OFICINA
SEGUIDOR DE LINHA

Ministrante: Gabriel Meneses Vieira

01, 02 e 03 de Julho de 2015

OFICINA
SEGUIDOR DE LINHA

Material Escrito Por:
Gabriel Meneses Vieira
Mateus Simões
Paula Carolina
Tiago Mitraud

Sumário

1 Introdução **5**

2 O Projeto **6**

3 Alimentação **8**
3.1 Descrição . . . 8
3.2 Escolha da Fonte de Alimentação . . . 8
3.3 Regulador de Tensão . . . 9
3.4 Power Blocking . . . 9

4 Controladores **10**
4.1 Descrição . . . 10
4.2 Arduino . . . 11
4.3 Estrutura de código em um microcontrolador . . . 11
4.4 Funções do Arduino . . . 12

5 Comunicação Serial **13**
5.1 Descrição . . . 13
5.2 Iniciando uma conexão Serial . . . 13
5.3 Serial no Seguidor de Linha . . . 13

6 Sensoreamento **15**
6.1 Reconhecimento de linha . . . 15
6.1.1 Sensor Digital . . . 15
6.1.2 Sensor Analógico . . . 16
6.2 Sensor Ultrassônico . . . 17

7 Sistema Elétrico de Potência **19**
7.1 Controle de Motores Usando Transistor . . . 19
7.2 Controle de Motores Usando Uma Ponte H . . . 20
7.2.1 Funcionamento do L298N . . . 20
7.3 PWM (Pulse Width Modulation) . . . 21
7.4 Controle do Sentido e Velocidade de Rotação dos Motores . . . 21
7.5 Implementando o Power Blocking . . . 23

8 O Display LCD **24**
8.1 Descrição . . . 24
8.2 O LCD no Arduino . . . 25

9 Interrupções **27**
9.1 Descrição . . . 27
9.2 Usando Interrupções . . . 27
9.3 Ativando e desativando Interrupções . . . 28

10 Controle **29**
10.1 Controle utilizando velocidades constantes . . . 29
10.2 Controladores PID . . . 29

11 Apendices **31**
11.1 Softwares Uteis . . . 31
11.2 Links Uteis . . . 31

Lista de Figuras

2.1 Chassi projetado para um seguidor de linha 6

3.1 Pilhas 8
3.2 Bateria de 9V 8
3.3 Bateria de Polímero de Lítio 8
3.4 Bateria de íon de Lítio 8
3.5 Regulador de Tensão LM7805 9
3.6 Regulador de Tensão LM78L33 9
3.7 Ligação do regulador de tensão 9

4.1 PIC 10
4.2 MSP430 10
4.3 TivaC 10
4.4 Arduino UNO 10
4.5 Arduino Mega 2560 11
4.6 Raspberry Pi B+ 11
4.7 IDE do Arduino 11

5.1 Monitor Serial da IDE Arduino 14

6.1 LED e Fototransistor 15
6.2 Esquema para ligação do sensor Digital 16
6.3 LED e LDR 17
6.4 Esquema para ligação do sensor Analógico 17
6.5 Sensor Ultrassônico HC-SR04 18
6.6 Codigo para medir distancia Utilizando o sensor HC-SR04 18

7.1 Transistores de Junção Bipolar NPN BC547B e TIP31 19
7.2 Circuito para o controle de motores com um Transistor NPN 19
7.3 Módulo Ponte H L298N 20
7.4 Esquema de funcionamento de uma Ponte H 20
7.5 Conexões do Módulo L298N 21
7.6 Montagem do Arduino e Ponte H 22
7.7 Código em C Para Controle dos Motores. 22

8.1 Display LCD e um Shield com um display para o Arduino 24
8.2 Ligação do Display ao Arduino 25

Capítulo 1

Introdução

Robôs autônomos são máquinas inteligentes capazes de realizar tarefas sem controle humano contínuo e explícito sobre seus movimentos. Seguidores de linha, como o nome sugere, constituem uma classe de robôs autônomos que identificam e percorrem uma trilha, normalmente caracterizada por uma linha de dimensões bem definidas. A conclusão de um percurso e sucesso do projeto depende, é claro, de boa estrutura física e de tomadas de decisões que respondam aos diferentes tipos de trajetória que o robô possa enfrentar. Com o objetivo de auxiliar os estudantes na criação e no desenvolvimento de robôs seguidores de linha, o Programa de Educação Tutorial da Engenharia Elétrica - PET EE - promove a Oficina de Seguidores de Linha e disponibiliza esta apostila para ensinar os passos básicos e relevantes à criação desse tipo de robô.

Capítulo 2

O Projeto

O projeto do seguidor de linha deve ser realizado pensando na forma de detecção da linha, no controle do robô e em como identificar e superar os obstáculos propostos. Para isso, devemos decidir quantos e quais sensores utilizar, tipo de chassi, tipo de alimentação do circuito elétrico e diversas outras variáveis que compõem um robô autônomo seguidor de linha.

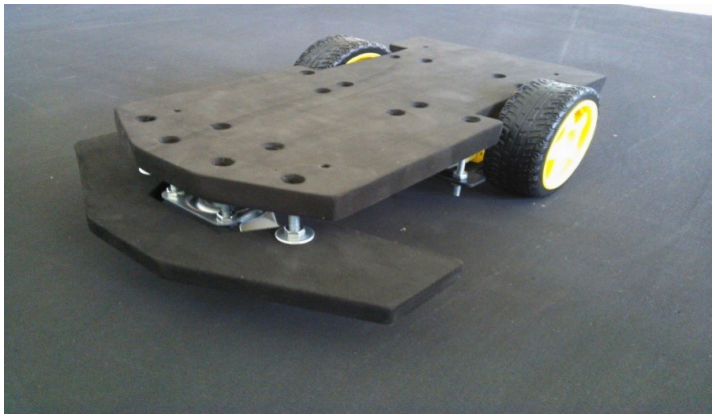
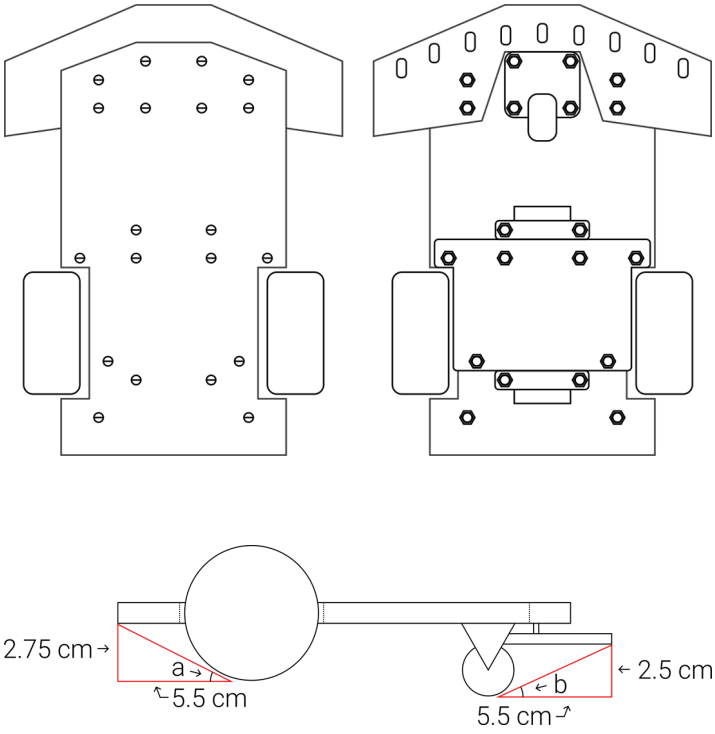


Figura 2.1: *Chassi projetado para um seguidor de linha*

Nos capítulos seguintes, iremos abordar separadamente cada parte de um seguidor de linha. Apresentaremos exemplos práticos para que o leitor, ao terminar de ler o material, tenha conhecimento suficiente para construir um robô seguidor de linha. Há várias formas de se construir um robô deste tipo. Detalhamos neste material um dos métodos mais comuns, com a utilização do microcontrolador Arduino, mas ressalvamos que não há impedimentos para a utilização das informações aqui apresentadas em outras ferramentas.

No Desing do Chassi é importante observar as dimensões, materiais utilizados e o peso de todo o conjunto para que tudo fique de acordo com as especificações desejadas.



Capítulo 3

Alimentação

3.1 Descrição

Um sistema embarcado, como um robô seguidor de linha, deve carregar a sua própria fonte de alimentação, limitando as fontes de energia a serem utilizadas a pilhas e baterias.

Figura 3.1: *Pilhas*

Figura 3.2: *Bateria de 9V*

Figura 3.3: *Bateria de Polímero de Lítio*

Figura 3.4: *Bateria de íon de Lítio*

3.2 Escolha da Fonte de Alimentação

Ao escolher a fonte de alimentação do seguidor, deve-se levar em conta alguns fatores. Primeiramente, a bateria deve ser capaz de fornecer tensão e corrente suficientes para que os motores, o controlador e todos os periféricos possam funcionar corretamente. A bateria também deve conter carga suficiente para que o sistema funcione por um tempo desejado (uma competição por exemplo).

E por último a bateria deve ser recarregável, o que é altamente recomendável, apesar de não ser vital para o funcionamento do veículo.

3.3 Regulador de Tensão

Como provavelmente a tensão da bateria será maior que a tensão suportada pelo controlador, é recomendado o uso de um regulador de tensão. O regulador tem um funcionamento simples: ele recebe uma tensão contínua maior que a tensão especificada na entrada e fornece a tensão especificada na saída.

Como a maior parte dos controladores funciona a 5V ou 3.3V recomenda-se reguladores com essa especificação. Um modelo de regulador de tensão de 5V é o LM7805 e de 3.3V temos o LM78L33.

Figura 3.5: *Regulador de Tensão LM7805*

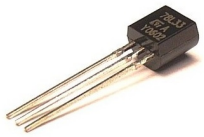


Figura 3.6: *Regulador de Tensão LM78L33*

Na figura 3.7 está o esquema de ligação do regulador para que a tensão de saída seja constante.

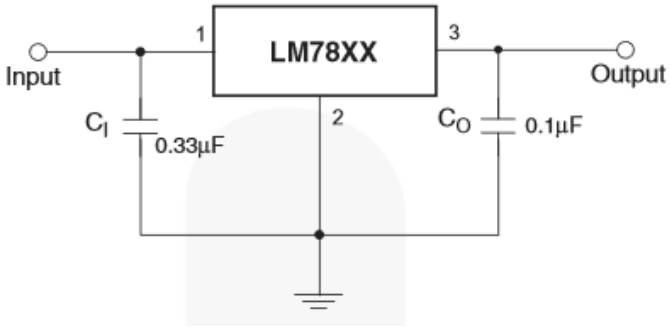


Figura 3.7: *Ligação do regulador de tensão*

3.4 Power Blocking

Como é possível perceber, o tempo que o robô pode ficar funcionando é de grande importância e, portanto, a economia de carga da bateria é essencial. A técnica conhecida como power blocking tem essa função. Ela consiste em fornecer potência a um periférico apenas no momento em que ele será utilizado. Como o tempo de funcionamento dos periféricos é bem curto em um seguidor de linha, uma grande economia de energia pode ser notada.

Se considerarmos que a corrente no periférico é aproximadamente constante enquanto está ligado, podemos perceber a grande redução na potência fornecida a ele.

Como implementar o Power Blocking será discutido mais adiante.

Capítulo 4

Controladores

4.1 Descrição

Existem vários controladores que podem ser usados em um seguidor de linhas, é possível fazer um seguidor até mesmo com amplificadores operacionais! Entre os mais comuns no mercado estão os microcontroladores PIC da Microchip, placas controladoras da Texas Instruments como o MSP430 (proprietário) ou o TivaC (Baseado em um microcontrolador ARM M4), Arduino (baseado em microcontroladores da Atmel), Raspberry Pi, BeagleBone e vários outros.

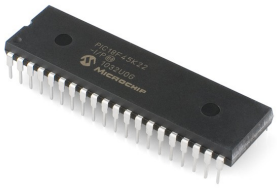


Figura 4.1: *PIC*

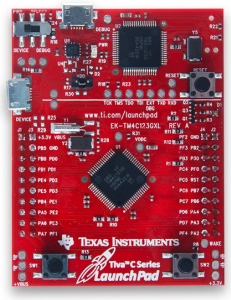


Figura 4.3: *TivaC*

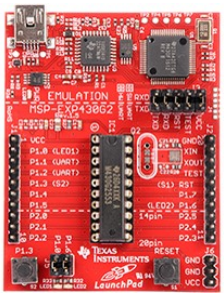


Figura 4.2: *MSP430*

Figura 4.4: *Arduino UNO*

Figura 4.5: *Arduino Mega 2560*

Figura 4.6: *Raspberry Pi B+*

4.2 Arduino

Para este curso foi escolhido o Arduino. Entre os motivos para essa escolha estão a facilidade de programar para plataforma, a grande quantidade de periféricos que pode ser conectada no Arduino e, por ser uma plataforma Open Source, a grande quantidade de referências que podem ser encontradas online.

Entre as melhores referências está o próprio site do Arduino (www.arduino.cc) em que pode ser encontrada sintaxe de todas as funções básicas do controlador e de suas bibliotecas. No decorrer deste material serão explicadas como usar várias dessas funções.

No site pode ser feito o download da IDE do Arduino, um ambiente simples e com interface amigável para programar e carregar os códigos no controlador.

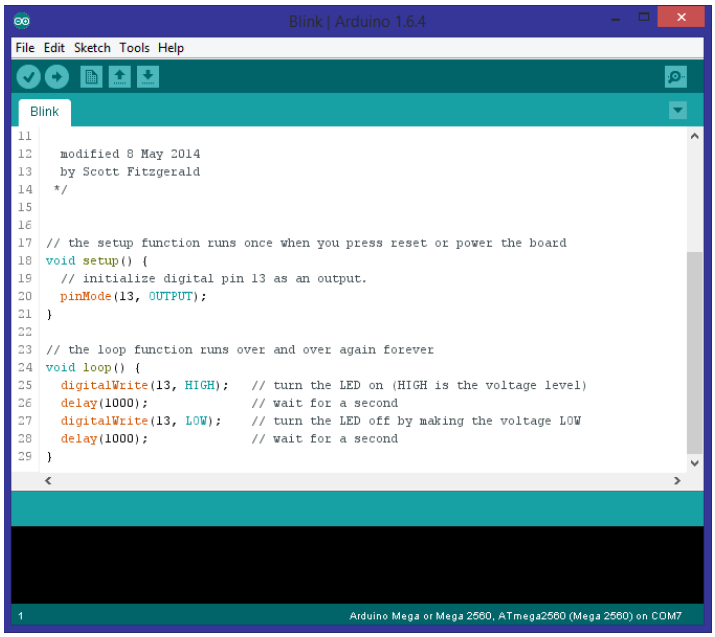


Figura 4.7: *IDE do Arduino*

4.3 Estrutura de código em um microcontrolador

Em geral, no código de um microcontrolador, há uma parte que é executada uma única vez quando o controlador for ligado e uma parte que roda continuamente. No Arduino essas partes são

bem separadas em duas funções:

- *setup()* - Essa função contém a parte do código que será executada uma única vez ao iniciar;
- *loop()* - E nessa, a parte executada continuamente.

No *setup()* são declarados os pinos que serão entrada e saída, inicializados os periféricos, os estados iniciais das saídas, interrupções, entre outras coisas. No *loop()* fica a parte principal do código que conterá as suas rotinas.

4.4 Funções do Arduino

Abaixo estão listadas algumas funções do Arduino. A sintaxe do código é a mesma da linguagem C e todas bibliotecas de C podem ser utilizadas. O funcionamento dessas funções será demonstrado em capítulos posteriores. Para uma lista mais completa, cheque o Apêndice 1.

- Digital I/O
  - pinMode()
  - digitalWrite()
  - digitalRead()
- Analog I/O
  - alogReference()
  - analogRead()
  - analogWrite()
- Advanced I/O
  - pulseIn()
- Tempo
  - millis()
  - micros()
  - delay()
  - delayMicroseconds()

Capítulo 5

Comunicação Serial

5.1 Descrição

O Arduino, assim como vários microcontroladores tem um módulo interno para realizar a comunicação serial UART (Universal Asynchronous Receiver Transmitter). Esse módulo possibilita a comunicação serial full-duplex (bidirecional), de modo que o Arduino possa comunicar com o computador e outros dispositivos e os dispositivos possam se comunicar com o Arduino.

Toda placa Arduino tem pelo menos uma porta serial que usa os pinos digitais 0(RX) e 1(TX) e o USB do computador para realizar essa comunicação. Caso utilize alguma função serial, os pinos 0 e 1 não podem ser utilizados para outras funções.

5.2 Iniciando uma conexão Serial

As conexões seriais devem ser inicializadas no setup() utilizando a função:

Serial.begin(baudRate)

e para o Arduino Mega:

Serial1.begin(baudRate)
Serial2.begin(baudRate)
Serial3.begin(baudRate)

Baud Rate representa a velocidade de comunicação entre os dispositivos. A IDE do Arduino já tem um monitor Serial Built-in que pode ser usado para receber dados. Para utilizá-lo é necessário escolher a mesma Baud Rate escolhida no *Serial.begin(baudRate)*.

5.3 Serial no Seguidor de Linha

No seguidor de linha utilizaremos as funções print para depuração do código.
Serial.print()
Serial.println()

Abaixo estão listadas várias funções que podem ser usadas em comunicação Serial (todas começadas por Serial.)

- available()

- begin()
- end()
- find()
- findUntil()
- flush()
- parseFloat()
- parseInt()
- peek()
- print()
- println()
- read()
- readBytes()
- readBytesUntil()
- readString()
- readStringUntil()
- setTimeout()
- write()

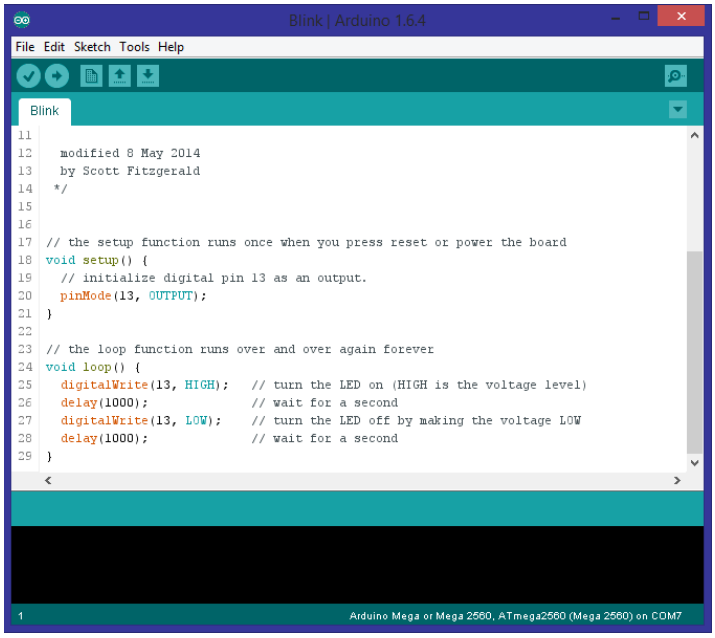


Figura 5.1: *Monitor Serial da IDE Arduino*

Capítulo 6

Sensoreamento

6.1 Reconhecimento de linha

Para reconhecer a linha utiliza-se a reflexão da luz. Em uma pista preta com linha branca a luz deverá refletir bem apenas onde há linhas ou marcações na pista. Serão mostrados dois tipos de sensores para realizar esse reconhecimento.

6.1.1 Sensor Digital

O sensor digital é composto por um LED e um fototransistor. O fototransistor trabalha em corte quando não é identificada luz refletida e como um amplificador quando a luz é detectada. A saída desse sistema tem níveis de tensão que podem ser reconhecidos como sinais digitais, e lidos pela função *digitalRead()* do Arduino.

Figura 6.1: *LED e Fototransistor*

Os resistores devem ser dimensionados para obter a corrente desejada no LED e para que o Pull Down do fototransistor funcione corretamente.

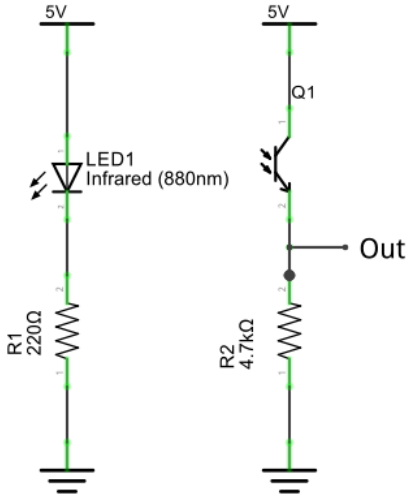


Figura 6.2: *Esquema para ligação do sensor Digital*

```
1  #define sensor 2
2  boolean s;
3
4  void setup() {
5    Serial.begin(9600);
6    pinMode(sensor, INPUT);
7  }
8
9  void loop() {
10   s = digitalRead(sensor);
11   if(s) Serial.print("sensor = HIGH");
12   else Serial.print("sensor = LOW");
13 }
```

6.1.2 Sensor Analógico

O funcionamento do sensor analógico é semelhante, utiliza-se um LED e um LDR (Resistor dependente de Luz). Nesse caso os níveis de tensão são variáveis, então é necessário calibrar a sensibilidade do sensor. A leitura do sensor é realizada utilizando a função *analogRead()* e ele deve ser conectado a uma entrada analógica.

O conversor Analógico-Digital (ADC) do Arduino tem resolução de 10 bits, de modo que os níveis de tensão são codificados com um valor entre 0 e 1023. Como o sensor é analógico, deve-se realizar testes para identificar qual nível de luminosidade a linha é identificada.

Figura 6.3: *LED e LDR*

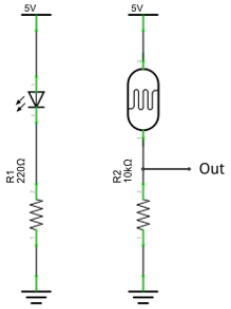


Figura 6.4: *Esquema para ligação do sensor Analógico*

```
#define sensor A0
int s;

void setup() {
  Serial.begin(9600);
}

void loop() {
  s = analogRead(sensor);
  Serial.print("sensor = ");
  Serial.print(s);
  delay(500);
}
```

6.2 Sensor Ultrassônico

O sensor Ultrassônico (HC-SR04) é utilizado para identificar obstáculos no caminho do robô. Essencialmente, ele é um sensor de distância: emite um pulso Ultrassônico e mede o Eco desse pulso. Utilizando um fator de escala podemos obter a distância.

O Sensor tem 4 pinos: VCC, GND, Trigger e Echo. Quando o nível lógico no trigger é alto, ocorre a emissão do pulso ultrassônico, o eco é detectado pelo pino echo utilizando a função *pulseIn()*

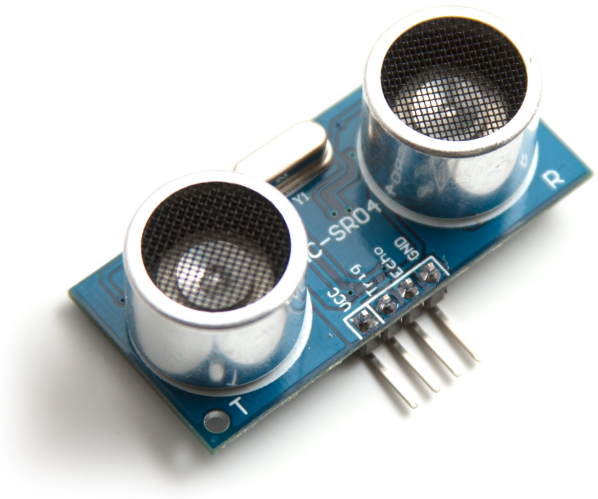


Figura 6.5: *Sensor Ultrassônico HC-SR04*

A ideia é que o veículo realize uma rotina de desvio caso um objeto seja encontrado a uma distância menor que a preestabelecida.

```
#define triggerPin 12
#define echoPin 13

double time, dist;

void setup() {
  pinMode (triggerPin, OUTPUT);
  pinMode (echoPin, INPUT);
  Serial.begin(9600);
}

void loop() {
  // Send an ultrasonic Pulse
  digitalWrite(triggerPin, LOW);
  delayMicroseconds(2);
  digitalWrite(triggerPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(triggerPin, LOW);
  //Measuring the time
  time = pulseIn(echoPin, HIGH);
  //Calculate the distance in cm
  dist = (time/2)/29.2;
  Serial.println(dist);
  delay(50);
}
```

Figura 6.6: *Codigo para medir distancia Utilizando o sensor HC-SR04*

Capítulo 7

Sistema Elétrico de Potência

7.1 Controle de Motores Usando Transistor

Um jeito simples de controlar motores é utilizando um transistor. Utilizando um transistor de junção bipolar como um amplificador na configuração emissor comum pode-se controlar cargas indutivas utilizando tensões e correntes maiores que o Arduino pode fornecer.

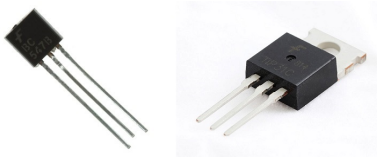


Figura 7.1: *Transistores de Junção Bipolar NPN BC547B e TIP31*

O esquemático do circuito utilizando um transistor NPN está na figura 7.2

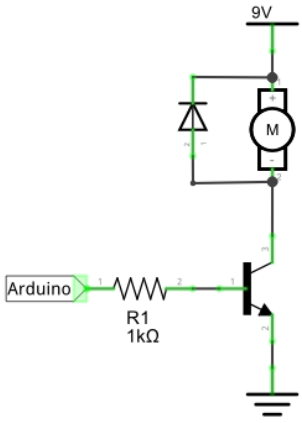


Figura 7.2: *Circuito para o controle de motores com um Transistor NPN*

Este circuito possibilita o controle da velocidade do motor utilizando PWM, porém não permite o controle da direção de rotação do motor.

Devem-se tomar alguns cuidados ao utilizar-se o transistor, o diodo deve ser colocado em paralelo com o motor polarizado inversamente para evitar ruídos criados por cargas indutivas. Não inverta a polaridade do diodo, fazendo isso o circuito estaria em curto. Deve-se tomar cuidado com a corrente que o modelo do transistor suporta, caso uma corrente alta passe pelo transistor ele ira aquecer e queimar.

7.2 Controle de Motores Usando Uma Ponte H

A ponte H é um outro circuito que possibilita o controle dos motores de corrente contínua. Com a ponte H, podemos controlar não só a velocidade do motor, mas também o sentido de rotação do motor. Para controle de velocidade utilizam-se os pinos PWM do Arduino assim como no transistor de junção bipolar. Com isso, será possível ter controle sobre a movimentação do seguidor de linha. Nesta seção, utilizaremos um módulo com o CI L298N.

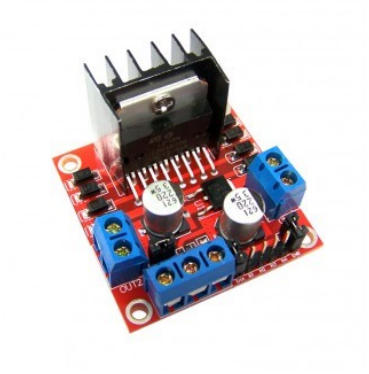


Figura 7.3: *Módulo Ponte H L298N*

A ponte H tem esse nome pois o seu circuito tem um formato semelhante ao de uma letra H. A ponte H da figura 7.4, por exemplo, possui quatro chaves (S1, S2, S3 e S4), que podem ser ativadas aos pares (S1 e S3) ou (S2 e S4). Para cada configuração, a corrente que fluirá pelo motor terá um sentido, o que fará com a rotação seja no sentido horário ou anti-horário. Caso nenhuma chave seja ativada, o motor permanecerá desligado.

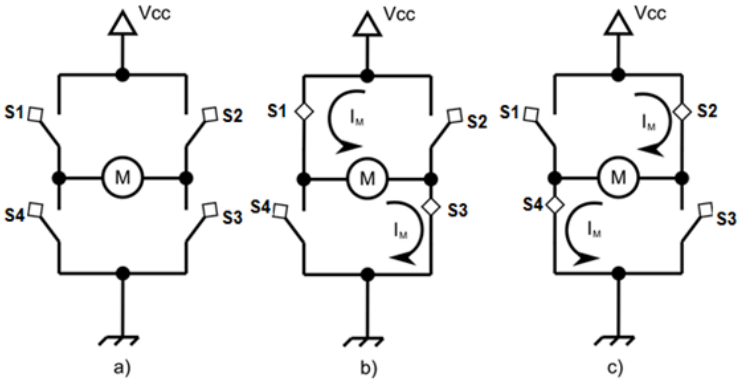


Figura 7.4: *Esquema de funcionamento de uma Ponte H*

7.2.1 Funcionamento do L298N

A seguir, iremos detalhar cada entrada do módulo da ponte H utilizado.

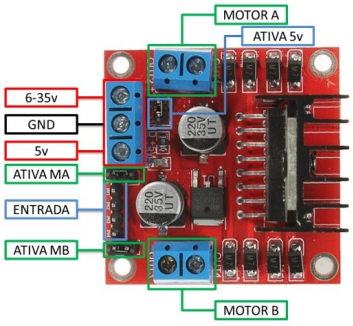


Figura 7.5: *Conexões do Módulo L298N*

Os motores DC serão conectados nas entradas Motor A e Motor B. Já o controle da velocidade dos motores, será feito através dos pinos Ativa MA e Ativa MB, pois são os responsáveis pelo controle PWM dos motores A e B respectivamente. Caso os pinos estejam com jumper, não haverá controle de velocidade, pois eles estarão ligados continuamente aos 5V. Esses pinos serão utilizados em conjunto com os pinos PWM do Arduino.

Este Driver Ponte H L298N possui um regulador de tensão embutido. Quando o driver está operando entre 6-35V, este regulador disponibiliza uma saída regulada de +5V no pino (5v) para um uso externo, 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.

Em (6-35V) e (GND) 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).

Por fim, as entradas IN1, IN2, IN3 e IN4 receberam um sinal digital do Arduino para controlar o acionamento dos motores. Sendo estes pinos responsáveis pela rotação do Motor A (IN1 e IN2) e Motor B (IN3 e IN4). A tabela abaixo 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.

7.3 PWM (Pulse Width Modulation)

Para entender o controle da velocidade dos motores, será necessário ter uma noção básica sobre PWM. A modulação por largura de pulso, mais conhecida como PWM, é uma técnica para obter resultados analógicos por meios digitais. O controle digital é usado para criar uma onda quadrada, um sinal alternado entre ligado e desligado. Este padrão on-off pode simular valores de tensões entre alto (5 volts) e baixo (0 volts) ao alterar a proporção do tempo em que o sinal alterna entre alto e baixo. A duração do desse "tempo"é chamado de largura de pulso. Para obter diferentes valores analógicos, basta controlar, ou modular, a largura de pulso. Se padrão de ligar-desligar é repetido com uma rapidez suficiente, em um LED por exemplo, o resultado é como se o sinal estivesse constante em um valor entre 0 e 5v, possibilitando o controle da luminosidade do LED. No arduino, o PWM é implementado utilizando a função *analogWrite()*.

7.4 Controle da Velocidade e Sentido de Rotação dos Motores

A montagem da ponte H L298N com o Arduino para controlar um motor pode ser vista na figura 7.6. Os pinos ENA, INI1 e INI2 estão ligados às entradas 6, 5 e 4 do Arduino. Alimentamos a ponte H em VCC com 9V, e o motor está conectado em OUT1 e OUT2;

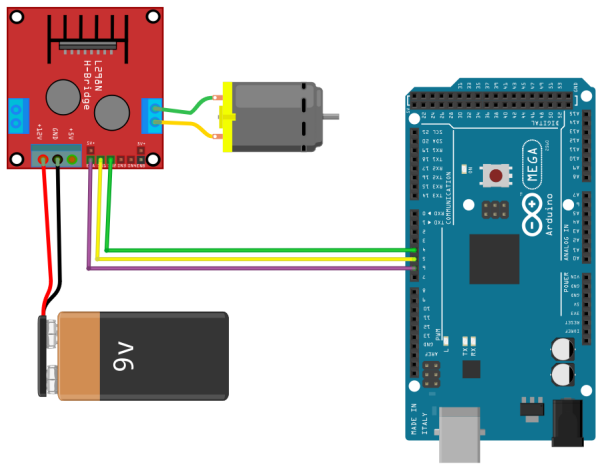


Figura 7.6: *Montagem do Arduino e Ponte H*

O Código em C para o Arduino para controlar tanto a velocidade quanto a rotação dos motores pode ser visto nas figuras 7.7.

```
1  //Define os pinos de entrada da ponte H
2  #define ENA 6
3  #define IN1 5
4  #define IN2 4
5
6  void setup()
7  {
8    //Configura os pinos da ponte H
9    pinMode(IN1, OUTPUT);
10   pinMode(IN2, OUTPUT);
11 }
12
13 //Controle da velocidade
14 void velocidade()
15 {
16   int duty_cycle = 100;
17   while (duty_cycle < 255)
18   {
19     analogWrite(ENA, duty_cycle);
20     duty_cycle = duty_cycle + 10;
21     delay(500);
22   }
23 }
24
```

```
22   }
23 }
24
25 void loop()
26 {
27   //Motor A gira no sentido horário
28   digitalWrite(IN1, HIGH);
29   digitalWrite(IN2, LOW);
30   velocidade();
31   //Para o motor A
32   digitalWrite(IN1, HIGH);
33   digitalWrite(IN2, HIGH);
34   delay(1000);
35   //Motor A gira no sentido anti-horário
36   digitalWrite(IN1, LOW);
37   digitalWrite(IN2, HIGH);
38   velocidade();
39   //Para o motor A
40   digitalWrite(IN1, HIGH);
41   digitalWrite(IN2, HIGH);
42   delay(1000);
43 }
44
45
```

Figura 7.7: *Código em C Para Controle dos Motores.*

7.5 Implementando o Power Blocking

Agora já temos base para implementar o power blocking no Arduino. A implementação é simples, usa-se um pino adicional do Arduino para controlar o fornecimento de potência para um periférico. Abaixo a função de leitura de sensores digitais com a utilização de power blocking:

```
131 void readSensors() {
132   digitalWrite(powerBlockSensors, HIGH);
133   delayMicroseconds(10); // ajustar delay
134   s0 = digitalRead(sensor0);
135   s1 = digitalRead(sensor1);
136   s2 = digitalRead(sensor2);
137   s3 = digitalRead(sensor3);
138   s4 = digitalRead(sensor4);
139   s5 = digitalRead(sensor5);
140   s6 = digitalRead(sensor6);
141   s7 = digitalRead(sensor7);
142   digitalWrite(powerBlockSensors, LOW);
143 }
```

Note que na implementação é necessário um pequeno delay para garantir que o periférico esteja ligado e não seja afetado por efeitos transitórios.

Capítulo 8

O Display LCD

8.1 Descrição

O Display de Cristal Líquido, ou simplesmente LCD (do inglês liquid crystal display), é um painel muito utilizado para exibir informações como textos, imagens e vídeos. O display LCD oferece uma maneira simples de adicionar uma interface visual ao projeto. Utilizaremos neste curso, um display LCD 16x2 caracteres, que significa que ele possui 2 linhas com 16 caracteres para escrita cada.

Figura 8.1: *Display LCD e um Shield com um display para o Arduino*

A comunicação desse display com o Arduino é feita de forma simples, ele tem 16 pinos, sendo 3 pinos de alimentação, 3 de controle, 8 de dados e 2 para acionar o backlight (luz de fundo). Na tabela 8.1, você pode conferir as informações referentes a cada pino do LCD. A numeração dos pinos foi feita da esquerda para a direita, assim, o pino 1 é o que está na extrema esquerda.

O pino RS tem a função de descrever que tipo de dados estamos enviando. Em nível baixo (0), os dados serão tratos como comandos, já em nível alto (1), os dados serão tratados como caracteres. Já o pino R/W (Read/Write) é utilizado para determinar se queremos ler (1) ou escrever (0) dados no display. O pino E, de Enable, deve ser habilitado quando queremos iniciar a transferência de dados entre o Arduino e o display. Os três pinos citados são os responsáveis pelo controle do display, já os pinos de que compõem o barramento de dados são D0 a D7.

Tabela 8.1: *Display LCD 16x2*

Pino	Simbolo	Função
1	VSS	GND
2	VDD	5V
3	V_0	Ajuste de Contraste
4	RS	Habilita/Desabilita Seletor de Registrador
5	R/W	Leitura/Escrita
6	E	Enable
7	D0	Dados
8	D1	Dados
9	D2	Dados
10	D3	Dados
11	D4	Dados
12	D5	Dados
13	D6	Dados
14	D7	Dados
15	A	5V (Backlight)
16	K	GND(Backlight)

8.2 O LCD no Arduino

A montagem dos display utilizando Arduino pode ser vista na figura 8.2.

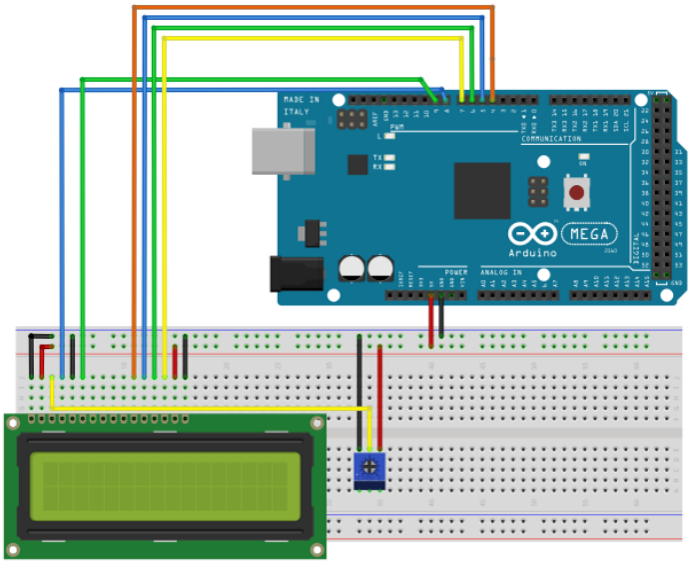


Figura 8.2: *Ligação do Display ao Arduino*

Como podemos ver na imagem, alimentamos o LCD com 5v nos pinos VDD e A, já os pinos VSS e K são ligado ao GND. O potenciômetro conectado ao pino V0 do LCD tem a função de configurar o contraste do display. Os pinos V0 e R/W foram conectados aos pinos 8 e 9 do Arduino respectivamente. Não utilizaremos o pino RS, então conectamos ele ao terra. Não é necessário utilizar todos os oito pinos de dados do LCD, para este exemplo utilizaremos apenas quatro, os pinos D4 a

D7, conectados aos pinos 4, 5, 6 e 7 do Arduino respectivamente.

O Arduino possui uma biblioteca especifica para o uso de Displays LCD: <LiquidCrysta.h>. Esta biblioteca possui várias funções úteis para configurar e tratar os dados enviados ao display. Algumas dessas funções parecidas com as funções serial.

O código para esta montagem pode ser conferido abaixo

```
#include <LiquidCrystal.h> //Inclui a biblioteca do LCD.
//Para mais informações sobre a bilbioteca acesse: https://www.arduino.cc/en/Reference/LiquidCrystal

LiquidCrystal lcd(8, 9, 4, 5, 6, 7); //Configura os pinos do Arduino para se comunicar com o LCD.

int tempo = 0; //Variável inteira que irá armazenar a contagem do tempo.
void setup()
{
  lcd.begin(16, 2); //Inicia o LCD com dimensões 16x2(Colunas x Linhas).
  lcd.setCursor(0, 0); //Posiciona o cursor na primeira coluna (0) e na primeira linha (0) do LCD.
  lcd.print("OFICINA - PETEE"); //Escreve no LCD "OFICINA - PETEE"
  lcd.setCursor(0, 1); //Posiciona o cursor na primeira coluna (0) e na segunda linha (1) do LCD.
  lcd.print("TEMPO:"); //Escreve no LCD "TEMPO:"
}

void loop()
{
  lcd.setCursor(8, 1); //Posiciona o cursor na oitava coluna (8) e na segunda linha (1) do LCD.
  lcd.print(tempo); //Escreve o valor atual da variável tempo painel no LCD.
  delay(1000); //Delay de 1000 ms ou 1 s.
  tempo++; //Incrementa o valor da variável tempo.
  if(tempo == 10) //Se a variável tempo chegar a 10 (10 segundos),...
  {
    tempo = 0; //...zera o valor da variável
  }
}

```

Capítulo 9

Interrupções

9.1 Descrição

Uma interrupção é um sinal assíncrono externo ao controlador CPU, indicando a necessidade de atenção ou a realização de alguma ação. Os pedidos de interrupção podem ser realizados por algum periférico do próprio controlador ou por um sinal elétrico externo gerado por outro sistema digital ligado ao Arduino através de qualquer um de seus pinos de entrada.

Em muitas aplicações de sistemas embarcados necessita-se de uma interrupção externa para informar ao microcontrolador que algum periférico acabou de fazer sua tarefa e está pedindo a atenção dela ou que algum evento externo através de um pino de entrada ocorreu.

Na ocorrência de algum evento externo, se o controlador aceitar interrupções naquele momento, o fluxo do programa é interrompido e o controlador executa uma rotina associada àquela interrupção

As interrupções são associadas a alguns pinos do Arduino, o número de interrupções que podem ser utilizadas varia de acordo com o modelo

Placa	int.0	int.1	int.2.	int.3	int.4	int.5
Uno, Ethernet	2	3				
Mega2560	2	3	21	20	19	18
Leonardo	3	2	0	1	7	

Tabela 9.1: *Pinos para interrupções em algumas Placas*

O Arduino Due tem uma capacidade maior de interrupções que permite associar uma interrupção a qualquer pino disponível

Nota: Durante uma interrupção a função delay() não funcionará e o valor retornado pela função millis() não será incrementado. Dados recebidos por conexão Serial poderão ser perdidos.

O uso de interrupções pode melhorar o desempenho do controlador, fazendo com que o código que é executado continuamente fique menor.

9.2 Usando Interrupções

Interrupções são uteis para fazer algumas coisas acontecerem automaticamente e podem ajudar a resolver alguns problemas temporais. Em um seguidor de linha as interrupções podem ser usadas para leitura de alguns sensores de obstáculos.

A interrupção depende de um tipo especifico de função chamada de ISR (Interrupt Service Routines). Uma ISR não deve receber nenhum parâmetro e não deve retornar nada.

Em geral, Uma ISR deve ser o mais curta e rápida possível. Se o seu código utiliza múltiplas interrupções, apenas uma pode ser executada por vez. Por depender de interrupções, as funções delay() e millis() não funcionarão enquanto outra ISR é executada. A função delayMicroseconds() funcionará normalmente já que não depende de interrupções.

Para passar dados entre a ISR e o programa principal tipicamente usa-se variáveis globais. Variáveis declaradas dentro de uma ISR devem ser declaradas como volatile.

Uma interrupção é declarada na função setup() e tem a sintaxe abaixo

attachInterrupt(interrupção, ISR, modo)

parâmetros: Interrupção: o número da interrupção (int) ISR: a função ISR que é chamada quando a interrupção ocorre mode: define o modo como a interrupção deve ser ativada. Quatro constantes são predefinidas como validas: LOW: a interrupção é ativada sempre que o pino estiver em nível lógico baixo CHANGE: a interrupção é ativada sempre que o pino mudar de nível lógico RISING: a interrupção é ativada sempre que o pino mudar de nível lógico baixo para alto FALLING: a interrupção é ativada sempre que o pino mudar de nível lógico alto para baixo

9.3 Ativando e desativando Interrupções

Por padrão, as interrupções são ativadas no Arduino. Caso alguma tarefa essencial esteja sendo executada é recomendado que as interrupções sejam desabilitadas. Após a execução da rotina deve-se reativar as interrupções. Com essa finalidade temos as funções abaixo:

noInterrupts() interrupts()

No caso de desativar uma interrupção especifica, pode-se usar a função:

detachInterrupt(interrupção)

Capítulo 10

Controle

Existem vários métodos de controlar o robô. O método a ser utilizado deve ser pensado na fase de projeto, de modo que o controle seja otimizado. Nesse capítulo serão mostrados dois exemplos de como controlar o seguidor de linha.

10.1 Controle utilizando velocidades constantes

Este é o método simples de controlar o seguidor. Considerando um robô com duas rodas e com 3 sensores. Nesse caso temos as seguintes possibilidades:

- Quando o sensor central está ativado ambas as rodas giram com a mesma velocidade.
- Quando o sensor da direita está ativado, a roda da esquerda deve estar mais rápida que a da direita, realizando assim uma curva para direita
- Quando o sensor da esquerda está ativado, a roda da direita deve estar mais rápida que a da esquerda, realizando assim uma curva para esquerda
- Se mais de um sensor estiver ativo simultaneamente a ação a ser realizada deve ser decidida de acordo com o caso.

As velocidades devem ser escolhidas tendo o em mente o comportamento desejado para o robô. Se a velocidade média for alta, o controle com esse método fica mais difícil.

10.2 Controladores PID

Os controladores PID tem uma aplicação muito vasta no controle de processos. Ele é composto por 3 tipos de ações:

- Uma ação proporcional ao erro, responsável por minimizar este erro;
- Uma ação integrativa, que leva em conta os momentos anteriores, responsável por anular o erro;
- Uma ação derivativa, que leva em conta o momento atual e é responsável por melhorar o tempo de resposta do sistema.

O algoritmo deve implementar esses 3 termos. Na equação abaixo o termo integrativo foi determinado pela regra do Trapézio e o derivativo pela inclinação da reta:

$$a(n) = k_p e(n) + k_i \sum_{i=0}^{n} \left(\frac{(e(i) + e(i-1))T}{2}\right) + k_d \frac{(e(i) - e(i-1))}{T}$$

Em que:

- $a(n)$ é a ação;
- $e(n)$ é o erro;
- k_p, k_i e k_d são as constantes dos termos proporcional, integrativo e derivativo respectivamente;
- e T é o período de amostragem....

As constantes k_p, k_i e k_d são determinadas empiricamente. Recomenda-se determina-las nessa ordem para determinar quais valores otimizam a performance. Vale lembrar que, como o algoritmo é implementando computacionalmente, o termo integrativo pode sofrer overflow com o tempo e que a ação é limitada pela resolução do driver de PWM do Arduino.

Capítulo 11

Apendices

11.1 Softwares Uteis

- Labcenter Proteus Design Suite (Isis e Ares) - Para Simulações de circuitos e desing de placas de circuito impresso
- Cadsoft EAGLE PCB Design - Também para desing de placas de circuito impresso (Possui uma versão gratuita)
- Fritzing - Um software amigavel para desenho de circuitos e placas de circuito impresso.

11.2 Links Uteis

- Site Oficial do Arduino - `https://www.arduino.cc/`
- Serie de tutoriais para Arduino por Jeremy Blum (em ingles) - `https://www.youtube.com/watch?v=fCxzA9_kg6s&list=PLA567CE235D39FA84`
- Serie de tutoriais para Eagle por Jeremy Blum (em ingles) - `https://www.youtube.com/watch?v=1AXwjZoyNno&list=PL868B73617C6F6FAD`
- Blog FilipeFlop - `http://blog.filipeflop.com/`
- Eletrodex - Site para compra de componentes com preços em conta e entrega rapida para Belo Horizonte `http://www.eletrodex.com.br/`