Automação do Mecanismo XYZ para Gerenciamento de Armazém- Warehouse

Este projeto utiliza um Raspberry Pi Pico para criar uma interface HTML capaz de controlar o mecanismo XYZ de captura de pallets e monitorar a localização dos pallets no rack.

Funcionalidades

* **Servidor Web Integrado:** Utiliza o Wi-Fi para a criação de uma interface html acessível para controle do Mecanismo XYZ por meio de botões e sliders, o que permite um acesso remoto por qualquer dispositivo conectado à rede.
* **Controle em Tempo Real:** Controla os

motores de passo ou servos nas direções X, Y e Z, resposta instantânea ao comando dado pela interface e monitoramento de sensores para feedback.
* **Automação e Lógica de Armazém:** Mapeamento dos espaços ocupados pelos pallets no rack.

Funcionamento do Código
Inicialmente foram adicionadas as bibliotecas padrão da Pico, GPIO, ADC, I2C, PWM, do display OLED e fontes, FreeRTOS e tasks, e as bibliotecas de entrada e saída padrão.
Após isso, stdio_init_all() inicializa o Pico, vTaskStartScheduler() inicializa o FreeRTOS, e implementa o servidor HTTP.

Na implementação do HTTP, send_next_chunk() envia pedaços de resposta HTTP de até 1024 bytes, http_sent() é o callback chamado quando o cliente confirma que recebeu um pedaço de resposta, http_recv() é o callback chamado quando o cliente envia uma requisição, connection_callback() é chamado quando chega uma nova conexão e start_http_server() cria um socket TCP (pcb), faz o bind na porta 80, coloca em modo listei e define connection_callback() como função que lida com novas conexões.
Além das funções principais do servidor HTTP, o código também implementa uma task chamada vPollingTask para executar periodicamente a função cyw43_arch_poll(). Essa chamada é fundamental para manter a pilha de rede do módulo Wi-Fi CYW43 ativa, garantindo

que eventos de rede, como recebimento de pacotes e manutenção de conexões, sejam tratados corretamente dentro do ambiente multitarefa do FreeRTOS.
Além disso, é importante ressaltar a estrutura http_state, usada para armazenar o estado de cada conexão HTTP. Ela permite que as respostas sejam enviadas de forma segmentada, em blocos de até 1024 bytes, utilizando o controle de fluxo do protocolo TCP. Isso é feito pelas funções send_next_chunk() e http_sent(), que coordenam o envio assíncrono da resposta armazenada em hs->response, possibilitando maior robustez no tratamento de grandes quantidades de dados. A função preencher_html() é responsável por gerar dinamicamente o conteúdo HTML da página web no buffer html[], o qual é posteriormente transferido para hs-

>response para envio ao cliente. Por fim, o endpoint /web já está estruturado para retornar uma resposta JSON, que será utilizado posteriormente para fornecer dados em tempo real sobre o estado físico do armazém, como posições dos motores, status do eletroímã e sensores ativos.
O controle de movimentação do mecanismo XYZ e a ativação do eletroímã são representados atualmente por variáveis globais (como current_x, current_y, electromagnet_active). A função preencher_html() monta dinamicamente o HTML com os dados atuais do sistema, enquanto o endpoint /web oferece uma API em JSON que permite à interface JavaScript atualizar o estado visual do armazém em tempo real. O controle físico dos motores e do eletroímã ainda será integrado em versões futuras.

O arquivo HTML.c é responsável por gerar a página web que será utilizada. Nesse arquivo, há um buffer html [8192] que serve para armazenar todo o conteúdo da página e uma função preencher_html que usa a snprintf para montar uma resposta HTTP completa, além de uma interface gráfica com layout do rack, botões de operação, históricos de movimentações entre outros. Por fim, o JavaScript usa a função fetchHistorico() para atualizar o painel de histórico com os registros retornados e a função enviarAcao(acao) chama a função fetchHistorico() para atualizar o histórico.