Carrinho Pet com Monitoramento de Vídeo

Etapa 3

Caio Vitor Carneiro de Oliveira
Guilherme Achilles de Oliveira e Aguiar
Humberto Alves Mesquita

Brasília
2025

1. Construção do Protótipo

1.1 Montagem Física

- Estrutura: montagem do chassi com as rodas, motores DC e suporte para os módulos e os microcontroladores.

- Microcontroladores: fixação da BitDogLab (Raspberry Pi Pico W) e da ESP32-CAM em posição estratégica.

- Alimentação: Feita atualmente por meio de 4 pilhas AA e um power back.

- Drivers do Motor: conexão dos motores para controle de direção e velocidade.

- Sensores: instalação do MPU6050.

1.2 Integração de Software

O programa inicializa o carrinho, conectando ao Wi-Fi e a um broker MQTT na rede local, assina os tópicos de controle e mantém um loop ocioso enquanto a pilha de rede processa mensagens em background

Principais características:

- Configuração do hardware: inicializa PWM e pinos dos motores via myCar.configureCar().

- Conectividade: estabelece Wi-Fi (SSID/Senha) e sessão MQTT (IP/porta/usuário/senha).

- Controle do carrinho: a lógica de movimento é delegada às bibliotecas car.h (drivers/pinos) e mqtt_control.h (assinaturas e callbacks).

- Telemetria: envia temperatura interna a cada 5 s com mqtt_start_periodic_publish_temp_ms(5000).

1.3 Mapeamento de pinos dos motores:

- Esquerda: 16, 17
- Direita: 18, 19
- Traseiro/Centro: 8, 9

1.3.1 Fluxo de execução do programa:

- Inicializa IO: stdio_init_all() e atraso breve.
- Configura PWM dos motores: myCar.configureCar().
- Conecta Wi-Fi + MQTT e assina tópicos de controle: mqtt_begin(...).
- Inicia telemetria periódica (5 s).
- Entra em loop ocioso (tight_loop_contents()), enquanto a pilha de rede trata assinaturas/callbacks de controle recebidos via MQTT.

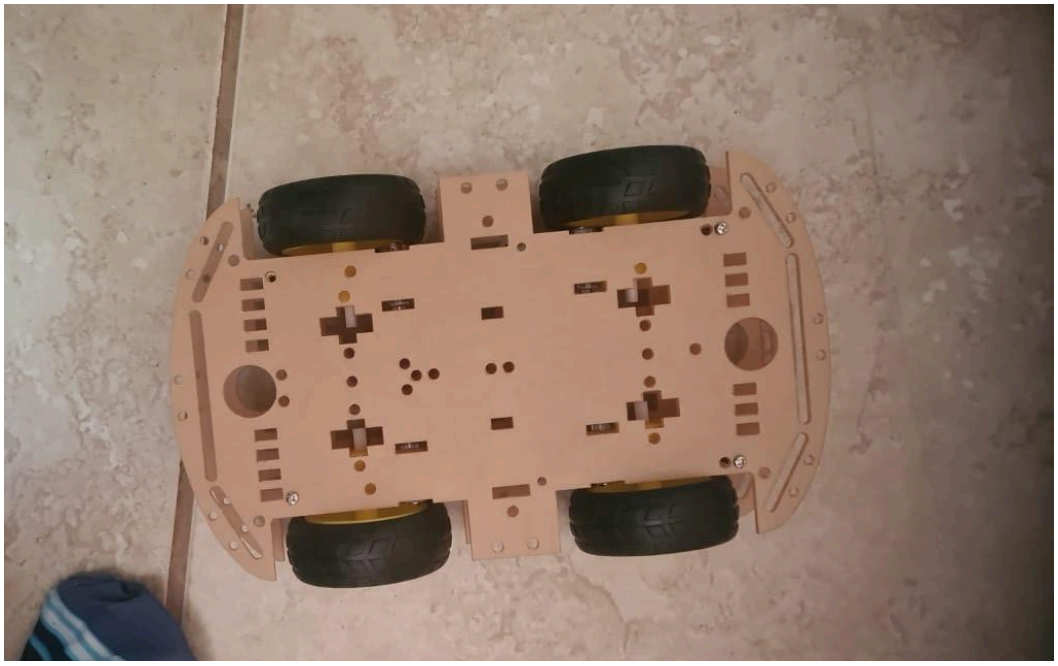

Imagem 1 - Chassi

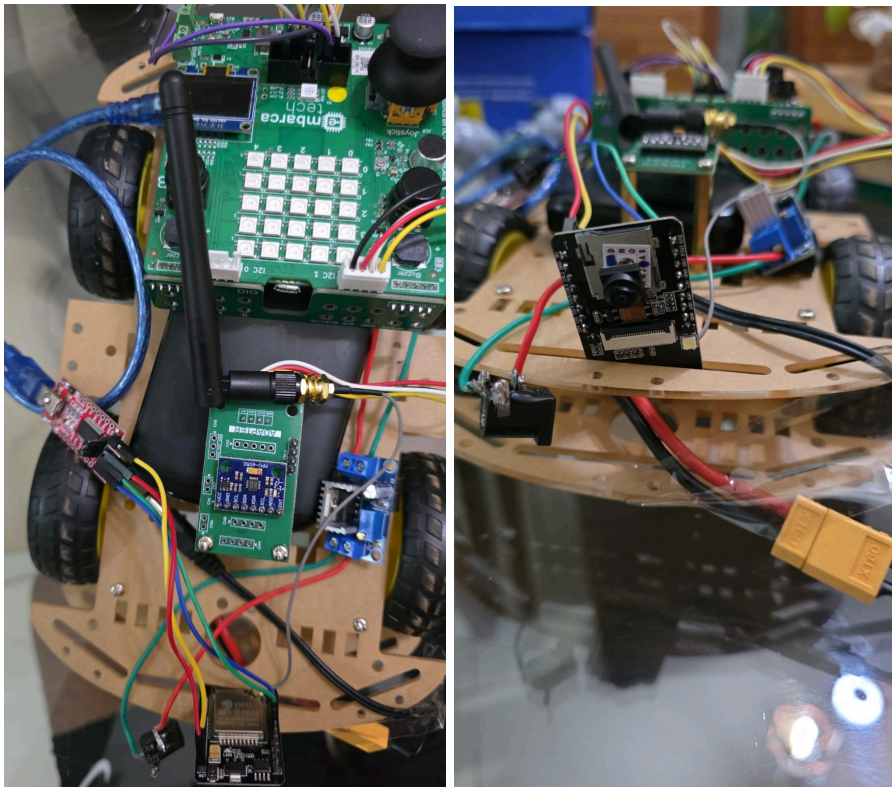

Imagem 2 - Posicionamento do módulo MPU6050 e da ESP32-CAM

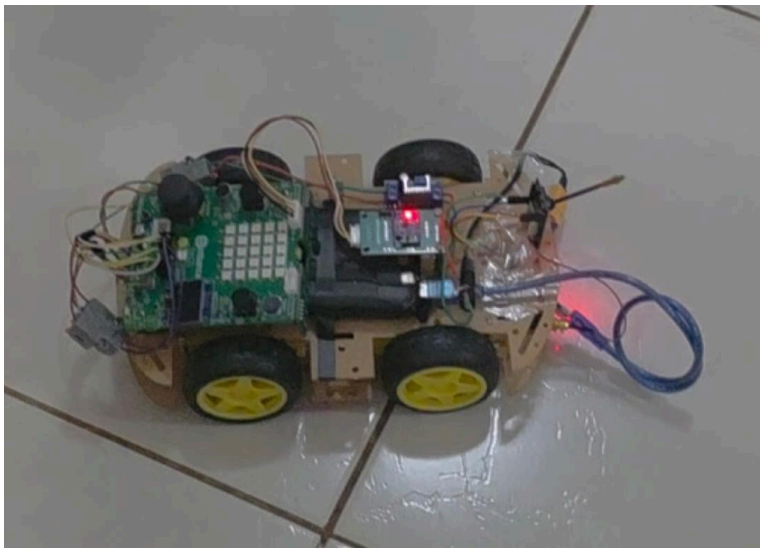

Imagem 3 - Protótipo montado

1.4 CameraWebServe

O diretório *CameraWebServer* contém a implementação do servidor web embarcado na ESP32-CAM, cuja função principal é disponibilizar o streaming de vídeo e uma

interface de controle acessível via navegador.

- Inicialização da câmera (OV2640): configura resolução, taxa de quadros e formato (MJPEG).

- Servidor HTTP embarcado: responde a requisições GET/POST de usuários conectados à rede Wi-Fi.

- Streaming de vídeo em tempo real: disponibilizado via protocolo MJPEG, exibido em um <img> na interface web.

- Interface HTML/CSS/JS: contém botões para controle de movimento do carrinho (frente, trás, esquerda, direita).

- Integração com AWS IoT: os comandos de movimento recebidos na interface são encaminhados via HTTP para o broker, que os retransmite para a BitDogLab por MQTT.

- Monitoramento do sistema: inclui funções de status, como feedback da câmera e estado da conexão.

1.4.1 Estrutura dos arquivos

- app_httpd.cpp: implementa o servidor HTTP e os endpoints de controle/stream

- camera_pins.h: define a pinagem da ESP32-CAM (mapeamento da câmera OV2640).

- camera_index.h: contém a página HTML embarcada servida pelo módulo.

- CameraWebServer.ino: função principal que inicializa Wi-Fi, câmera e servidor.

1.5 Fluxo de funcionamento

- Inicialização: ESP32-CAM conecta ao Wi-Fi configurado.

- Câmera: é ativada e configurada para capturar imagens.

- Servidor HTTP: disponibiliza a página web embarcada.

- Streaming: frames são transmitidos via MJPEG para o navegador.

- Controle: entradas do usuário (cliques/botões) são transformadas em comandos e enviados ao carrinho via AWS IoT + MQTT.

1.6 Controle via terminal (Python + MQTT)

O código controlar.py implementa uma interface de controle do carrinho pelo terminal, usando Python com a biblioteca *curses* (para captura de teclado) e paho-mqtt (para comunicação MQTT).

Funcionamento:

- Conexão MQTT: conecta ao broker Mosquitto com autenticação (usuário/senha).

- escola/sala1/carro/cmd: recebe comandos de direção (UP, DOWN, LEFT, RIGHT, STOP).

- escola/sala1/carro/speed: recebe valor da velocidade (0 a 4095).

Interface de teclado (curses):

- Setas ou WASD: Barra de espaço → parar imediatamente.

- +/- : aumenta ou reduz a velocidade.

- q: encerra o programa.

- Modo “hold”: se nenhuma tecla for pressionada por um período definido (200 ms), o comando “PARAR” é enviado automaticamente, evitando que o carrinho continue em movimento por erro de operador.

- Feedback visual: exibe no terminal a velocidade atual e o último comando enviado.

Parâmetros configuráveis durante os testes

- Broker MQTT: 192.168.15.9 (porta 1883, sem TLS).

- Credenciais: usuário: aluno, senha: ******.

- Velocidade inicial: 3000 (de 0 a 4095).

- Tempo ocioso (hold mode): 200 ms.

1.6.1 Fluxo de funcionamento

- Inicializa MQTT e conecta ao broker.

- Inicia interface curses no terminal.

- Captura continuamente as teclas pressionadas e envia comandos via MQTT.

- Garante envio automático do comando “PARAR” quando nenhuma tecla é mantida pressionada.

- Exibe em tempo real a velocidade e o último comando publicado.

- Finaliza de forma limpa, encerrando o loop MQTT ao sair.

- Observações e melhorias futuras

- Segurança: migrar para MQTT com TLS (porta 8883).

- Portabilidade: permitir configurar broker, usuário, senha e tópicos via arquivo externo (config.json) em vez de hardcode.

- Interface gráfica opcional: adicionar versão com PyQt ou Tkinter para facilitar uso em usuários não técnicos.

- Controle refinado: incluir comandos diagonais (ex.: UP+LEFT) ou pré-ajustes de velocidade.

2. Testes Realizados

Durante o desenvolvimento do protótipo, foram conduzidos testes para validar as funcionalidades principais do carrinho, o registro em vídeo pode ser acessado através do seguinte link: https://www.youtube.com/watch?v=KwAwa5XKUWc

2.1 Testes Realizados

- Motores: verificou-se que os motores respondem adequadamente aos comandos de frente, trás, esquerda e direita.

- Câmera: a ESP32-CAM iniciou corretamente e transmitiu o vídeo em tempo real.

- Conectividade Wi-Fi: a comunicação com o AWS IoT demonstrou estabilidade durante os testes.

- Alcance de rede: o sistema funcionou de forma satisfatória dentro da área de cobertura testada.

- Avaliação da interface web, confirmando responsividade e execução correta dos comandos e a sincronização entre streaming de vídeo e comandos de movimento, comprovando que ambos podem ser executados ao mesmo tempo.

3. Ajustes Identificados

3.1 Movimento para os lados

Uma das melhorias planejadas é a movimentação, atualmente o carrinho executa apenas deslocamentos para frente, trás e curvas, com a adaptação da estrutura será possível deslocar o carrinho com mais facilidade para os lados, aumentando a manobrabilidade em ambientes internos e estreitos.

3.2 Retirada da Power Bank

O protótipo atual utiliza uma power bank como solução de alimentação, pretende substituí-la por um sistema dedicado de bateria recarregável, integrado diretamente ao chassi, isso permitirá uma alimentação mais confiável, maior autonomia e um design mais limpo sem componentes externos adaptados.

3.3 Redução para uso exclusivo da ESP32

Atualmente, o sistema faz uso combinado da BitDogLab (Raspberry Pi Pico W) e da ESP32-CAM. Uma das melhorias em estudo é simplificar a arquitetura, mantendo apenas a ESP32 como microcontrolador central, responsável tanto pelo streaming de vídeo quanto pelo controle dos motores, essa alteração reduzirá custos, consumo energético e complexidade do software.

3.4 Ajuste do Chassi

O chassi atual ainda apresenta limitações de espaço e fixação, pretende-se redesenhar sua estrutura para melhorar a disposição dos componentes, reduzir vibrações e garantir maior estabilidade durante o movimento, esse ajuste também contribuirá para um visual mais compacto e profissional do protótipo.

3.5 Exposição da Porta AWS para Acesso Remoto

Atualmente, o controle do carrinho depende da rede Wi-Fi local, uma melhoria futura será a exposição da porta de comunicação com a AWS IoT, permitindo que o carrinho seja operado remotamente de qualquer lugar, mesmo fora da residência, essa modificação amplia o alcance de uso e proporciona maior flexibilidade ao usuário.

3.6 Inclusão de Carregador por Indução

Outra melhoria considerada é a implementação de carregamento por indução, com isso, o carrinho poderá ser recarregado automaticamente ao se posicionar em uma base apropriada, eliminando a necessidade de desconectar e reconectar cabos manualmente, essa funcionalidade aumenta a autonomia operacional e a praticidade do uso diário.

4. Referências:

Github Disponível em:
https://github.com/EmbarcaTech-2025/projeto-final-caio_guilherme_humberto_projetofinal

Imagem 1 - Disponível em:
https://github.com/EmbarcaTech-2025/projeto-final-caio_guilherme_humberto_projetofinal/blob/main/Etapa%203/Imagens/imagem%201.jpeg

Imagem 2 - Disponível em:
https://github.com/EmbarcaTech-2025/projeto-final-caio_guilherme_humberto_projetofinal/blob/main/Etapa%203/Imagens/imagem%202.jpeg

Figura 2.1 - Disponível em:
https://github.com/EmbarcaTech-2025/projeto-final-caio_guilherme_humberto_projetofinal/blob/main/Etapa%203/Imagens/imagem%202.1.jpeg

Imagem 3 - Disponível em:
https://github.com/EmbarcaTech-2025/projeto-final-caio_guilherme_humberto_projetofinal/blob/main/Etapa%203/Imagens/imagem%203.jpeg

Link para o vídeo - Disponível em:
https://www.youtube.com/watch?v=KwAwa5XKUWc