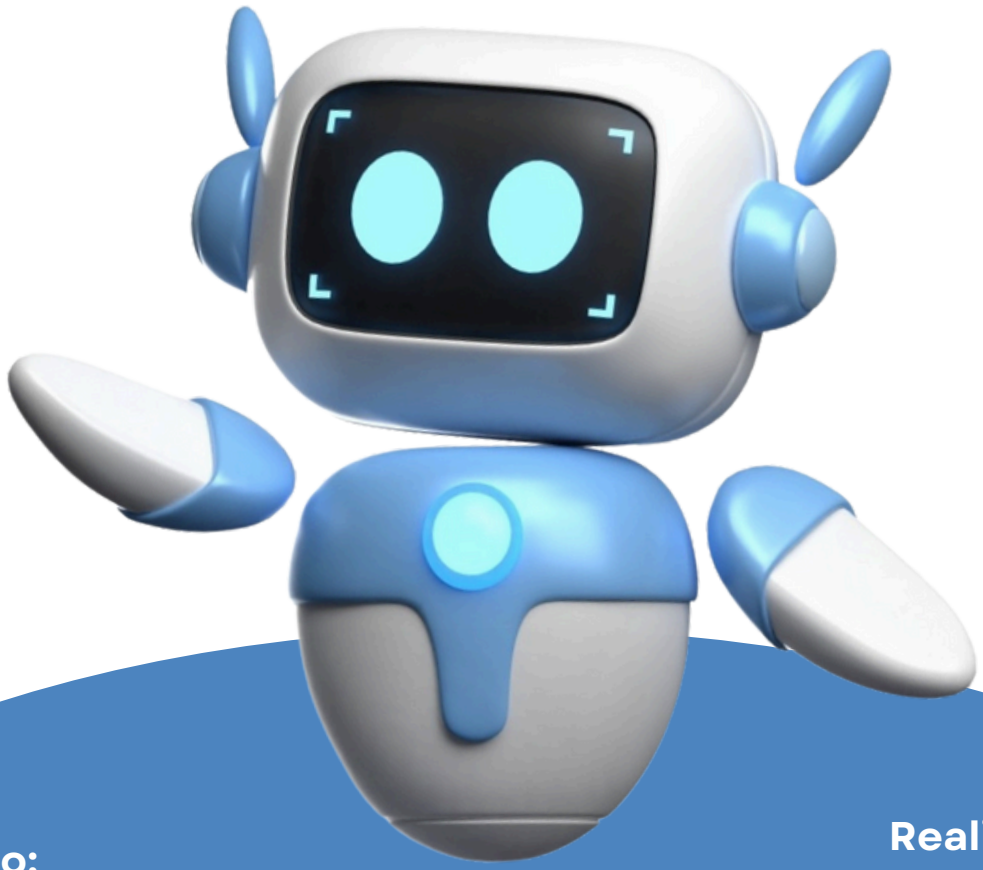
GUIA PARA PROGRAMAÇÃO

DO ROBÔ

Este manual foi desenvolvido para ajudar você a configurar e controlar seu robô, com foco em sensores, motores e lógica de combate, usando Arduino.



Apoio:

INSTITUTO FEDERAL
Farroupilha
Campus Santo Ângelo

GOVERNO DO ESTADO
RIO GRANDE DO SUL
SECRETARIA DE INOVAÇÃO, CIÊNCIA E TECNOLOGIA

COREDE MISSÕES

CNPq

Realização:

tecnoURI missões

URI
SANTO ÂNGELO

GUIA PARA PROGRAMAÇÃO DO ROBÔ

Produção
Flávia Figueiró
Lorenzo Callegaro

Apoio
Secretária de Inovação, Ciência e Tecnologia do RS - SCIT
COREDE Missões
Conselho Nacional de Desenvolvimento Científico e Tecnológico - CNPq
Instituto Federal Farroupilha - IFFar Santo Ângelo

Realização
URI Santo Ângelo
TECNOURI Missões

2024

SOBRE O PROJETO

A **1ª Olimpíada Missioneira de Robótica (OMR)** é uma iniciativa educacional que promove uma competição científica regional, utilizando a robótica como ferramenta central de aprendizado. Destinada a estudantes de 10 a 19 anos, matriculados em escolas públicas ou privadas de ensino fundamental, médio ou técnico, a Olimpíada busca fortalecer os laços entre a universidade e as escolas, incentivando o pensamento computacional entre crianças e adolescentes.

O principal objetivo é **aproximar os jovens do universo da tecnologia e das carreiras científicas**, identificando talentos e democratizando o acesso ao conhecimento científico e tecnológico. Através de atividades educativas, a 1ª Olimpíada Missioneira de Robótica cria ambientes de aprendizado colaborativo, onde alunos, professores e a comunidade escolar participam ativamente, ampliando a troca de experiências e saberes.

As estratégias adotadas exploram o potencial do pensamento computacional para estimular a curiosidade, a experimentação e a resolução de problemas, promovendo uma compreensão inclusiva da ciência e da tecnologia. Apoiada pelo Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq) e pelo Ministério da Ciência, Tecnologia e Inovação (MCTI), a Olimpíada contribui diretamente para a popularização da ciência e a melhoria do ensino, incentivando jovens a considerarem carreiras nas áreas científicas e tecnológicas.

Além disso, a iniciativa promove a inclusão social, garantindo a participação de estudantes de diversas origens. Por meio da Olimpíada, a URI desempenha um papel fundamental no desenvolvimento humano, estimulando a cooperação e o surgimento de novos talentos, reafirmando seu compromisso com a educação, a ciência e a tecnologia.

Cristina Paludo Santos
Coordenadora Geral da OMR

INTRODUÇÃO

Com o objetivo de auxilia-los na jornada de entendimento sobre os materiais recebidos e principalmente na construção dos robô, criamos este Guia. Nosso propósito é garantir que cada participante tenha acesso a informações claras e detalhadas, facilitando o entendimento dos componentes recebidos nos kits e proporcionando uma experiência de aprendizado rica e interativa.

Neste material, vocês encontrarão uma explicação completa sobre cada parte que compõe os kits recebidos, desde os elementos eletrônicos até as peças mecânicas, ajudando a familiarização com suas funções e características. Além disso, oferecemos um guia passo a passo que orienta o processo de montagem do robô, detalhando cada etapa de forma prática e acessível, para que a construção seja tranquila e satisfatória.

Por fim, dedicamos uma seção especial à programação do robô, onde explicamos, de maneira simples e estruturada, como configurar e programar o sistema para que ele funcione conforme o esperado. Nossa intenção é que este Guia sirva como uma referência na montagem do robô ajudando vocês a superar desafios e a desenvolver, com confiança, suas habilidades em robótica.

AULA 1

Pensamento Computacional

De acordo com **Brackmann (2017)**, o Pensamento Computacional é uma capacidade humana única que combina criatividade, criticidade e estratégia, aplicando os princípios da Computação em diversas áreas do conhecimento. Seu objetivo é **identificar e resolver problemas**, seja individualmente ou em equipe, através de etapas claras que podem ser executadas tanto por uma pessoa quanto por uma máquina.

Em outras palavras, pensar computacionalmente vai além da simples programação de código: trata-se de **abordar problemas de forma sistemática**, observando diferentes níveis de abstração, desde o hardware até algoritmos de alto nível. É entender a interação entre essas camadas, reconhecer padrões e desenvolver soluções eficientes. Envolve, ainda, a habilidade de decompor problemas complexos em partes menores e raciocinar sobre elas, permitindo projetar sistemas robustos e otimizados.

A importância do Pensamento Computacional reside no fato de que ele **transcende a Ciência da Computação** e se aplica a várias áreas do conhecimento. Desenvolve habilidades essenciais como resolução de problemas, criatividade e pensamento crítico, capacitando as pessoas a lidar com desafios complexos de maneira sistemática e eficiente.

Em um mundo cada vez mais digital e interconectado, essa habilidade se torna fundamental, pois prepara os indivíduos para não apenas utilizar a tecnologia, mas também entender e influenciar os sistemas e processos ao seu redor.

Em 2006, a cientista da computação **Jeannette Wing** introduziu o termo Pensamento Computacional como uma forma de raciocínio que ajuda a decompor problemas complexos em partes menores para resolvê-los com eficiência. Wing identificou **cinco pilares** principais do Pensamento Computacional:

- **Decomposição:** Dividir um problema em partes menores e mais gerenciáveis. Ao planejar uma festa, por exemplo, você separa as tarefas, como lista de convidados, local, comida, etc.
- **Reconhecimento de Padrões:** Identificar semelhanças em problemas diferentes, aproveitando soluções passadas para desafios futuros semelhantes.
- **Abstração:** Focar nos elementos mais relevantes e ignorar os detalhes desnecessários. Como em um mapa, onde apenas o caminho importa, não cada elemento da paisagem.
- **Algoritmos:** Criar uma sequência lógica de passos que levam à solução, como uma receita de bolo, onde seguir o passo a passo resulta no resultado esperado.
- **Avaliação:** Testar, ajustar e otimizar as soluções para garantir que sejam eficientes e eficazes.

AULA 1

Pensamento Computacional

Na **competição** de robôs de sumô, onde o objetivo é **empurrar o adversário para fora da arena**, esses cinco pilares são fundamentais. O Pensamento Computacional é utilizado em todas as fases, desde a montagem do robô até a programação final, garantindo que ele seja **autônomo, preciso e estratégico**. Veja como cada pilar é aplicado nesse contexto:

- **Decomposição:** Essa habilidade é essencial para montar a estrutura do robô, aplicar sensores e motores, e planejar sua programação, quebrando o projeto em etapas mais fáceis de gerenciar.
- **Reconhecimento de Padrões:** Analisar e reconhecer padrões nos movimentos e estratégias dos adversários ajuda a determinar a percepção e os movimentos ideais do robô.
- **Abstração:** Focar nas funções essenciais do robô, como detectar o oponente e empurrá-lo para fora da arena, permite simplificar a programação e a construção, sem distrações com detalhes que não impactam o desempenho.
- **Algoritmos:** A programação do robô de sumô envolve algoritmos que guiam seus movimentos e ações. As sequências de ação e repetição são planejadas com base em condições lógicas.
- **Avaliação:** Após a montagem e a programação, é fundamental avaliar a resposta dos sensores, a eficiência em combate e o desempenho contra diferentes adversários. Com essa análise, são feitos ajustes para melhorar a performance do robô, otimizando tanto o hardware quanto o software.

Cada um desses pilares **organiza o processo de construção e otimização do robô**, assegurando que todas as etapas estejam bem planejadas e executadas **para alcançar o sucesso na competição.**

AULA 2

Introdução ao Arduino

Introdução ao IDE do Arduino
O Arduino IDE (Integrated Development Environment) é a plataforma de desenvolvimento usada para escrever, compilar e carregar códigos nas placas Arduino. Criado para ser acessível e intuitivo, o IDE é ideal para iniciantes e avançados programarem dispositivos Arduino de maneira prática e eficiente.

Funcionalidades do Arduino IDE

- **Editor de Código:** onde você escreve as instruções que a placa Arduino executará.
- **Compilador:** converte o código escrito na linguagem do Arduino (baseada em C/C++) em uma forma compreensível pelo microcontrolador da placa.
- **Monitor Serial:** permite visualizar dados enviados da placa para o computador, sendo uma ferramenta essencial para depuração e monitoramento.

Com o Arduino IDE, é possível desenvolver projetos que variam de controle básico a criações complexas, ampliando as possibilidades no campo da programação e da eletrônica.

Instalando o Arduino IDE
Passo 1: Acesse o site oficial para download do Arduino IDE
Link: https://www.arduino.cc/en/software.
Passo 2: Escolha a versão compatível com o sistema operacional do seu computador

- Windows: selecione "Windows ZIP file" ou a opção recomendada.
- MacOS: clique na versão para Mac.
- Linux: escolha a distribuição compatível.

Passo 3: Instale o Arduino IDE
Após o download, inicie a instalação clicando no arquivo baixado e siga as instruções exibidas na tela.

Utilizando o Arduino IDE em Chromebooks
Para Chromebooks, é recomendável utilizar o Arduino Web Editor, acessível online e sem a necessidade de instalação completa do IDE.

1. Acesse o <https://docs.arduino.cc/learn/starting-guide/the-arduino-web-editor/?
2. Vá ao Arduino Create e siga as instruções para configurar o editor no Chromebook.

Dica: No Chromebook, pode ser necessário conceder permissões adicionais e ativar o modo desenvolvedor para concluir a instalação.
Essa introdução cobre os primeiros passos para usar o Arduino IDE e permite que você comece a explorar o mundo da programação e da eletrônica com Arduino.

AULA 3

Introdução à Algoritmos

Instruções no Arduino
As instruções são os **comandos** que dizem ao microcontrolador o que ele deve fazer, com cada uma representando um passo a ser seguido.
Por exemplo, funções como **PinMode ()**, **digitalRead ()**, **e digitalWrite ()** permitem configurar, ler e escrever valores nos pinos do microcontrolador. A função **delay ()** é outro exemplo, utilizada para controlar pausas no programa. Essas instruções possibilitam definir o comportamento de cada pino, especificando se ele irá **enviar ou receber informações**.

Variáveis no Arduino
As variáveis são áreas de memória onde **armazenamos dados temporários**, como números ou estados que precisamos manter enquanto o programa está em execução. Ao criar uma variável, escolhemos um tipo específico, como **int** para inteiros, **float** para decimais, ou **char** para caracteres. Isso ajuda o programa a entender qual tipo de dado ele está manipulando e a alocar o espaço de memória adequado.

Por que as Instruções em C/C++ terminam com Ponto e Vírgula (;)
Em C/C++, as instruções geralmente terminam com um ponto e vírgula (;). Esse símbolo indica ao compilador que aquela **linha de código está completa**, permitindo que ele **passe para a próxima instrução**. Sem o ponto e vírgula, o compilador não sabe onde a instrução termina, causando erros no código.

Condições no Arduino
As condições permitem que o **microcontrolador tome decisões com base em testes lógicos**. Usamos estruturas como **if**, **else if**, e **else** para verificar se uma condição é verdadeira ou falsa e, com base nisso, escolher qual caminho seguir.

Exemplo:

```
if (sensor > 100) {
    digitalWrite (LED, HIGH);
    }
```

Aqui, se o valor do sensor for maior que 100, o LED é aceso. Essas condições controlam o fluxo do programa, respondendo de maneira adequada a diferentes situações.

AULA 3

Introdução à Algoritmos

Parênteses e Chaves no Código

- Parênteses (): Usados para **agrupar argumentos** de funções e **definir a ordem** das operações. Por exemplo, em **digitalWrite (LED, HIGH);** os parênteses delimitam os argumentos da função.
- Chaves {}: Utilizadas para **delimitar blocos de código**, como em funções ou loops, organizando o conteúdo que deve ser executado em conjunto.
  Exemplo:

```
{
  digitalWrite (LED, HIGH);
  delay (1000);
}
```

Os **parênteses** indicam os argumentos das funções, enquanto as **chaves** agrupam instruções que devem ser executadas juntas.

Por que Indentar o Código?

A indentação **organiza o código** e torna sua **estrutura mais clara**, facilitando a visualização dos blocos de código e a compreensão do fluxo do programa. Essa prática é **essencial**, especialmente em projetos mais complexos, com múltiplas funções, loops e condições, pois **torna o código mais fácil de entender e de manter**.

AULA 4

Primeiros Passos com a Programação no Arduino

Setup(): Função que **roda apenas uma vez quando o Arduino é ligado ou reiniciado**. É usada para **configurar pinos** como entrada ou saída e **iniciar componentes**. É usado para preparar o Arduino para o que ele vai fazer.

Exemplo cotidiano:
O setup() é como a sua rotina matinal. Quando você acorda, tem coisas que você faz apenas uma vez, no início do dia, para se preparar. Por exemplo:

- Levantar da cama.
- Escovar os dentes.
- Colocar a roupa.

Essas são coisas que você só faz uma vez quando acorda. Vamos dizer que eu fosse lá e apertasse um botão ou ligasse você na tomada, após isso você iria levantar da cama, escovar os dentes e trocar de roupa porque eu te programei para isso, mesma coisa que vocês vão fazer com o robô.

Loop(): Função que **roda repetidamente**. É onde o **código principal** do programa fica e onde as **ações contínuas ocorrem**. Coração do programa. Ele executa repetidamente o código que você programou, como verificar se um botão foi pressionado ou se um sensor detectou algo. Cada vez que o Arduino termina o que está no loop(), ele recomeça e continua verificando os sensores ou tomando ações baseadas nas entradas que recebe, de forma contínua.

Exemplo cotidiano:
Agora, o loop() é como o que você faz durante o resto do seu dia. Por exemplo:

- Você repete certas ações, como comer, beber água ou checar o celular, várias vezes ao longo do dia.
- A cada momento, você verifica se está com fome e, se estiver, vai comer.
- Verifica se está com sede e, se estiver, vai beber água

AULA 5

Explicando as Funções - MOTOR

Controle de Motores para Robô de Sumô com Arduino
Em um robô de sumô, o sistema de movimentação é fundamental para **garantir que o robô execute ações rápidas e precisas durante a competição**. Neste projeto, utilizamos motores controlados pelo Arduino para realizar movimentos básicos como avançar, recuar, girar e parar, além de ações específicas, como um "ataque" rápido contra o oponente. Através da configuração da ponte H e das portas lógicas, podemos ajustar a direção e a velocidade dos motores, proporcionando controle completo sobre o movimento do robô.

Configuração dos Motores no Arduino
Para controlar os motores do robô, configuramos no **setup()** os pinos que ligam a ponte H às portas lógicas do Arduino como **OUTPUT**, permitindo que **enviem** sinais para os motores.

```
void setup() {
    pinMode(5, OUTPUT);  // Ponte H B-IA (traseiro, esquerda)
    pinMode(6, OUTPUT);  // Ponte H B-IB (frontal, esquerda)
    pinMode(9, OUTPUT);  // Ponte H A-IA (traseiro, direita)
    pinMode(10, OUTPUT);  // Ponte H A-IB (frontal, direita)
}
```

Funções Básicas de Movimento

- Função frente (): Mover o robô para frente.

```
void frente () {
    analogWrite (5, 0);  // Motor B desligado
    analogWrite (6, 150);  // Motor B ligado com velocidade média
    analogWrite (9, 0);  // Motor A desligado
    analogWrite (10, 150);  // Motor A ligado com velocidade média
}
```

Ambos os motores ligados para frente em velocidade média (150) e desligados para trás;

- Função ré (): Mover o robô para trás.

```
void re () {
    analogWrite (5, 150); // Motor B ligado com velocidade média
    analogWrite (6, 0); // Motor B desligado
    analogWrite (9, 150); // Motor A ligado com velocidade média
    analogWrite (10, 0); // Motor A desligado
}
```

Ambos os motores ligados para trás em velocidade média (150) e desligados para frente

AULA 5

Explicando as Funções - MOTOR

- Função stop (): Parar o robô.

```
void stop () {
    analogWrite (5, 0);  // desliga todos os motores
    analogWrite (6, 0);
    analogWrite (9, 0);
    analogWrite (10, 0);
}
```

Todos os motores são desligados, interrompendo o movimento do robô.

- Função girar (): Fazer o robô girar em torno de si mesmo.

```
void girar () {
    analogWrite (5, 0); // Motor B desligado
    analogWrite (6, 150); // Motor B ligado com velocidade média
    analogWrite (9, 150); // Motor A ligado com velocidade média DIREITA
    analogWrite (10, 0); // Motor A desligado
}
```

O motor traseiro do lado esquerdo fica desligado e o frontal fica ligado em velocidade média, enquanto o motor traseiro do lado direito fica ligado em velocidade média e o frontal desligado, fazendo o robô girar em uma direção fixa.

- Função atacar (): Fazer o robô avançar rapidamente, simulando um "ataque".

```
void atacar () {
    analogWrite (5, 0); // Motor B desligado
    analogWrite (6, 200); // Motor B ligado com alta velocidade
    analogWrite (9, 0); // Motor A desligado
    analogWrite (10, 200); // Motor A ligado com alta velocidade
}
```

Os motores frontais de ambos os lados ficam ligados em alta velocidade enquanto os traseiros ficam desligados, fazendo o robô avançar rapidamente para frente, simulando um movimento de "ataque".

- Essas funções permitem controlar os movimentos básicos do robô para diferentes propósitos, como explorar, recuar, parar, girar ou “atacar”.

AULA 6

Explicando as Funções - DE LINHA

Primeiramente, o robô **possui sensores de linha conectados aos pinos 11, 12 e 13**. Estes sensores ajudam o robô a **identificar e reagir à presença de uma linha ou borda**, o que é especialmente útil para competições onde o robô precisa se mover dentro de uma área delimitada.

Olhe como se o robô estivesse de costas para você:

```
setup(){
    pinMode(11, INPUT);  // Configura o pino 11 para receber sinais do sensor traseiro
    pinMode(12, INPUT);  // Configura o pino 12 para receber sinais do sensor frontal esquerdo
    pinMode(13, INPUT);  // Configura o pino 13 para receber sinais do sensor frontal direito
}
```

Esses passos abaixo fazem o robô reagir à presença de uma linha detectada pelos sensores, recuando e girando para evitar cruzar essa linha, essencial em competições como sumô, onde sair da arena pode desclassificar o robô.

```
void loop() {  // Se o sistema estiver ligado, executa o código
    if (estadoSistema) {
      if (!digitalRead(12) || !digitalRead(13)) {
         Serial.println("Linha detectada");
         stop();
        delay(1);
         re();
        delay(300);
         stop();
        delay(1);
        girar();
        delay(600);
        stop();
      }
    }
}
```

AULA 6

Explicando as Funções - DE LINHA

Verificação dos Sensores de Linha:
if (!digitalRead(12) || !digitalRead(13))
Esta linha lê os pinos 12 e 13, que estão conectados aos sensores de linha.

- digitalRead(12) e digitalRead(13) retornam LOW (0) quando uma linha é detectada.
- O operador ! (não) inverte o valor, tornando-o verdadeiro (true) quando uma linha é detectada.

Comportamento do Robô ao Detectar Linha:
Quando a condição acima é verdadeira, o robô inicia uma sequência de ações:

- **Exibe uma Mensagem:** Serial.println("Linha detectada"); imprime a mensagem "Linha detectada" no monitor serial.
- **Para o Movimento:** stop(); chama a função stop, que desativa os motores para evitar cruzar a linha.
- **Dá Ré:** A função re(); faz o robô se afastar da linha. Em seguida, aguarda 300ms para garantir que o robô recue o suficiente.
- **Gira:** girar(); é chamada para que o robô faça uma rotação, reposicionando-se longe da linha. Ele aguarda 600ms para que o giro seja concluído.
- **Parada Final:** Por fim, o robô chama stop(); novamente para garantir que o movimento foi totalmente concluído.

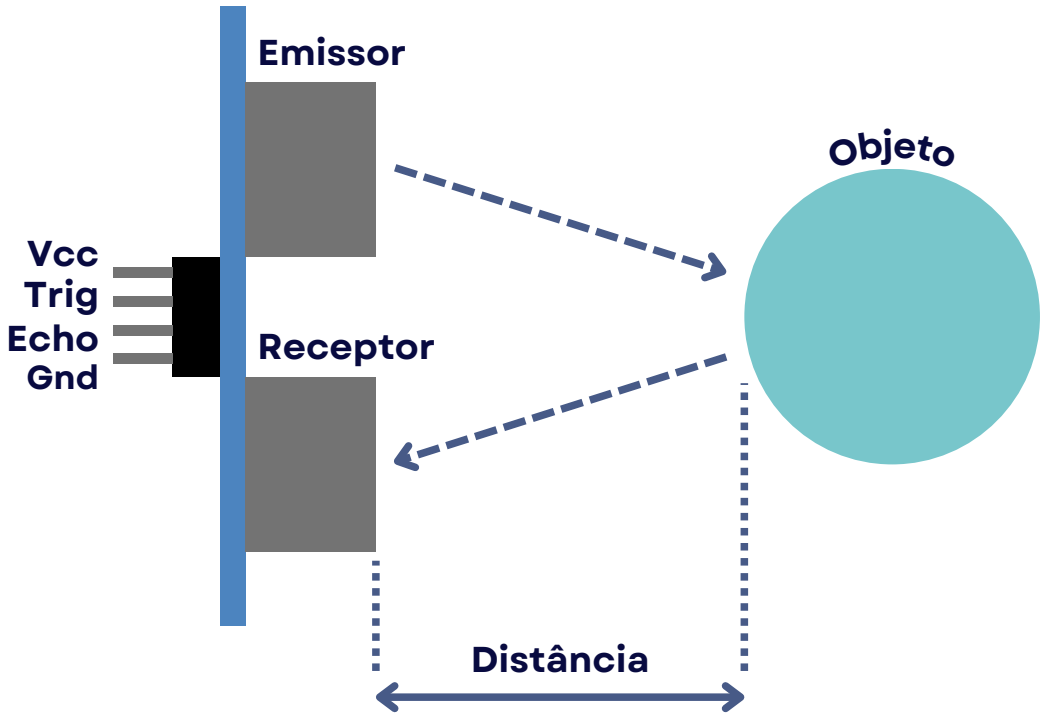
AULA 7

Explicando as Funções - ULTRASSÔNICO

Uso do Sensor Ultrassônico no Robô de Sumô com Arduino
O sensor ultrassônico é um componente essencial para que o robô de sumô **detecte objetos ao seu redor**, usando **ondas sonoras** para **medir a distância**.
Esse sensor permite ao robô identificar a posição de um adversário na arena e reagir rapidamente.

Funcionamento do Sensor Ultrassônico
O sensor possui dois pinos principais:

- **Trigger**: Esse é o pino que **inicia a medição de distância**. Quando o Arduino envia um pulso de alta frequência para o Trigger, **o sensor emite uma onda ultrassônica em direção ao objeto.**
- **Echo**: Esse pino **recebe o sinal de retorno**. Após a **onda ultrassônica** ser emitida, ela viaja até encontrar um objeto e é **refletida de volta para o sensor**. O pino Echo permanece ativo pelo tempo que a onda leva para ir até o objeto e retornar. Com isso, é possível calcular a distância com precisão.



AULA 7

Explicando as Funções - ULTRASSÔNICO

Incluir a Biblioteca e Declarar o Sensor
Primeiro, incluímos a biblioteca Ultrasonic para simplificar o uso do sensor. Para isso, vá em Rascunho -> Incluir Biblioteca -> Ultrasonic e adicione a seguinte linha no código:

```
#include <Ultrasonic.h>
```

Em seguida, criamos uma instância do sensor ultrassônico, definindo os pinos Trigger e Echo:

```
Ultrasonic ultrasonic (8, 7);  // Pino 8 para Trigger, pino 7 para Echo
```

Configuração dos Pinos no setup()
No setup(), configuramos o pino **Trigger** como **saída** e o pino **Echo** como **entrada**. Também inicializamos a **comunicação serial** para monitorar as leituras de distância.

```
void setup () {
    pinMode (8, OUTPUT);  // Pino Trigger como saída
    pinMode (7, INPUT);  // Pino Echo como entrada
    Serial.begin(9600);  // inicializa a comunicação serial para monitoramento
}
```

Ler a Distância no loop()
No loop(), utilizamos o sensor ultrassônico para medir a distância apenas quando o sistema está ligado (representado pela variável estadoSistema).

```
void loop () {
    if (estadoSistema) {  // executa apenas se o sistema estiver ligado
        int centimetros = ultrasonic. read (CM); // lê a distância em cm
```

Ações com Base na Distância
Se a distância detectada for menor que 30 cm, o robô executa ações específicas, como parar e avançar rapidamente ("atacar") para confrontar o oponente.

```
    if (centimetros < 30) { // Verifica se há um obstáculo a menos de 30 cm
        Serial.println("Oponente detectado");
        stop ();
        delay (1);
        atacar ();
        delay (300);
        stop ();
        }
    }
}
```

AULA 8

Explicando as Funções - BOTÃO

Explicação da parte do código que trata do controle do estado do sistema por meio de um botão:
Declarando as variáveis do sistema para funcionamento do botão:

```
bool estadoSistema = false;
bool botaoPressionado = false;
unsigned long tempoPressionado = 0;
const unsigned long tempoNecessario = 500;
bool estadoAnteriorBotao = LOW;
```

- **estadoSistema:** Armazena o estado atual do sistema (ligado/desligado), alternando seu valor sempre que o botão é pressionado e segurado por um tempo suficiente.
- **botaoPressionado:** Indica se o botão foi pressionado, ajudando a controlar a duração da pressão e a evitar alternâncias múltiplas indesejadas.
- **tempoPressionado:** Registra o momento exato em que o botão foi pressionado, usando a função millis(). Essa variável é usada para calcular a duração da pressão do botão.
- **tempoNecessario:** Define o tempo mínimo que o botão deve ser pressionado para que o sistema alterne seu estado, configurado para 500 ms (ou 0,5 segundos).
- **estadoAnteriorBotao:** Armazena o estado do botão na última leitura do loop(), permitindo detectar mudanças entre pressionado (HIGH) e solto (LOW), essenciais para capturar a pressão inicial do botão.

Configuração do botão no modo INPUT_PULLUP:

```
 void setup() {
      pinMode(2, INPUT_PULLUP); // Configura o pino 2 como entrada com resistor pull-up para o botão
      Serial.begin(9600);  // Inicializa a comunicação serial
      stop(); // Robô inicia desligado
 Serial.println("Sistema Desligado - Aguardando primeiro clique para ligar...");
}
```

- **pinMode(2, INPUT_PULLUP):** Configura o pino 2 como entrada com resistor pull-up para o botão.
- **Serial.begin(9600):** Inicializa a comunicação serial.
- **stop():** Robô inicia desligado.
- **Serial.println("Sistema Desligado - Aguardando primeiro clique para ligar...");** Entrega a mensagem ao serial quando ligar o arduino.

AULA 8

Explicando as Funções - BOTÃO

Esta parte do código lê o estado de um botão e alterna o estado de um sistema **(liga/desliga)** quando o botão é pressionado e segurado por mais de 0,5 segundos. Ele verifica se o botão foi pressionado e registra o momento da pressão. Se o botão permanecer pressionado pelo tempo necessário, o estado do sistema é alternado. Chama a função **stop()** para parar quando necessário e uma breve pausa **delay** ajuda a evitar múltiplas leituras indesejadas.

```
 void loop() {
      bool estadoAtualBotao = digitalRead(2);
      if (estadoAtualBotao == HIGH && estadoAnteriorBotao == LOW) {
         tempoPressionado = millis();
         botaoPressionado = true;
}

 if (estadoAtualBotao == HIGH && botaoPressionado && (millis() - tempoPressionado >= tempoNecessario)) {
      estadoSistema = !estadoSistema;

      if (estadoSistema) {
         Serial.println("Sistema Ligado");
      }
      else {
         Serial.println("Sistema Desligado");
         stop();
      }

      botaoPressionado = false;
      delay(300);
}

 if (estadoAtualBotao == LOW && estadoAnteriorBotao == HIGH) {
      botaoPressionado = false;
}

 estadoAnteriorBotao = estadoAtualBotao;
}
```

AULA 8

Explicando as Funções - BOTÃO

Leitura do Estado Atual do Botão:

- **estadoAtualBotao = digitalRead(2)**: Lê o estado atual do botão conectado ao pino 2. Retorna HIGH quando o botão é pressionado e LOW quando está solto.

Detecção de Pressionamento Inicial:

- **if (estadoAtualBotao == HIGH && estadoAnteriorBotao == LOW)**: Verifica uma mudança de estado de LOW para HIGH, indicando que o botão acabou de ser pressionado.
- **tempoPressionado = millis()**: Armazena o momento exato em que o botão foi pressionado.
- **botaoPressionado = true**: Marca que o botão está atualmente pressionado.

Verificação de Pressão Prolongada:

- **if (estadoAtualBotao == HIGH && botaoPressionado && (millis() - tempoPressionado >= tempoNecessario))**: Verifica se o botão está pressionado continuamente pelo tempo necessário (0,5 segundos).
- **estadoSistema = !estadoSistema**: Alterna o estado do sistema. Se estiver desligado, liga, e vice-versa.
- **Serial.println("Sistema Ligado" ou "Sistema Desligado")**: Exibe o estado atual do sistema no monitor serial.
- **stop()**: Chama a função stop para garantir que o robô pare se o sistema for desligado.
- botaoPressionado = false: Redefine a variável para evitar múltiplas detecções do mesmo pressionamento.
- **delay(300)**: Atraso para evitar o "bouncing" do botão (leituras erradas causadas por instabilidade mecânica).

Detecção de Soltura do Botão:

- **if (estadoAtualBotao == LOW && estadoAnteriorBotao == HIGH)**: Detecta a transição de HIGH para LOW, ou seja, o botão foi solto.
- **botaoPressionado = false**: Redefine a variável para permitir a próxima detecção de pressão prolongada.

Atualização do Estado Anterior do Botão:

- **estadoAnteriorBotao = estadoAtualBotao**: Armazena o estado atual do botão para comparações na próxima iteração do loop.

```
#include <Ultrasonic.h>

Ultrasonic ultrasonic(8, 7); // Trigger e Echo

bool estadoSistema = false; // Variável para armazenar o estado do
sistema
bool botaoPressionado = false; // Variável para verificar se o botão foi
pressionado

void setup() {
  pinMode(11, INPUT); // Sensor de linha tras
  pinMode(12, INPUT); // Sensor de linha esquerda frente
  pinMode(13, INPUT); // Sensor de linha direita frente

  pinMode(2, INPUT_PULLUP); // Botão de Ligar (utilizando INPUT_PULLUP)

  pinMode(5, OUTPUT);  // Ponte H B-IA // tras
  pinMode(6, OUTPUT);  // Ponte H B-IB // frente
  pinMode(9, OUTPUT);  // Ponte H A-IA // tras
  pinMode(10, OUTPUT); // Ponte H A-IB // frente

  pinMode(8, OUTPUT); // Trigger Sensor Ultrassônico
  pinMode(7, INPUT);  // Echo Sensor Ultrassônico

  Serial.begin(9600);

  // Robô está desligado no início
  stop ();
  Serial.println("Sistema Desligado - Aguardando primeiro clique para
ligar...");
}

void loop() {
  // Verifica se o botão foi pressionado
  if
 (digitalRead(2) == HIGH){
    // Se o botão estava solto e agora foi pressionado
    if (!botaoPressionado) {
      botaoPressionado = true; // Marca que o botão foi pressionado
      estadoSistema = !estadoSistema; // Alterna o estado do sistema
(liga/desliga)

      delay(300);
```

```
 if (estadoSistema) {
        Serial.println("Sistema Ligado");
      } else {
        Serial.println("Sistema Desligado");
        stop();  // Para o robô quando desligar
      }
    }
  }

  // Se o botão foi solto
  if (digitalRead(2) == LOW && botaoPressionado) {
    botaoPressionado = false; // Reseta a variável para detectar a
próxima pressão
  }

  // Se o sistema estiver ligado, executa o código
  if (estadoSistema) {
    if (!digitalRead(12) || !digitalRead(13)) { // Verifica sensores de
linha
      Serial.println("Linha detectada");
      stop();
      delay(1);
      re();
      delay(300);
      stop();
      delay(1);
      girar();
      delay(600);
      stop();
    }

    int centimetros = ultrasonic.read(CM);

    if (centimetros < 30) {
      Serial.println("Oponente detectado");
      stop();
      delay(1);
      atacar();
      delay(300);
      stop();
    }
    frente();
    delay(1);
  }
}
```

```
void re() {
  Serial.println("Re");
  analogWrite(5, 150);
  analogWrite(6, 0);
  analogWrite(9, 150);
  analogWrite(10, 0);
}

void frente() {
  Serial.println("Frente");
  analogWrite(5, 0);
  analogWrite(6, 150);
  analogWrite(9, 0);
  analogWrite(10, 150);
}

void girar() {
  Serial.println("Girando");
  analogWrite(5, 0);
  analogWrite(6, 150);
  analogWrite(9, 150);
  analogWrite(10, 0);
}

void atacar() {
  Serial.println("Atacar");
  analogWrite(5, 0);
  analogWrite(6, 200);
  analogWrite(9, 0);
  analogWrite(10, 200);
}

void stop() {
  Serial.println("Parado");
  analogWrite(5, 0);
  analogWrite(6, 0);
  analogWrite(9, 0);
  analogWrite(10, 0);
}
```