	0 a 291º

A Figura 2 mostra o conector DB fêmea de 25 pinos, que fica localizado na base fixa do robô manipulador. É através desse conector que ocorre a troca de dados entre o robô manipulador e o microcontrolador do Arduino Mega. A Tabela 2 relaciona os pinos para alimentação dos motores de corrente contínua e os que recebem os sinais de saída do potenciômetro e também a alimentação dos potenciômetros (5V).

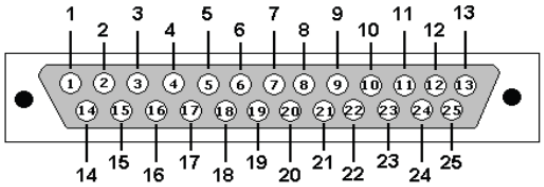


Figura 2 – Relação de pinos do RD5NT

Tabela 3 - Pinagem do RD5NT

Pinos	Função	Pinos	Função
1	Motor 1	14	Motor 1
2	Motor 2	15	Motor 2
3	Chave de fim-de-curso	16	Não utilizado
4	Motor 3	17	Motor 3
5	Motor 4	18	Motor 4
6	Motor 5	19	Motor 5
7	Não utilizado	20	Potenciômetro 4
8	Não utilizado	21	Potenciômetro 3
9	Não utilizado	22	Potenciômetro 4
10	Não utilizado	23	Não utilizado
11	Não utilizado	24	Potenciômetro 1
12	Não utilizado	25	Energização potenciômetro
13	Energização potenciômetro		

3 ESTRATÉGIA DE CONTROLE PID

Tendo como objetivo geral do projeto o controle de posicionamento dos eixos, escolheu-se a estratégia de controle proporcional-integral-derivativo (PID) devido a sua difusão nesse tipo de objetivo.

O PID é um mecanismo de controle de malha fechada amplamente utilizada em sistemas industriais. Basicamente, um controlador PID efetua a subtração entre uma variável de processo medida e um ponto de ajuste desejado.

O algoritmo de controle PID envolve três parâmetros: o proporcional, os valores integrais e derivados. Simplificando, estes valores podem ser interpretados em termos de tempo: P depende do erro presente, I sobre a acumulação de erros no passado, e D é uma previsão de futuros erros, com base na taxa atual de mudança. A soma ponderada destas três ações é usada para ajustar o processo por meio de um elemento controlado, tal como a posição de uma válvula de controle, um amortecedor, ou a potência fornecida a um elemento de aquecimento[4].

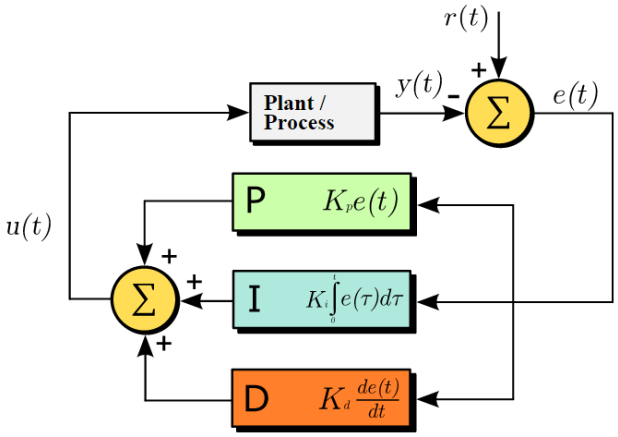


Figura 3- Malha de controle PID

Na ausência de conhecimento do modelo matemático do processo, o controlador PID tem sido aplicado com mais frequência. Ao ajustar os três parâmetros do algoritmo controlador PID, o controlador pode proporcionar uma ação de controle concebido para as necessidades específicas do processo. A resposta do controlador pode ser descrita em termos de capacidade de resposta do controlador a um erro, o grau em que o controlador ultrapassa o valor de ajuste, e o grau de oscilação do sistema. Nota-se que o uso do algoritmo PID para controle não garante um controle otimizado do sistema ou sistema de estabilidade.

4 COMPONENTES DO CIRCUITO

O diagrama de blocos e componentes do circuito é mostrado na Figura 4. A escrita e gravação do programa no microcontrolador é efetuada com o auxílio no notebook. O microcontrolador lê o posicionamento de cada eixo rotativo e executa o programa. Em seguida, envia para as saídas PWM valores de tensão que serão amplificados no circuito amplificador com L298 para então acionar os motores cc. A posição angular é modificada com o acionamento do motor cc e sua nova posição é lida no próximo ciclo do microcontrolador.

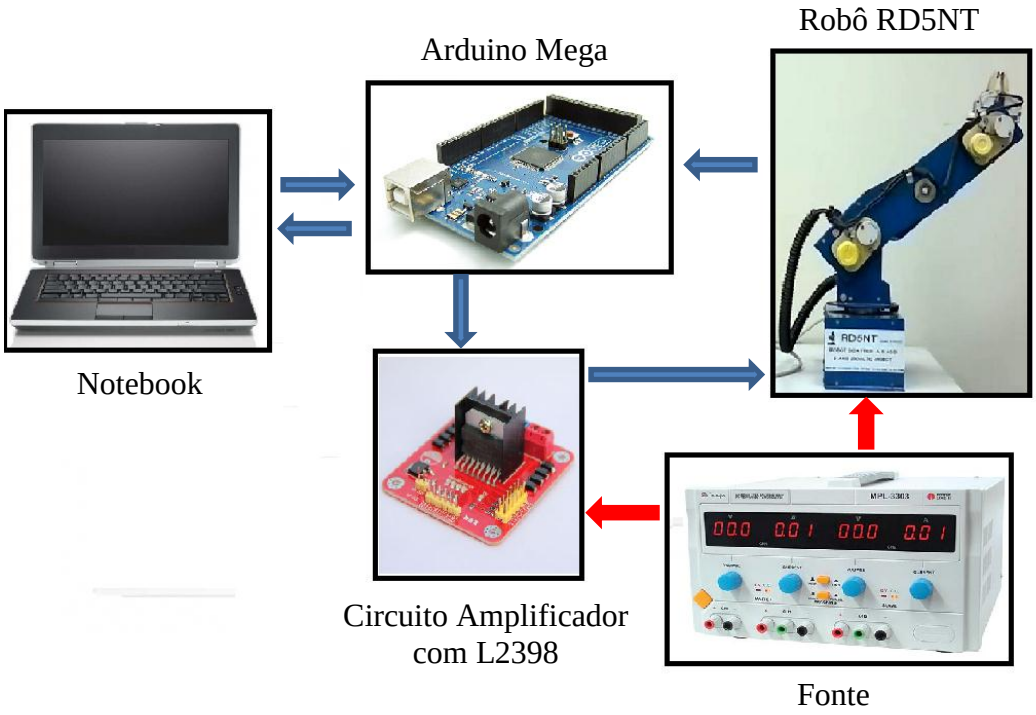


Figura 4- Diagrama de blocos e componentes do circuito

4.1 O MICROCONTROLADOR – ARDUINO MEGA

O circuito de acionamento dos motores necessita de um microcontrolador para efetuar a leitura do posicionamento angular de cada eixo, comparar com a posição desejada (*Setpoint*) e efetuar o controle PID. Sendo assim, devido a praticidade, utilizaremos o Arduino Mega como elemento de processamento do circuito.

O Arduino Mega é uma placa de microcontrolador baseado no ATmega1280. Ele possui 54 pinos digitais de entrada / saída (dos quais 14 podem ser usados como saídas PWM), 16 entradas analógicas, 4 UARTs (portas seriais de *hardware*), um cristal oscilador de 16 MHz, uma conexão USB, um *jack* de alimentação, protocolo de comunicação ICSP, e um botão de reset. Ele contém tudo que é necessário para apoiar o microcontrolador, basta conectá-lo a um computador com um cabo USB ou ligá-lo com um adaptador AC para DC ou bateria para começar. O Mega é compatível com a maioria dos *shields* projetados para o Arduino Duemilanove ou Diecimila.

Figura 5 - Arduino Mega

Fonte: Imagem disponível em: <http://arduino.cc/en/Main/arduinoBoardMega>. Acesso em 14/09/2014

Para o projeto de acionamento do robô RD5NT utilizaremos as saídas PWM para acionamento dos motores DC por meio de pontes H, as entradas analógicas para leitura de posição do eixo rotativo através do potenciômetro linear e as saídas digitais para controle dos *enables* dos respectivos motores através da ponte H. As saídas PWM serão usadas para gerar tensões com valores controlados pelo programa.

4.2 PONTE H DUPLA L298

O L298 é um circuito integrado de ponte H dupla desenvolvida para aceitar padrões de nível lógico TTL e acionar cargas indutivas tais como relés, solenoides, motores DC e motores de passo. Dois "*enables*" foram desenvolvidos para permitir ou não permitir o acionamento da carga independentemente uma da outra. O L298 foi utilizado no projeto para acionar os motores CC presente nos elos do braço do robô RD5NT.

O diagrama de blocos é mostrado na Figura 6. Os emissores dos transistores mais inferiores de cada ponte são conectados juntos e no terminal correspondente pode ser adicionado um resistor externo como um sensor de corrente.

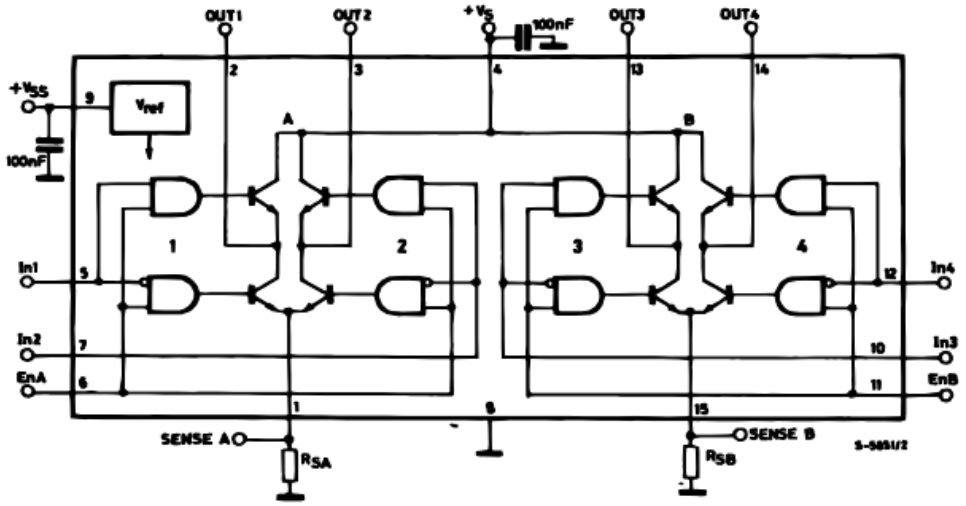


Figura 6 - Diagrama de blocos do L298

Fonte – Datasheet L298. STMicroelectronics

O CI foi configurado de forma que a alimentação lógica seja independente da alimentação de força, fazendo com que o circuito lógico trabalhe numa tensão menor. Existe dois tipos de circuito fabricado pela STMicroelectronics: a Multiwatt15 e a PowerSO20. Neste projeto utilizou-se a Multiwatt15 mostrada na Figura 7.

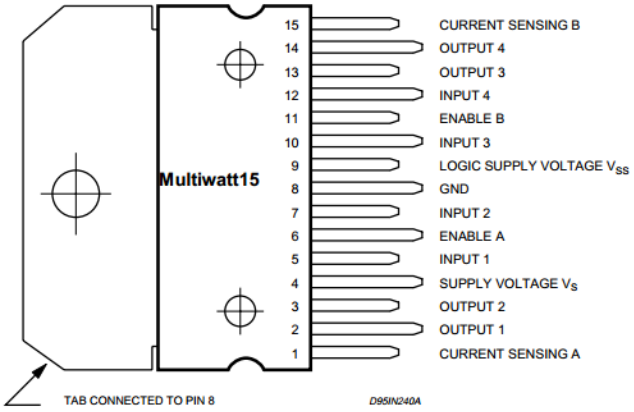


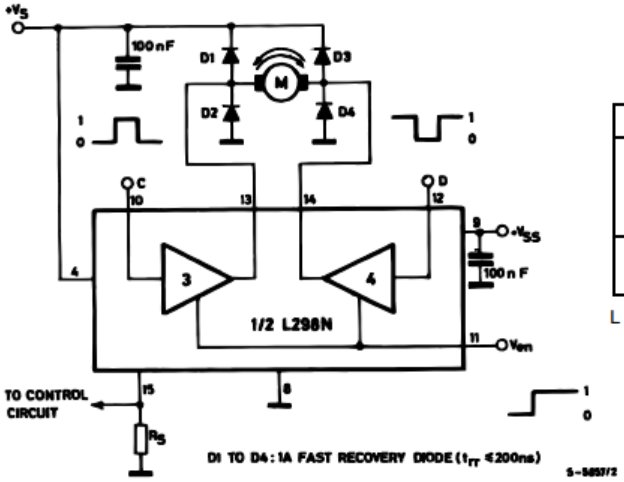
Figura 7- Configuração Multiwatt15 utilizado no projeto do robô

Fonte – Datasheet L298. STMicroelectronics

As tensões de operação que são recomendadas pelo fabricante são: tensão de alimentação 12V e tesão lógica 5V. E são essas tensões que foram utilizadas no projeto.

O método de acionamento de um motor CC bidirecional é mostrado na Figura 8, juntamente com a combinação de entradas e a direção de rotação do motor. Os diodos

efetuam o papel de proteção contra correntes reversas e o capacitor o de tornar a tensão mais estável.



Inputs		Function
V_{en} = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
V_{en} = L	C = X ; D = X	Free Running Motor Stop

L = Low H = High X = Don't care

Figura 8 - Método de acionamento de um motor DC bidirecional

Fonte – Datasheet L298. STMicroelectronics

Cada L298 é capaz de acionar dois motores CC bidirecionalmente, sendo, portanto, necessário três CI L298 para acionarem cinco motores CC presentes na estrutura do braço automático RD5NT.

4.3 FONTE

No projeto, foi utilizada a fonte de alimentação simétrica DC Digital Minipa. Trata-se de um equipamento digital de bancada, com quatro displays de 3 dígitos (duas de tensão e duas de corrente), capaz de fornecer duas saídas variáveis com tensão de 0 a 30V DC e corrente de 0 a 3A DC, e uma saída fixa de 5V / 3A DC. Possui proteção de sobrecarga e inversão de polaridade, e a duas saídas variáveis podem ser ligadas em série ou paralelo através do painel frontal.

A fonte foi responsável por alimentar o circuito de força (7-12V) e o circuito lógico 5V. Já o Arduino foi alimentado através da porta USB.

Figura 9 - Fonte de alimentação utilizada.

Fonte: Imagem e descrição disponível em:< http://www.mreferramentas.com.br/>

5 CONTROLE DO POSICIONAMENTO DO PULSO

Inicialmente, estudaremos o controle de posição de somente um motor. O esquemático da Figura 10 mostra o circuito de acionamento do motor DC do Pulso do robô manipulador. Os pinos que serão utilizados do Arduino são:

- Entrada analógica: A1
- Saídas PWM de acionamento: 6 e 7
- Saída digital para o *enable*: 33

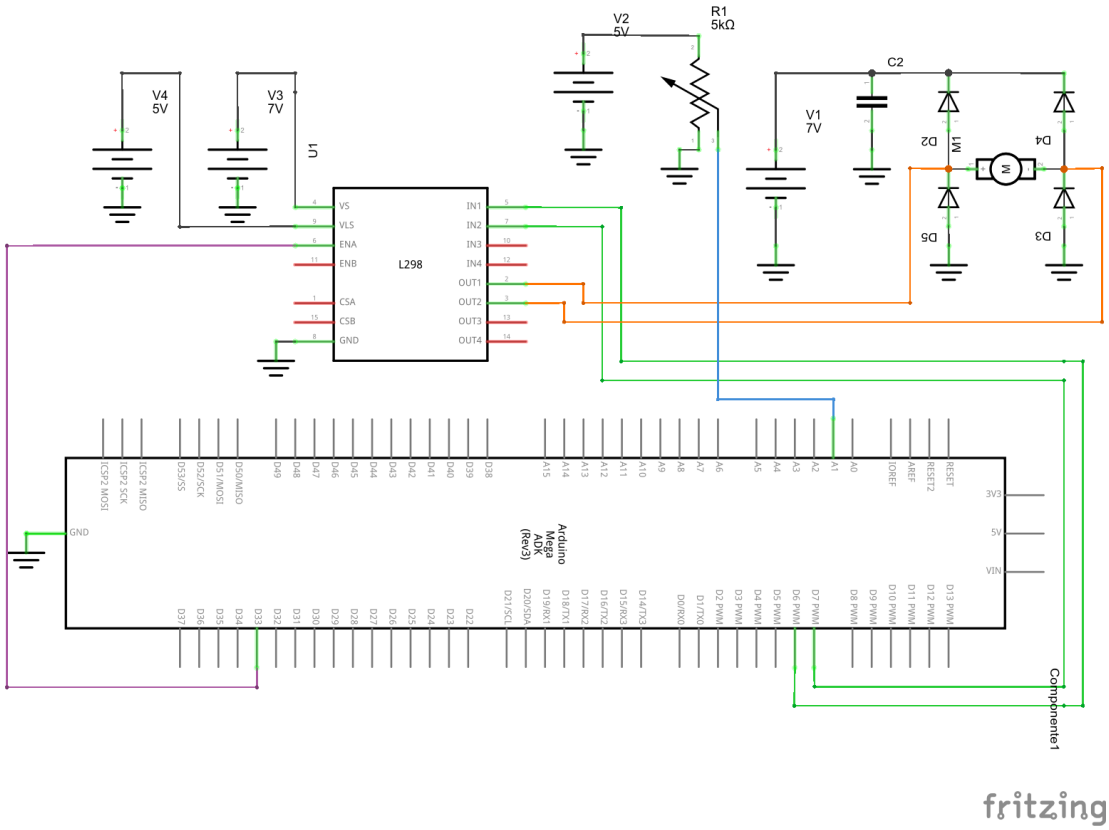


Figura 10 - Esquemático de acionamento do um motor

Fonte: Circuito montado com o auxílio do *software Fritzing*

Basicamente o programa deve ler o valor do potenciômetro, transformar esse valor no seu equivalente em graus e comparar com o valor de posição angular desejado. A diferença entre esses dois valores deve ser aplicada a um controle PID e a saída deve ser enviada em valores de tensão em PWM. Esse valor é amplificado e então aplicado ao motor cc.

Iniciamos a escrita do programa definindo as variáveis que serão utilizadas conforme mostra a Figura 11. As variáveis *Kp*, *Ki* e *Kd* são as constantes do controle PID e são ajustadas de acordo com o desempenho do controle.

```
//Programa para acionamento do motor do Pulso - RD5NT

unsigned long lastTime;         //Utilizadas no controle PID
double errSum, lastErr;         //Utilizadas no controle PID
double Input, Output, Setpoint; //Variáveis de entrada, saída e valor desejado
double kp=5, ki=0, kd=0;        //Constantes do controle PID
```

Figura 11 - Definição das variáveis

Em seguida, inicializaremos o programa e a placa através da função *setup* conforme mostra Figura 12. Quando a placa é ligada ou resetada, essa função é executada. É através dela que informa-se ao *hardware* do Arduino o que será utilizado dele. Dessa forma, informamos que utilizaremos os pinos 6 e 7 como saídas PWM de acionamento do motor, o pino 33 como *enable* e o pino 1 como entrada do sinal medido do potenciômetro. A Figura 10 mostra as ligações desses pinos.

Ainda nesta etapa, efetua-se um *print* para comunicação da necessidade de entrada do valor desejado do posicionamento angular em graus. O valor em graus é armazenado na variável x através da função *Serial.parseInt()* e em seguida é transformada em binário através da função *map(x,0,360,1023)*.

```
void setup(){

  Serial.end();          //Desabilitando e eliminando qualquer comunicação serial anterior

  Serial.begin(57600);  //Habilitando comunicação serial com 57600bps

  pinMode(6,OUTPUT);    // Saida do controlador (PWM) = Output sentido 1
  pinMode(7,OUTPUT);    // Saida do controlador (PWM) = Output sentido 2
  pinMode(33,OUTPUT);   // Enable
  pinMode(1,INPUT);     // Sinal medido (potenciômetro) = Input

  Serial.print("Angulo do Pulso desejado em graus: ");  //Comunicando com o usuário
  while(!Serial.available());      //Aguardando o valor desejado em graus
  double x = Serial.parseInt();    //Armazedando o valor na variável x
  Setpoint = map(x,0,360,0,1023);  //Mapeando o valor de x para binário e armazenando em Setpoint
}
```

Figura 12 - Função Setup

A próxima etapa do programa é o *loop* principal do programa. Esta etapa é executada indefinidamente até que a placa seja desligada ou resetada. No entanto, como

dentro desse *loop* uma sub-rotina responsável pelo cálculo PID será chamada, debruçaremos primeiro sobre ela, a função *Compute()* mostrada na Figura 13.

Quando a sub-rotina *Compute()* é chamada, ela inicia definindo a variável *now* através da função *millis()*, que retorna o valor em milissegundos desde que a placa começou a executar o programa atual. Esse valor será a referência para o cálculo da variação do tempo. Em seguida, define-se a variável *timeChange* que irá armazenar a variação do tempo calculada pela diferença entre as variáveis *now* e *lastTime*. A diferença entre o *Setpoint* e o *Input* é armazenada na variável *error* e será usada para o cálculo das parcelas proporcional, integral e derivativa. A integral é dada pelo produto da variável *error* e *timeChange* mais o somatório das parcelas anteriores e é armazenado em *errSum*. A parcela derivativa é calculada pela divisão da diferença entre *error* e *lastErr* por *timeChange*. Por fim, a soma do produto dessas parcelas por seus respectivos coeficientes será armazenada em *Output* e em seguida é tomado seu módulo. Os valores de *error* e *now* são armazenadas em *lastErr* e *lastime*, respectivamente, para que no próximo ciclo sejam tomadas como referência.

```
void Compute(){
   unsigned long now = millis();   //Definindo a variável de tempo agora
   double timeChange = (double)(now - lastTime); // Definindo Delta tempo
   double error = Setpoint - Input;  //Cálculo do erro
   errSum += (error * timeChange);   //Cálculo integral
   double dErr = (error - lastErr) / timeChange;  //Cálculo derivativo
   Output = kp * error + ki * errSum + kd * dErr; //Cálculo - PID
   Output = abs(Output); //Módulo do resultado
   lastErr = error;  //Armazenamento do último erro antes do próximo ciclo
   lastTime = now;   //Armazenamento do último tempo antes do próximo ciclo
}
```

Figura 13 - Função *Compute()* onde efetua-se o calculo PID

O *loop* principal inicia efetuando a leitura do potenciômetro e armazenando esse valor em *Input*, vide Figura 14. De posse deste valor, já pode-se efetuar o cálculo PID através da função *compute()*, e assim é feito. Após a execução da função *compute()*, o resultado do PID já está armazenado na variável *Output*, bastando efetuar seu tratamento e enviar para saída do Arduino. A primeira etapa é limitar o valor de *Output* no intervalo binário de 0 a 1023. Em seguida, faz-se uma conversão do valor em binário para o intervalo de 0 a 255 que é o padrão de saída do PWM. O valor 0 possui uma saída de 0V ao longo do tempo. Já a saída 255 possui o valor de 5V ao longo do tempo. Mas se a saída for 191, tem-se na saída uma onda quadrada com um *duty cicle* de 75% em 5V. Por fim, basta definir qual o sentido de rotação do motor cc. Para isso, basta comparar o valor de *Input* que é o valor lido pelo potenciômetro e o *Setpoint* que é o valor desejado. Se *Input* for maior ou igual a *Setpoint* escreve-se o valor *Output* padronizado em PWM no pino 7

e o valor nulo no pino 6, seguido da habilitação do *enable*. Assim, define-se um sentido de rotação. Caso *Input* seja menor que *Setpoint* os valores escritos nos pinos 7 e 6 são invertidos, seguido da habilitação do *enable*. Tem-se portanto o sentindo inverso do anterior.

```
void loop()
{
  Input = analogRead(A1); //lendo o valor do potenciometro em binário
  Compute();              //Chamando a função de cálculo PID

  Output = constrain(Output,0,1023); //limitando o valor absoluto do PID de 0 a 1023
  Output = map(Output,0,1023,0,255); //convertendo o valor PID para padrão PWM

  if (Input >= Setpoint)
    {
    analogWrite(6, 0);
    analogWrite(7, Output); //Acionando o sentido de rotação 1
    digitalWrite(33,HIGH); //habilitando enable
    delay(1);
    }
  else
    {
    analogWrite(6, Output);  //Acionando o sentido de rotação 2
    analogWrite(7, 0);
    digitalWrite(33,HIGH); //habilitando enable
    delay(1);
    }
}
```

Figura 14 - Função *loop*

O programa completo está em anexo a este trabalho. A implementação do posicionamento dos outros eixos é baseada no programa escrito para o posicionamento do pulso.

6 PROGRAMA DE ACIONAMENTO DO ROBÔ RD5NT

O programa é o Código 2 – Programa de acionamento do RD5NT e encontra-se no tópico Anexo deste trabalho. Ele pode ser subdividido em 3 etapas: Definição de variáveis, *setup* e *loop*.

A primeira delas é a definição de variáveis. Nesta etapa é definida a maioria das variáveis que serão utilizadas ao longo do programa. As variáveis *Input, Output, Setpoint, lastTime, errSum* são definidas como vetores com quatro posições, referentes a cada motor. Já as variáveis *K, pino* e limites (referentes aos limites dos arcos) além de possuírem as posições referentes a cada motor, devem contemplar as constantes de PID, os pinos que serão utilizados por cada motor, e os arcos, respectivamente. Sendo assim são definidas como matrizes. Algumas variáveis de tempo utilizadas no controle PID serão comuns aos motores, não sendo necessária a distinção pra cada um.

A segunda etapa é o *setup*. Nela define-se como cada pino será utilizado. A matriz pino é submetida a uma função *for* que define os pinos de entrada e saída. Os pinos que serão utilizados pela garra foram definidos separadamente. Na função *setup* as funções *condicaogarra()* e *converteangulo()* são executadas de modo a comunicar com o usuário do programa a necessidade da entrada de dados que são referentes a condição e posicionamento da garra e dos motores de cada eixo integrante do robô. Nessas funções são informadas os *setpoints* desejados e se gostaríamos de fechar ou abrir a garra.

A terceira etapa é a função *loop*. Aqui os potenciômetros são lidos em valores binários e diferentemente do programa anterior em que tinha-se uma função dedicada para o controle PID, essa etapa foi integrada ao *loop* por conveniência.

7 ANÁLISE DOS RESULTADOS E CONCLUSÕES

Os passos da metodologia e dos objetivos do trabalho foram executados com êxito. Inicialmente foi esquematizado e montado o circuito amplificador com o L298N e posteriormente testado o funcionamento de cada ponte H integrante do CI, mostrando-se como resultado o funcionamento conforme previsto no planejamento. Ao aplicar a tensão em nível lógico, juntamente com o *enable* habilitado, as pontes H efetuaram o devido acionamento dos motores cc.

Os ensaios e medições dos valores dos potenciômetros lidos pela placa microcontroladora também permitiram o devido mapeamento dos arcos disponíveis para o movimento de cada eixo. Sendo assim, o próximo passo para implementação do programa que efetuará o controle PID pôde ser executado.

O procedimento de elaboração do programa responsável pelo cálculo PID seguiu uma lógica simples e de fácil compreensão. Conforme já discutido, o programa iniciou-se através da definição das variáveis, seguido da execução da função *setup*, onde informou-se ao Arduino aquilo que dele seria usado e então efetuou-se a lógica principal do programa, a função *loop*. Dentro desta lógica, o cálculo PID é executado a cada ciclo, ou seja, todo ciclo os potenciômetros são lidos, é efetuado o PID e atualizado o *Output*. Em seguida, o resultado obtido e armazenado na variável *Output* é então escrito na saída PWM do Arduino. O ajuste dos coeficientes proporcional, integral e derivativo foi obtido experimentalmente, o que levou a um controle puramente proporcional devido a sua melhor atuação.

Contudo, o processo evolutivo do acionamento do robô manipulador alcançou os objetivos que foram estabelecidos no início do trabalho, ou seja, a colocação do robô numa posição desejada foi alcançada com sucesso.

REFERÊNCIAS BIBLIOGRÁFICA

[1] PINTO, Carlos R. A. **Controle adaptativo aplicado em dois elos de um robô eletromecânico de cinco graus de liberdade**. 2011. 136 f. Tese (Doutorado em Mecânicaa). Unifersidade Federal da Paraíba, João Pessoa, 2011.

[2] DA COSTA E SILVA, V. e M.; MONTENEGRO, P. H. de M. **Modelamento em tempo real de três elos de um robô manipulador eletromecânico de cinco graus de liberdade**. Asociacón Argentina de Mecânica Computacional, Bueno Aires, v. 29, p. 2229-2241, Nov. 2010.

[3] **Motor Control using PWM and PID: Embedded Design**. Disponível em: < http://coactionos.com/embedded%20design%20tips/2013/10/15/Tips-Motor-Control-using-PWM-and-PID/ >. Acesso em 10/09/2014.

[4] MITUHIKO, A. **Control System, Robotics and Automation: PID Control.** Vol-II Kyoto University, Japan.

[5] **PID Controller**. Disponível em: < http://en.wikipedia.org/wiki/PID_controller >. Acesso em 11/09/2014.

[6] GUILHERME, P. **Como os robôs estão mudando o mundo?**. Tecmundo Beta, 2012. Disponível em:< http://www.tecmundo.com.br/robotica/28854-como-os-robos-estao-mudando-a-industria-.htm >. Acesso em 11/09/2014.

[7] BASTOS FILHO, T. F. **Aplicação de Robôs nas Indústrias**. Departamento de Engenharia Elétrica, Universidade Federal do Espírito Santo, 1999.

ANEXO - PROGRAMAS

Código 1 - Programa para acionamento do motor do Pulso - RD5NT

```
//Programa para acionamento do motor do Pulso - RD5NT

unsigned long lastTime;         //Utilizadas no controle PID
double errSum, lastErr;         //Utilizadas no controle PID
double Input, Output, Setpoint; //Variáveis de entrada, saída e valor desejado
double kp=5, ki=0, kd=0;        //Constantes do controle PID

void setup(){

  Serial.end();         //Desabilitando e eliminando qualquer comunicação serial anterior

  Serial.begin(57600);  //Habilitando comunicação serial com 57600bps

  pinMode(6,OUTPUT);    // Saida do controlador (PWM) = Output sentido 1
  pinMode(7,OUTPUT);    // Saida do controlador (PWM) = Output sentido 2
  pinMode(33,OUTPUT);   // Enable
  pinMode(1,INPUT);     // Sinal medido (potenciômetro) = Input

  Serial.print("Angulo do Pulso desejado em graus: ");  //Comunicando com o usuário
  while(!Serial.available());      //Aguardando o valor desejado em graus
  double x = Serial.parseInt();    //Armazedando o valor na variável x
  Setpoint = map(x,0,360,0,1023);  //Mapeando o valor de x para binário e armazenando em Setpoint
}

void loop()
{
  Input = analogRead(A1); //lendo o valor do potenciometro em binário
  Compute();              //Chamando a função de cálculo PID

  Output = constrain(Output,0,1023); //limitando o valor absoluto do PID de 0 a 1023
  Output = map(Output,0,1023,0,255); //convertendo o valor PID para padrão PWM

  if (Input >= Setpoint)
    {
    analogWrite(6, 0);
    analogWrite(7, Output); //Acionando o sentido de rotação 1
    digitalWrite(33,HIGH); //habilitando enable
    delay(1);
    }
  else
    {
    analogWrite(6, Output);  //Acionando o sentido de rotação 2
    analogWrite(7, 0);
    digitalWrite(33,HIGH); //habilitando enable
    delay(1);
    }
}

void Compute(){
   unsigned long now = millis();   //Definindo a variável de tempo agora
   double timeChange = (double)(now - lastTime); // Definindo Delta tempo
   double error = Setpoint - Input;  //Cálculo do erro
   errSum += (error * timeChange);   //Cálculo integral
   double dErr = (error - lastErr) / timeChange;  //Cálculo derivativo
   Output = kp * error + ki * errSum + kd * dErr; //Cálculo - PID
   Output = abs(Output); //Módulo do resultado
   lastErr = error;  //Armazenamento do último erro antes do próximo ciclo
   lastTime = now;   //Armazenamento do último tempo antes do próximo ciclo
}
```

Código 2 – Programa de acionamento do RD5NT

```
//Programa para acionamento de todos motores do RD5NT

//  Configurando array para medida dos ultimos tempos medidos
  unsigned long lastTime[4];
//-------------------------------------------------------------------------------------------//
/*
  Definindo variavel para controle de inputs e outputs e setpoints
  cada linha representa o elo
*/
  double Input[4], Output[4], Setpoint[4];
//-------------------------------------------------------------------------------------------//
/*
  Definido arrays para somatorio dos erros e ultimos erros
*/
  double errSum[4]={0,0,0,0},lastErr[4]={0,0,0,0};
//-------------------------------------------------------------------------------------------//
/*
  Configuração do array de ganho dos elos:
  o numero da linha corresponde o do numero do elo a ser controlado
  a coluna corresponde a função de ganho na ordem
    1)proporcional
    2)integral
    3)derivativo
*/
double k[4][3]={{5,0,0},{5,0,0},{5,0,0},{5,0,0}};//arrays com os valores de kp,kd,ki
//-------------------------------------------------------------------------------------------//
/*Configuração do array de pinos para controle dos elo

    linhas representam o elo a ser controlado e colunas as a funcionalidade do tipo

    controle do elo 1 (linha 0) o pulso:
      pino potenciometro 1 será 1
      pino do input 1 do motor 2 será 6
      pino do input 2 do motor 2 será 7
      pino do enable motor 2 será 33

    controle do elo 2 (linha 1)o cotovelo:
      pino potenciometro 2 será 2
      pino do input 1 do motor 3 será 8
      pino do input 2 do motor 3 será 9
      pino do enable motor 3 será 35

    controle do elo 3 (linha 2)o ombro:
      pino potenciometro 3 será 3
      pino do input 1 do motor 4 será 10
      pino do input 2 do motor 4 será 11
      pino do enable motor 4 será 37

    controle do elo 4 (linha 3)o ombro:
      pino potenciometro 4 será 4
      pino do input 1 do motor 5 será 12
      pino do input 2 do motor 5 será 13
      pino do enable motor 5 será 39
*/
int pino[4][4] = {{1,6,7,33},{2,8,9,35},{3,10,11,37},{4,12,13,39}};
```

```
//-------------------------------------------------------------------------------------------//
/*Configuração de valores medidos
  para leitura minima e maxima da entrada analogica
  Alimentação do fonte Vs=4.9V
  Alimentação da fonte Vss=4.9V

  vetor chamado de limites
  1)linhas indicam o potenciometro
  2)1ª Coluna indica valor maximo de deslocamento angular em relação a referencia
  3)2ª Coluna indica valor minimo de setpoint medido pelo arduino
  4)3ª COluna indica valor maximo de setpoint medido pelo arduino
*/
double limites[4][3]={{360,0,1023},{293,66,900},{96,376,648},{291,101,928}};
//-------------------------------------------------------------------------------------------//
unsigned long now;
double timeChange;
double error;
double dErr;
double lasttime;

int condicao;
void setup(){

  Serial.end(); //eliminando qualquer configuração serial anterior

  //configurando pinos de entrada
  for(int i = 0;i<4;i++){
    pinMode(pino[i][0],INPUT);
    //configurando pinos de saída
    for (int j=1;j<4;j++){
      pinMode(pino[i][j],OUTPUT);
    }
  }

  pinMode(41,OUTPUT);   //Enable do motor 1 - garra
  pinMode(43,OUTPUT);   //Input 2 do motor 1
  pinMode(45,OUTPUT);   //Input 1 do motor 1
  pinMode(47,OUTPUT);   //Fonte Switch
  Serial.begin(57600); //configurando entrada serial

  //Verificar estado inicial que se quer da garra
  condicaogarra();
  //Impondo a codição angular desejada usando a funcao converte angulo detalhes abaixo
  for(int i=0;i<4;i++){
    Setpoint[i] = converteangulo(i);
  }
  //iniciando os valores do tempo dos lastTime medidos
    for(int j=0;j<4;j++){
      lastTime[j]=millis();
    }
  }
//-------------------------------------------------------------------------------------------/
void loop()
{
  //Laço para leitura de todos os potenciometros
  Input[0] = analogRead(A1); //potenciometro 1
  Input[1] = analogRead(A2); //potenciometro 2
  Input[2] = analogRead(A3); //potenciometro 3
  Input[3] = analogRead(A4); //potenciometro 4
```

```
  for(int l=0;l<4;l++){
     now = millis();
     timeChange = (double)(now - lastTime[l]);
     error = Setpoint[l] - Input[l];
     errSum[l] += (error * timeChange);
     dErr = (error - lastErr[l])/timeChange;
     Output[l] = (k[l][0])*error + (k[l][1])*errSum[l] + (k[l][2])*dErr;
     lastErr[l] = error;
     lastTime[l] = now;
     if(Output[l]>0){
       Output[l]=abs(Output[l]);
       Output[l]=constrain(Output[l],0,1023);
       Output[l]=map(Output[l],0,1023,0,255);
       analogWrite(pino[l][1], 0); //acionando input 1
       analogWrite(pino[l][2], Output[l]); //acionando input 2
       digitalWrite(pino[l][3],HIGH); //habilitando enable
     }
     else{
       Output[l]=abs(Output[l]);
       Output[l]=constrain(Output[l],0,1023);
       Output[l]=map(Output[l],0,1023,0,255);
       analogWrite(pino[l][1], Output[l]); //acionando input 1
       analogWrite(pino[l][2], 0); //acionando input 2
       digitalWrite(pino[l][3],HIGH); //habilitando enable
     }
  }

  if (Serial.available())
  {
    //Impondo a codição angular desejada usando a funcao converte angulo
    for(int i=0;i<4;i++)
      {
      Setpoint[i] = converteangulo(i);
      }
  }
}
//--------------------------------------------------------------------------//
/*
  Funcao que fornece o deslocamento angular desejado
*/
double converteangulo(int elo){
  Serial.print("Digite o deslocamento para ");
  switch(elo){
   case 0:
     Serial.print("o Pulso 0-");
     break;
   case 1:
     Serial.print("o Cotovelo 0-");
     break;
   case 2:
     Serial.print("o Ombro 0-");
     break;
```

```
   case 3:
     Serial.print("a Base 0-");
     break;
  }
  Serial.print(limites[elo][0]);Serial.println(" em graus:");
  while(!Serial.available());
  double x =Serial.parseInt();
  double deslocamento = map(x,0,limites[elo][0],limites[elo][1],limites[elo][2]);
  return(deslocamento);
}
void condicaogarra(){
  Serial.print("Deseja-se garra aberta( digite 1) ou fechada(digite 2): ");
  while(!Serial.available());
    condicao=Serial.parseInt();
    if(condicao==1){
        abrirgarra();
    }
    else{
        fechargarra();
    }
  Serial.println();
}
void abrirgarra(){
  digitalWrite(41,HIGH);//Enable do motor 1
  digitalWrite(43,HIGH);//Input1 do motor 1 alto
  digitalWrite(45,LOW);//Input2 do motor 2 baixo
}

void fechargarra(){
  digitalWrite(41,HIGH);//Enable do motor 1
  digitalWrite(43,LOW);//Input1 do motor 1 baixo
  digitalWrite(45,HIGH);//Input2 do motor 2 alto
  digitalWrite(47,HIGH); //Alimentando Switch
  double estaaberto;
  do{
     estaaberto =analogRead(A5);
     delay(50);
  }while(estaaberto == 0);
  digitalWrite(47,LOW); //Alimentando Switch
  digitalWrite(41,LOW);//Input2 do motor 2 baixo
}
//-------------------------------------------------------------------------------------//
```