1. Aula 19

Nesta aula, iremos conhecer um pouco mais sobre os códigos do S4A, criando regras que simulam a configuração do nosso robô seguidor de linha.

Veja a estrutura completa.

```
  int AIA = 9;  // (pwm) pino 9 conectado ao pino A-IA do Módulo
  int AIB = 8;  // (pwm) pino 8 conectado ao pino A-IB do Módulo
  int BIA = 7; // (pwm) pino 7 conectado ao pino B-IA do Módulo
  int BIB = 6;  // (pwm) pino 6 conectado ao pino B-IB do Módulo
byte frente = 255;
byte voltar = 200;
byte speed1 = 255;
byte speed2 = 255;// Mude este valor (0-255) para controlar a velocidade dos motores
byte speedv1 = 255;
byte speedv2 = 255;

byte sensor2 = 12;//Pino que deve se conectar a saída (out) do sensor
byte sensor1 = 11; //Pino que deve se conectar a saída (out) do sensor

void setup() {
  pinMode(AIA, OUTPUT); // Colocando os pinos como saída
  pinMode(AIB, OUTPUT);
  pinMode(BIA, OUTPUT);
  pinMode(BIB, OUTPUT);
}

void loop() {

 if(!digitalRead(sensor2))
    {
 speed2=frente;
        speedv2=0;
    }else{
       speed2=0;
        speedv2=voltar;
      }

        if(!digitalRead(sensor1))
    {

speed1=frente;
speedv1=0;
    }else{
```

Vamos entender o que faz o void loop().

Na programação, um **loop()** é conjunto de instruções que um programa de computador percorre e repete um significativo número de vezes até que sejam alcançadas as condições desejadas.

No Arduino, após a função setup(), que inicializa e declara os valores iniciais, a função loop() faz precisamente o que seu nome indica: ela repete-se continuamente, permitindo que seu programa funcione dinamicamente. É utilizada para controlar de forma ativa a placa Arduino.

Dentro do **LOOP(),** temos a função **IF()**, que serve como uma condição. Nesse caso, ela verifica se o sensor 2 estiver detectando a linha, siga em frente e acelere até 255.

As estruturas de condição possibilitam ao programa tomar decisões e alterar o seu fluxo de execução. É por meio delas que podemos dizer ao sistema: “execute a instrução A caso a expressão X seja verdadeira; caso contrário, execute a instrução B”.

if/else

A estrutura condicional if/else permite ao programa avaliar uma expressão como sendo verdadeira ou falsa e, de acordo com o resultado dessa verificação, executar uma ou outra rotina.

Sintaxe do if/else:

```
if (expressão booleana) {
   // bloco de código 1
} else {
   // bloco de código 2
}
```

As instruções presentes no bloco de código 1 serão executadas caso a expressão booleana seja verdadeira. Do contrário, serão executadas as instruções presentes no bloco de código 2.

As chaves são delimitadores de bloco, tendo a função de agrupar um conjunto de instruções. Apesar do uso desses delimitadores ser opcional caso haja apenas uma linha de código, ele é recomendado, pois facilita a leitura e manutenção do código, tornando-o mais legível.

else if

Complementar ao if/else temos o operador else if. Esse recurso possibilita adicionar uma nova condição à estrutura de decisão para atender a lógica sendo implementada.

Sintaxe do if/else com else if:

```
if (expressão booleana 1) {
   // bloco de código 1
} else if (expressão booleana 2) {
   // bloco de código 2
} else {
   // bloco de código 3
}
```

Dessa forma, se a expressão booleana 1 for verdadeira, o bloco de código 1 será executado. Caso seja falsa, o bloco de código 1 será ignorado e será testada a expressão booleana 2. Se ela for verdadeira, o bloco de código 2 será executado. Caso contrário, o programa vai ignorar esse bloco de código e executar o bloco 3, declarado dentro do else.

Podemos utilizar quantos else if forem necessários. Entretanto, o else deve ser adicionado apenas uma vez, como alternativa ao caso de todos os testes terem falhado.