Smart Crossing

Protótipo - Semáforo inteligente com acessibilidade para pessoas com deficiência

Nícolas Marçal
Vinícius Esperança Mantovani

Campinas, SP

25/08/2025

1. Introdução

Este relatório detalha o desenvolvimento de um protótipo de semáforo inteligente, concebido para otimizar o fluxo de tráfego e aumentar a segurança dos pedestres. O sistema utiliza a placa Raspberry Pi Pico W para gerenciar dinamicamente os estados dos semáforos de veículos e pedestres.

O objetivo central do projeto é o desenvolvimento de um sistema embarcado que auxilie pedestres, especialmente aqueles com mobilidade reduzida, na travessia de faixas. Para isso, uma característica principal é a comunicação via Bluetooth Low Energy (BLE), que permite a um aplicativo Android interagir com o semáforo para ajustar o tempo de travessia de forma personalizada. Adicionalmente, o sistema emprega um sensor de distância para a detecção automática de pedestres e um display OLED para fornecer informações visuais claras.

O firmware foi desenvolvido em C/C++ sobre o sistema operacional de tempo real FreeRTOS, que permite uma gestão eficiente e concorrente das múltiplas tarefas do sistema.

2. Materiais e Componentes

O protótipo foi construído com os seguintes componentes:

- **Microcontrolador (MCU):** Raspberry Pi Pico W.
- **Sensor de Distância:** Sensor ToF (Time-of-Flight) VL53L0X para detecção de pedestres.
- **Sinalização Visual:**
  - Uma matriz de LEDs endereçáveis (Neopixel) de 5x5, funcionando como os semáforos para carros e pedestres.
  - LEDs indicadores de estado: azul para conexão Bluetooth (GPIO 12), vermelho para desconexão (GPIO 13) e verde para indicar o tempo de travessia (GPIO 11).
- **Display de Informações:** Display OLED monocromático de 128x64 pixels com comunicação I2C.
- **Software e Frameworks:**
  - Firmware desenvolvido em C/C++ utilizando o SDK da Raspberry Pi Pico e FreeRTOS.
  - Framework BTstack com driver CYW43 para a funcionalidade Bluetooth.
  - Aplicativo Android nativo em Kotlin, utilizando Jetpack Compose e API BluetoothGatt.

3. Arquitetura e Funcionamento

O sistema opera com base em uma arquitetura multitarefa gerenciada pelo FreeRTOS, com três tarefas principais rodando concorrentemente.

3.1. Pinos Utilizados (Raspberry Pi Pico)

- **I2C (i2c0) - Sensor:**
  - GPIO 0 (SDA): Conectado ao pino SDA do sensor VL53L0X.
  - GPIO 1 (SCL): Conectado ao pino SCL do sensor VL53L0X.
- **I2C (i2c1) - Display:**
  - GPIO 2 (SDA): Conectado ao pino SDA do display OLED.
  - GPIO 3 (SCL): Conectado ao pino SCL do display OLED.
- **Controle da Matriz de LEDs (PIO):**
  - GPIO 28: Pino de dados para a matriz de LEDs Neopixel.
- **LEDs Indicadores:**
  - GPIO 11: LED verde.
  - GPIO 12: LED azul.
  - GPIO 13: LED vermelho.

3.2. Lógica do Software (FreeRTOS)

O arquivo “main.c” inicializa o sistema e cria três tarefas principais:

1. **traffic_lights_task:** Funciona como a máquina de estados que controla a lógica do semáforo, alternando entre os estados (verde para carro, amarelo, vermelho, etc.). O fluxo normal é feito com temporização entre os estados carro/pedestre: vermelho/verde, vermelho/amarelo, vermelho/vermelho, verde/vermelho, amarelo/vermelho, vermelho/vermelho e assim por diante. Quando a detecção de um pedestre ocorre (seja pelo sensor ou por uma solicitação BLE), o

semáforo está no estado verde:vermelho e o tempo mínimo de abertura do semáforo para carros já se passou, esta tarefa inicia a sequência para fechar o sinal dos carros e abrir para os pedestres.

2. **sensor_task:** Dedicada exclusivamente à leitura do sensor de distância VL53L0X. Se um pedestre é detectado a menos de 1 metro, a tarefa envia uma notificação para a *traffic_lights_task.c*

3. **display_task:** Gerencia todas as atualizações no display OLED, exibindo mensagens como "ABERTO!" e "FECHADO!.

3.3. Comunicação Bluetooth para Acessibilidade

Para atender às necessidades de pessoas com mobilidade reduzida, o módulo BLE da Pico W foi ativado.

- **Configuração BLE:** A placa atua como um servidor BLE customizado, utilizando o framework BTstack. Foi criada uma GATT customizada com um serviço (UUID 0xFFF0) e uma característica de escrita (UUID 0xFFF1).

- **Aplicativo Android:** Um aplicativo desenvolvido em Kotlin permite ao usuário se conectar ao semáforo. O app solicita as permissões necessárias (

  BLUETOOTH_SCAN, BLUETOOTH_CONNECT, ACCESS_FINE_LOCATION), escaneia dispositivos e se conecta automaticamente ao serviço da Pico W.

- **Ajuste Dinâmico de Tempo:** A interface do app possui botões de 1 a 5, representando o nível de mobilidade do usuário. Ao selecionar um nível, o valor correspondente é enviado à placa. O sistema interpreta esse valor para ajustar o tempo de travessia. Por exemplo:

  - Nível 1 → 10 segundos.
  - Nível 3 → 15 segundos.
  - Nível 5 → 30 segundos.

- **Feedback Visual:** LEDs externos indicam o estado da conexão BLE: o LED azul acende quando há conexão e o vermelho quando não há. O LED verde pisca por um tempo proporcional ao nível de mobilidade enviado pelo app, simulando o tempo de travessia estendido.

4. Desafios Encontrados no Desenvolvimento

- **Inicialização do Bluetooth:** A ativação do BLE exigiu ajustes finos no arquivo CMakeLists.txt e a inclusão correta das bibliotecas pico_btstack_ble, pico_btstack_cyw43 e pico_cyw43_arch_none. A ausência de qualquer uma delas resultava em falha total na inicialização.

- **Gestão de Tarefas Concorrentes:** A abordagem inicial para controlar o piscar do LED verde usava sleep_ms(), o que bloqueava o loop principal do BTstack e o funcionamento do sistema. A solução foi migrar para um sistema baseado em timers assíncronos [50], um desafio similar ao que motivou o uso do FreeRTOS para gerenciar múltiplas tarefas sem travamentos.

- **Recepção de Dados BLE:** A função de callback att_write_callback apresentou falhas ao interpretar valores enviados como texto. Foi necessário ajustar a lógica para interpretar corretamente os valores binários enviados no formato uint8.

- **Permissões no Android:** As versões mais recentes do Android (API 31+) exigem permissões explícitas e em tempo de execução para operações de Bluetooth, o que tornou obrigatória a inclusão de um requestPermissionLauncher no código do aplicativo.

5. Resultados Obtidos

Ao final do desenvolvimento, obteve-se um sistema funcional com as seguintes características:

- Um protótipo de semáforo inteligente, com lógica de estados gerenciada eficientemente pelo FreeRTOS.

- Detecção de presença de pedestres funcional através do sensor ToF.

- Feedback visual claro para o pedestre por meio da matriz de LEDs e do display OLED.

- A placa Pico W atuando como um servidor BLE customizado e estável.

- Um aplicativo Android funcional capaz de detectar o dispositivo, conectar-se automaticamente e enviar valores de 1 a 5 para ajustar o comportamento da placa em tempo real[59].

6. Melhorias Planejadas

- **Sincronização entre Múltiplos Semáforos:** Para garantir que ambos os lados de uma faixa de pedestres operem de forma sincronizada, planeja-se a comunicação entre duas placas Pico W. Isso pode ser implementado via Wi-Fi para estabelecer uma comunicação direta (P2P) ou criando uma rede BLE onde as placas trocam dados sobre a presença de pedestres. Quando um sensor detectar um pedestre, um sinal será enviado para a outra placa, e ambas iniciarão a rotina de travessia simultaneamente.

- **Segurança e Autenticação:** Implementação de emparelhamento seguro (pairing/bonding) no BLE para evitar que conexões não autorizadas possam controlar o semáforo.

- **Sensores Integrados Adicionais:** O sistema já utiliza um sensor de presença, mas poderia ser expandido com a inclusão de sensores ultrassônicos ou infravermelhos para redundância ou para cobrir uma área maior.

7. Conclusão

O projeto demonstrou a viabilidade de utilizar a Raspberry Pi Pico W como uma solução robusta e de baixo custo para um semáforo inteligente focado em acessibilidade. A comunicação entre o aplicativo Android e a placa mostrou-se eficiente para o ajuste personalizado do tempo de travessia. A arquitetura baseada em FreeRTOS se mostrou eficaz na gestão de tarefas concorrentes e o sistema se provou escalável, pronto para incluir novos recursos, como a sincronização entre múltiplos dispositivos e sensores adicionais.