Robótica Educacional

VERSÃO 2

Módulo 3

Aula
35

Robô habilidoso - I

Diretoria de Tecnologia e Inovação

PARANÁ
GOVERNO DO ESTADO
SECRETARIA DA EDUCAÇÃO

GOVERNADOR DO ESTADO DO PARANÁ
Carlos Massa Ratinho Júnior

SECRETÁRIO DE ESTADO DA EDUCAÇÃO
Roni Miranda Vieira

DIRETOR DE TECNOLOGIA E INOVAÇÃO
Claudio Aparecido de Oliveira

COORDENADOR DE TECNOLOGIAS EDUCACIONAIS
Marcelo Gasparin

Produção de Conteúdo
Cleiton Rosa
Darice Alessandra Deckmann Zanardini
Roberto Carlos Rodrigues

Validação de Conteúdo
Cleiton Rosa

Revisão Textual
Kellen Pricila dos Santos Cochinski

Projeto Gráfico e Diagramação
Edna do Rocio Becker

2025

Sumário

Aula 35 Robô habilidoso - I

Introdução

No mundo da Robótica, da tecnologia e até no nosso dia a dia, estamos sempre em busca de melhorias. Pense em smartphones ou outros gadgets, por exemplo, que recebem atualizações de software o tempo todo, novos modelos são lançados com recursos mais avançados, nova interface, sistema operacional mais robusto... Agora pense em um jogo em que você começa com o personagem básico e, ao longo da partida, recebe novas habilidades, ferramentas e armaduras. Tudo isso são exemplos de **upgrades**!

Um **upgrade** é basicamente uma melhoria ou aprimoramento em algo que já existe. E não se trata apenas de deixar isso mais bonito ou mais potente, muitas vezes o upgrade serve para adicionar novas funcionalidades, melhorar o desempenho, aumentar a durabilidade ou mesmo otimizar a experiência do usuário.

Na Robótica, o conceito de **upgrade** é fundamental para o desenvolvimento contínuo, tal como temos feito em alguns projetos. É um processo de construir, testar, identificar o que pode ser melhorado e então aplicar essas melhorias – e é assim que os protótipos se transformam em itens cada vez mais incríveis!

Em um robô que já se movimenta, a adição de novas funcionalidades pode ser a inclusão de outros sensores ou atuadores. Outro upgrade pode ser em relação à melhoria do seu desempenho, por exemplo, revendo a potência dos motores ou a precisão de movimentos. Além disso, podemos pensar na estrutura física do robô, fazendo com que ele tenha mais sua durabilidade e funcione de forma mais consistente. Também podemos rever a programação para tornar o seu controle mais fácil, intuitivo ou mesmo divertido!

Aula 35 Robô habilidoso - I

Objetivos desta aula

- Ampliar e aprimorar o robô com sensor ultrassônico para atribuição de novas funções;
- Criação de funções;
- Personalização de projetos.

Lista de materiais

- Notebook;
- Arduino IDE.

Aula 35 Robô habilidoso - I

Roteiro da aula

1. Contextualização

Vocês se lembram dos nossos primeiros projetos com sensores ultrassônicos? Aquele robô que conseguia "enxergar" os obstáculos, desviar deles, medindo distâncias? E como saber qual movimento ele faria na sequência, ao detectar um obstáculo?

Provavelmente, no primeiro projeto do robô ultrassônico, a gente poderia incrementar o código e habilitar o monitor serial do Arduino para imprimir os valores do sensor... ou então, conforme a programação, ao parar e virar, deduzíamos que o robô detectou um obstáculo. Os movimentos do robô podem ser bem precisos na detecção dos obstáculos, porém, como mostrar sua movimentação para nós de forma mais rápida e clara?

É exatamente aí que entra a ideia de adicionar LEDs para indicar as manobras do robô – então pense nisso como upgrade de linguagem visual ou mesmo upgrade de "comunicação" do nosso robô ultrassônico.

Na versão anterior do robô ultrassônico, ele detectava obstáculo e virava. Mas não havia uma forma visual para dizer "olha, tem algo na frente!" ou "estou virando para esquerda porque desviei de um obstáculo". Agora, o robô fará as manobras que comandarmos via programação, também mostrando o que está fazendo pela sua estrutura física. E o upgrade desse robô terá "mais um plus": a atribuição da função seguidor de linha!

Vamos lá?

Iniciaremos a programação de upgrade do robô com a definição de todos os pinos para motores, sensor ultrassônico, LEDs, potenciômetro, botão e sensores infravermelhos no preâmbulo como constantes, ou seja, valores de leitura que não se alteram durante a execução da programação.

```
/* Definições de pinos. */
#define motor1A 9
#define motor1B 6
#define motor2A 5
#define motor2B 3
#define trigPin 10
#define echoPin 11
#define re 2
#define farolLED 7
#define freioLED 8
#define piscaDireita 12
#define piscaEsquerda 13
#define potPin A2
#define botaoPin 4
#define sensorIREsq 14 // A0 como
digital
#define sensorIRDir 15 // A1 como
digital
```

Aula 35 Robô habilidoso - I

Na sequência do preâmbulo, declaramos as variáveis globais para serem usadas em diferentes partes do código. Essas variáveis são importantes por permitirem que diferentes funções da nossa programação compartilhem e atualizem informações, garantindo que o robô tenha acesso aos dados mais recentes para tomar suas decisões:

- **duracao** e **distancia** vão armazenar os resultados do sensor ultrassônico.
- **modo** vai guardar se o robô está no "modo obstáculo" (0) - inicial - ou "modo seguidor de linha" (1), decidindo o comportamento principal do robô.

```
long duracao;
int distancia;
bool  modo = 0; // 0 = obstáculo, 1 = seguidor de linha
```

Finalizamos o preâmbulo com as variáveis para o ***debounce*** do botão. Ao pressionarmos o botão para escolha da função do robô - modo seguidor de linha ou modo desvio de obstáculo -, os contatos do botão podem oscilar antes de se firmarem no estado estável de pressionado (LOW) ou solto (HIGH), o que pode levar o Arduino a ler múltiplos sinais - ruídos - e interpretar como pressionamentos, mesmo que tenhamos apertado o botão apenas uma vez. Aplicando a técnica do ***debounce***, o Arduino ignora qualquer ruído que ocorra dentro de um curto período após a primeira mudança de estado, garantindo que cada apertar do botão seja lido como um único comando:

- **ultimoTempoBotao** armazena o último momento em que o estado do botão foi alterado de forma confiável e é usada para calcular o tempo decorrido.
- **debounceDelay** define o tempo de espera após uma mudança de estado para considerar a próxima leitura como válida e estável.
- **estadoBotaoAnterior** armazena a última leitura estável do botão.

```
/* Variáveis para o debounce do botão. */
long ultimoTempoBotao = 0;
long debounceDelay = 50; /* Atraso de 50ms para o debounce. */
bool estadoBotaoAnterior = HIGH;
```

Aula 35 Robô habilidoso - I

Na sequência, faremos as configurações iniciais no **void setup()**, como as configurações de entrada e saída e outras funções de início:

- **pinMode()** para dizer ao Arduino se um pino vai ser usado para enviar energia/sinal (OUTPUT, como para os motores e LEDs) ou para receber um sinal (INPUT para os sensores e o potenciômetro). Como conectamos o botão diretamente ao Arduino, utilizamos o parâmetro INPUT_PULLUP, ativando a resistência interna como já realizado em montagens anteriores.
- **digitalWrite(farolLED, HIGH)** para acionar os "faróis" assim que o robô inicializar.
- **Serial.begin(9600)** para iniciar a comunicação serial, fundamental para monitorarmos o que o Arduino está lendo, como a distância do sensor ultrassônico, auxiliando no debugging (saberemos mais sobre este tema na próxima Aula!).

Aula 35 Robô habilidoso - I

```
void setup() {
  pinMode(motor1A, OUTPUT);
  pinMode(motor1B, OUTPUT);
  pinMode(motor2A, OUTPUT);
  pinMode(motor2B, OUTPUT);

  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  pinMode(re, OUTPUT);
  pinMode(farolLED, OUTPUT);
  pinMode(freioLED, OUTPUT);
  pinMode(piscaDireita, OUTPUT);
  pinMode(piscaEsquerda, OUTPUT);

  pinMode(potPin, INPUT);
  pinMode(botaoPin, INPUT_PULLUP);
  pinMode(sensorIREsq, INPUT);
  pinMode(sensorIRDir, INPUT);

  digitalWrite(farolLED, HIGH);  /* Faróis sempre ligados. *
/
  Serial.begin(9600);
}
```

Aula 35 Robô habilidoso - I

Chegamos à programação do **void loop()**, o "coração" do programa, e vamos analisar as partes de sua estrutura:

- Iniciamos com a criação da variável local **statusBotao**, essencial para o ***debounce*** ao armazenar a leitura do pino no qual o botão está conectado, usado para alternar os modos do robô entre desvio de obstáculo ou seguidor de linha:
    - Se a leitura atual for diferente da **estadoBotaoAnterior** (ou seja, se houve uma mudança de status - possivelmente o início de um **bounce** ou um pressionamento real), o tempo atual é registrado em **ultimoTempoBotao** pela função **millis()**.
    - A cada ciclo do loop, o estado atual do botão é lido: se o botão for pressionado, passando de alto para baixo, a variável **modo** inverte seu valor.
    - A condição **if ((millis() - ultimoTempoBotao) > debounceDelay)** verifica se já se passaram 50 milissegundos de atraso, definido no preâmbulo por **debounceDelay**, para garantir que a leitura atual seja estável e não apenas ruído.
    - Se o tempo for validado e o botão estiver pressionado **(if (leituraBotao == LOW))**, a variável **modo** é invertida, alternando entre os modos do robô e o código entra em um loop controlado pelo **while(digitalRead(botaoPin) == LOW)** para "travar" a execução do programa e impedir múltiplas trocas de **modo** enquanto o botão estiver pressionado. Ao ser solto, o **while()** finaliza com um pequeno delay para garantir que o botão se estabilize completamente para leitura de um único comando e seu **status** é atualizado e impresso no monitor serial para **debugging** (tema da próxima aula).
- Leitura do potenciômetro e mapeamento para controle dos motores:
    - O potenciômetro permite um controle suave e gradual do robô e, pela função **map()**, pegaremos o valor do intervalo de 0 a 1023 do potenciômetro para um novo intervalo de 0 a 255, ideal para o controle PWM da potência dos motores.
- Modos de operação:
    - Pela lógica condicional **if... else**, criamos duas formas de operação do robô – **modo == 0** para **desvio de obstáculos** ou **modo == 1** para **seguidor de linha**, sendo executado apenas o código correspondente ao modo.
- Modo desvio de obstáculos:
    - Gerencia o comportamento do robô quando um obstáculo é detectado pelo

sensor ultrassônico – caso a distância for menor que 10 cm, ele reage com uma sequência de ações cujas funções criaremos no decorrer da programação: para, com o indicativo de "freio", dá ré, sinalizando o movimento, para novamente, acende o "freio" de novo e então decide aleatoriamente para qual lado virar, piscando a seta correspondente. Pela função **random()**, adicionamos a imprevisibilidade ao comportamento e a utilização dos LEDs permite o feedback visual das ações do robô, ideia do nosso upgrade.

- Modo seguidor de linha:
  - Controla o robô para seguir, usando os sensores infravermelhos, uma linha. Os sensores fornecem um sinal digital alto ou baixo, indicando se estão sobre uma superfície clara ou escura, e o robô toma decisões com base na combinação do estado desses dois sensores:
    - Ambos na linha: o robô para.
    - Um sensor sobre a linha: ajusta a direção do robô.
    - Nenhum sensor na linha: o robô segue em frente.
  - Pela função **analogWrite()**, controlamos os motores para correções de direção do robô no modo seguidor de linha.
- Delay final:
  - Pausa entre cada loop para otimizar controles e a execução de ações.

```
void loop() {

  /* Leitura do botão evitando múltiplas leituras. */

  bool statusBotao = digitalRead(botaoPin);

  if (statusBotao != estadoBotaoAnterior) {

    ultimoTempoBotao = millis();

  }

  if ((millis() - ultimoTempoBotao) > debounceDelay) {

    if (statusBotao == LOW) {

      modo = !modo;

      /* Aguarda o botão ser solto para evitar múltiplas trocas. */
```

Aula 35 Robô habilidoso - I

```
      while(digitalRead(botaoPin) == LOW);
      delay(50); /* Pequeno delay para garantir que o botão foi solto. */
    }
  }
  estadoBotaoAnterior = statusBotao;
  /* Impressão do modo selecionado para o robô. */
  Serial.print("Modo: ");
  Serial.println(modo);
  int potencia = map(analogRead(potPin), 0, 1023, 0, 255);
  if (modo == 0) {
    /* Modo desvio de obstáculos. */
    distancia = medirDistancia();
    if (distancia < 10) {
      parar();
      digitalWrite(freioLED, HIGH);
      delay(300);
      moverTras(potencia);
      digitalWrite(freioLED, LOW);
      digitalWrite(re, HIGH);
      delay(500);
      parar();
      digitalWrite(re, LOW);
      digitalWrite(freioLED, HIGH);
      delay(300);
     digitalWrite(freioLED, LOW);
```

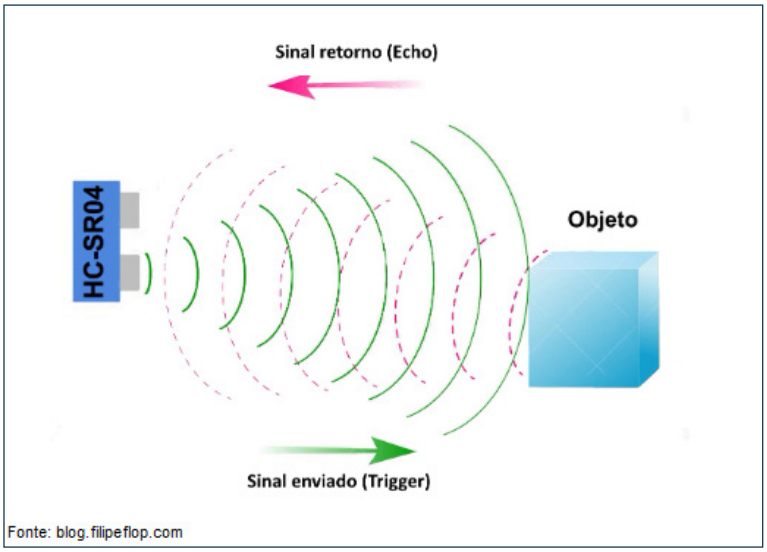
Aula 35 Robô habilidoso - I

```
      if (random(0, 2) == 0) {
        piscar(piscaDireita);
        virarDireita(potencia);
      } else {
        piscar(piscaEsquerda);
        virarEsquerda(potencia);
      }
    } else {
      moverFrente(potencia);
    }
  } else {
    /* Modo seguidor de linha com controle direto dos motores. */
    int irEsq = digitalRead(sensorIREsq);
    int irDir = digitalRead(sensorIRDir);

    if (irEsq == LOW && irDir == LOW) {
      moverFrente(potencia);
    } else if (irEsq == HIGH && irDir == LOW) {
      /* Corrige curva para esquerda. */
      analogWrite(motor1A, 0);
      analogWrite(motor1B, potencia);
      analogWrite(motor2A, potencia);
      analogWrite(motor2B, 0);
    } else if (irEsq == LOW && irDir == HIGH) {
```

Aula 35 Robô habilidoso - I

```
      /* Corrige curva para direita. */
      analogWrite(motor1A, potencia);
      analogWrite(motor1B, 0);
      analogWrite(motor2A, 0);
      analogWrite(motor2B, potencia);
    } else {
      parar();
    }
  }
  delay(100);
}
```



Vamos agora às **funções auxiliares** que criamos para nosso robô. Como em aulas anteriores, poderíamos criar abas específicas para cada uma ou deixá-las no sketch principal. Como serão três tipos de funções auxiliares, vamos indicá-las em nossa programação após o **void loop()** e abordá-las aqui pelo sensor ou atuador relacionado.

Para o **sensor ultrassônico**, criaremos a função **int medirDistancia()** para retornar um valor inteiro, em centímetros, correspondente à medição do sensor. Essa função não exige nenhum argumento para funcionar e talvez você recorde que em outros projetos com sensor ultrassônico utilizamos uma biblioteca, porém aqui focaremos no trabalho direto com o sensor e seus pinos **trig** e **echo** para emissão e recepção, lembrando dos detalhes das ondas ultrassônicas como aprendemos na **Aula 34 - Sensor de distância**, do Módulo 1 de Robótica Educacional.

Aula 35 Robô habilidoso - I

Para **emissão de pulso**, as funções **digitalWrite(trigPin, LOW)** e **delayMicroseconds(3)** garantem que o pino trig esteja em estado baixo por um breve momento, limpando qualquer sinal anterior. Então, **digitalWrite(trigPin, HIGH) e delayMicroseconds(10)** colocam o pino trig em estado alto por exatamente 10 microssegundos, desligando-o depois. É essa pulsação de 10 microssegundos que instrui o sensor ultrassônico a enviar um "ping" sônico, como se o robô desse um "gritinho" rápido.

Saiba mais!!!

Na nossa jornada pela Robótica, já conhecemos duas funções temporizadoras: delay() e millis(). No projeto dessa aula, mais uma! delayMicroseconds() permite uma pausa micro e seu parâmetro é um número em microssegundos (acima de 3 microssegundos para funcionamento mais correto). E quanto vale o microssegundo?

1 microssegundo (µs) = 0.001 milissegundo (ms) ou 0.000001 segundo (s)

1000 microssegundos (µs) = 1 milissegundo (ms)

1000000 microssegundos (µs) = 1 segundo (s)

Para a **recepção do pulso**, utilizamos outra função do Arduino, **pulseIn()**, destinada à medição do tempo em microssegundos que o pino **echo** ficou aguardando até o sinal "retornar a", aguardando até o sinal retornar a ele. Essa duração do pulso do eco é armazenada na variável **duracao**.

Para cálculo da distância captada pelo sensor ultrassônico, chegou o momento de associarmos a Física à Matemática! A velocidade do som no ar é de aproximadamente 340 metros por segundo, ou 0,034 centímetros por microssegundo. Multiplicamos a duração (em microssegundos) por essa velocidade para obter a distância total (ida e volta) que o som percorreu. Então, para termos o valor de distância que queremos até o objeto detectado, dividimos por 2 porque a variável **duracao** está com o valor de ida e volta. Por fim, essa função que criamos retorna o valor final inteiro calculado para ser usado no **void loop()**.

Aula 35 Robô habilidoso - I

```
int medirDistancia() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(3);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  duracao = pulseIn(echoPin,
HIGH);
  return duracao * 0.034 / 2;
}
```

Na sequência, temos um conjunto de funções auxiliares responsáveis pelos movimentos do robô. Em cada uma delas, como realizamos em programações anteriores, é feita uma combinação de potências a cada um dos pinos dos motores e todas as funções de movimento – exceto parar() – recebem um argumento **int pot**. Isso significa que podemos controlar a potência do robô para cada tipo de movimento de forma dinâmica, usando o potenciômetro que recebe um valor e guarda na variável **potencia**. Outro controle que, caso necessário, pode ser ajustado, diz respeito ao delay aplicado nas manobras do robô. Nas funções **virarDireita()** e **virarEsquerda()**, aplicamos 300 milissegundos e esse valor pode ser ajustado para manobras mais eficientes.

```
void moverFrente(int pot) {
  analogWrite(motor1A, pot);
  analogWrite(motor1B, 0);
  analogWrite(motor2A, pot);
  analogWrite(motor2B, 0);
}
void moverTras(int pot) {
  analogWrite(motor1A, 0);
  analogWrite(motor1B, pot);
  analogWrite(motor2A, 0);
  analogWrite(motor2B, pot);
}
void parar() {
  analogWrite(motor1A, 0);
  analogWrite(motor1B, 0);
  analogWrite(motor2A, 0);
  analogWrite(motor2B, 0);
}
void virarDireita(int pot) {
  analogWrite(motor1A, pot);
  analogWrite(motor1B, 0);
  analogWrite(motor2A, 0);
  analogWrite(motor2B, pot);
  delay(300);
}
void virarEsquerda(int pot) {
  analogWrite(motor1A, 0);
  analogWrite(motor1B, pot);
  analogWrite(motor2A, pot);
  analogWrite(motor2B, 0);
  delay(300);
}
```

Aula 35 Robô habilidoso - I

Como havíamos falado sobre o upgrade do robô quanto ao seu aspecto de "feedback visual", a função **void piscar(int pino)** é o que dá ao nosso robô a capacidade de comunicar visualmente algumas de suas ações, de forma semelhante às setas de um carro ou às luzes de alerta de uma máquina. Para as demais ações do robô fluírem, utilizaremos aqui a função **millis()**, já explorada em aulas anteriores.

```
void piscar(int pino) {
  for (int i = 0; i < 4; i++) {
    digitalWrite(pino, HIGH);
    delay(200);
    digitalWrite(pino, LOW);
    delay(200);
  }
}
```

Agora, vocês têm uma compreensão completa de como o robô se move, percebe o ambiente e se comunica. Confira como ficou a **programação completa dessa aula** para, na próxima, realizarmos a **montagem** do robô. Até lá!

Desafios:

Outra sugestão de comunicação visual refere-se à inserção de um painel OLED nas portas do Arduino ainda não utilizadas... programe o OLED para exibir, por exemplo, o modo de operação do robô, a potência dos motores e a direção da manobra.

Que tal adicionar mais uma comunicação ao seu robô? Verifique quais portas do Arduino seguem disponíveis e insira um buzzer para sinalização sonora dos movimentos.

Você também pode pensar em outros incrementos, como ativar um modo de esquivo no seu robô, instalando o sensor ultrassônico em um suporte móvel, controlado por servomotor, para aumentar seu campo de "observação".

E se...

O código apresentar algum erro? Verifique as sintaxes e parâmetros aplicados, bem como a criação das funções auxiliares.

2. Feedback e finalização

Assim como o sensor ultrassônico deu "visão" ao nosso primeiro robô, os LEDs darão a este robô a capacidade de expressar suas ações de forma visual e clara.

Esse é mais um exemplo de projeto que demonstra o quanto a Robótica não se esgota e, cada vez mais, podemos ampliar possibilidades!

REFERÊNCIAS

ARDUINO. **Documentação de Referência da Linguagem Arduino**. Disponível em: https://www.arduino.cc/reference/pt/. Acesso em: 13 mai. 2025.

DIRETORIA DE TECNOLOGIAS E INOVAÇÃO (DTI)

COORDENAÇÃO DE TECNOLOGIAS EDUCACIONAIS (CTE)

EQUIPE ROBÓTICA PARANÁ

- Adilson Carlos Batista
- Ailton Lopes
- Andrea da Silva Castagini Padilha
- Cleiton Rosa
- Darice Alessandra Deckmann Zanardini
- Edna do Rocio Becker
- Enzo Enrico Giacomini Piolla
- Kellen Pricila dos Santos Cochinski
- Marcelo Gasparin
- Michele Serpe Fernandes
- Michelle dos Santos
- Roberto Carlos Rodrigues
- Sandra Aguera Alcova Silva
- Viviane Dziubate Pittner

Os materiais, aulas e projetos da "Robótica Paraná", foram produzidos pela Coordenação de Tecnologias Educacionais (CTE), da Diretoria de Tecnologia e Inovação (DTI), da Secretaria de Estado da Educação do Paraná (SEED), com o objetivo de subsidiar as práticas docentes com os estudantes por meio da Robótica.
Este material foi produzido para uso didático-pedagógico exclusivo em sala de aula.

DTI - DIRETORIA DE TECNOLOGIA E INOVAÇÃO

PARANÁ
GOVERNO DO ESTADO
SECRETARIA DA EDUCAÇÃO