INSTITUTO FEDERAL DO ESPÍRITO SANTO

CAMPUS SERRA

CURSO ENGENHARIA DE CONTROLE E AUTOMAÇÃO

PROF. LEONARDO AZEVEDO SCARDUA
PEDRO HENRIQUE DAMASCENA
RONNI NASCIMENTO BERMUDES

PROJETO: CARRINHO CONTROLADO POR APP VIA BLUETOOTH COM SENSORES DE COLISÃO E "ABISMO"

SERRA

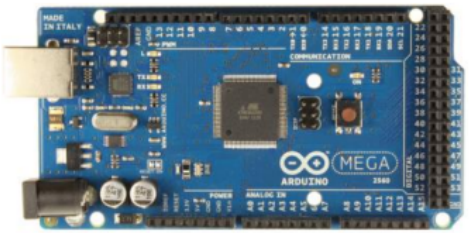
2024

1. Introdução

Este projeto envolveu a montagem e programação de um carrinho robótico que pode ser controlado por aplicativo via Bluetooth. Ele foi equipado com sensores avançados para detecção de obstáculos e bordas, além de utilizar recursos como PWM, interrupções e temporização. Utilizamos a plataforma Arduino, especificamente o Microcontrolador ARDUINO MEGA 2560, para implementar essas funcionalidades.

2. Objetivo.

Desenvolver um protótipo de carrinho robótico controlado por um aplicativo móvel via Bluetooth, utilizando a plataforma Arduino e recursos avançados como a biblioteca SoftwareSerial(). Essa biblioteca facilita a comunicação bidirecional entre o Arduino Mega 2560 e o módulo Bluetooth HC-05, essencial para receber comandos do usuário através de um joystick virtual no aplicativo móvel. O protótipo inclui sensores de detecção de obstáculos e bordas, permitindo ao Arduino Mega 2560 tomar decisões em tempo real para evitar colisões e quedas. Este projeto não apenas demonstra a aplicação prática de recursos como PWM e interrupções para controle preciso dos motores, mas também ilustra como a integração de hardware e software pode viabilizar soluções robustas em automação e robótica.



Arduino Mega 2560

3. Recursos.

Este projeto utiliza recursos avançados do Arduino, incluindo a comunicação serial, modulação por largura de pulso (PWM) e interrupções para criar um sistema de controle remoto via Bluetooth para um carrinho robótico.

Porta Serial: A comunicação serial é essencial para estabelecer uma conexão Bluetooth estável entre o Arduino e o dispositivo de controle remoto. Utilizando a biblioteca SoftwareSerial, o Arduino pode receber comandos e enviar dados de volta através da porta serial, estabelecendo uma interface bidirecional que permite o controle remoto do carrinho.

PWM (Modulação por Largura de Pulso): A modulação por largura de pulso é empregada para controlar a velocidade dos motores do carrinho. O Arduino gera sinais PWM nos pinos conectados aos controladores dos motores. Isso permite variar a velocidade dos motores ajustando o ciclo de trabalho do sinal PWM, mantendo a direção constante.

Interrupção: A interrupção é utilizada para capturar os pulsos ultrassônicos emitidos pelo sensor de distância. Através da função attachInterrupt(), o Arduino pode detectar mudanças no estado do pino conectado ao sensor ultrassônico e iniciar uma rotina de medição de distância. Essa abordagem permite uma resposta rápida às alterações no ambiente ao redor do carrinho, como a aproximação de obstáculos.

4. Materiais Necessários.

- 1 Arduino Mega 2560
- 2 Motores DC
- 1 Módulo Bluetooth HC-05
- 1 Sensor Ultrassônico HC-SR04
- 1 Sensor de Obstáculo (Infravermelho)
- 1 Protoboard e cabos de conexão
- 1 Mini Driver Motor Ponte H
- 1 Chassi para Carrinho Robô
- 4 Pilhas 1,5V
- 1 Botão Liga-Desliga - Resistores - Leds

5. Funcionamento e Lógica.

Para o funcionamento do nosso carrinho, foi necessário o uso de um app de controle remoto, no caso optamos pelo **Bluetooth Rc Car APK** (disponível para baixar no navegador). Com esse app em mãos, nós conseguimos enviar de nosso dispositivo celular, comandos que por sua vez seria enviado para o arduino via Bluetooth, uma vez enviado o comando, o arduino passa a ser o responsável pela interpretação e pela tomada de decisão do que deve ser feito com base na lógica fornecida para ele.

Após termos feito a declaração de variáveis e dos pinos, partimos para a comunicação do dispositivo com o arduino, usamos a biblioteca “SoftwareSerial” para criar uma porta serial em pinos digitais específicos no Arduinos (pinBRx e pinBTx), dentro da função “setup”, a comunicação foi inicializada e dentro do loop principal “void loop”, o código verifica se há dados disponíveis para leitura na porta serial do Bluetooth. Se houver, ele lê o comando e armazena na variável “mensagem”.

```
1   #include <SoftwareSerial.h>

29  SoftwareSerial bluetooth(pinBRx, pinBTx);  // RX, TX

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

176
177 void loop() {
178   // Captura mensagens via Bluetooth
179   if (bluetooth.available()) {
180     mensagem = bluetooth.read();
181   }
182
```

Com a comunicação realizada, partimos para a lógica que faria o nosso carrinho se movimentar. Nesse ponto nós usamos um switch case para conectar o comando recebido (armazenado na variável “mensagem”) com a ação que o carrinho deveria realizar. O código abaixo aponta o que ele deve fazer para cada comando que ele receber.

```
234  // Controle dos motores
235  switch (mensagem) {
236    case 'S':  // Parar
237      comando = 0;
238      break;
239    case 'D':  // Parar tudo
240      comando = 0;
241      break;
242    case 'F':  // Para frente
243      comando = 1;
244      break;
245    case 'B':  // Ré
246      comando = 2;
247      break;
248    case 'L':  // Esquerda
249      comando = 3;
250      break;
251    case 'R':  // Direita
252      comando = 4;
253      break;
254    case 'G':  // Para frente e esquerda
255      comando = 5;
256      break;
257    case 'I':  // Para frente e direita
258      comando = 6;
259      break;
260    case 'H':  // Ré e esquerda
261      comando = 7;
262      break;
263    case 'J':  // Ré e direita
264      comando = 8;
265      break;
266  // Controle de velocidade de "0" a "q"
267  case '0':
268    Vel = 0;
269    break;
270  case '1':
271    Vel = 1;
272    break;
273  case '2':
274    Vel = 2;
275    break;
276  case '3':
277    Vel = 3;
278    break;
279  case '4':
280    Vel = 4;
281    break;
282  case '5':
283    Vel = 5;
284    break;
285  case '6':
286    Vel = 6;
287    break;
288  case '7':
289    Vel = 7;
290    break;
291  case '8':
292    Vel = 8;
```

Esse switch case aí seria o principal (o que tem contato direto com os comandos solicitados pelo usuário), temos um switch case secundário que pega a atualização da variável “comando” feita pelo switch case principal, esse switch secundário é responsável pelo controle dos motores, ele envia um sinal para a ponte H e ela por sua vez comuta as suas portas para fazer os motores funcionar.

```
344   // Controle dos motores com base no comando
345   switch (comando) {
346     case 0:  // Parado
347       digitalWrite(pinM1A, LOW);
348       digitalWrite(pinM1B, LOW);
349       digitalWrite(pinM2A, LOW);
350       digitalWrite(pinM2B, LOW);
351       break;
352     case 1:  // Para frente
353       analogWrite(pinM1A, velocidade);
354       analogWrite(pinM1B, 0);
355       analogWrite(pinM2A, velocidade);
356       analogWrite(pinM2B, 0);
357       break;
358     case 2:  // Ré
359       analogWrite(pinM1A, 0);
360       analogWrite(pinM1B, velocidade);
361       analogWrite(pinM2A, 0);
362       analogWrite(pinM2B, velocidade);
363       break;
364     case 3:  // Esquerda
365       analogWrite(pinM1A, 0);
366       analogWrite(pinM1B, velocidade);
367       analogWrite(pinM2A, velocidade);
368       analogWrite(pinM2B, 0);
369       break;
370     case 4:  // Direita
371       analogWrite(pinM1A, velocidade);
372       analogWrite(pinM1B, 0);
373       analogWrite(pinM2A, 0);
374       analogWrite(pinM2B, velocidade);
375       break;
376     case 5:  // Para frente e esquerda
377       analogWrite(pinM1A, velocidade / 3);
378       analogWrite(pinM1B, 0);
379       analogWrite(pinM2A, velocidade);
380       analogWrite(pinM2B, 0);
381       break;
382     case 6:  // Para frente e direita
383       analogWrite(pinM1A, velocidade);
384       analogWrite(pinM1B, 0);
385       analogWrite(pinM2A, velocidade / 3);
386       analogWrite(pinM2B, 0);
387       break;
388     case 7:  // Ré e esquerda
389       analogWrite(pinM1A, 0);
390       analogWrite(pinM1B, velocidade);
391       analogWrite(pinM2A, 0);
392       analogWrite(pinM2B, velocidade / 3);
393       break;
394     case 8:  // Ré e direita
395       analogWrite(pinM1A, 0);
396       analogWrite(pinM1B, velocidade / 3);
397       analogWrite(pinM2A, 0);
398       analogWrite(pinM2B, velocidade);
399       break;
400   }
```

Nosso projeto ainda conta com controle de velocidade, usando o recurso de pinos PWM e a função analogWrite conseguimos fazer isso. No aplicativo usado temos uma função que permite fazer isso, é como se fosse o acelerador do carrinho, o que a gente precisou fazer foi basicamente dividir o intervalo fornecido em faixas (de 0 à 10) que seria enviado para o nosso switch case principal e assim criar uma fórmula para ajustar a velocidade do carrinho, essa velocidade é usada no switch case secundário, que ativa os motores e ainda diz a velocidade que eles devem se movimentar. Admitindo a velocidade mínima (VelMin) como 100 para o começar a se deslocar, a fórmula ficou da seguinte forma.

```
319     // Cálculo da velocidade
320     velocidade = velMin + (((255 - velMin) / 10) * Vel);
```

O carrinho possui 2 sensores, o primeiro que temos é o ultrassônico (para medir distância), ele é encarregado de ativar a parte de reduzir a velocidade ou parar quando um objeto é detectado, o ato de redução de velocidade ou de frenagem depende da distância do carrinho até o objeto . Para a implementação do ultrassônico foi necessário o uso de interrupção, a interrupção foi necessária para medir a distância usando o sensor ultrassônico de forma precisa e eficiente. A interrupção permite que o microcontrolador responda imediatamente a eventos importantes (neste caso, a mudança no estado do pino de eco do sensor ultrassônico) sem a necessidade de ficar constantemente verificando o estado desse pino dentro do loop principal do programa. Usando o **attachinterrupt**, conseguimos fazer a leitura desse sinal e para reutilizar dentro da função que calcula a distância. Com o cálculo da distância foi possível implementar o controle da frequência de um buzzer, quanto mais próximo maior a frequência dele.

```
108     // CONFIGURA A INTERRUPÇÃO PARA SENSOR DE DISTANCIA
109     attachInterrupt(digitalPinToInterrupt(pinEcoD), Distancia, CHANGE);
110   }
111
```

```
167   void Eco() {
168     // ENVIA O SINAL PARA O MÓDULO INICIAR O FUNCIONAMENTO
169     digitalWrite(pinTriggerD, HIGH);
170     // AGUARDAR 10 uS PARA GARANTIR QUE O MÓDULO VAI INICIAR O ENVIO
171     delayMicroseconds(10);
172     // DESLIGA A PORTA PARA FICAR PRONTO PARA PROXIMA MEDIÇÃO
173     digitalWrite(pinTriggerD, LOW);
174     // INDICA O MODO DE FUNCIONAMENTO (AGUARDAR PULSO)
175     modo = 0;
176   }
```

```
153   void Distancia() {
154     switch (modo) {
155       case 0:
156         eco0 = micros();
157         modo = 1;
158         break;
159       case 1:
160         distancia = (float)(micros() - eco0) / 58.3;  // distancia em CM
161         eco0 = 0;
162         modo = -1;
163         break;
164     }
165   }
166
```

O segundo sensor que temos é o sensor infravermelho, no nosso código colocamos ele para operar como detector de borda/abismo, caso ele encontre alguma descontinuidade no plano, ele trava o movimento e impede o usuário de continuar seguindo naquela direção.ao mesmo tempo ele é responsável por ativar um buzzer e um led, como formas alternativas de alerta para o usuário.

```
136  // Controla o buzzer baseado na distância medida
137  void controlaBuzzerBorda() {
138    unsigned long currentMillisBorda = millis();
139    int intervalo = 100;  // Limita o intervalo mínimo para 100ms
140    if (currentMillisBorda - previousMillisBorda >= intervalo) {
141      previousMillisBorda = currentMillisBorda;
142      buzzerState = !buzzerState;
143      if (buzzerState) {
144        tone(pinBuzzer, 1000);
145      } else {
146        noTone(pinBuzzer);  // Desliga o buzzer temporariamente
147      }
148    } else {
149      noTone(pinBuzzer);  // Desliga o buzzer se a distância for maior que 10 cm
150    }
151  }
```

```
328        if (SensorBordaD) {
329        digitalWrite(alertasensor1, HIGH);
330        controlaBuzzerBorda();
331        } else {
332        digitalWrite(alertasensor1, LOW);
333      }
```

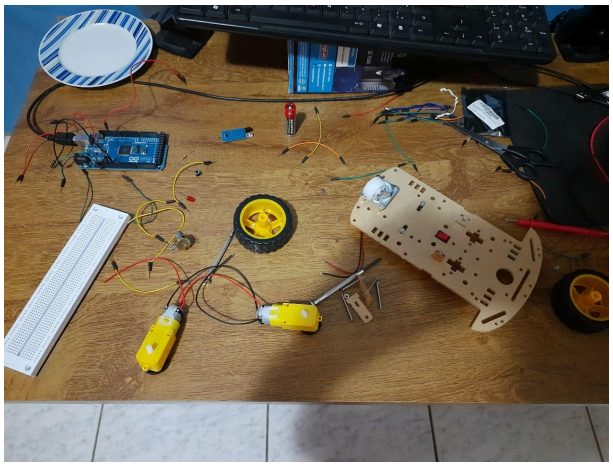
Como mencionado anteriormente usamos leds como ferramenta de alerta ou sinalização. No modo de sinalização, ele opera quando o carrinho executa algum movimento para o lado (Esquerdo, Direito, Frente Esquerda e Frente Direita), os leds funcionam como esquema de seta, que piscam por um determinado tempo e frequência e quando o comando cessa ele para. Como isso exige muita precisão usamos a função **millis()** para estar fazendo essa temporização sem correr o risco de bloquear o nosso código, a mesma função também foi usada para controlar a temporização do buzzer que serve de alerta para distância crítica. Código completo encontra- se em:

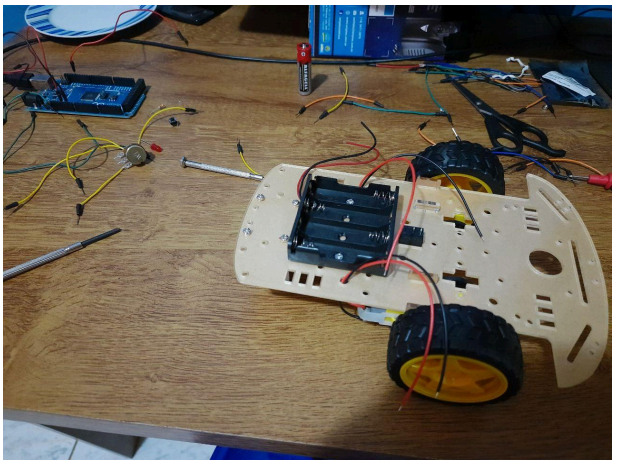
Código do Projeto:

GitHub: https://github.com/ronnibermudes/carrinho.

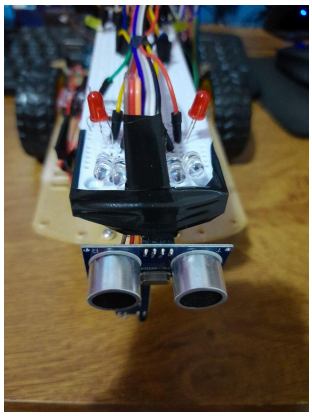
6. Resultado e Conclusão.

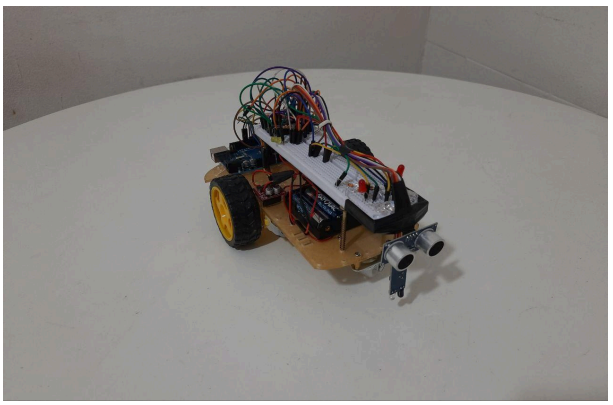
6.1- Montagem.





6.2- Finalizado





Este projeto não apenas mostra como é possível automatizar processos usando o Arduino, mas também demonstra como tecnologias como o controle de velocidade por PWM e o uso de interrupções podem ser combinadas para criar sistemas robustos e eficientes. Essa integração permite desenvolver soluções de automação que são versáteis e capazes de se adaptar às necessidades específicas, como controle remoto e detecção de obstáculos em tempo real.

Referências:

Arduino: https://www.arduino.cc/reference/pt/
Referência e Crédito: Prof. Flávio Guimarães
Especialista em Programação e robótica
YouTube: https://www.youtube.com/watch?v=EpOzvxkM-cQ