Oficina Seguidor de Linha

O Guia Definitivo do Mochileiro Maker

PETEE UFMG

OFICINA DE SEGUIDOR DE LINHA, UNIVERSIDADE FEDERAL DE MINAS GERAIS

HTTP://PETEE.CPDEE.UFMG.BR/

Este texto foi escrito como complemento para a oficina de seguidor de linha, promovida pelo Programa de Educação Tutorial da Engenharia Elétrica (PETEE) da Universidade Federal de Minas Gerais.

Mateus Simões, Junho 2017

Sumário

I Tópicos Básicos

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.

Figura 1.1: Operação de robô autônomo em uma industria.

A conclusão de um percurso e sucesso do projeto depende, é claro, da eficiência mecânica do protótipo, do controle, autonomia energética e diversos outros fatores, como a estruturação de tomadas de decisões que respondam aos diferentes tipos de trajetória que o robô possa enfrentar. Sendo assim, o desenvolvimento de um robô seguidor de linha pode se tornar um projeto bastante complexo, dependendo do nível de exigência de precisão.

Com o objetivo de auxiliar os estudantes na criação e no desenvolvimento de robôs seguidores de linha, o PETEE promove a Oficina de Seguidores de Linha e disponibiliza esta apostila para detalhar os passos básicos e relevantes à criação desse tipo de robô.

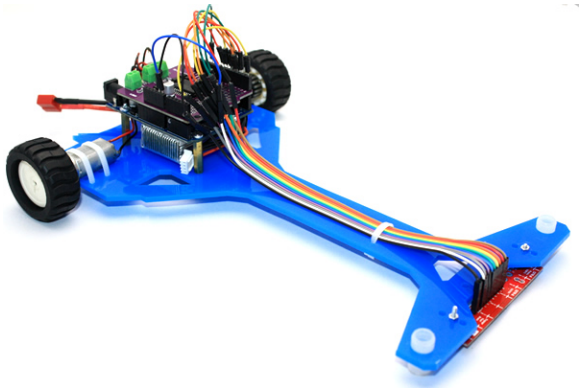


Figura 1.2: Exemplo de um robô seguidor de linha.

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, é necessário 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.

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 o seu próprio robô.

Há diversas formas de se construir um robô deste tipo. Com fins didáticos, neste material há uma ênfase maior em um dos métodos mais comuns, com a utilização do microcontrolador Arduino.

Entretanto, não há impedimentos para a utilização das informações aqui apresentadas em outras plataformas e até mesmo a aplicação de métodos totalmente diferentes para o mesmo fim. Este guia é introdutório ao assunto e o conhecimento não se limita à este guia. Encorajamos que o leitor tente sempre algo novo e diferente do conteúdo deste guia.

Muitos temas são temas já apresentados em diversos fóruns e sites. Já outros tópicos são frutos da experiência do grupo na construção de robôs seguidores de linha.

2. Introdução à Programação

A programação envolve um conjunto de métodos e instruções que caracterizam o funcionamento do robô. Grande parte do desempenho do projeto depende da programação. É na da programação que técnicas de controle como o PID (*proportional–integral–derivative*) são implementadas.

A linguagem de programação do projeto está relacionada à plataforma de desenvolvimento. Entre as diversas linguagem disponíveis, uma das mais comuns é a linguagem C, muito utilizada na programação de microcontroladores, como Arduino.

A comunidade em volta da linguagem de programação C é bem extensa, o que reforça a sua popularidade. Dessa forma, não é difícil encontrar conteúdo relacionada à linguagem, caso o desejo do leitor seja aprofundar seus conhecimentos na linguagem, uma vez que apenas as funções básicas serão trabalhadas aqui.

2.1 Linguagem C

O código em linguagem C é descrito por palavras chaves, que são palavras reservadas que possuem, cada uma, um significado específico para o compilador. Palavras chaves são parte da sintaxe e não podem ser utilizadas como um identificador.

Identificadores são nomes que o programador dá às entidades como variáveis, funções, estruturas.

```
int idade;
double altura;
```

No código, peso e altura são identificadores. E lembre-se, os identificadores devem ser diferentes das palavras reservadas e é recomendável que o seu nome tenha alguma relação como o seu uso.

int	double	float	char
struct	enum	void	return
while	else	do	continue
if	for	false	true

Tabela 2.1: Exemplos de palavras chaves em C

2.2 Bibliotecas

Uma biblioteca em C é um grupo de funções e declarações criadas para utilização em outros programas. Portanto, uma biblioteca consiste da implementação de varias tarefas em um arquivo com extensão .c. Por exemplo, para utilizar as funções de entrada e saída de dados ou as funções de controle de um periférico qualquer, que não fazem parte da lista de comandos padrões da linguagem, é necessário utilizar uma biblioteca específica.

O trecho de código abaixo ilustra a utilização de bibliotecas em um código, que devem estar no cabeçalho do código e fora de qualquer função do programa.

```
#include <stdio.h>  //Para funcoes fprintf() and EOF
#include "SevSeg.h" //Para utilizar o display de sete seguimentos
```

2.3 Tipos de Dados

Em programação C, variáveis devem ser declaradas antes de serem utilizadas. Variável é uma área da memória onde é armazenado dados do programa. Cada variável tem um tipo de dado associado, que depende da informação que será armazenada.

Tipos de dados se refere ao tipo e ao tamanho de um dado associado à uma variável ou função. Há três tipos de dados fundamentais: inteiros, ponto flutuante e caractere. Mas há também os tipos de dados derivados, como os arrays e ponteiros.

2.3.1 Fundamentais

Há três tipos de dados fundamentais, que são os mais básicos e utilizados em qualquer programa.

Inteiro

Inteiros são quaisquer números, positivos e negativos, que não possuem casas decimais, como 0, -7, 10. Em C, a palavra int é reservada para declarar variáveis do tipo inteiro.

```
int id;
```

Nesse exemplo, id é um variável do tipo inteiro. Também é possível declarar mais de uma variável do tipo inteiro de uma vez.

```
int id, idade;
```

Ponto Flutuante

Variáveis do tipo ponto flutuante são capazes de armazenar números reais com casas decimais. Em C é possível declarar um variável usando a palavra chave float ou double, por exemplo:

```
float saldoBanco;
double preco;
```

A Diferença entre float e double está relacionada ao tamanho ocupado na memória por cada tipo de dado. O tamanho de um float é 4 bytes e tem uma precisão de 6 dígitos. Já o tamanho do double é 8 bytes e uma precisão de 14 dígitos.

Caractere

A palavra chave char é utilizada para declaração de caracteres e tem um tamanho de 1 byte, por exemplo:

```
char letra = 'M';
```

2.3.2 Ponteiros

Ponteiros em C é uma variável que armazenam/apontam para endereços de outras variáveis. Um ponteiro é definido com o tipo de dado para o qual irá apontar.

Para obter o endereço de uma variável VAR basta utilizar o operador de referência &. Dessa forma, &VAR irá retornar o endereço de VAR.

Para saber o valor da variável para a qual o ponteiro aponta utiliza-se o operador *. Para definir um ponteiro utilize a sintaxe abaixo.

```
tipo_de_dado *nome_ponteiro;
```

Para obter o valor apontado por um ponteiro

```
#include <stdio.h>
int main()
{
   int *ptr, q;
   q = 50;

   // Endereco associado a ptr
   ptr = &q;

   // Exibe o valor de q usando o ponteiro
   printf("%d", *ptr);

   return 0;
}
```

Código 2.1: Utilização de ponteiros

O código 2.1 produz a seguinte saída.

Saída do Código 2.1

```
50
```

O valor do endereço obtido será diferente para a cada execução do programa.

2.4 Arranjos

Um arranjo é uma coleção de dados do mesmo tipo. Há dois tipos de arranjos, unidimensional ou multidimensional.

2.4.1 Unidimensional

Se você deseja armazenar a nota de 100 alunos em uma prova, a melhor opção é utilizar um arranjo.

```
double nota[100];
```

A sintaxe geral para a declaração de um arranjo de dados é ilustrado abaixo

```
tipo_de_dado nome_do_arranjo[tamanho_do_arranjo];
```

Após a declaração, o nome e o tamanho do arranjo não pode ser alterado. Portanto, defina estes parâmetros com cautela.

Os elementos de um arranjo podem ser acessados por meio de índices. Por exemplo, a nota do primeiro aluno é acessado por nota[0], do segundo aluno será nota[1] e assim por diante.

O primeiro índice de um arranjo será sempre 0 e não 1, como no exemplo, a nota do primeiro aluno é nota[0]. Se o tamanho do arranjo é N, então o último índice acessível será N - 1. No exemplo, a nota do último aluno será nota[99].

Também é possível inicializar um arranjo. Para isso, utilize como referência o código abaixo.

```
int nota[5] = {19, 10, 30, 17, 24};
```

2.4.2 Multidimensional

Em C, é possível criar arranjos de arranjos, que são chamados de arranjos multidimensionais. Por exemplo

```
float x[3][4];
```

Aqui, x é um arranjo de duas dimensões que pode armazenar até 12 elementos. Como analogia, pense em uma tabela de 3 linhas e 4 colunas.

Também é possível inicializar um arranjo multidimensional. Para isso, utilize como referência o código abaixo, que apresenta a inicialização de um arranjo de duas dimensões.

```
int matriz_identidade[3][3] = {{1, 0, 0}, {0, 1, 0},  {0, 0, 1}};
```

2.5 Operadores

Operadores são palavras reservadas para a realização de operações matemáticas específica. Serão abordados dois tipos de operadores, os aritméticos e os lógicos.

2.5.1 Operadores Aritméticos

Um operador aritmético realiza operações matemáticas como adição, subtração e multiplicação em valores numéricos, seja ele constante ou variável.

Operador	Função
+	soma
-	subtração
*	multiplicação
/	divisão
%	resto da divisão

Tabela 2.2: Operadores aritméticos.

```
#include <stdio.h>
void main()
{
    int a = 9,b = 4, c;

    c = a+b;
    printf("a+b = %d \n",c);

    c = a-b;
    printf("a-b = %d \n",c);

    c = a*b;
    printf("a*b = %d \n",c);

    c=a/b;
    printf("a/b = %d \n",c);

    c=a%b;
    printf("Resto de a dividido por b = %d \n",c);
}
```

Código 2.2: Demonstração dos operadores aritméticos

O código 2.2 produz a seguinte saída.

Saída do Código 2.2

```
a+b = 13
a-b = 5
a*b = 36
a/b = 2
Resto de a dividido por b=1
```

Também é possível realizar o incremento e o decremento de valores utilizando as variáveis aritméticas, como no Código 2.3

```c
#include <stdio.h>
int main()
{
    int a = 10, b = 100;
    float c = 10.5, d = 100.5;

    printf("++a = %d \n", ++a);

    printf("--b = %d \n", --b);

    printf("++c = %f \n", ++c);

    printf("--d = %f \n", --d);

    return 0;
}
```

Código 2.3: Demonstração de incremento e decremento.&

O código 2.3 produz a seguinte saída.

Saída do Código 2.3

```
++a = 11
–b = 99
++c = 11.500000
++d = 99.500000
```

Os operadores de incremento ++ e decremento – foram utilizados como prefixo, mas também é possível utiliza-los como sufixo. Como a++ ou a–.

Uma outra aplicação dos operadores aritméticos é a atribuição. O principal operador de atribuição é o sinal de igualdade =.

Para realizar a comparação de valores, utiliza-se os operadores apresentados na Tabela 2.4. Onde uma função retornar 0 indica falso e retornar 1 indica verdadeiro.

Operador	Exemplo	Equivalente a
=	a = b	a = b
+=	a += b	a = a+b
-=	a -= b	a = a-b
*=	a *= b	a = a*b
=	a /= b	a = a/b
%=	% = b	a = a%b

Tabela 2.3: Operadores aritméticos utilizados em atribuições.

Operador	Função	Exemplo
==	Igual a	5 == 3 retorna 0
>	Maior que	5 > 3 retorna 1
<	Menor que	5 < 3 retorna 0
!=	Diferente de	5 != 3 retorna 1
>=	Maior ou igual a	5 >= 3 retorna 1
<=	Menor ou igual a	5 <= 3 retorna 0

Tabela 2.4: Operadores aritméticos utilizados em atribuições.

2.5.2 Operadores Lógicos

Os operadores lógicos são apresentados na Tabela 2.5.

Operador	Função
&&	AND Lógico, retorna verdadeiro se todos os valores são verdadeiros
||	OR Lógico, retorna verdadeiro se pelo menos um valor é verdadeiro
!	NOT Lógico, retorna verdadeiro se o valor for falso

Tabela 2.5: Operadores lógicos.

Como um exemplo da aplicação do operador AND, se c = 5 e d = 2, então a expressão ((c == 5) && (d > 5)) é igual a 0 pois d > 5 é falso.

Já para o operador OR, se c = 5 e d = 2, então a expressão ((c == 5) || (d > 5)) é igual a 1 pois pelo menos um dos valores é verdadeiro, neste caso c == 5.

A operação NOT inverte o estado lógico do valor, logo, se c = 5, então a expressão !(c == 5) é igual a 0.

2.6 Condicionais

Operadores condicionais são uma classe de funções que controlam o fluxo de execução do programa.

2.6.1 if

O comando if avalia uma expressão entre parenteses. Se a espressão avaliada é verdadeira, um bloco de comandos é executado, se a expressão é falsa, esse bloco de comandos é ignorado.

```
if (expressao)
{
   // bloco de comandos
}
```

O Código 2.4 exibe o número informado se o usuário inseriu um valor menor que zero. Se o usuário inserir um valor positivo, o número não é exibido.

```
#include <stdio.h>
int main()
{
    int numero;

    printf("Insira um valor inteiro: ");
    scanf("%d", &numero);

    // Verdadeiro se numero menor que zero
    if (numero < 0)
    {
        printf("Voce inseriu %d.\n", numero);
    }
    return 0;
}
```

Código 2.4: Demonstração do comando if

O código 2.4 produz a seguinte saída se o valor informado é menor que zero.

Saída do Código **??** para um número inserido menor que zero.

```
Insira um valor inteiro: -2
Voce inseriu -2.
```

O código 2.4 produz a seguinte saída se o valor informado é maior que zero.

Saída do Código 2.4 para um número inserido menor que zero.

```
Insira um valor inteiro: 4
```

2.6.2 if...else

O comando if...else executa um bloco de códigos caso uma expressão seja verdadeira ou executa um outro bloco caso a expressão seja falsa.

```
if (expressao) {
    // bloco de comandos A
}
else {
    // bloco de comandos B
}
```

O Código 2.5 informa se o número inserido pelo usuário é par ou impar.

```
#include <stdio.h>
int main()
{
    int numero;
    printf("Insira um valor inteiro: ");
    scanf("%d",&numero);

    // Verdadeiro se resta 0
    if( numero%2 == 0 )
        printf("%d : numero par.",numero);
    else
        printf("%d : numero impar.",numero);
    return 0;
}
```

Código 2.5: Demonstração do comando if...else

O código 2.4 produz a seguinte saída se o valor informado é menor que zero.

Saída do Código 2.5.

```
Insira um valor inteiro: 7
7 : numero impar.
```

2.6.3 if...elsif...else

O comando if...elsif...else executa um bloco de códigos caso uma expressão seja verdadeira, e verifica a próxima expressão por meio do comando elsif e assim por diante, até que alguma condição seja verdadeira ou não haja mais verificações.

```
if (expressao1)
{
   // bloco executado se expressao1 for verdadeira
}
else if(expressao2)
{
   // bloco executado se expressao2 for verdadeira
}
else if (expressao3)
{
   // bloco executado se expressao3 for verdadeira
}
```

```
.
.
else
{
   // bloco executado se todas as expressoes sao falsas
}
```

O Código 2.6 verifica se um primeiro número informado pelo usuário é igual, maior ou menor que um segundo número informado.

```
#include <stdio.h>
int main()
{
    int numero1, numero2;
    printf("Insira dois valores inteiros: ");
    scanf("%d %d", &numero1, &numero2);

    //Avalia se forem iguais.
    if(numero1 == numero2)
    {
        printf("Resultado: %d = %d",numero1,numero2);
    }

    //Avalia se numero1 maior que numero2.
    else if (numero1 > numero2)
    {
        printf("Resultado: %d > %d", numero1, numero2);
    }

    // Se as duas expressoes sao falsas
    else
    {
        printf("Resultado: %d < %d",numero1, numero2);
    }

    return 0;
}
```

Código 2.6: Demonstração do comando if...elsif...else

O código 2.6 produz a seguinte saída.

Saída do Código 2.6.

```
Insira dois valores inteiros: 12
23
Resultado: 12 < 23
```

2.6.4 switch...case

Os comandos if...elsif...else oferecem grande opções de fluxo de dados, mas se é necessário avaliar o valor de uma única variável, é melhor utilizar os comandos switch..case, que oferece uma velocidade de execução muitas vezes maior que o if...elsif...else, além de possuir uma sintaxe mais limpa e fácil de entender.

```
switch (n)
{
    case constante1:
        // Bloco executado se n = constante1;
        break;

    case constante2:
        // Bloco executado se n = constante2;
        break;
        .
        .
        .
    default:
        // Bloco executado se n diferente de todas as constantes;
}
```

Código 2.7: Demonstração do comando switch...case

Quando a constante avaliada é igual ao valor da expressão em um caso, o fluxo é passado para o bloco associado àquele caso. Como exemplo, há o Código 2.7. Uma vez selecionado um caso, o compilador irá executar os blocos associados a partir do caso selecionado até chegar ao fim do switch ou encontrar o comando break.

Portanto, o comando break é utilizado para evitar a execução do blocos relacionados ao casos posteriores ao bloco do caso selecionado.

2.7 Estruturas de Repetição

As estruturas de repetição são comando utilizados para repetir blocos de comandos em uma rotina até que a condição de parada seja atendida.

2.7.1 for

A sintaxe do comando loop é a seguinte

```
for (inicializacao; expressao_Teste; expressao_Atualizacao)
{
       // bloco de codigos
}
```

O Código 2.8 calcula a soma dos dos N primeiro números naturais.

```c
#include <stdio.h>
int main()
{
    int num, cont, soma = 0;
    printf("Insira um valor inteiro: ");
    scanf("%d", &num);

    // for loop termina quando n <= cont
    for(cont = 1; cont <= num; ++cont)
    {
        soma += cont;
    }

    printf("Soma = %d", soma);

    return 0;
}
```

Código 2.8: Demonstração do comando for

O código 2.8 produz a seguinte saída.

Saída do Código 2.8.

```
Insira um valor inteiro: 10
Soma = 55
```

O valor informado é armazenado na variável num. Suponha que o usuário informou o número 10. O contador cont é inicializado com o valor 1 e a expressão de teste é avaliada. Como cont <= num (1 é menor ou igual a 10) é verdadeiro, o bloco de código do comando loop é executado.

Após ser executado, ocorre a atualização ++cont é executado e o valor de cont será atualizado para 2. Novamente a expressão de teste será avaliada. Como 2 é menor que 10, o bloco de códigos do loop será executado novamente e o valor de cont será atualizado para 3 e assim por diante.

Quando cont é igual a 11, a expressão de teste será falsa e o bloco de loop for será interrompido e a soma total será exibida.

2.7.2 while

O laço while avalia uma expressão de teste. Se essa expressão é verdadeira, o bloco de códigos no laço são executados e a expressão de testes são executadas novamente até que essa expressão seja falsa.

```c
while (expressaoTeste)
{
    //codigos
}
```

O Código 2.9 é o exemplo de um código que calcula o valor fatorial de um número informado pelo usuário.

```c
#include <stdio.h>
int main()
{
    int numero;
    int fatorial;

    printf("Insira um valor inteiro: ");
    scanf("%d",&numero);

    fatorial = 1;

    // Termina quando numero < 0
    while (numero > 0)
    {
        fatorial *= numero;  // fatorial = fatorial*numero;
        --numero;
    }

    printf("Fatorial= %d", fatorial);

    return 0;
}
```

Código 2.9: Demonstração do comando for

O Código 2.9 produz a seguinte saída.

Saída do Código 2.9.

Insira um valor inteiro: : 5
Fatorial = 120

2.7.3 do..while

O comando do..while é similar ao laço comando while com uma única importante diferença. O bloco do comando do..while é executado uma vez antes de checar a expressão de teste, pois a avaliação ocorre após a execução do bloco. Já no comando while, a expressão de teste é avaliada antes de executar o bloco de comandos.

```c
do
{
   // codigos
}
while (expressaoTeste);
```

O Código 2.10 é o exemplo de um código que soma valores informados pelo usuário até que este valor seja 0.

```c
#include <stdio.h>
int main()
{
```

```
    double numero, soma = 0;

    // loop executado pelo menos uma vez
    do
    {
        printf("Informe um numero: ");
        scanf("%lf", &numero);
        soma += numero;
    }
    while(numero != 0.0);

    printf("Soma = %.2lf",soma);

    return 0;
}
```

Código 2.10: Demonstração do comando do...while

O Código 2.10 produz a seguinte saída.

Saída do Código 2.10.

```
Informe um numero: 1.5
Informe um numero: 2.4
Informe um numero: -3.4
Informe um numero: 4.2
Informe um numero: 0
Soma = 4.70
```

2.8 Funções

Conforme o código em C se torna maior e mais complexo, é uma boa prática dividir um código em C em vários blocos menores chamados de funções. Além disso, as funções permitem que um bloco de códigos em C sejam reutilizados em várias parte do programa de forma simples, tornando o código mais enxuto, limpo e de fácil compressão.

Uma função em C é expressa entre chaves “ ” e um conjunto de funções produzem um programa. A sintaxe da definição de de uma função é ilustrada abaixo.

```
tipo_retorno nome_funcao(lista de argumentos)
{
  //Corpo da funcao;
}
```

A definição de uma função deve ser feita fora de qualquer outra função. Uma vez declarada, uma função pode ser chamada em qualquer parte do código, inclusive dentro da própria função, o que caracteriza um comportamento recursivo. A sintaxe da chamada de uma função qualquer é ilustrado abaixo.

```
nome_funcao(lista de argumentos);
```

Há duas formas de chamar uma função em C, chamada por valor e chamada por referência, serão discutidos ambos os tipos a seguir.

2.8.1 Chamada por valor

Neste tipo de chamada o valor da variável é passada como argumento da função. O valor atual da variável não pode ser alterado pela função pois o valor passado como argumento é copiado para outro espaço da memória e este novo valor será manipulado. Por exemplo o Código 2.11, os valores das variáveis "m"e "n"são passados para a função troca. Em seguida, "m"e "n"são copiados para novos parâmetros chamados "a"e "b"que possuem espaços na memória distintos de "n"e "m". Dessa forma, a manipulação de dados de "a"e "b"não afetam o valor de "m"e "n".

```
#include<stdio.h>
int main()
{
    int m = 22, n = 44;

    // chamada da fncao troca
    printf("Valores antes da troca  m = %d e n = %d", m, n);

    troca(m, n);

    printf("\nValores depois da troca m = %d\n e n = %d", m, n);
}

void troca(int a, int b)
{
    int tmp;
    tmp = a;
    a = b;
    b = tmp;
}
```

Código 2.11: Demonstração da utilização das funções

O Código 2.11 produz a seguinte saída.

Saída do Código 2.11.

```
Valores antes da troca m = 22 e n = 44
Valores antes da troca m = 22 e n = 44
```

2.8.2 Chamada por referência

Na chamada por referência o endereço do argumento é passado em vez do seu valor. O valor da variável pode ser alterado pela função pois o mesmo espaço de memória é utilizado. Para isso, é feito o uso de ponteiros.

Por exemplo o Código 2.12, os valores dos endereços das variáveis "m"e "n"são passados para a função troca. A função copia o endereço de "m"e "n"para os ponteiros "a"e "b"que irão manipular

diretamente os dados de "m"e "n". Dessa forma, a manipulação de dados de "a"e "b"afetam o valor de "m"e "n".

```
#include<stdio.h>
int main()
{
    int m = 22, n = 44;

    // chamada da fncao troca
    printf("Valores antes da troca  m = %d e n = %d", m, n);

    troca(&m, &n);

    printf("\nValores depois da troca m = %d\n e n = %d", m, n);
}

void troca(int *a, int *b)
{
    int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
}
```

Código 2.12: Demonstração da utilização das funções

O Código 2.12 produz a seguinte saída.

Saída do Código 2.12.

```
Valores antes da troca m = 22 e n = 44
Valores antes da troca m = 44 e n = 22
```

3. Eletrônica Básica

3.1 Grandezas Físicas

Muitas vezes uma grandeza assume valores muito grandes ou muito pequenos, tornando inviável a sua representação na unidade corrente. Dessa maneira, existem alguns "multiplicadores "que ajudam a representar os valores das grandezas de forma mais "agradável". Vejamos alguns múltiplos e submúltiplos fundamentais em eletrônica.

Prefixo	Símbolo	Potência
pico	p	10^{-12}
nano	n	10^{-9}
micro	μ	10^{-6}
mili	m	10^{-3}
quilo	k	10^{+3}
mega	M	10^{+6}
giga	G	10^{+9}
tera	T	10^{+12}

Tabela 3.1: Multiplicadores.

3.1.1 Tensão

A tensão elétrica em um circuito é a diferença de potencial entre dois pontos distintos. Em todo circuito elétrico é necessário a existência de uma fonte de tensão para fornecer energia ao circuito. A unidade da tensão elétrica é o volts (V).

Tensão Contínua

Tensão que descreve uma constante, ou seja, seu valor não varia ao longo do tempo.

Figura 3.1: Símbolo da fonte de tensão contínua.

Tensão alternada

É aquela que varia de valor com o passar do tempo, alternando a sua polaridade. A mais comum das tensões alternadas é a tensão senoidal, que assume uma infinidade de valores no decorrer do tempo e oscila em uma determinada frequência.

Figura 3.2: Símbolo da fonte de corrente alternada.

3.1.2 Corrente Elétrica

Podemos definir uma corrente elétrica como sendo o fluxo ordenado de elétrons por um meio condutor. De fato, ao submetermos um material condutor a uma diferença de potencial, os elétrons fluirão do ponto de maior concentração de elétrons para o ponto de menor concentração com sentido ordenado.

No fluxo convencional a corrente flui do polo positivo para o negativo, de modo que esse sentido não condiz com a realidade. No entanto, para efeito prático e didático o fluxo convencional será utilizado. A unidade da corrente elétrica é o ampère (A).

Figura 3.3: Símbolo da fonte de corrente.

3.2 Resistor

Pode-se definir a resistência elétrica como sendo um obstáculo à passagem da corrente elétrica. Em todo circuito elétrico existe uma resistência elétrica qualquer que dificulta a passagem da corrente. Até mesmo um pequeno pedaço de fio de cobre possui sua resistência à corrente. A resistência elétrica , cujo símbolo é a letra R, é medida em Ohm (Ω).

3.2.1 Associação de Resistores

Os resistores podem ser associados para se obter novos valores de resistências equivalentes R_{eq} no circuito.

Associação em Série

A resistência total, R_{eq}, de resistores conectados em serie é a soma das suas resistências.

$$R_{eq} = R_1 + R_2 + ... + R_n$$

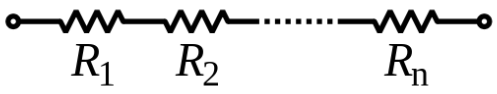


Figura 3.4: Associação de resistores em série

Associação em Paralelo

A resistência equivalente R_{eq} de resistores conectados em paralelo é inverso da soma dos inversos de cada resistência.

$$\frac{1}{R_{eq}} = \frac{1}{R_1} + \frac{1}{R_2} + ... + \frac{1}{R_n}$$

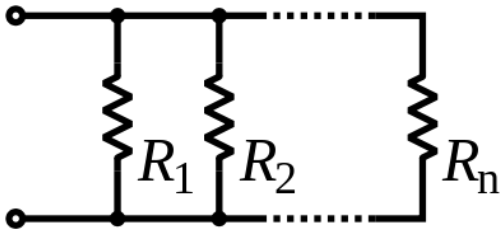


Figura 3.5: Associação de resistores em Paralelo

3.2.2 Código de Cores

Os resistores vendidos comercialmente não possuem seus valores impressos numericamente, mas em um código de cores. Para descobrir qual o valor de resistor, utilize a tabela 3.6.

Por exemplo, imagina um resistor que possui quatro faixas com as seguintes cores, em ordem da esquerda para direita: marrom, preto, vermelho e dourado. A cor marrom representa 1, a cor preto 0, a terceira cor é um multiplicador de 100 e a quarta faixa representa a tolerância, que é 5%. Portanto, o resistor lido é:

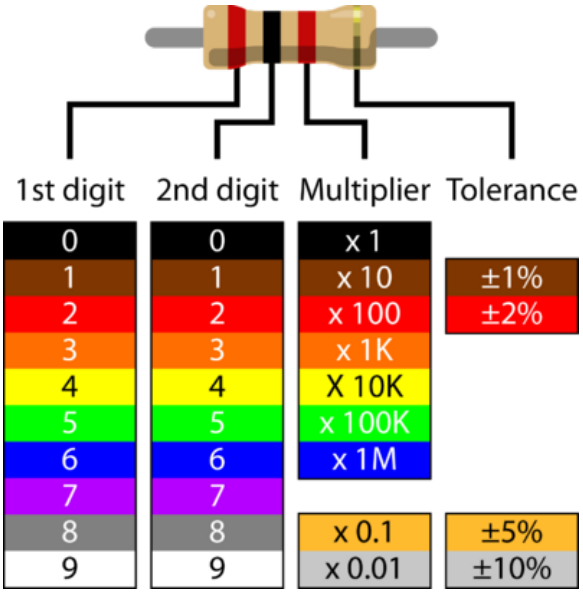


Figura 3.6: Tabela de cores para determinar valores de resistores.

$$10 \cdot 100 \pm 5\% = (1000 \pm 5\%)\Omega = (1k \pm 5\%)\Omega$$

3.3 A Lei de Ohm

A Lei de Ohm é a lei de Ohm é umas das leis fundamentais da eletrônica. Num circuito elétrico fechado, a intensidade de corrente elétrica é diretamente proporcional à tensão aplicada ao circuito e inversamente proporcional à resistência do mesmo. Pode ser expressa matematicamente pela equação 3.1.

$$A = \frac{V}{R} \tag{3.1}$$

3.4 Potência Elétrica

Fisicamente podemos definir potência como sendo a energia consumida ou liberada em um intervalo de tempo. Em eletricidade, diz-se que a energia de um Joule (1J), liberada ou consumida em um segundo (1s) equivale a um watt (1W). De fato, a potência elétrica, cujo símbolo é a letra P, é medida em Watt (W). Matematicamente, a potência elétrica pode ser representada como:

$$P = V \cdot I = \frac{V^2}{R} = RI^2 \tag{3.2}$$

3.5 Diodos

O diodo é um componente eletrônico de dois terminais, que conduz corrente elétrica preferivelmente em um só sentido, bloqueando a sua passagem no sentido oposto. Esse comportamento unidirecional é chamado retificação, sendo utilizado para converter corrente alternada em corrente contínua ou extrair a informação de um sinal modulado em amplitude (AM), por exemplo.

3.5.1 Retificadores

São os diodos mais comuns, fabricados com o objetivo primordial de permitirem a passagem da corrente elétrica em um só sentido (polarização direta), cumprindo um papel indispensável na transformação de corrente alternada em corrente contínua. Possuem vários tamanhos e formatos, de acordo com a sua potência nominal.

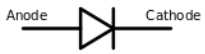


Figura 3.7: Símbolo de um diodo retificador.

3.5.2 LEDs

São diodos semicondutores que, quando energizados, emitem luz. A luz não é monocromática (como em um laser), mas consiste de uma banda espectral relativamente estreita, sendo produzida pelas interações energéticas dos elétrons. O processo de emissão de luz pela aplicação de uma fonte de energia elétrica é chamado eletroluminescência.

Figura 3.8: Símbolo de um LED.

3.5.3 Zener

São diodos fabricados para conduzir a corrente elétrica em sentido inverso (polarização inversa). Este efeito é chamado de "ruptura zener"e ocorre em um valor de tensão bastante preciso, permitindo que esse diodo seja utilizado com uma referência de tensão. São bastante empregados em circuitos reguladores de tensão em fontes de alimentação.

Figura 3.9: Símbolo de um diodo zener.

3.6 Transistor Bipolar

O principio do transistor é poder controlar a corrente. Ele é montado numa estrutura de cristais semicondutores, de modo a formar duas camadas de cristais do mesmo tipo intercaladas por uma camada de cristal do tipo oposto, que controla a passagem de corrente entre as outras duas.

Cada uma dessas camadas recebe um nome em relação à sua função na operação do transistor. As extremidades são chamadas de emissor e colector, e a camada central é chamada de base. Os aspectos construtivos simplificados e os símbolos eléctricos dos transistores são mostrados na figura abaixo. Observe que há duas possibilidade de implementação.

Figura 3.10: Transistor bipolar PNP.

Figura 3.11: Transistor bipolar NPN.

3.7 Eletrônica Digital

Eletrônica Digital é definida como eletrônica que emprega a utilização de sinais elétricos em apenas dois níveis de corrente (ou tensão) para definir a representação de valores binários.

Circuitos Lógicos baseiam seu funcionamento na lógica binária, que consiste no fato de que toda informação deve ser expressa na forma de dois dígitos (tanto armazenada, como processada), sendo tais dígitos, 0 (zero) ou 1 (um).

A partir daí surge intuitivamente à nomeação “digital” (dois dígitos). Este fato auxilia para a representação de estados de dispositivos que funcionam em dois níveis distintos, sendo estes: ligado/desligado (on/off), alto/baixo (high/low), verdadeiro/falso (true/false) entre outros.

3.7.1 Álgebra Booleana

A lógica binária surgiu Desenvolvida por George Boole, esta álgebra é uma série de postulados e operações simples para resolver muitos problemas. Sua maior importância só foi percebida com o surgimento da eletrônica. Para Boole, só existiam duas condições possíveis ou estados para analisar qualquer situação. As variáveis lógicas só podem adquirir dois estados, que podem ser verdadeiro ou falso; 0 ou 1; alto ou baixo; etc.

A eletrônica digital faz uso da lógica Booleana para representar valores. É muito comum em zeros e uns quando se fala de lógica binária, mas o que significa 0 e 1? Em eletrônica, 0 pode significar 0V e 1 pode representar 5V, como é o caso do Arduino, por exemplo.

A diversos dispositivos que realizam operações matemáticas com lógica binária, como portas lógicas, multiplexadores, flip flops e etc. Será abordado apenas portas lógicas neste texto.

3.7.2 Portas Lógicas

Em eletrônica, uma porta lógica é um dispositivo capaz de implementar funções lógicas, isso é, realizar operações matemáticas com números binários, de forma que múltiplas entradas lógicas produzam apenas uma saída lógica.

Porta AND

A saída será 1 se e somente se todas as variáveis de entrada forem 1. A expressão matemática da operação ANS é $A \cdot B = C$ e lê-se A e B igual a C.

Porta OR

A saída estará no nível alto (1) se uma ou mais das entradas estiverem no nível alto. A expressão matemática da operação OR é $A + B = C$ e lê-se A ou B igual a C.

Porta NOT

Se a entrada for 1 ou nível lógico alto, a saída será 0 ou nível lógico baixo. A expressão matemática da operação NOT é $\overline{A} = B$ e lê-se não A igual a B.

3.7.3 Tabela Verdade

A tabela verdade lista todas as combinações de valores que as variáveis de entrada podem assumir e os correspondentes valores da saída. Uma tabela verdade para uma lógica de dois bits é ilustrada abaixo. Onde A e B são entradas.

A	B	$A+B$	$A \cdot B$	$\overline{A}$	$\overline{B}$
0	0	1	0	1	1
0	1	0	0	1	0
1	0	1	0	0	1
1	1	1	1	0	0

Tabela 3.2: Tabela verdade para as operações AND, OR e NOT.

3.8 Circuitos Integrados

Todo circuito eletrônico é formados por uma certa configuração de componentes como transistores, diodos e resistores. O modo como esses componente são conectados determinam o comportamento do circuito. A ideia do circuito integrado (CI) é fabricar num processo único, sobre uma pequena pastilha de silício esses componentes já interligados para exercer uma função específica como um amplificador, um regulador de tensão, um oscilador, etc.

Assim, os circuitos integrados são diferentes uns dos outros no sentido de que cada um deles é feito para exercer uma determinada função. Essa função é dada pelo seu número ou identificação.

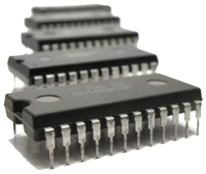


Figura 3.12: Circuito Integrado.

4. Plataformas de Desenvolvimento

Existem diversas plataformas de desenvolvimento que podem ser usados em um seguidor de linha. Entretanto, vale ressaltar que é possível utilizar também circuito analógicos para tal função.

Entre as plataformas de desenvolvimento 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: MSP430

Figura 4.2: Arduino UNO

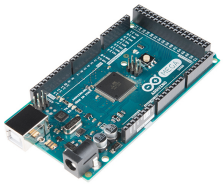


Figura 4.3: Arduino Mega 2560

Figura 4.4: Raspberry Pi

4.1 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 disponíveis 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 [1] 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 as principais funções no desenvolvimento do código em Arduino.

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. As principais funções o Arduino serão descritas a seguir. São elas as principais utilizadas na execução das rotinas mais básicas e estão presentes em todas as aplicações envolvendo o Arduino. A sintaxe do código é a mesma da linguagem C e todas bibliotecas de C podem ser utilizadas.

4.1.1 Estrutura de código

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() - Função que contém o código que será executada uma única vez. Geralmente utilizada na configuração das portas das GPIO (*general purpose input/output*) e iniciação das variáveis globais e descrição de periféricos como sensores, *displays* e etc;
- loop() - Função continuamente até que todo o processo seja interrompido.Sua execução ocorre logo após a função setup().

4.1.2 Funções Digitais

pinMode()

Configura um pino específico do Arduino para se comportar como uma entrada ou saída. Por padrão, os pinos são configurados como entradas (*input*), portando, não é necessário defini-los como entradas explicitamente.

O modo INPUT também é chamado de estado de alta impedância. Em outras palavras, a demanda de corrente por este estado é bastante reduzida, equivalente a um resistor de $100M\Omega$ conectado em série com o pino. Esse comportamento é útil pois reduz o efeito de carga e torna o pino útil para aplicações onde é necessário ler pequenos sinais, como sensores capacitivos e fotodiodos.

Muitas vezes é útil definir o pino em um estado conhecido se nenhuma entrada está presente. Isso pode ser feito adicionando um resistor de *pullup* (entre o pino e +5V), ou um resistor de *pulldown* (entre o pino e terra). Geralmente, o valor desse resistor é $10k\Omega$.

O modo OUTPUT também é chamado de estado de baixa impedância. Isso significa que ele pode fornecer uma quantidade significativa de corrente. Os pinos do ATMEGA podem fornecer uma corrente de até $40mA$, que é suficiente para alimentar LEDs (não se esqueça do resistor em série)

e muitos sensores. Entretanto, 40mA não é suficiente para operar a maioria dos relés, solenoides e motores, por exemplo.

O curto circuito nos pinos do Arduino, ou seja, a demanda por corrente elevada, pode danificar os componente internos da placa o que pode causar a "morte"do pino. Por este motivo, é uma boa prática conetar entre o pino de saída e o circuito ou dispositivos externos um resistor limitador de corrente. Caso não saiba calcular o valor deste resistor, utilize um com certa de 470Ω ou até mesmo 1$k\Omega$.

A sintaxe do código é exibida a seguir.

```
pinMode(pin, mode);
```

Os parâmetros são:

- pin: O número do pino desejado.
- mode: INPUT ou OUTPUT.

Há também o modo INPUT_PULLUP, que é utilizado para habilitar o resistor *pullup* do microcontrolador, não será detalhado neste texto.

digitalWrite()

Define o estado binário do pino digital. Se o pino está configurado como OUTPUT, sua tensão será 5V (ou 3,3V em algumas placas) se o estado é **HIGH**, ou 0V (terra) se o estado é **LOW**.

Se o pino está configurado como INPUT, a função digitalWrite() ir á habilitar (HIGH) ou desabilitar (LOW) o resistor de *pullup* do pino. Entretanto, é recomendável utilizar o modo INPUT_PULLUP da função pinMode() para habilitar o resistor de *pullup*.

A sintaxe do código é exibida a seguir.

```
digitalWrite(pin, value);
```

Os parâmetros são:

- pin: O número do pino desejado.
- value: HIGH ou LOW.

O código a seguir faz com que o LED acenda, espere 1s por meio da função delay(), e então apague e espere novamente 1s. Como essa rotina este inserida na função loop(), o processo é repetido fazendo com que o LED pisque a cada 1s.

```
int ledPin = 13;                  // LED conectado ao pino digital 13

void setup()
{
  pinMode(ledPin, OUTPUT);       // define o pino digital como saida
}
```

```
void loop()
{
  digitalWrite(ledPin, HIGH);   // define o pino estado HIGH (ligado
     )
  delay(1000);                  // aguarda 1000ms (1s)
  digitalWrite(ledPin, LOW);    // define o pino estado LOW (
     desligado)
  delay(1000);                  // aguarda 1000ms (1s)
}
```

Código 4.1: Exemplo da utilização da função digitalWrite()

digitalRead()

Ler o valor de um pino digital específico. A função retorna HIGH ou LOW.

A sintaxe do código é exibida a seguir.

```
digitalRead(pin);
```

O parâmetro é:

- pin: O número do pino desejado.

O exemplo a seguir reliza a leitura de um botão conectado ao pino 7 e armazena em uma variável. Como o botão pode assumir apenas dois estados (pressionado ou não), é feita a leitura digital e esse valor é armazenada em uma variável, que pode assumir o valor HIGH, valor inteiro igual a 1, ou LOW, valor inteiro igual a 0.

O valor da variável é aplicado à saída digital 13, que é um pino conectado ao LED da placa. Dessa forma, a lógica do programa é acender o LED quando o botão está pressionado e desliga-lo quando o botão é solto.

```
int ledPin = 13; // LED conectado ao pino digital 13
int inPin = 7;   // pushButton conectado ao pino digital 7
int val = 0;     // variavel que armazena o valor

void setup()
{
  pinMode(ledPin, OUTPUT);     // define o pino digital 13 como
     OUTPUT
  pinMode(inPin, INPUT);      // define o pino digital 7 como INPUT
}

void loop()
{
  val = digitalRead(inPin);   // leitura do pino digital 7
  digitalWrite(ledPin, val);  // define o LED com o estado do
     pushButton.
}
```

Código 4.2: Exemplo da utilização da função digitalRead()

Exercício 4.1 Programe o Arduino para criar uma linha de 6 LEDs que devem ser acessos em série com intervalo de 500ms e então desligados em série, com o mesmo intervalo. Utilize estruturas de repetições. ■

Exercício 4.2 Como é o código para que LED opere na de forma inversa, ou seja, se mantenha ligado enquanto o botão esteja pressionado e desligue caso contrário? ■

4.1.3 Funções Analógicas

analogRead()

Realiza a leitura de um pino analógico específico. A resolução do conversor analógico/digital do Arduino é 10 bits. Isso significa que o valor do sinal analógico lido entre 0 e 5V, que é o intervalo de leitura padrão, será convertido em valores entre 0 e $2^{10} - 1$. Portanto, a função retorna um valor entre 0 e 1023.

C A resolução da leitura é $\frac{5V}{2^10} = 4,9mV$ por unidade. O intervalo de leitura e a resolução pode ser alterados por meio do comando analogReference().

O tempo de leitura é de aproximadamente $100\mu s$, portanto a taxa de leitura máxima é cerca de 10000 leituras/segundo.

A sintaxe do código é exibida a seguir.

```
analogRead(pin);
```

O parâmetro é:

- pin: O número do pino desejado.

analogWrite()

Escreve um valor analógico (PWM) em um pino. Pode ser utilizado para a variação do brilho de um LED ou controle de velocidade de um motor, por exemplo. Após ser chamada a função analogWrite(), o pino irá gerar um sinal PWM constante com o *duty cycle* especificado até a próxima chamada da função analogWrite() para aquele mesmo pino.

A frequência do sinal PWM é 490Hz para a maioria dos pinos. Os pinos 5 e 6 possuem uma frequência de 980Hz. Os pinos que podem operar com a função PWM estão indicados na placa.É necessário definir o pinMode() como OUTPUT antes de utilizar a função analogWrite();

```
analogWrite(pin, value);
```

Os parâmetros são:

- pin: O número do pino desejado.
- value: O duty cycle: Entre 0 (constante em 0V) e 255 (constante em 5V).

O código a seguir realiza o valor de um potenciômetro conectado ao pino 3 cujos terminais também estão conectados a 0V e 5V. O valor lido é salvo em uma variável e esse valor pode apresentar qualquer valor entre 0 e 1023. O valor lido é dividido por 4 e escrito no pino que contém um LED. Dessa forma, é possível controlar o brilho do LED por meio do potenciômetro.

```
int ledPin = 13;     // define o pino 13 como OUTP
int analogPin = 3; // trimpot conectado ao pino 3
int val = 0;       // variavel para armazenar o valor lido

void setup()
{
  pinMode(ledPin, OUTPUT);   // define o pino como OUTPUT
}

void loop()
{
  val = analogRead(analogPin);   // leitura do pino
  analogWrite(ledPin, val / 4);  // analogRead varia de 0 to 1023,
     analogWrite varia de 0 to 255
}
```

Código 4.3: Utilização de funções analógicas

Exercício 4.3 Por que o brilho do LED é alterado quando varia-se o *duty cycle* do sinal PWM? ■

Exercício 4.4 Pesquise sobre o controle de outros periféricos, como os motores, por meio de PWM. ■

II Projeto

5. Chassi

O chassi é uma das principais partes das quais um seguidor de linha é constituído. É nele que serão instalados todos os componentes, e seu design deve levar em conta a dimensão, peso e posição de todos esses elementos, para garantir que o robô tenha uma boa estabilidade e consiga desenvolver todos os comandos programados para ele.

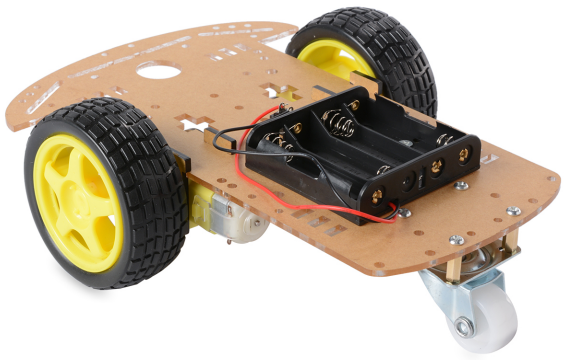


Figura 5.1: Exemplo de um chassi com elementos básicos já acoplados.

Em relação ao posicionamento dos elementos no chassi, é interessante que a alimentação esteja

próxima do centro de carga do robô para evitar a utilização de muitos fios condutores. Entretanto, quase sempre, as baterias serão os elementos de maior massa sobre a estrutura, tendo que ser observado a possibilidade delas deslocarem muito o centro de massa do robô e torná-lo instável.

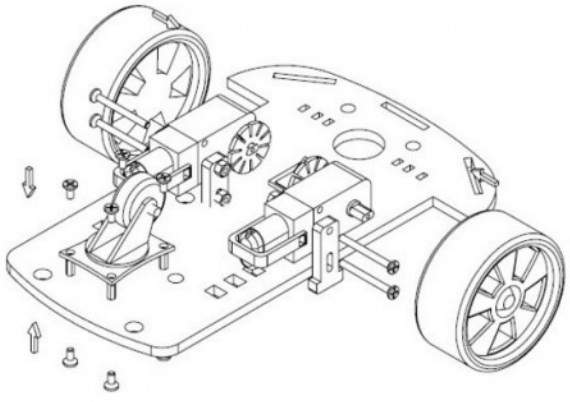


Figura 5.2: Posicionamento dos elementos básicos em um chassi.

6. Alimentação

Um robô seguidor de linha, assim como outros sistemas embarcados, necessita de uma fonte de alimentação própria, que seja capaz de fornecer energia suficiente para a realização de todas as atividades. No caso deste tipo de robô, é válido restringir tais fontes a baterias e pilhas.

Figura 6.1: Pilha

Figura 6.2: Bateria de Lítio

6.1 Escolha da Fonte de Alimentação

Para garantir que o seguidor será capaz de completar todo o seu trajeto de maneira eficiente, é necessário levar em conta alguns fatores no momento de escolha da fonte de alimentação.

As baterias devem ser de preferência recarregáveis, e capazes de fornecer tensão e corrente suficiente para todos os motores, controladores e sistemas periféricos, para que os mesmos funcionem com uma autonomia satisfatória. Pilhas, Figura 6.1, funcionam bem neste caso, pois é fácil encontrar modelos recarregáveis à um preço não muito elevado.

Entretanto, a corrente fornecida pelas pilhas podem não ser satisfatória em muitos casos. Nesse caso é interessante utilizar uma bateria de Lítio Polímero, a Figura 6.2 ilustra um exemplo. Essas

baterias, se bem dimensionadas, podem oferecer um desempenho muito elevado ao robô, além de serem recarregáveis.

Em todo caso, é interessante saber dimensionar qual será a potência necessária para que o robô funcione corretamente, para então definir qual tipo de bateria utilizar.

6.2 Regulador de Tensão

Geralmente a faixa de valores de tensão das baterias disponíveis no mercado, é superiora aos valores de tensão de operação dos controladores. Como exemplo, suponha que você tenha disponível uma bateria de 12V, mas o seu microcontrolador pode ser alimentado com no máximo 5V. Para solucionar esse problema, será necessário utilizar um regulador de tensão.

Reguladores de tensão são dispositivos que possuem um princípio de funcionamento simples: na entrada recebem uma tensão contínua, superiora ao valor de tensão especificada para um outro componente, e na saída fornecem este valor de tensão especificada.

O Arduino já possui reguladores de tensão embutidos na própria placa, o que torna possível uma alimentação de até 20V, no caso do Arduino UNO e MEGA. Entretanto, é recomendável que a placa seja alimentada com tensão entre 7V e 12V.

A maior parte dos controladores funcionam a 5V ou 3,3V, tornando recomendável a utilização de reguladores que possuam essa especificação. Um modelo de regulador de tensão de 5V é o LM7805, cuja tensão máxima a ser aplicada na entrada é de 35V e a máxima corrente obtida na saída é da ordem de 1A. Já para uma saída regulada de 3,3V, temos o regulador L78L33, por exemplo.

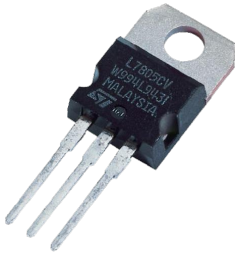
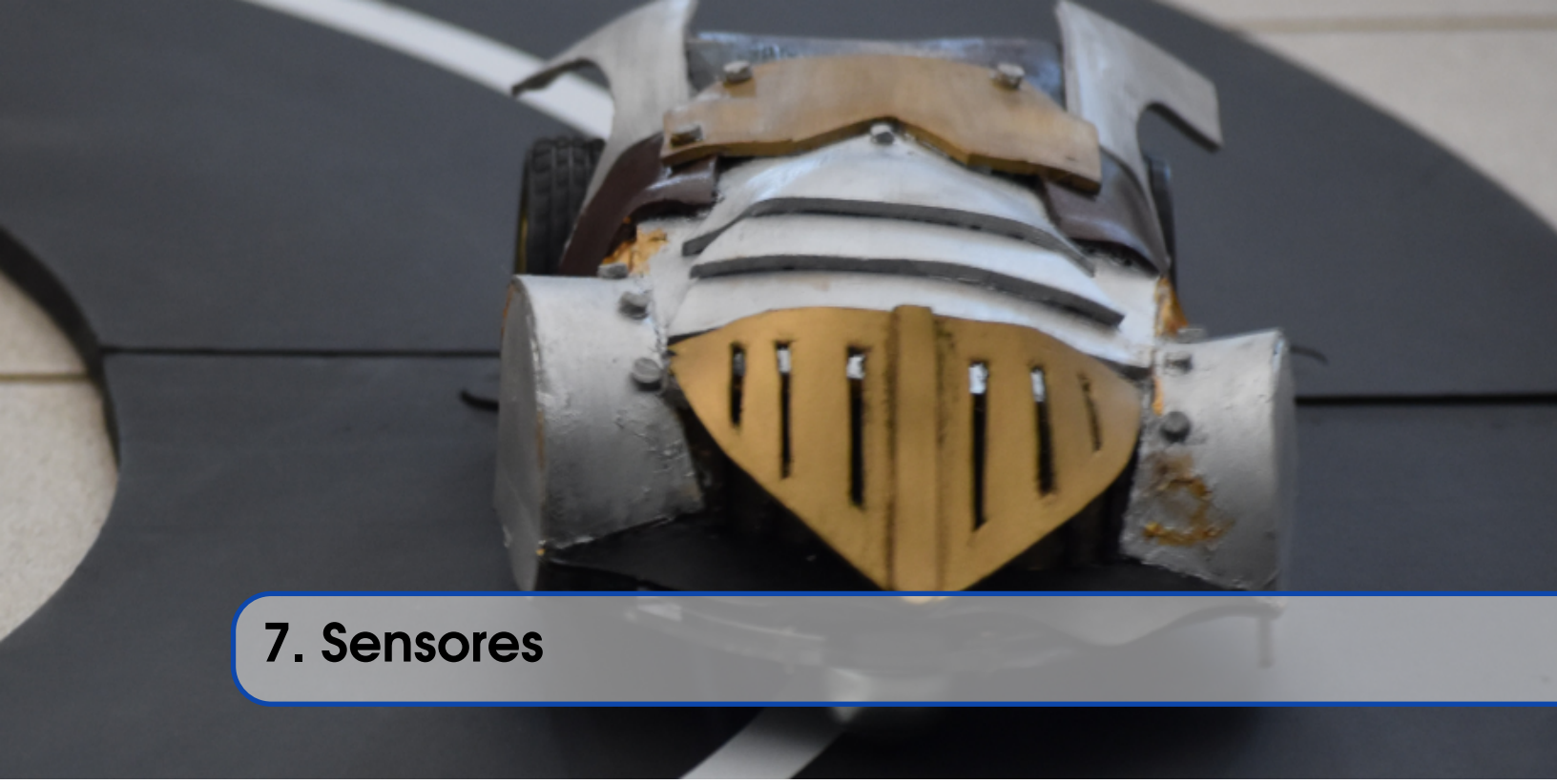


Figura 6.3: Regulador de tensão LM7805.



7. Sensores

7.1 Sensor de Luz Infravermelha

O sensor de luz infravermelha tem uma grande importância no projeto de robôs seguidores de linha pois esse sensor é o mais comum para a implementação de um sistema de detecção 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.

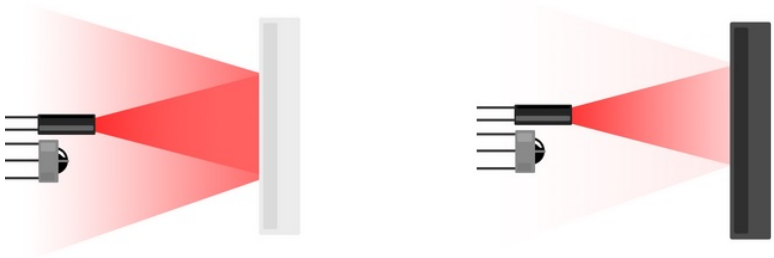


Figura 7.1: Reflexão da luz de um sensor infra vermelho para uma superfície branca e uma superfície preta.

O circuito mais comum na detecção de linhas é ilustrado na Figura 7.2. O circuito se trata de um LED emissor infra vermelho e um foto diodo, que ira receber a luz fornecida pelo LED que foi refletida pela superfície.

O valor do sinal pode ser lido de forma digital ou analógica. Entretanto, é recomendável a leitura analógica, que permite a calibração simples da leitura do sinal do fotodiodo. Essa característica é

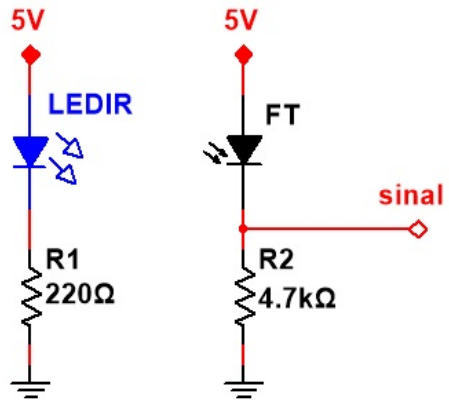


Figura 7.2: Circuito de operação do conjunto LED infra vermelho e fototransistor.

desejada pois em muitas situações a operação do robô é afetada pela alteração de luminosidade do local.

```
int objeto = 0;
#define SENSOR A0
void setup()
{
  pinMode(SENSOR, INPUT); //Pino que recebe o sinal
  Serial.begin(9600);
}

void loop()
{
  objeto = analogRead(SENSOR);
  if (objeto == 0)
  {
    Serial.println("Objeto: Detectado!");
  }
  else
  {
    Serial.println("Objeto: Ausente!");
  }
  delay(500);
}
```

Código 7.1: Utilização do sensor infravermelho.

Saída do Código **??**

Objeto: Detectado! Objeto: Ausente! Objeto: Detectado!

7.2 Sensor Ultrassônico

O sensor Ultrassônico (HC-SR04) é utilizado para identificar obstáculos no caminho do robô. Essencialmente, ele é um sensor de distância.

Figura 7.3: Sensor ultrassônico HC-SR04.

Funciona da seguinte forma: O sensor emite um sinal sonoro e aguarda o eco desse pulso. Um timer é responsável por contar o tempo entre a emissão do sinal e o recebimento do eco. Sabendo esse tempo, como a velocidade do som é conhecida, aproximadamente $v_{som} = 340m/s$, é possível descobrir a distância entre o objeto por meio da equação 7.1.

$$\text{distância} = \frac{\text{velocidade} \cdot \text{tempo}}{2} \tag{7.1}$$

A divisão por dois na equação ocorre pois a distância percorrida é o dobro da distância desejada, uma vez que a onda viaja até o obstáculo percorrendo uma certa distância e é então refletida, percorrendo novamente a mesma distância

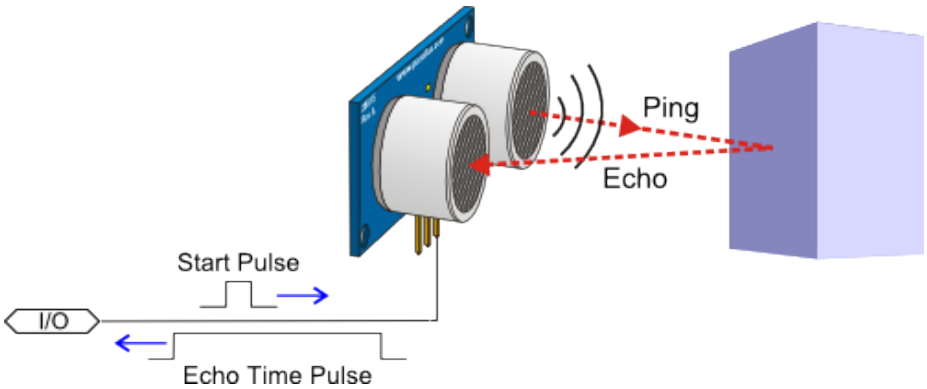


Figura 7.4: 000.

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

Entretanto, a utilização do sensor se torna mais fácil se utilizada a biblioteca própria para o sensor. O código abaixo apresenta um exemplo de utilização do sensor ultrassônico HC-SR04.

```
#include <Ultrasonic.h> //biblioteca do sensor ultrassonico

//Define os pinos para o trigger e echo
#define TRIGGER 4 //pino TRIGGER na porta digital 4
#define ECHO 5 //pino ECHO na porta digital 5

Ultrasonic ultrasonic(TRIGGER, ECHO); //Inicializa o sensor HC-SR04

void setup()
{
  Serial.begin(9600);
  Serial.println("Sensor de Distancia.");
}

void loop()
{
  float distancia;
  long microsec = ultrasonic.timing();
  distancia = ultrasonic.convert(microsec, Ultrasonic::CM);

  //Exibe informacoes no serial monitor
  Serial.print("Distancia: ");
  Serial.print(distancia);
  Serial.println("cm");
  delay(1000);
}
```

Código 7.2: Utilização do HC-SR04.

Saída do Código 7.2

```
Sensor de Distancia.
Distancia: 11cm
Distancia: 15cm
Distancia: 17cm
```

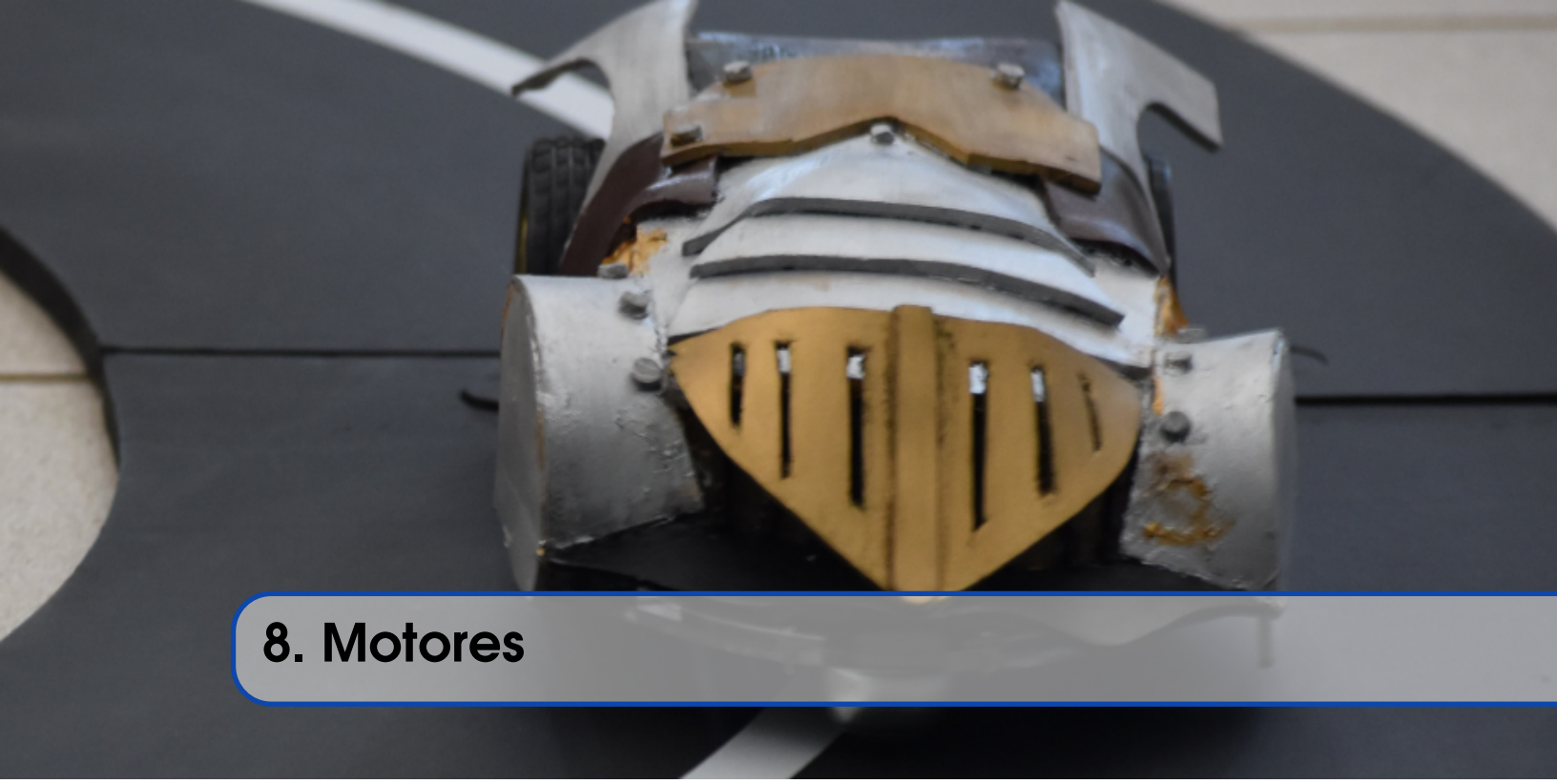
7.3 Encoder

Um encoder é um dispositivo eletromecânico que produz pulsos elétricos de acordo com a rotação d motor. É utilizado para medir a velocidade de rotação do motor. Essa informação pode ser útil

dependendo do método de controle utilizado. São quatro tipos de encoders, sendo que o absoluto e o incremental são os que mais nos interessam. Utilizando um encoder no eixo de cada motor, é possível obter com precisão a velocidade e a posição do robô, e tais informações associadas com a lógica de programação aplicada, será capaz de decidir as próximas ações do seguidor.

Utilizando o princípio de funcionamento deste tipo de sensor, que se baseia na emissão de um feixe de luz em direção a um disco opaco com várias aberturas transparentes, e medido a frequência de oscilação entre opaco e transparente, é possível substituir um encoder comercializado a preços altos, por um constituído pelo disco na Figura 7.5, e um sensor de luminosidade.

Figura 7.5: Disco encoder.



8. Motores

8.1 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.

O esquemático do circuito utilizando um transistor NPN está na figura **??**

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.

8.2 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.a

Figura 8.1: 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 8.2, 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.

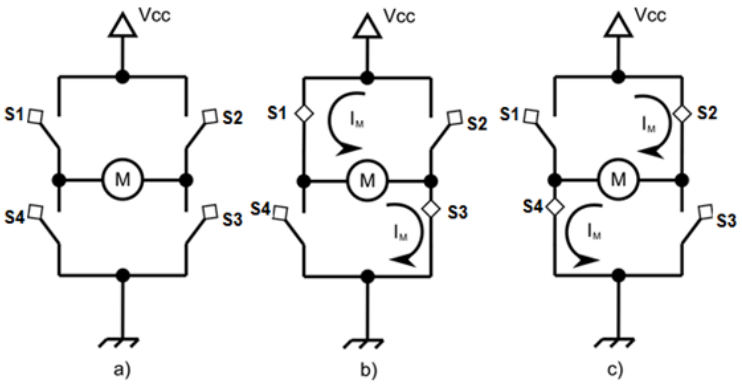


Figura 8.2: Esquema de funcionamento de uma Ponte H

8.2.1 Funcionamento do L298N

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

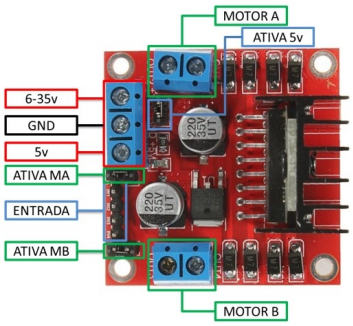


Figura 8.3: Conexões do Módulo L298N

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.

8.3 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()*.

8.4 Movimentação

A Figura 8.4 ilustra três casos de movimentação do robôs.

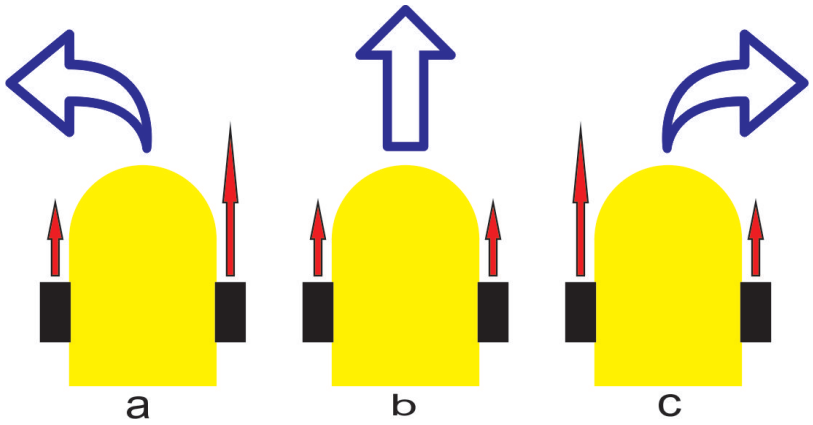


Figura 8.4: Movimentação.

A seta vermelha infica a velocidade do robô em cada roda. A movimentação do no caso b é o mais comum, a velocidade das duas é igual e o robô se movimenta em linha reta. Já nos casos a e c ocorre que a velocidade de uma das rodas é maior que a outra. No caso a, por exemplo, a velocidade da roda direita é maior. Nesse caso, o robô irá fazer uma curva para esquerda. Para o caso c o comportamento é análogo.

Vale a pena notar que quanto maior a velocidade de uma roda em relação a outra, menor é o raio da curva realizada. Em outras palavras, se é desejado realizar uma curva fechada, a velocidade de uma roda deve ser bem maior em relação a outra. Por exemplo, em uma curva cujo angulo é reto, uma boa estratégia é manter a velocidade de uma roda é nula e a outra numa velocidade definida, que depende da rapidez com a qual se deseja realizar o movimento.

Para que o robô gire no próprio eixo, realizando um retorno, por exemplo, a velocidade das duas roda deve ser a mesma, mas o sentido de uma é invertido em relação ao outro.

8.5 Leitura dos Sensores e Movimentação

Para se movimentar seguindo a linha, é necessário saber o básico de movimentação do robô, ou seja, a realização de curvas. Imagine um robô seguidor de linha com dois sensores infravermelhos. Esse robô é colocado num trajeto como indicado na Figura 8.5. Inicialmente, ambos os sensores não detectam a linha, nesse caso, o robô foi programado para andar em linha reta.

Após percorrer uma certa distância, o sensor do lado direito passa a reconhecer a linha. Isso pode inficar que há uma curva á direita a frente e então o robô passa a acelerar o motor direito para relizar o movimento e se alinhar à curva.

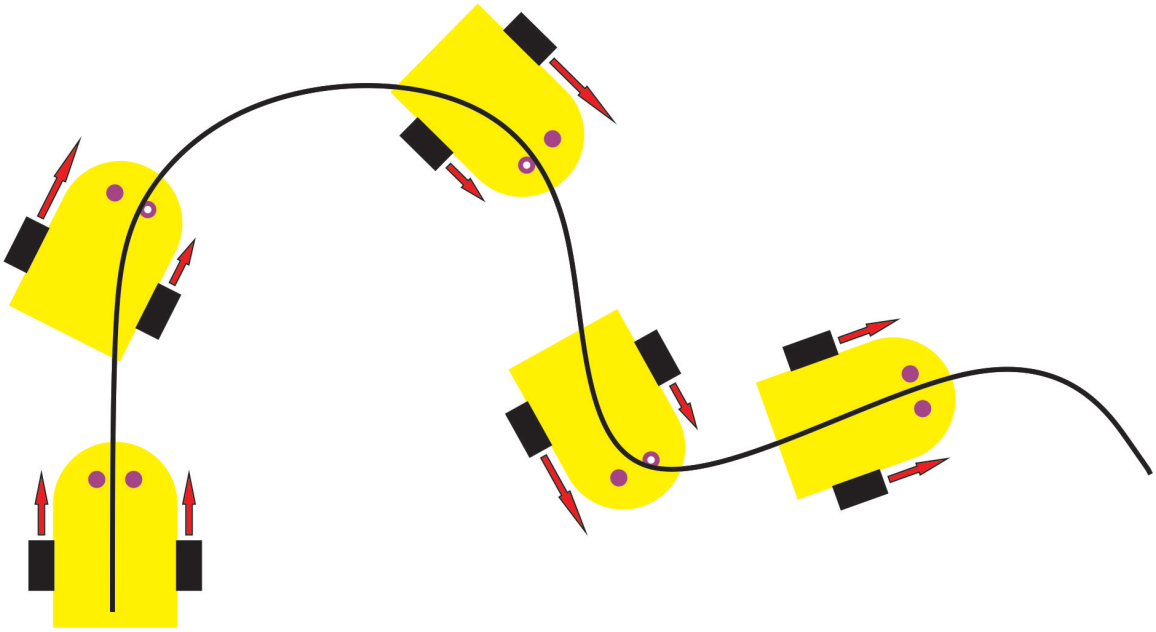


Figura 8.5: Detecção dos sensores e movimentação.

Esse é o modo como o robô se movimenta em conjunto com a detecção dos sensores. O robô deve se manter alinhado à linha, para isso, sempre que o sensor de um certo lado por detectado, o robô pode entender que uma curva para esse mesmo lado se encontra a frente.

Entretanto, é de grande importância conhecer a pista onde o robô se encontra. Uma vez que a configuração da pista pode indicar informações através das linhas, o que pode ser chamado também de marcações.

Como exemplo, imagine que um pequeno quadrado próximo à linha que deve ser seguida indique que haverá uma curva de 90^{o} no mesmo lado onde a linha foi detectada, como na Figura 8.6.

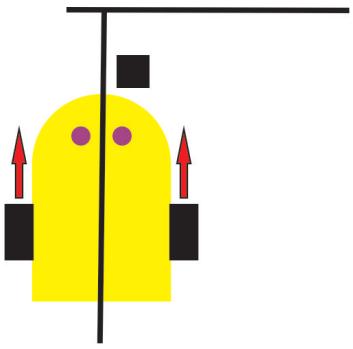


Figura 8.6: Exemplo de marcação na pista.

Essa informação pode ser valiosa no momento de realizar a manobra indicada, pois o programador pode reconhecer essa informação e realizar o controle devido à movimentação do robô para ultrapassar esse obstáculo. Assim, cabe ao programador entender o trajeto que será percorrido e preparar um algoritmo compatível.

C É possível e bastante indicado a construção de um robô seguidor de linha com um arranjo de mais de dois sensores. Não há uma quantidade ideal, mas as dimensões do robô e da linha a ser seguida influência de forma muito significativa na quantidade e espaçamento dos sensores. Além disso, uma quantidade maior de sensores significa um maior número de referências para o robô se localizar na pista.

9. Comunicação Serial

Será detalhada aqui a comunicação serial utilizada no Arduino, que utiliza os pinos TX/RX que possuem a lógica de nível TTL, que opera entre 0 e 5V.

Não conecte periféricos que se comuniquem de forma serial com sinais maiores que 5V, como uma porta serial RS232 que opera com $\pm 12V$, pois isso pode danificar o Arduino.

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. O Arduino se comunica por meio dos pinos 0 (RX) e 1 (TX) ou com um computador via USB.

Não é permitido a utilização dos pinos 0 e 1 como entradas e saídas digitais quando a comunicação serial é utilizada.

Também é possível se comunicar com a IDE do Arduino via monitor serial. Para isso é necessário definir o mesmo *baud rate* e iniciar a comunicação serial.

9.1 Iniciando uma conexão Serial

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

SERIAL.BEGIN(BAUDRATE)

O *baud rate* representa a velocidade de comunicação entre os dispositivos. A IDE do Arduino já tem um monitor serial embutido que pode ser usado para receber dados. Para utilizá-lo é necessário escolher a mesma *baud rate* definida para a comunicação.

9.2 Monitor Serial

No projeto em Arduino, o monitor serial pode ser utilizado para a depuração de código. Uma função bastante utilizada para isso é a print. A função print escreve dados na porta serial e pode assumir diversas formas. Um parâmetro opcional da função print é a base de dados utilizada. É permitido a base BIN (binária, ou base 2), OCT (octal, ou base 8), DEC (decimal, ou base 10), HEX (hexadecimal, ou base 16). Para dados do tipo float, este parâmetro define a quantidade de casas decimais que será utilizada.

A sintaxe do código é exibida a seguir.

SERIAL.PRINT(VAL, FORMAT)

Os parâmetros são:

- val: Valor a ser escrito.
- format: (**OPCIONAL**) Define a base para dados inteiros e número de casas decimais para *floats*.

O código a seguir escreve dados em diversos formatos.

```
int x = 0;     // variavel

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

void loop() {
  for(x=0; x< 64; x++){     // percorre os 65 termos da tabela ASCII.
    Serial.print(x);        // ASCII decimal
    Serial.print("\n");     // quebra de linha

    Serial.print(x, BIN);   // ASCII binario
    Serial.print("\n");     // quebra de linha

    Serial.print(x, OCT);   // ASCII octal
    Serial.print("\n");     // quebra de linha

    Serial.print(x, DEC);   // ASCII decimal
    Serial.print("\n");     // quebra de linha
```

```
    Serial.print(x, HEX);  // ASCII hexadecimal
    Serial.print("\n");    // quebra de linha

    delay(200);            // delay 200ms
  }
  Serial.println("");      // Escreve o texto e logo apos quebra a
     linha
}
```

Código 9.1: Exemplo da utilização da função Serial.print() digitalWrite()