UNIVERSIDADE DO VALE DO ITAJAÍ, CAMPUS ITAJAÍ
ENGENHARIA DE COMPUTAÇÃO

LARISSA DE SOUSA GOUVEA

THIAGO YUKIO HORITA PACHECO

Modelagem STR - Projeto 2

ITAJAÍ
2024

```c
// Bibliotecas necessárias para executar programa
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/timers.h"
#include "driver/gpio.h"
#include "esp_timer.h"

// Definir os pinos dos sensores
#define SENSOR_INJECAO_PIN 32  // GPIO para sensor de injeção eletrônica
#define SENSOR_TEMPERATURA_PIN 33  // GPIO para sensor de temperatura do motor
#define SENSOR_ABS_PIN 34  // GPIO para sensor de ABS
#define SENSOR_AIRBAG_PIN 35  // GPIO para sensor de airbag
#define SENSOR_CINTO_PIN 36  // GPIO para sensor de cinto de segurança
#define SENSOR_VELOCIDADE_PIN 37 // Exemplo de GPIO para sensor de velocidade
#define SENSOR_CONSUMO_PIN 38 // Exemplo de GPIO para sensor de consumo

// Definir os tempos de resposta em milissegundos
#define TEMPO_INJECAO_MS 15 // 500 us = 0.5 ms
#define TEMPO_TEMPERATURA_MS 20 // 20 ms
#define TEMPO_ABS_MS 100 // 100 ms
#define TEMPO_AIRBAG_MS 100 // 100 ms
#define TEMPO_CINTO_MS 1000 // 1 segundo = 1000 ms
#define TEMPO_VELOCIDADE_MS 100 // Intervalo de amostragem para velocidade
#define TEMPO_CONSUMO_MS 100 // Intervalo de amostragem para consumo

#define AMOSTRAS 200 // Número de amostras para cálculo da média

static bool motor_ativo = false;
static bool frenagem_ativo = false;
static bool vida_ativa = false;

static float amostras_velocidade[AMOSTRAS];
static float amostras_consumo[AMOSTRAS];
static int contagem_velocidade = 0;
```

```c
static int contagem_consumo = 0;
static float velocidade_media = 0;
static float consumo_media = 0;

// Configuração dos sensores
void configurar_sensores() {
    // Configurar os pinos dos sensores como entradas
    esp_rom_gpio_pad_select_gpio(SENSOR_INJECAO_PIN);
    gpio_set_direction(SENSOR_INJECAO_PIN, GPIO_MODE_INPUT);

    esp_rom_gpio_pad_select_gpio(SENSOR_TEMPERATURA_PIN);
    gpio_set_direction(SENSOR_TEMPERATURA_PIN, GPIO_MODE_INPUT);

    esp_rom_gpio_pad_select_gpio(SENSOR_ABS_PIN);
    gpio_set_direction(SENSOR_ABS_PIN, GPIO_MODE_INPUT);

    esp_rom_gpio_pad_select_gpio(SENSOR_AIRBAG_PIN);
    gpio_set_direction(SENSOR_AIRBAG_PIN, GPIO_MODE_INPUT);

    esp_rom_gpio_pad_select_gpio(SENSOR_CINTO_PIN);
    gpio_set_direction(SENSOR_CINTO_PIN, GPIO_MODE_INPUT);
}


void monitoramento_injecao(void *pvParameter) {
    while (1) {
        if (gpio_get_level(SENSOR_INJECAO_PIN)) {

            int64_t start_time = esp_timer_get_time();

            motor_ativo = true;
            printf("\033[32mInjeção eletrônica acionada!\033[0m\n");

            int64_t end_time = esp_timer_get_time();  // Captura o
tempo após a ação
            printf("\033[32mTempo da Injeção Eletrônica: %lld
µs\033[0m\n", end_time - start_time);

            // Aguarda por 500 µs
            // while ((esp_timer_get_time() - start_time) < 500);
        }
```

```
        vTaskDelay(pdMS_TO_TICKS(TEMPO_INJECAO_MS)); // Atraso para
simulação
        // vTaskDelay(pdUS_TO_TICKS(TEMPO_INJECAO_MS));
    }
}

// Função para monitorar o sensor de temperatura do motor
void monitoramento_temperatura(void *pvParameter) {
    while (1) {
        if (gpio_get_level(SENSOR_TEMPERATURA_PIN)) {
            int64_t start_time = esp_timer_get_time();
            motor_ativo = true;
            printf("\033[31mTemperatura do motor acima do
limite!\033[0m\n");
            // printf("Temperatura do motor acima do limite!\n");
            int64_t end_time = esp_timer_get_time();  // Captura o
tempo de fim
            printf("\033[31mTempo da Temperatura: %lld µs\033[0m\n",
end_time - start_time);
        }
        vTaskDelay(pdMS_TO_TICKS(TEMPO_TEMPERATURA_MS)); // Atraso de
acordo com o deadline da temperatura
    }
}

// Função para monitorar o sensor de ABS
void monitoramento_abs(void *pvParameter) {
    while (1) {
        if (gpio_get_level(SENSOR_ABS_PIN)) {
            int64_t start_time = esp_timer_get_time();
            frenagem_ativo = true;
            // printf("ABS acionado!\n");
            printf("\033[34mABS acionado!\033[0m\n");
            int64_t end_time = esp_timer_get_time();  // Captura o
tempo de fim
            printf("\033[34mTempo da ABS: %lld µs\033[0m\n", end_time
- start_time);
        }
        vTaskDelay(pdMS_TO_TICKS(TEMPO_ABS_MS)); // Atraso de acordo
com o deadline do ABS
    }
}
```

```c
// Função para monitorar o sensor de airbag
void monitoramento_airbag(void *pvParameter) {
    while (1) {
        if (gpio_get_level(SENSOR_AIRBAG_PIN)) {
            int64_t start_time = esp_timer_get_time();
            vida_ativa = true;
            // printf("Airbag acionado!\n");
            printf("\033[35mAirbag acionado!\033[0m\n");
            int64_t end_time = esp_timer_get_time();  // Captura o
tempo de fim
            printf("\033[35mTempo da Airbag: %lld µs\033[0m\n",
end_time - start_time);
        }
        vTaskDelay(pdMS_TO_TICKS(TEMPO_AIRBAG_MS)); // Atraso de
acordo com o deadline do airbag
    }
}

// Função para monitorar o sensor de cinto de segurança
void monitoramento_cinto(void *pvParameter) {
    while (1) {
        if (gpio_get_level(SENSOR_CINTO_PIN)) {
            int64_t start_time = esp_timer_get_time();
            vida_ativa = true;
            // printf("Cinto de segurança acionado!\n");
            printf("\033[36mCinto de segurança acionado!\033[0m\n");
            int64_t end_time = esp_timer_get_time();  // Captura o
tempo de fim
            printf("\033[36mTempo da cinto: %lld µs\033[0m\n",
end_time - start_time);
        }
        vTaskDelay(pdMS_TO_TICKS(TEMPO_CINTO_MS)); // Atraso de
acordo com o deadline do cinto de segurança
    }
}

void monitoramento_velocidade(void *pvParameter) {
    while (1) {
        if (contagem_velocidade < AMOSTRAS) {
            // Simular leitura de velocidade do sensor
            float velocidade = (float)(rand() % 100); // Exemplo de
```

```
velocidade aleatória
            amostras_velocidade[contagem_velocidade++] = velocidade;
        } else {
            // Calcular a média quando atingir 200 amostras
            float soma = 0;
            for (int i = 0; i < AMOSTRAS; i++) {
                soma += amostras_velocidade[i];
            }
            velocidade_media = soma / AMOSTRAS;
            contagem_velocidade = 0; // Resetar contagem
        }
        vTaskDelay(pdMS_TO_TICKS(TEMPO_VELOCIDADE_MS));
    }
}


void monitoramento_consumo(void *pvParameter) {
    while (1) {
        if (contagem_consumo < AMOSTRAS) {
            // Simular leitura de consumo do sensor
            float consumo = (float)(rand() % 15); // Exemplo de
consumo aleatório
            amostras_consumo[contagem_consumo++] = consumo;
        } else {
            // Calcular a média quando atingir 200 amostras
            float soma = 0;
            for (int i = 0; i < AMOSTRAS; i++) {
                soma += amostras_consumo[i];
            }
            consumo_media = soma / AMOSTRAS;
            contagem_consumo = 0; // Resetar contagem
        }
        vTaskDelay(pdMS_TO_TICKS(TEMPO_CONSUMO_MS));
    }
}

// Função para atualizar o estado dos subsistemas
void atualizar_display(void *pvParameter) {
    while (1) {
        printf("Estado dos subsistemas:\n");
        printf("Motor: %s\n", motor_ativo ? "Ativo" : "Inativo");
        printf("Frenagem: %s\n", frenagem_ativo ? "Ativo" :
```

```
"Inativo");
        printf("Vida: %s\n", vida_ativa ? "Ativo" : "Inativo");
        printf("Velocidade média: %.2f km/h\n", velocidade_media);
        printf("Consumo médio: %.2f L/100km\n", consumo_media);

        // Reseta o estado dos subsistemas para o próximo ciclo
        motor_ativo = false;
        frenagem_ativo = false;
        vida_ativa = false;

        vTaskDelay(pdMS_TO_TICKS(1000));  // Atualiza a cada 1
segundo
    }
}

// Função principal
void app_main() {

    // Configuração dos sensores
    configurar_sensores();

    // Criação das tarefas de monitoramento dos sensores com
prioridades baseadas nos deadlines
    xTaskCreate(monitoramento_injecao, "monitoramento_injecao", 2048,
NULL, 6, NULL); // Alta prioridade
    xTaskCreate(monitoramento_temperatura,
"monitoramento_temperatura", 2048, NULL, 5, NULL); // Prioridade
média-alta
    xTaskCreate(monitoramento_abs, "monitoramento_abs", 2048, NULL,
3, NULL); // Prioridade média
    xTaskCreate(monitoramento_airbag, "monitoramento_airbag", 2048,
NULL, 4, NULL); // Prioridade média-alta
    xTaskCreate(monitoramento_cinto, "monitoramento_cinto", 2048,
NULL, 2, NULL); // Prioridade baixa
    xTaskCreate(atualizar_display, "atualizar_display", 2048, NULL,
1, NULL); // Prioridade mais baixa
    xTaskCreate(monitoramento_velocidade, "monitoramento_velocidade",
2048, NULL, 2, NULL); // Prioridade baixa
    xTaskCreate(monitoramento_consumo, "monitoramento_consumo", 2048,
NULL, 2, NULL); // Prioridade baixa
}
```

4)

Tarefas com Requisitos Temporais Hard

Monitoramento de Injeção Eletrônica: Com um tempo máximo de ação de 100 ms 500 µs A injeção eletrônica é crítica para o funcionamento do motor. Se não for detectada em tempo hábil, pode haver perda de potência ou falhas mecânicas graves.

Monitoramento de Temperatura do Motor: Com um tempo máximo de ação de 20 ms, o superaquecimento do motor pode causar danos irreparáveis. A detecção rápida é necessária para ativar medidas corretivas (como desligar o motor).

Monitoramento de ABS: Com um tempo máximo de ação de 100 ms, o sistema de freios precisa funcionar em situações de emergência. A resposta rápida é fundamental para garantir a segurança do motorista e prevenir acidentes de trânsito.

Monitoramento de Airbag: Com um tempo máximo de ação de 100 ms, a ativação do airbag deve ser imediata em caso de colisão. Qualquer atraso pode resultar em ferimentos graves para os ocupantes do veículo.

Tarefas com Requisitos Temporais Soft

Monitoramento de Cinto de Segurança: Com um tempo máximo de ação de 1000 ms, embora importante para a segurança, a detecção do estado do cinto pode tolerar atrasos, pois não interfere diretamente em uma situação de emergência. O sistema pode alertar o motorista com um pequeno atraso.

Tarefa de Monitoramento de Velocidade Média: Com um tempo máximo de ação de 1000 ms Embora seja importante, uma perda de atualização ocasional não resulta em um perigo imediato. A velocidade deve ser atualizada, mas não necessariamente dentro de um prazo rigoroso.

Tarefa de Monitoramento do Consumo Médio de Gasolina: Com um tempo máximo de ação de 1000 ms**,** o consumo médio de gasolina, não é tão crítico em termos de segurança imediata caso esta tarefa atrasar ou não for executada exatamente dentro do tempo esperado, embora importante para a eficiência do veículo.

Atualização do Display: Com um tempo máximo de ação de 1000 ms, a atualização do estado dos subsistemas é útil, mas não crítica. Um atraso aqui não afeta a operação do veículo e serve mais para informação do usuário.

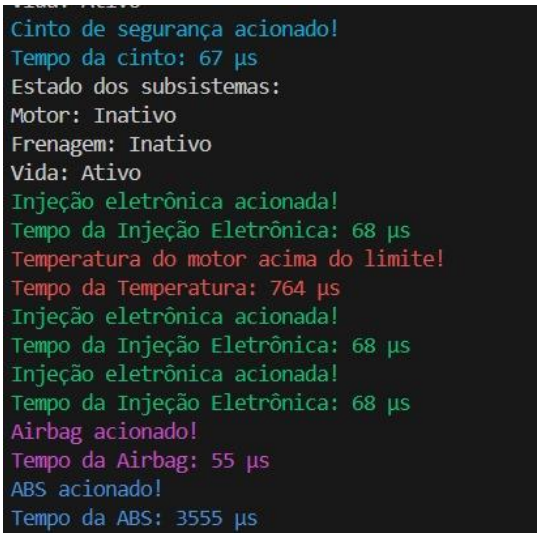
Operações sem Requisito Temporal

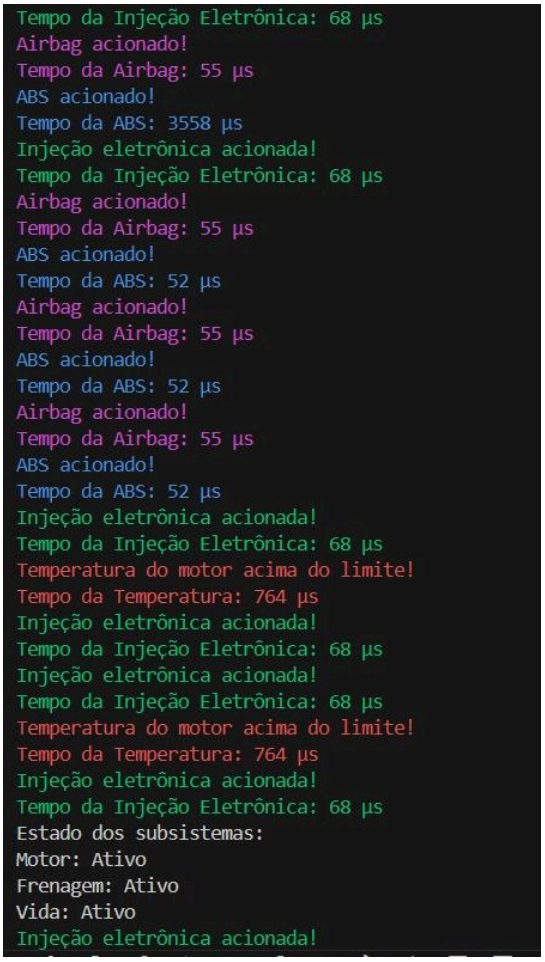
Configuração dos Sensores: A configuração dos sensores é realizada uma vez na inicialização do sistema. Não há necessidade de resposta em tempo real, pois a configuração não afeta diretamente a operação contínua.

Leitura de Estado de Subsistemas: As leituras e registros de estados podem ser feitas de maneira assíncrona, e o tempo em que essas informações são processadas não impacta o funcionamento imediato do sistema.

Reseta de Estado dos Subsistemas: A atualização do estado dos subsistemas (motor, frenagem, vida) não exige um tempo específico, pois essas variáveis são resetadas periodicamente sem consequências críticas.

5)





É possível verificar nas imagens que os requisitos de tempo estabelecidos para o sistema de monitoramento do comportamento dos subsistemas do veículo estão sendo cumpridos. A análise das imagens demonstra que cada sensor e atuador está operando dentro dos prazos definidos, assegurando a eficácia do monitoramento.

6)
Versão Executivo Cíclico

```c
#include <stdio.h>
#include <unistd.h> // para usar sleep

// Definir tempos de ciclo em milissegundos
#define PERIODO_DISPLAY 1000        // Atualização do display a cada 1 segundo
#define PERIODO_CINTO_SEGURANCA 1000 // Cinto de segurança a cada 1 segundo
#define PERIODO_ABS 100             // ABS a cada 100 ms
#define PERIODO_AIRBAG 100          // Airbag a cada 100 ms
#define PERIODO_TEMPERATURA 20      // Temperatura do motor a cada 20 ms
#define PERIODO_INJECAO 0.5         // Injeção eletrônica a cada 0.5 ms

// Funções simulando cada subsistema
void atualiza_injecao_eletronica() {
    printf("Atualizando injeção eletrônica...\n");
}

void monitora_temperatura_motor() {
    printf("Monitorando temperatura do motor...\n");
}

void monitora_abs() {
    printf("Monitorando ABS...\n");
}

void monitora_airbag() {
    printf("Monitorando airbag...\n");
}

void monitora_cinto_seguranca() {
    printf("Monitorando cinto de segurança...\n");
}

void atualiza_display() {
    printf("Atualizando display...\n");
}
```

```c
// Função para simular espera em milissegundos
void wait_ms(int ms) {
    usleep(ms * 1000); // Usar usleep para milissegundos
}

void app_main() {
    int timer_injecao = 0;
    int timer_temperatura = 0;
    int timer_abs = 0;
    int timer_airbag = 0;
    int timer_cinto = 0;
    int timer_display = 0;

    while (1) {
        // Incrementa timers (simulação de temporização)
        timer_injecao += 1;
        timer_temperatura += 1;
        timer_abs += 1;
        timer_airbag += 1;
        timer_cinto += 1;
        timer_display += 1;

        // Executar tarefas dentro do período designado

        // Injeção eletrônica (executar a cada 0.5 ms)
        if (timer_injecao >= PERIODO_INJECAO) {
            atualiza_injecao_eletronica();
            timer_injecao = 0;
        }

        // Temperatura do motor (executar a cada 20 ms)
        if (timer_temperatura >= PERIODO_TEMPERATURA) {
            monitora_temperatura_motor();
            timer_temperatura = 0;
        }

        // ABS (executar a cada 100 ms)
        if (timer_abs >= PERIODO_ABS) {
            monitora_abs();
            timer_abs = 0;
        }
```

```
        // Airbag (executar a cada 100 ms)
        if (timer_airbag >= PERIODO_AIRBAG) {
            monitora_airbag();
            timer_airbag = 0;
        }

        // Cinto de segurança (executar a cada 1 segundo)
        if (timer_cinto >= PERIODO_CINTO_SEGURANCA) {
            monitora_cinto_seguranca();
            timer_cinto = 0;
        }

        // Atualizar o display (executar a cada 1 segundo)
        if (timer_display >= PERIODO_DISPLAY) {
            atualiza_display();
            timer_display = 0;
        }

        // Aguarda 1 ms antes de repetir o ciclo
        wait_ms(1);
    }
}
```

Laço de repetição com tratador de interrupções

```
#include <stdio.h>
#include "driver/gpio.h"
#include "esp_timer.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"

// Definir os pinos dos sensores
#define SENSOR_INJECAO_PIN 32
#define SENSOR_TEMPERATURA_PIN 33
#define SENSOR_ABS_PIN 34
#define SENSOR_AIRBAG_PIN 35
#define SENSOR_CINTO_PIN 36

// Definir os tempos de resposta em milissegundos
#define TEMPO_INJECAO_MS 15
```

```c
#define TEMPO_TEMPERATURA_MS 20
#define TEMPO_ABS_MS 100
#define TEMPO_AIRBAG_MS 100
#define TEMPO_CINTO_MS 1000

static bool motor_ativo = false;
static bool frenagem_ativo = false;
static bool vida_ativa = false;

// Configuração dos sensores
void configurar_sensores() {
    esp_rom_gpio_pad_select_gpio(SENSOR_INJECAO_PIN);
    gpio_set_direction(SENSOR_INJECAO_PIN, GPIO_MODE_INPUT);

    esp_rom_gpio_pad_select_gpio(SENSOR_TEMPERATURA_PIN);
    gpio_set_direction(SENSOR_TEMPERATURA_PIN, GPIO_MODE_INPUT);

    esp_rom_gpio_pad_select_gpio(SENSOR_ABS_PIN);
    gpio_set_direction(SENSOR_ABS_PIN, GPIO_MODE_INPUT);

    esp_rom_gpio_pad_select_gpio(SENSOR_AIRBAG_PIN);
    gpio_set_direction(SENSOR_AIRBAG_PIN, GPIO_MODE_INPUT);

    esp_rom_gpio_pad_select_gpio(SENSOR_CINTO_PIN);
    gpio_set_direction(SENSOR_CINTO_PIN, GPIO_MODE_INPUT);
}

void monitoramento_injecao() {
    if (gpio_get_level(SENSOR_INJECAO_PIN)) {
        motor_ativo = true;
        printf("Injeção eletrônica acionada!\n");
    }
}

void monitoramento_temperatura() {
    if (gpio_get_level(SENSOR_TEMPERATURA_PIN)) {
        motor_ativo = true;
        printf("Temperatura do motor acima do limite!\n");
    }
}

void monitoramento_abs() {
    if (gpio_get_level(SENSOR_ABS_PIN)) {
```

```
        frenagem_ativo = true;
        printf("ABS acionado!\n");
    }
}

void monitoramento_airbag() {
    if (gpio_get_level(SENSOR_AIRBAG_PIN)) {
        vida_ativa = true;
        printf("Airbag acionado!\n");
    }
}

void monitoramento_cinto() {
    if (gpio_get_level(SENSOR_CINTO_PIN)) {
        vida_ativa = true;
        printf("Cinto de segurança acionado!\n");
    }
}

void atualizar_display() {
    printf("Estado dos subsistemas:\n");
    printf("Motor: %s\n", motor_ativo ? "Ativo" : "Inativo");
    printf("Frenagem: %s\n", frenagem_ativo ? "Ativo" : "Inativo");
    printf("Vida: %s\n", vida_ativa ? "Ativo" : "Inativo");

    // Reseta o estado dos subsistemas para o próximo ciclo
    motor_ativo = false;
    frenagem_ativo = false;
    vida_ativa = false;
    vTaskDelay(pdMS_TO_TICKS(1000)); // Ajuste o tempo conforme
necessário
}

void app_main() {
    configurar_sensores();

    while (1) {
        monitoramento_injecao();
        monitoramento_temperatura();
        monitoramento_abs();
        monitoramento_airbag();
        monitoramento_cinto();
```

```
        atualizar_display();

        // Atraso para evitar sobrecarga da CPU e permitir que outras
tarefas sejam executadas
        vTaskDelay(pdMS_TO_TICKS(100)); // Ajuste o tempo conforme
necessário
    }
}
```

Microkernel

```
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/queue.h"
#include "driver/gpio.h"
#include "esp_timer.h"

// Definir os pinos dos sensores
#define SENSOR_INJECAO_PIN 32
#define SENSOR_TEMPERATURA_PIN 33
#define SENSOR_ABS_PIN 34
#define SENSOR_AIRBAG_PIN 35
#define SENSOR_CINTO_PIN 36
#define SENSOR_VELOCIDADE_PIN 37
#define SENSOR_CONSUMO_PIN 38

// Definir os tempos de resposta em milissegundos
#define TEMPO_INJECAO_MS 15
#define TEMPO_TEMPERATURA_MS 20
#define TEMPO_ABS_MS 100
#define TEMPO_AIRBAG_MS 100
#define TEMPO_CINTO_MS 1000
#define TEMPO_VELOCIDADE_MS 100
#define TEMPO_CONSUMO_MS 100

#define AMOSTRAS 200

// Estruturas para mensagens
typedef struct {
    float velocidade;
    float consumo;
```

```
} SensorData;

QueueHandle_t queue_sensor_data;

// Prototipos das funções
void monitoramento_injecao(void *pvParameter);
void monitoramento_temperatura(void *pvParameter);
void monitoramento_abs(void *pvParameter);
void monitoramento_airbag(void *pvParameter);
void monitoramento_cinto(void *pvParameter);
void monitoramento_velocidade(void *pvParameter);
void monitoramento_consumo(void *pvParameter);
void atualizar_display(void *pvParameter);

// Configuração dos sensores
void configurar_sensores() {
    esp_rom_gpio_pad_select_gpio(SENSOR_INJECAO_PIN);
    gpio_set_direction(SENSOR_INJECAO_PIN, GPIO_MODE_INPUT);
    esp_rom_gpio_pad_select_gpio(SENSOR_TEMPERATURA_PIN);
    gpio_set_direction(SENSOR_TEMPERATURA_PIN, GPIO_MODE_INPUT);
    esp_rom_gpio_pad_select_gpio(SENSOR_ABS_PIN);
    gpio_set_direction(SENSOR_ABS_PIN, GPIO_MODE_INPUT);
    esp_rom_gpio_pad_select_gpio(SENSOR_AIRBAG_PIN);
    gpio_set_direction(SENSOR_AIRBAG_PIN, GPIO_MODE_INPUT);
    esp_rom_gpio_pad_select_gpio(SENSOR_CINTO_PIN);
    gpio_set_direction(SENSOR_CINTO_PIN, GPIO_MODE_INPUT);
}

// Funções de monitoramento (simplificadas)
void monitoramento_injecao(void *pvParameter) {
    while (1) {
        if (gpio_get_level(SENSOR_INJECAO_PIN)) {
            printf("Injeção eletrônica acionada!\n");
        }
        vTaskDelay(pdMS_TO_TICKS(TEMPO_INJECAO_MS));
    }
}

void monitoramento_temperatura(void *pvParameter) {
    while (1) {
        if (gpio_get_level(SENSOR_TEMPERATURA_PIN)) {
            printf("Temperatura do motor acima do limite!\n");
        }
```

```
        vTaskDelay(pdMS_TO_TICKS(TEMPO_TEMPERATURA_MS));
    }
}

void monitoramento_abs(void *pvParameter) {
    while (1) {
        if (gpio_get_level(SENSOR_ABS_PIN)) {
            printf("ABS acionado!\n");
        }
        vTaskDelay(pdMS_TO_TICKS(TEMPO_ABS_MS));
    }
}

void monitoramento_airbag(void *pvParameter) {
    while (1) {
        if (gpio_get_level(SENSOR_AIRBAG_PIN)) {
            printf("Airbag acionado!\n");
        }
        vTaskDelay(pdMS_TO_TICKS(TEMPO_AIRBAG_MS));
    }
}

void monitoramento_cinto(void *pvParameter) {
    while (1) {
        if (gpio_get_level(SENSOR_CINTO_PIN)) {
            printf("Cinto de segurança acionado!\n");
        }
        vTaskDelay(pdMS_TO_TICKS(TEMPO_CINTO_MS));
    }
}

void monitoramento_velocidade(void *pvParameter) {
    float amostras_velocidade[AMOSTRAS] = {0};
    int contagem_velocidade = 0;

    while (1) {
        if (contagem_velocidade < AMOSTRAS) {
            float velocidade = (float)(rand() % 100); // Simulação de
leitura de velocidade
            amostras_velocidade[contagem_velocidade++] = velocidade;
        } else {
            float soma = 0;
            for (int i = 0; i < AMOSTRAS; i++) {
```

```
                soma += amostras_velocidade[i];
            }
            SensorData data;
            data.velocidade = soma / AMOSTRAS;
            xQueueSend(queue_sensor_data, &data, portMAX_DELAY);
            contagem_velocidade = 0; // Resetar contagem
        }
        vTaskDelay(pdMS_TO_TICKS(TEMPO_VELOCIDADE_MS));
    }
}

void monitoramento_consumo(void *pvParameter) {
    float amostras_consumo[AMOSTRAS] = {0};
    int contagem_consumo = 0;

    while (1) {
        if (contagem_consumo < AMOSTRAS) {
            float consumo = (float)(rand() % 15); // Simulação de
leitura de consumo
            amostras_consumo[contagem_consumo++] = consumo;
        } else {
            float soma = 0;
            for (int i = 0; i < AMOSTRAS; i++) {
                soma += amostras_consumo[i];
            }
            SensorData data;
            data.consumo = soma / AMOSTRAS;
            xQueueSend(queue_sensor_data, &data, portMAX_DELAY);
            contagem_consumo = 0; // Resetar contagem
        }
        vTaskDelay(pdMS_TO_TICKS(TEMPO_CONSUMO_MS));
    }
}

void atualizar_display(void *pvParameter) {
    SensorData data;
    while (1) {
        if (xQueueReceive(queue_sensor_data, &data, portMAX_DELAY)) {
            printf("Velocidade média: %.2f km/h\n", data.velocidade);
            printf("Consumo médio: %.2f L/100km\n", data.consumo);
        }
        vTaskDelay(pdMS_TO_TICKS(1000)); // Atualiza a cada 1 segundo
    }
```

```
}

void app_main() {
    queue_sensor_data = xQueueCreate(10, sizeof(SensorData));

    configurar_sensores();

    xTaskCreate(monitoramento_injecao, "monitoramento_injecao", 2048,
NULL, 6, NULL);
    xTaskCreate(monitoramento_temperatura, "monitoramento_temperatura",
2048, NULL, 5, NULL);
    xTaskCreate(monitoramento_abs, "monitoramento_abs", 2048, NULL, 4,
NULL);
    xTaskCreate(monitoramento_airbag, "monitoramento_airbag", 2048,
NULL, 3, NULL);
    xTaskCreate(monitoramento_cinto, "monitoramento_cinto", 2048, NULL,
2, NULL);
    xTaskCreate(monitoramento_velocidade, "monitoramento_velocidade",
2048, NULL, 1, NULL);
    xTaskCreate(monitoramento_consumo, "monitoramento_consumo", 2048,
NULL, 1, NULL);
    xTaskCreate(atualizar_display, "atualizar_display", 2048, NULL, 1,
NULL);
}
```

```
Temperatura do motor acima do limite!
Temperatura do motor acima do limite!
Temperatura do motor acima do limite!
Velocidade média: 1.96 km/h
Consumo médio: 7.06 L/100km
Temperatura do motor acima do limite!
Injeção eletrônica acionada!
Injeção eletrônica acionada!
Temperatura do motor acima do limite!
Injeção eletrônica acionada!
Temperatura do motor acima do limite!
Injeção eletrônica acionada!
Injeção eletrônica acionada!
Temperatura do motor acima do limite!
```