Relatório de Sistemas Embarcados

Bruno H. Antunes, Esdras F.Cruz, Leonardo F. Pacuola

25 de agosto de 2024

1 Objetivo da Atividade

O trabalho da disciplina de Sistemas Embarcados consiste na criação de um projeto que atenda a alguns requisitos mínimos. O projeto deve incluir a criação de uma PCB (Placa de Circuito Impresso), a utilização de pelo menos dois periféricos, uma interação homem-máquina e a implementação de uma interrupção.

2 Introdução

Para o projeto, desenvolveu-se um carrinho que opera por controle Bluetooth através de um aparelho celular. Esse sistema incorpora tecnologia de sistemas embarcados, utilizando um microcontrolador ESP-32.

2.1 Descrição Geral

Foram utilizados diversos componentes eletrônicos, incluindo uma Ponte H Dupla L298N para controlar o sentido do carrinho e um servo motor para o controle da direção. Um motor DC foi responsável pela movimentação. O microcontrolador escolhido foi o ESP-32, que possui um módulo Bluetooth em sua estrutura, sendo ideal para este caso. Adicionalmente, um buzzer ativo foi incorporado para funcionar como buzina e um sonar para a medição de distância. Resistores e capacitores foram utilizados para estabilizar o circuito e minimizar ruídos. A estrutura do carrinho foi composta pela base e suas rodinhas. Para a PCB, foi utilizada uma placa universal de tamanho 7x5. Para controlar o carrinho foi utilizado um aplicativo de celular, cujo o nome é "Arduino Bluetooh Control", disponível para instalação na Play Store.

3 Montagem

3.1 Montagem Física

A montagem consistiu na utilização de uma placa universal, onde foram colocados o ESP-32, o capacitor, os resistores e o buzzer. O motor foi instalado dentro de um compartimento traseiro da própria estrutura do carrinho (Figura 2), enquanto o servo motor foi posicionado na parte frontal, acoplado a uma estrutura de direção desenvolvida manualmente. Todos os componentes foram fixados no carrinho com cola ou fita.

3.2 Montagem do Código

Para controlar os servomotores e a comunicação via Bluetooth, foram incluídas bibliotecas específicas e configurados os pinos dos sensores e atuadores do carrinho. Na função de configuração inicial (‘setup‘), a comunicação serial e Bluetooth foram iniciadas, e as tarefas necessárias foram criadas. O hardware foi configurado definindo os pinos como entrada ou saída e inicializando o servo na posição neutra (90 graus).

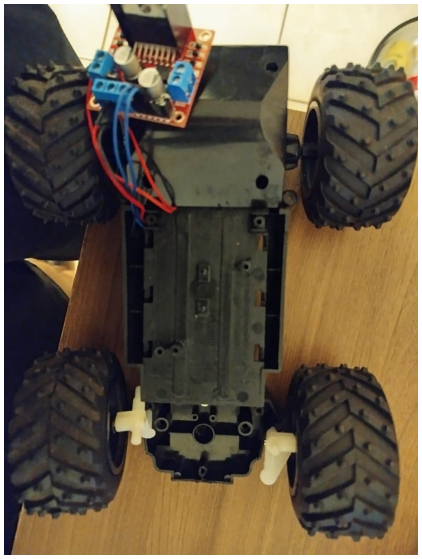

Figura 1: Estrutura do carrinho.

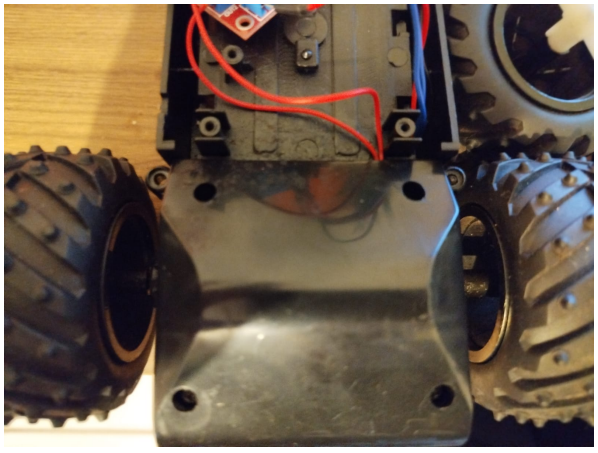

Figura 2: Compartimento do carrinho.

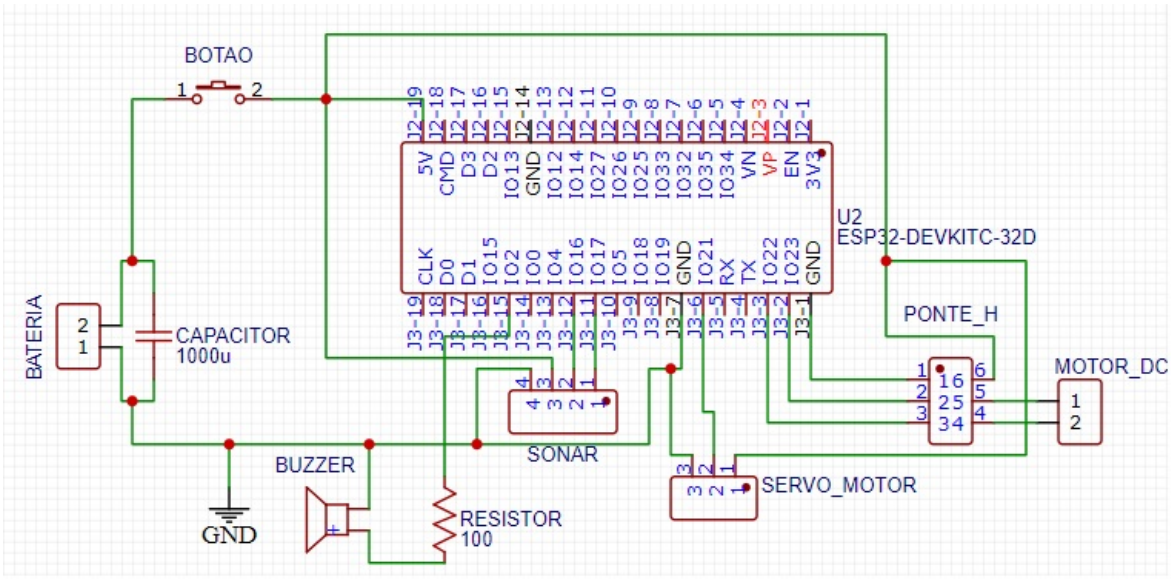

Figura 3: Esquema do circuito.

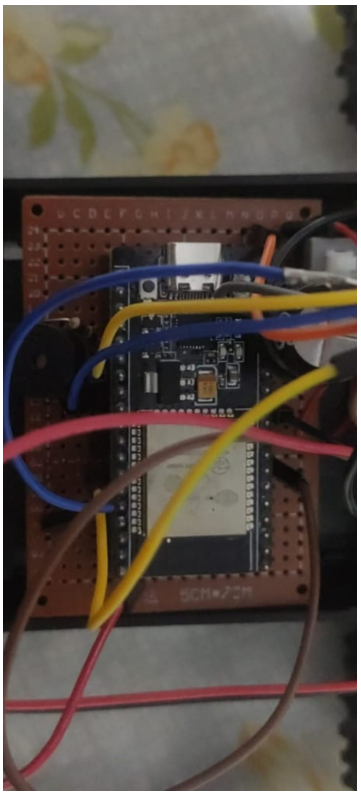

Figura 4: PCB com os componentes.

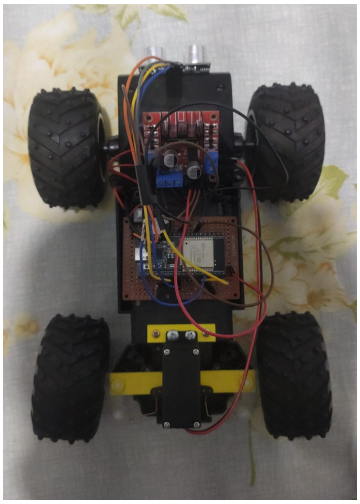

Figura 5: Carrinho completo.

4 Implementação do Software

4.1 Controle de Direção

A primeira tarefa ajusta a direção do carrinho utilizando um servomotor baseado nos comandos recebidos via Bluetooth. A posição do servomotor é ajustada gradualmente para a esquerda ou direita, permitindo que o carrinho mude de direção suavemente.

4.2 Controle de Movimento

A segunda tarefa controla o sentido de movimento do carrinho, permitindo que ele avance ou recue. Este controle é realizado através de uma ponte H, que altera a polaridade da corrente fornecida ao motor, possibilitando a mudança de sentido do movimento.

4.3 Controle de Distância

A terceira tarefa mede a distância de objetos à frente do carrinho usando um sensor ultrassônico. Se um objeto é detectado a uma distância inferior a um limite pré-estabelecido, a tarefa aciona a buzina do carrinho como sinal de alerta. Essa funcionalidade é crucial para evitar colisões e garantir a segurança do sistema.

4.4 Controle da Buzina

A quarta tarefa é responsável por ligar e desligar a buzina com base nas instruções recebidas da tarefa de controle de distância. Isso permite uma sinalização sonora quando um obstáculo é detectado próximo ao carrinho, alertando sobre a proximidade de objetos.

4.5 Recepção de Comandos Bluetooth

A quinta tarefa lê os comandos enviados via Bluetooth, atualizando as variáveis globais que controlam a direção e o movimento do carrinho. Esta tarefa utiliza um semáforo para garantir que a leitura dos comandos seja feita de forma segura e sem interferências, proporcionando um controle remoto eficiente e responsivo.

5 Problemas no Desenvolvimento

5.1 Problemas Físicos

Durante a montagem do carrinho, diversos desafios foram enfrentados. O desenvolvimento de um mecanismo eficaz de direção, acoplado ao servo motor, mostrou-se particularmente complexo. Além disso, ocorreram problemas comuns, como má conexão, conexões erradas e portas sem a funcionalidade desejada. Além desses, surgiram problemas com o servo motor, que, por motivos desconhecidos, girava sem parar, o que acarretou danos no seu giro, além de problemas na estrutura de direção.

5.2 Problemas de Software

Inicialmente, o projeto foi planejado para ser sequencial. Entretanto, observou-se que, dessa forma, a execução dos comandos demoraria muito. Outros problemas incluíram a sincronização correta das tarefas gerenciadas pelo FreeRTOS e a implementação eficiente dos comandos Bluetooth. Houve dificuldades na comunicação entre o ESP-32 e o Bluetooth, além de problemas de falta de memória devido ao tamanho das pilhas das tarefas.

6 Soluções Implementadas

6.1 Soluções para Problemas Físicos

O problema de direção foi resolvido por tentativa e erro. A questão das conexões foi solucionada com testes em outras portas e com o uso de aparelhos como o multímetro, utilizado para identificar problemas. Em vez de fazer a movimentação direta do servo no ESP32, foi utilizado outras portas da ponte H para controlá-lo, similarmente ao que foi feito com o motor DC. Assim, foi possível realizar o giro do servo; para isso, o servo foi aberto e conectadas as portas de saída da ponte H.

6.2 Soluções para Problemas de Software

O problema de software foi solucionado utilizando tarefas com o uso do FreeRTOS. Com essa estrutura, foi obtido um bom desempenho e um ótimo tempo de resposta, com o atraso sendo quase imperceptível. Quanto ao problema de memória, a solução foi aumentar o tamanho das pilhas das tarefas, e para o sincronismo foram utilizados semáforos.

7 Código Fonte

```
#include "BluetoothSerial.h"
#include "ESP32Servo.h"
//Servo myservo;

#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
  #error Bluetooth is not enabled! Please run `make menuconfig` to enable it
#endif

BluetoothSerial SerialBT;

#define echoPin 17
#define trigPin 16
#define buzzer 2
#define controle_de_sentido1 22
#define controle_de_sentido2 23
#define controle_de_direcao1 27
#define controle_de_direcao2 32
#define enB 21

int distance;
unsigned long duration;
```

```
22 int teste;
23 SemaphoreHandle_t xSerialSemaphore;
24
25 char lastDirectionCommand = 'Q'; // Stop
26 char lastMovementCommand = 'Q'; // Stop
27 char lastBuzzerCommand = 'Q'; // Stop
28
29 void vTarefa1(void *pvParameters); // controle de dire  o
30 void vTarefa2(void *pvParameters); // controle de sentido
31 void vTarefa3(void *pvParameters); // controle de distancia
32 void vTarefa4(void *pvParameters); // controle de buzina
33 void vTarefaBluetooth(void *pvParameters); // Tarefa para ler comandos
       Bluetooth
34
35 void setupHardware();
36 void createTasks();
37
38 void setup() {
39   Serial.begin(115200);
40   SerialBT.begin("RedesESP32"); // Nome do dispositivo Bluetooth
41
42   setupHardware();
43   createTasks();
44
45   //Serial.println("Setup completed");
46 }
47
48 void loop() {
49   // O loop principal fica vazio, as tarefas s o gerenciadas pelo FreeRTOS
50 }
51
52 void setupHardware() {
53   pinMode(controle_de_sentido1, OUTPUT);
54   pinMode(controle_de_sentido2, OUTPUT);
55   pinMode(controle_de_direcao1, OUTPUT);
56   pinMode(controle_de_direcao2, OUTPUT);
57   pinMode(echoPin, INPUT);
58   pinMode(trigPin, OUTPUT);
59   pinMode(buzzer, OUTPUT);
60   pinMode(enB, OUTPUT);
61   analogWrite(enB, 90);
62
63   xSerialSemaphore = xSemaphoreCreateMutex();
64   if (xSerialSemaphore == NULL) {
65     //Serial.println("Erro ao criar o sem foro");
66     while (1);
67   }
68 }
69
70 void createTasks() {
71   BaseType_t result;
72
73   result = xTaskCreate(vTarefa1, "Tarefa1", 1024, NULL, 1, NULL);
74   if (result != pdPASS) {
75     //Serial.println("Erro ao criar Tarefa1");
76     while (1);
77   }
78
79   result = xTaskCreate(vTarefa2, "Tarefa2", 1024, NULL, 1, NULL);
80   if (result != pdPASS) {
81     //Serial.println("Erro ao criar Tarefa2");
82     while (1);
```

```
  }

  result = xTaskCreate(vTarefa3, "Tarefa3", 1024, NULL, 1, NULL);
  if (result != pdPASS) {
    //Serial.println("Erro ao criar Tarefa3");
    while (1);
  }

  result = xTaskCreate(vTarefa4, "Tarefa4", 1024, NULL, 1, NULL);
  if (result != pdPASS) {

    while (1);
  }

  result = xTaskCreate(vTarefaBluetooth, "TarefaBluetooth", 1024, NULL, 1,
      NULL);
  if (result != pdPASS) {

    while (1);
  }
}

//tarefa do servo motor direcao
void vTarefa1(void *pvParameters) {
  while (1) {
   if (lastDirectionCommand == 'L'){
      digitalWrite(controle_de_direcao1, HIGH);
      digitalWrite(controle_de_direcao2, LOW);

    }
    else if (lastDirectionCommand == 'R') {
      digitalWrite(controle_de_direcao1, LOW);
      digitalWrite(controle_de_direcao2, HIGH);
    }
    else{
      digitalWrite(controle_de_direcao1, HIGH);
      digitalWrite(controle_de_direcao2, HIGH);

    }

    vTaskDelay(10 / portTICK_PERIOD_MS);
  }
}

//tarefa para sentido ponte H
void vTarefa2(void *pvParameters) {
  while (1) {
    if (lastMovementCommand == 'S'){
      digitalWrite(controle_de_sentido1, HIGH);
      digitalWrite(controle_de_sentido2, LOW);
    }
    else if (lastMovementCommand == 'X') {
      digitalWrite(controle_de_sentido1, LOW);
      digitalWrite(controle_de_sentido2, HIGH);
    }
    else{
      digitalWrite(controle_de_sentido1, HIGH);
      digitalWrite(controle_de_sentido2, HIGH);
    }

    vTaskDelay(100 / portTICK_PERIOD_MS);
  }
```

```
144 }
145 //tarefa para o sonar:
146
147 void vTarefa3(void *pvParameters) {
148   while (1) {
149     digitalWrite(trigPin, LOW);
150     vTaskDelay(2 / portTICK_PERIOD_MS);
151     digitalWrite(trigPin, HIGH);
152     vTaskDelay(10 / portTICK_PERIOD_MS);
153     digitalWrite(trigPin, LOW);
154
155     duration = pulseIn(echoPin, HIGH);
156     distance = duration / 58.2;
157     teste = distance;
158    // Serial.println(distance);
159
160     if (distance <= 10 && distance > 0) {
161       lastBuzzerCommand = 'H'; // Aciona o buzzer
162       //digitalWrite(led, HIGH);
163
164     }
165     else {
166       lastBuzzerCommand = 'M'; // Desliga o buzzer
167       //digitalWrite(led, LOW);
168     }
169     vTaskDelay(50 / portTICK_PERIOD_MS);
170   }
171 }
172
173 //tarefa para condi  o de parametros do sonar e ver se o buzzer liga
174 void vTarefa4(void *pvParameters) {
175   while (1) {
176     if (lastBuzzerCommand == 'H') {
177       digitalWrite(buzzer, HIGH);
178     }
179     else {
180       digitalWrite(buzzer, LOW);
181     }
182     vTaskDelay(50 / portTICK_PERIOD_MS);
183   }
184 }
185
186 void vTarefaBluetooth(void *pvParameters) {
187   while (1) {
188     char recebido;
189     if (SerialBT.available()) {
190       if (xSemaphoreTake(xSerialSemaphore, (TickType_t) 10) == pdTRUE) {
191         recebido = SerialBT.read();
192
193
194         if (recebido == 'R' || recebido == 'L') {
195           lastDirectionCommand = recebido; // Atualiza o  ltimo  comando de
                dire  o
196         }
197
198         else if (recebido == 'S' || recebido == 'X') {
199
200           lastMovementCommand = recebido; // Atualiza o  ltimo  comando de
                movimento
201         }
202         else{
203           lastMovementCommand = recebido;
```

```
            lastDirectionCommand = recebido;
          }

          xSemaphoreGive(xSerialSemaphore);
        }
      }
      vTaskDelay(10 / portTICK_PERIOD_MS);
    }
}
```

8 Conclusão

O desenvolvimento do carrinho controlado por Bluetooth utilizando o microcontrolador ESP-32 proporcionou uma experiência abrangente e desafiadora no campo de Sistemas Embarcados. A montagem física, envolvendo a questão de solda em uma PCB e a integração de diversos periféricos, juntamente com a implementação de tarefas gerenciadas pelo FreeRTOS, demonstrou a complexidade e a necessidade de diversos testes para garantir as funcionalidades exigidas.

Apesar dos desafios enfrentados na sincronização das tarefas e na montagem dos componentes, o uso de semáforos e a configuração adequada do Bluetooth para assegurar a estabilidade e eficiência do projeto, este trabalho não apenas cumpriu os requisitos mínimos da disciplina, mas também proporcionou uma compreensão dos aspectos práticos e teóricos necessários para o desenvolvimento de sistemas, evidenciando a importância da integração entre hardware e software, sendo de grande valia para nossa vida profissional como futuros engenheiros de computação

9 Referências Bibliográficas

TANNUS, Alexandre. Arduino: Servo Motores, 2018.

VALVANO, Jonathan.Embedded Systems: Real-Time Interfacing to Arm® Cortex™-M Microcontrollers. 2ª ed. Createspace, 2011.

MIELI, Alexandre; SANTOS, Raul; MIRANDA, Brenda; SANTOS, Davi; GAUTHIER, André; ROCHA, Pablo; SILVA, Pâmela. Controle de Veículo por Aplicativo.2ª ed. 2023.

BABIUCH, Marek ; FOLTYNEK, Petr ;SMUTNY, Pavel. Usando o microcontrolador ESP32 para processamento de dados. 2019.