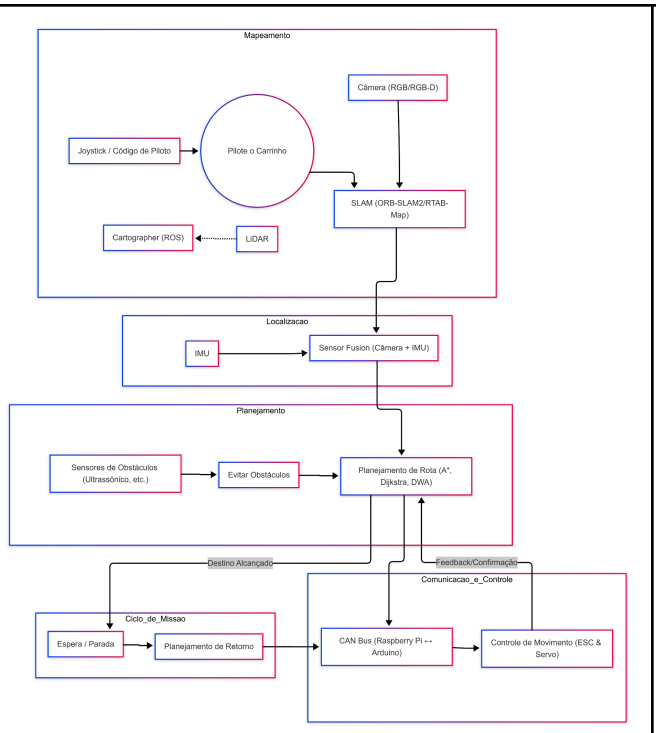
Projeto RC Autônomo

Este projeto transforma um carro RC impresso em 3D em uma plataforma de demonstração de sistemas ADAS e navegação autônoma, usando sensores reais e software embarcado, com funcionalidades presentes em veículos reais.

🧠 Arquitetura do Sistema




```
flowchart LR
   %% Mapeamento (SLAM)
   subgraph Mapeamento
      direction LR
      A["Joystick / Código de Piloto"] --> B(("Pilote o Carrinho"))
      C_Camera["Câmera (RGB/RGB-D)"] --> D_SLAM["SLAM (ORB-SLAM2/RTAB-Map)"]
      B --> D_SLAM
      %% Futuro: LiDAR com Cartographer
      F_Lidar["LiDAR"] -.-> G_Cartographer["Cartographer (ROS)"]
   end

   %% Localização com Sensor Fusion
   subgraph Localizacao
      direction TB
      D_SLAM --> H_SensorFusion["Sensor Fusion (Câmera + IMU)"]
      I_IMU["IMU"] --> H_SensorFusion
   end

   %% Planejamento da Rota e Obstáculos
   subgraph Planejamento
      direction TB
      H_SensorFusion --> J_Rota["Planejamento de Rota (A*, Dijkstra, DWA)"]
      K_Obs["Sensores de Obstáculos (Ultrassônico, etc.)"] --> L_Evit["Evitar Obstáculos"]
      L_Evit --> J_Rota
   end

   %% Comunicação CAN e Controle
   subgraph Comunicacao_e_Controle
      direction TB
      J_Rota --> M_CAN["CAN Bus (Raspberry Pi ↔ Arduino)"]
      M_CAN --> N_Controle["Controle de Movimento (ESC & Servo)"]
      %% Feedback do Atuador para Ajuste Dinâmico
      N_Controle -- "Feedback/Confirmação" --> J_Rota
   end

   %% Ciclo de Missão: Destino e Retorno
   subgraph Ciclo_de_Missao
      direction TB
      J_Rota -- "Destino Alcançado" --> O_Espera["Espera / Parada"]
      O_Espera --> P_Retorno["Planejamento de Retorno"]
      P_Retorno --> M_CAN
   end
```

🔍 Como funciona

- 🔹 **1. Mapeamento (SLAM)**
  - Usa-se ORB-SLAM2 ou RTAB-Map (com câmera RGB ou RGB-D) no Raspberry Pi para mapear o ambiente inicialmente
  - `Cartographer` do ROS se usar LiDAR (futuro)
  - Inicialmente, você **pilota o carrinho** (via joystick ou código) para **gravar o mapa**
- 🔹 **2. Localização**
  - Durante a navegação, o sistema compara o que vê com o **mapa previamente criado**
  - A posição é ajustada com dados da câmera e IMU
- 🔹 **3. Planejamento de rota**
  - A posição atual e o destino são usadas para calcular a melhor rota (A*, Dijkstra ou DWA)
  - Pode usar bibliotecas do ROS como `nav2d` ou `move_base`

🔹 **4. Comunicação CAN**

- O Raspberry envia comandos ao Arduino por rede CAN, simulando uma rede veicular real
- Protocolo leve, tolerante a falhas e determinístico, ideal para sistemas automotivos

🔹 **5. Controle de movimento**

- O Raspberry Pi envia comandos ao Arduino via CAN (ex: `velocidade = 30, direção = -15°`).
- O Arduino interpreta os comandos CAN e atua no ESC e servo

A cada passo, o planejamento decide:

- ângulo de direção (servo)
- velocidade (ESC)

🧪 **Exemplo de comportamento:**

- O carrinho inicia no mapa com orientação e posição estimadas via IMU na `origem`
- O `destino` é definido
- O sistema planeja a rota (A*)
- Enquanto navega, sensores monitoram obstáculos (evita pets ou móveis novos), controla a tração e AEB
- Ao chegar ao `destino`, o carrinho espera e depois inicia o trajeto de volta para `origem.`

🧩 Componentes necessários

Recurso	Função	Status
Raspberry Pi 4	Computação principal (navegação, planejamento)	✅
Sensor Ultrassônico	Evitar obstáculos dinâmicos	✅
IMU (ex: MPU6050)	Estimar orientação / odometria	🔶 Recomendado
Câmera USB	Visão do ambiente (opcional)	✅
Encoder nas rodas	Medir deslocamento (odometria precisa)	🔶 Opcional
Placa CAN para RPi e Arduino	Comunicação entre módulos embarcados	✅
Módulo Gps Neo-6m V2 Com Antena	GPS para melhorar localização	🔶 Opcional

🧰 Tecnologias Utilizadas

Função	Ferramenta sugerida
SLAM visual	ORB-SLAM2, RTAB-Map, OpenVSLAM

Planejamento	A*, Dijkstra, ROS move_base
Controle de motor	Arduino (PWM + PID via CAN)
Localização	Fusion de IMU + visão (EKF/Complementary filter) + GPS
Evasão de obstáculos	Sensor ultrassônico em tempo real

📊 Impacto Profissional

- Conhecimento em percepção, controle, localização e comunicação veicular
- Demonstra integração de sensores reais com algoritmos de controle e redes embarcadas
- Entendimento de arquitetura embarcada distribuída (Raspberry + Arduino via CAN)
- Alinha com requisitos como:
  - **trajectory planning**
  - **driver and road model**
  - **software in the loop**
  - **CAN communication**
  - **C++ ou Python com percepção embarcada**

🔄 Fluxograma Funcional — RC Autônomo com Navegação Baseada em SLAM, AEB e Comunicação CAN

[Início]
↓
[Inicializar sensores, IMU, câmeras, motor e ESC]
↓
[Carregar mapa existente OU iniciar mapeamento SLAM]
↓
[Estimar posição atual (IMU + visão)]
↓
[Planejar rota até destino (ex: Quarto)]
↓
[Enquanto destino não for alcançado]:
├──> [Capturar dados sensoriais e imagem]
├──> [Atualizar posição e SLAM]
├──> [Verificar obstáculos à frente (AEB)]
├──> [Corrigir trajetória (se necessário)]
├──> [Ajustar controle de tração se roda patinar]
├──> [Replanejar rota se necessário]
├──> [Enviar comandos ao Arduino via rede CAN (velocidade e direção)]
↓
[Chegou ao destino?]
↓
[Sim] → [Esperar ou iniciar retorno à origem]

📦 Componentes Detalhados por Função

Categoria	Componente	Função	Status

Processamento	Raspberry Pi 4 (2GB)	SLAM, planejamento, controle de missão	✅
Visão	Câmera USB ou ESP32-CAM	Detecção visual, navegação SLAM	✅
Localização	IMU MPU6050	Estimativa de orientação e odometria	🔶 Recomendado
Atuação	Arduino Uno ou Mega	Geração de sinais PWM para ESC e servo	✅
Comunicação	2× MCP2515 CAN + cabos	Comunicação CAN entre Raspberry Pi e Arduino	✅
Controle de movimento	ESC 30A + motor brushed 240 27T	Tração principal	✅
Controle de direção	Servo SG996	Controle de direção do veículo	✅
Segurança (AEB)	Sensor ultrassônico HC-SR04	Detecção frontal de obstáculos	✅
Controle de tração	Algoritmo + detecção de variação de corrente no ESC (ou slip estimado via tempo/resposta PWM)	Ajuste dinâmico do torque para evitar derrapagem	🔶 Implementar software
Alimentação	Power Bank 5V 10.000mAh	Fonte para Raspberry Pi e periféricos	✅
Encoder nas rodas		Medir deslocamento (odometria precisa)	🔶 (adquirir ou estimar)

📐 Diagrama de Arquitetura Visual

```
graph TD
 subgraph Percepção
  CAM(Câmera USB / ESP32-CAM)
  US(Sensor Ultrassônico)
  IMU(IMU MPU6050)
 end
```

```
subgraph Processamento - Raspberry Pi
  SLAM[SLAM / Localização / Planejamento]
  CAN_TX[CAN TX (MCP2515)]
end

subgraph Controle - Arduino
  CAN_RX[CAN RX (MCP2515)]
  ESC[Controle ESC - Motor]
  SERVO[Controle de Direção - Servo]
end

CAM --> SLAM
IMU --> SLAM
US --> SLAM
SLAM --> CAN_TX
CAN_TX --> CAN_RX
CAN_RX --> ESC
CAN_RX --> SERVO
```

Códigos

📘 Estrutura de README para GitHub — Projeto RC Autônomo com ADAS e Navegação

```
# 🧠 Veículo RC Autônomo com SLAM, AEB e Controle de Tração

Este projeto transforma um carro RC impresso em 3D em uma plataforma de demonstração de sistemas ADAS e navegação autônoma, usando sensores reais e software embarcado, com funcionalidades presentes em veículos reais.

## 🚗 Funcionalidades
- AEB (Autonomous Emergency Braking) com sensor ultrassônico
- Navegação indoor baseada em SLAM e planejamento de rota
- Controle de tração simples para evitar patinação em acelerações
- Controle de movimento via Arduino (ESC + Servo)
- Processamento em Raspberry Pi 4 (2GB)
- Integração modular em Python e C++
- Comunicação CAN entre Raspberry Pi 4 e Arduino

## 📦 Hardware
- Raspberry Pi 4 (2GB)
- Câmera USB + ESP32-CAM
- Sensor Ultrassônico HC-SR04
- Arduino Uno ou Mega
- Servo SG996 + ESC + motor 240 27T
- Power Bank 5V 10.000mAh
- IMU (ex: MPU-6050) — recomendado
- Módulo Gps Neo-6m V2 Com Antena
```

🧰 Tecnologias Usadas
- Python, C++
- OpenCV
- ORB-SLAM2 / RTAB-Map
- ROS (opcional)
- PWM, UART, Serial

🎯 Motivação
Demonstrar, em escala reduzida, conceitos-chave usados em sistemas ADAS e veículos autônomos reais, com foco em percepção, localização, planejamento, controle de movimento e segurança funcional (AEB, tração).

📂 Estrutura do Projeto

├── perception/ │ ├── slam_module.py │ ├── localizer.py ├── control/ │ ├── motor_control.py │ ├── traction_control.py ├── sensors/ │ ├── ultrasonic.py │ ├── slip_detector.py ├── main.py ├── README.md

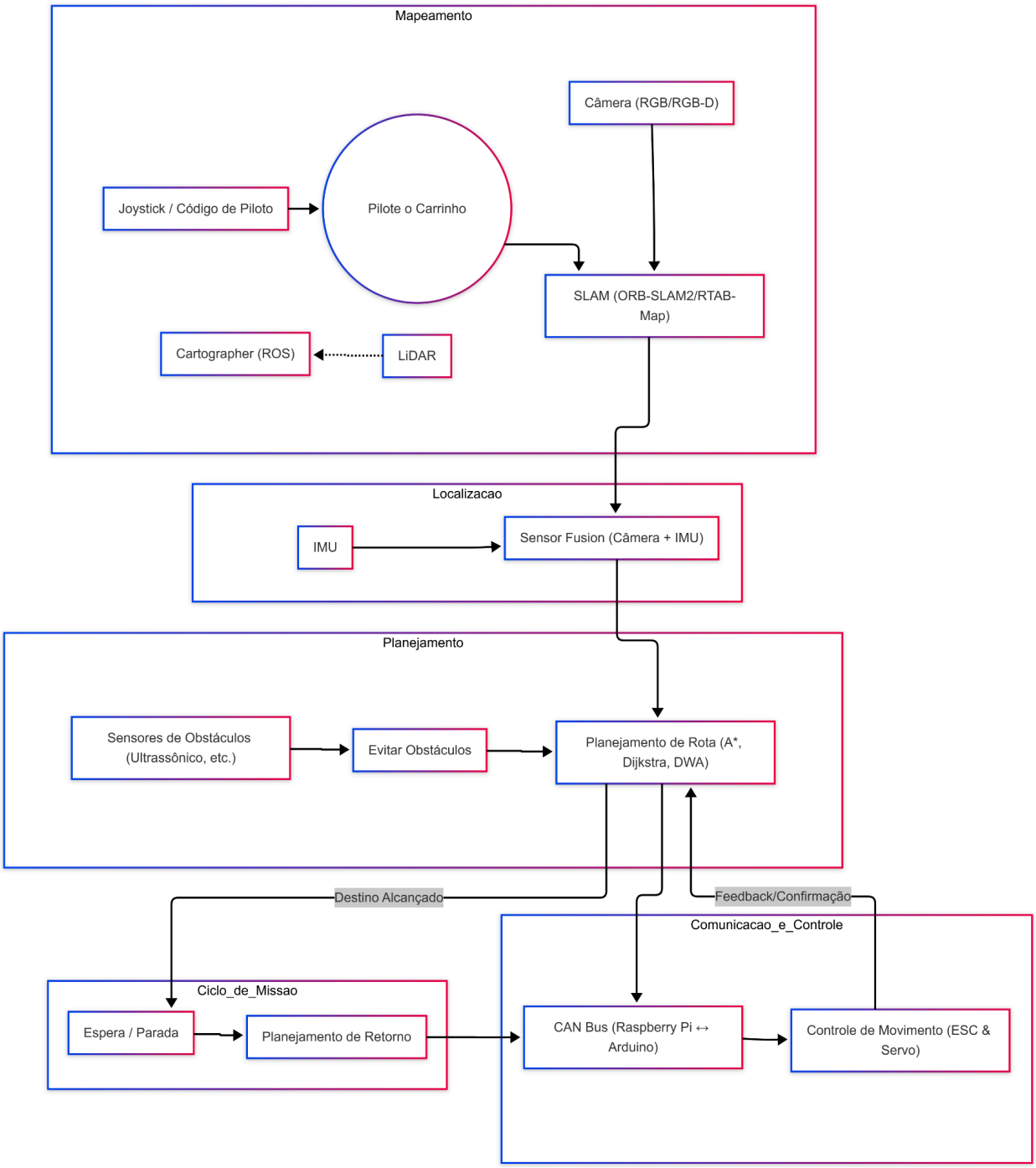
🧠 Arquitetura do Sistema

✅ Como adaptar o projeto para direção autônom com comunicação CANa

🧩 Componentes necessários

Recurso	Função	Status
Raspberry Pi 4	Computação principal (navegação, planejamento)	✅
Sensor Ultrassônico	Evitar obstáculos dinâmicos	✅
IMU (ex: MPU6050)	Estimar orientação / odometria	🔶 Recomendado
Câmera USB ou ESP32-CAM	Visão do ambiente (opcional)	✅
Encoder nas rodas	Medir deslocamento (odometria precisa)	🔶 Opcional

🧱 Arquitetura sugerida




```mermaid
flowchart LR
    %% Mapeamento (SLAM)
    subgraph Mapeamento
        direction LR
        A["Joystick / Código de Piloto"] --> B(("Pilote o Carrinho"))
        C_Camera["Câmera (RGB/RGB-D)"] --> D_SLAM["SLAM (ORB-SLAM2/RTAB-Map)"]
        B --> D_SLAM
        %% Futuro: LiDAR com Cartographer
```

```
    F_Lidar["LiDAR"] -.-> G_Cartographer["Cartographer (ROS)"]
  end

  %% Localização com Sensor Fusion
  subgraph Localizacao
    direction TB
    D_SLAM --> H_SensorFusion["Sensor Fusion (Câmera + IMU)"]
    I_IMU["IMU"] --> H_SensorFusion
  end

  %% Planejamento da Rota e Obstáculos
  subgraph Planejamento
    direction TB
    H_SensorFusion --> J_Rota["Planejamento de Rota (A*, Dijkstra, DWA)"]
    K_Obs["Sensores de Obstáculos (Ultrassônico, etc.)"] --> L_Evit["Evitar Obstáculos"]
    L_Evit --> J_Rota
  end

  %% Comunicação CAN e Controle
  subgraph Comunicacao_e_Controle
    direction TB
    J_Rota --> M_CAN["CAN Bus (Raspberry Pi ↔ Arduino)"]
    M_CAN --> N_Controle["Controle de Movimento (ESC & Servo)"]
    %% Feedback do Atuador para Ajuste Dinâmico
    N_Controle -- "Feedback/Confirmação" --> J_Rota
  end

  %% Ciclo de Missão: Destino e Retorno
  subgraph Ciclo_de_Missao
    direction TB
    J_Rota -- "Destino Alcançado" --> O_Espera["Espera / Parada"]
    O_Espera --> P_Retorno["Planejamento de Retorno"]
    P_Retorno --> M_CAN
  end
```

👨‍💻 Autor
Este projeto foi desenvolvido com fins didáticos para apresentar conhecimentos técnicos em ADAS, robótica móvel e software embarcado em contextos automotivos. Ideal para demonstrar competências em oportunidades como Engenheiro de Software ADAS.

🧪 Esboço de Código Python (Simplificado)

```
import serial
import time
from slam_module import SLAM, Planner, Localizer
from motor_control import MotorController
from sensor import read_ultrasonic, read_wheel_slip

# Inicializações
slam = SLAM()
```

```
planner = Planner()
localizer = Localizer()
motor = MotorController(port='/dev/ttyUSB0')

# Carregar mapa ou iniciar mapeamento
slam.load_or_start_mapping()

# Definir destino
goal = (2.5, 3.0)  # Coordenadas do quarto

while True:
    image = slam.capture_image()
    slam.update_map(image)

    position = localizer.estimate_position()
    path = planner.plan(position, goal)

    # AEB
    if read_ultrasonic() < 0.4:
        motor.stop()
        print("AEB acionado: obstáculo detectado")
        continue

    # Controle de tração (simples)
    if read_wheel_slip():
        motor.reduce_power()
        print("Controle de tração: reduzindo torque")

    command = planner.next_move(position, path)
    motor.send_command(command)

    if planner.reached_goal(position, goal):
        motor.stop()
        print("Destino alcançado!")
        break

    time.sleep(0.1)
```

🔧 Código Exemplo de Comunicação CAN (MCP2515)

Raspberry Pi (Python - envio de comandos)

```
import can
import time

bus = can.interface.Bus(channel='can0', bustype='socketcan')

msg = can.Message(arbitration_id=0x123, data=[30, 90], is_extended_id=False)  # 30:
velocidade, 90: direção
```

```
try:
    while True:
        bus.send(msg)
        print("Comando enviado via CAN")
        time.sleep(0.1)
except KeyboardInterrupt:
    print("Encerrando")
```

Arduino (Recepção com biblioteca MCP_CAN)

```
#include <mcp_can.h>
#include <SPI.h>

const int SPI_CS_PIN = 10;
MCP_CAN CAN(SPI_CS_PIN);

void setup() {
  Serial.begin(115200);
  if (CAN.begin(MCP_ANY, CAN_500KBPS, MCP_8MHZ) != CAN_OK) {
    Serial.println("Erro ao iniciar CAN");
    while (1);
  }
  CAN.setMode(MCP_NORMAL);
  Serial.println("CAN pronto");
}

void loop() {
  if (CAN_MSGAVAIL == CAN.checkReceive()) {
    byte len = 0;
    byte buf[8];
    CAN.readMsgBuf(&len, buf);

    int velocidade = buf[0];
    int direcao = buf[1];

    Serial.print("Velocidade: "); Serial.print(velocidade);
    Serial.print(" | Direcao: "); Serial.println(direcao);

    // PWM nos pinos do ESC e servo
  }
}
```

🧪 Simulação Tinkercad (proposta)

- Conecte o **Arduino Uno** com **MCP2515**
- Emule envio de dados CAN por outro Arduino (ou visualize entrada via Serial)
- Configure pinos: `CS = 10`, `MISO = 12`, `MOSI = 11`, `SCK = 13`

👉 Para o Raspberry, a simulação real exige hardware (não suportado diretamente pelo Tinkercad).

📄 Documento Técnico (Whitepaper)

Título: Plataforma Experimental de Veículo Autônomo Escalonado com Navegação Baseada em SLAM e Comunicação CAN

Resumo: Este projeto visa implementar um sistema funcional de direção autônoma em escala reduzida, utilizando arquitetura modular embarcada distribuída baseada em Raspberry Pi e Arduino, comunicação via CAN, SLAM para navegação indoor, e funcionalidades ADAS como AEB e controle de tração. O sistema se aproxima da lógica usada em protótipos reais de veículos autônomos.

Palavras-chave: ADAS, SLAM, CAN bus, Veículo Autônomo Escalonado, ESC, Servo, Sensor Ultrassônico, Python, C++, ROS

Seções:

✅ Documento ideal para anexar ao currículo como projeto de portfólio de ADAS.

Se desejar, posso exportar este documento como PDF ou gerar um modelo de apresentação profissional também.