

**UNIVERSIDADE DO VALE DO ITAJAÍ  
ESCOLA DO MAR, CIÊNCIA E TECNOLOGIA  
CURSO DE ENGENHARIA DE COMPUTAÇÃO  
PROJETO DE SISTEMAS EMBARCADOS**

**SISTEMA DE RASTREAMENTO E MOVIMENTAÇÃO  
AUTÔNOMA COM VISÃO COMPUTACIONAL E  
PROCESSAMENTO REMOTO**

por

Fernando Gabriel Trentin  
Nicolas Mendonça Melo Fajardo

Itajaí (SC), dezembro de 2025

**UNIVERSIDADE DO VALE DO ITAJAÍ  
ESCOLA DO MAR, CIÊNCIA E TECNOLOGIA  
CURSO DE ENGENHARIA DE COMPUTAÇÃO  
PROJETO DE SISTEMAS EMBARCADOS**

**SISTEMA DE RASTREAMENTO E MOVIMENTAÇÃO  
AUTÔNOMA COM VISÃO COMPUTACIONAL E  
PROCESSAMENTO REMOTO**

por

Fernando Gabriel Trentin  
Nicolas Mendonça Melo Fajardo

Relatório apresentado como requisito parcial da disciplina Projeto de Sistemas Embarcados do Curso de Engenharia de Computação para análise e aprovação.

Professores Responsáveis: Paulo Roberto Valim  
Felipe Viel

Itajaí (SC), dezembro de 2025

## SUMÁRIO

<b>1 INTRODUÇÃO.....</b>	<b>4</b>
1.1 DEFINIÇÃO DO PROBLEMA.....	4
1.2 SOLUÇÕES EXISTENTES.....	4
1.3 SOLUÇÃO PROPOSTA .....	7
1.4 ANÁLISE DE VIABILIDADE.....	8
<b>2 PROJETO .....</b>	<b>9</b>
2.1 VISÃO GERAL.....	9
2.2 PREMISSAS.....	11
2.3 ANÁLISE DE REQUISITOS.....	12
2.3.1 Requisitos funcionais.....	12
2.3.2 Requisitos não funcionais.....	13
2.3.3 Regras de negócio .....	14
2.4 ARQUITETURA DE HARDWARE.....	15
2.5 ARQUITETURA DE SOFTWARE.....	19
2.6 PLANEJAMENTO .....	22
2.7 CRONOGRAMA .....	24
2.8 ANÁLISE DE RISCOS.....	25
<b>3 DESENVOLVIMENTO .....</b>	<b>26</b>
3.1 IMPLEMENTAÇÃO .....	26
3.2 VERIFICAÇÃO.....	42
3.3 RESULTADOS.....	43
<b>4 CONSIDERAÇÕES FINAIS / CONCLUSÕES .....</b>	<b>ERRO! INDICADOR NÃO DEFINIDO.</b>

# 1 INTRODUÇÃO

## 1.1 DEFINIÇÃO DO PROBLEMA

A utilização de visão computacional em sistemas embarcados tem se tornado cada vez mais comum em projetos de robótica móvel e Internet das Coisas (IoT). A identificação de objetos, o reconhecimento de trajetos e a navegação autônoma são funcionalidades essenciais para aplicações como veículos inteligentes, robôs educacionais e sistemas de monitoramento. No entanto, implementar essas capacidades em plataformas de baixo custo, como o ESP32-CAM, apresenta desafios relevantes devido às limitações de processamento, memória e estabilidade da comunicação sem fio.

No entanto, surgem desafios que impedem o funcionamento eficiente do sistema. Dispositivos como o ESP32 possuem recursos limitados para análises visuais em tempo real, o que compromete tarefas como desvio de obstáculos, seguimento de linhas ou identificação de alvos. A transmissão contínua de imagens por Wi-Fi também introduz atrasos, perda de dados e instabilidade, fatores que afetam diretamente a tomada de decisão do robô.

Dessa forma, o problema central do projeto consiste em desenvolver um sistema embarcado capaz de capturar imagens, processá-las com eficiência e tomar decisões autônomas em tempo real, mesmo sob forte restrição de hardware e comunicação. O desafio inclui integrar a ESP32-CAM e o computador externo de forma estável, com baixa latência, alta confiabilidade e controle preciso dos motores do carrinho.

O código e a apresentação deste projeto estão disponíveis em:  
<https://github.com/fetrentin7/SelfDrivingCar>

## 1.2 SOLUÇÕES EXISTENTES

Existem diversas soluções que exploram o uso do módulo ESP32-CAM em conjunto com visão computacional para detecção de objetos e controle de robôs móveis, tanto em tutoriais abertos quanto em kits comerciais.

Uma das referências mais próximas do problema deste trabalho aparece no artigo do site *Electronic Clinic*. O autor apresenta um sistema em que o ESP32-CAM envia o fluxo de vídeo por Wi-Fi para um computador, e o processamento de detecção de objetos ocorre em Python

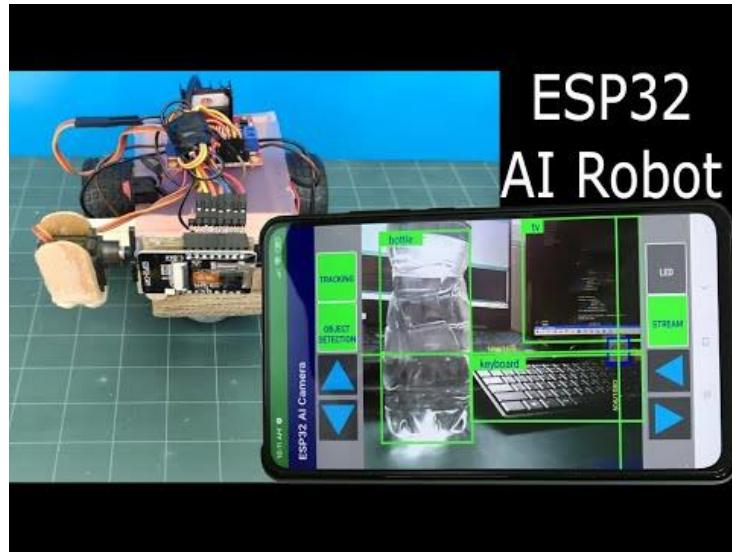
com OpenCV e o modelo YOLOv3. Nesse arranjo, o ESP32-CAM exerce apenas a função de captura e transmissão das imagens, enquanto o computador realiza a inferência e pode acionar alertas. Embora esse exemplo mostre detecção de objetos em tempo quase real com hardware de baixo custo, o foco recai sobre a identificação de classes em uma cena fixa, e não sobre o controle contínuo de um robô móvel que precisa reagir ao ambiente.

Figura 1 – Uso da esp32 com o Yolo V3



Em um vídeo do YouTube intitulado “*ESP32-CAM: Simple AI Robot (Object Detection | Object ...)*”, o autor apresenta um robô simples com ESP32-CAM que executa detecção de objetos e aciona a movimentação da plataforma com base nas informações visuais. [YouTube](#) O projeto demonstra que é possível integrar visão computacional e atuação em um carrinho autônomo de baixo custo. Porém, a solução se concentra em um protótipo fechado, sem detalhamento de arquitetura de software, estratégias de comunicação com o computador hospedeiro ou critérios quantitativos de avaliação de desempenho.

Figura 2 – Detecção de uma pista de corrida usando ESP32-CAM



Há também relatos em comunidades colaborativas, como o tópico “*Realtime object tracking robot using OpenCV*” no Reddit, em que usuários descrevem robôs com ESP32-CAM e OpenCV para rastreamento de objetos em tempo real. Nesses casos, o computador recebe as imagens, calcula a posição do alvo e envia comandos ao robô. Esses projetos funcionam como provas de conceito, mas normalmente carecem de documentação estruturada, análise formal de requisitos e discussão sistemática sobre latência, perdas na rede e limitações de hardware.

Figura 3 – Detecção de objetos com OpenCV e ESP32.



No âmbito acadêmico, alguns trabalhos recentes já avaliam o uso do ESP32-CAM em tarefas de detecção de objetos com modelos da família YOLO, destacando a viabilidade da placa para identificar múltiplas classes em fluxos de vídeo sob recursos limitados. Outros artigos descrevem plataformas móveis de inspeção e vigilância baseadas em câmeras e

processamento de visão autônoma ou semiautônoma, em geral com foco em monitoramento de ambientes internos e aplicações de segurança.

### **1.3 SOLUÇÃO PROPOSTA**

Nesta seção, defina a solução proposta sem apresentar detalhes (estes serão fornecidos posteriormente). Também procure comparar com as soluções existentes de modo a identificar o diferencial do que está sendo proposto e o grau/tipo de inovação da sua solução em relação às soluções similares. Faça um quadro comparativo com os atributos das soluções existentes e da solução proposta, incluindo as métricas e tecnologias que as caracterizam.

Quadro 1 – Comparativo da solução proposta com as soluções existentes

<b>Nome</b>	<b>Características</b>	<b>Funcionamento</b>	<b>Projeto</b>	<b>Documentação</b>
ESP32-CAM + YOLOv3	Processamento externo no computador	Detecção de objetos em tempo quase real	Projeto demonstrativo	Exige configuração manual extensa
Robô ESP32-CAM (YouTube)	Integra câmera e motores	Movimentação reativa simples	Projeto demonstrativo	Sem documentação técnica estruturada
Rastreamento ESP32 + OpenCV (Reddit)	Rastreamento de objetos no PC	Comandos enviados ao robô	Funciona como prova de conceito	Ausência de análise de latência e desempenho
Solução proposta	Coneção com duas câmeras ESP	Detecção de objetos e reconhecimento de trajetórias	Controle autônomo em tempo real	Análise de latência e desempenho

### **1.4 MERCADO**

A solução proposta se insere no mercado de visão computacional aplicada à automação e sistemas móveis inteligentes, um setor que cresce impulsionado pela adoção de algoritmos de detecção de objetos, rastreamento e navegação autônoma em diferentes áreas. Os potenciais clientes incluem empresas de tecnologia, startups, laboratórios de pesquisa e organizações que precisam testar rapidamente protótipos de robôs móveis, sistemas de monitoramento, inspeção visual ou aplicações de IA embarcada. Esses clientes valorizam soluções de baixo custo e fácil

integração, capazes de capturar imagens, estimar profundidade e realizar ajustes automáticos de orientação para acompanhamento de alvos.

O mercado global de visão computacional está crescendo, e mesmo uma pequena participação por meio de kits de prototipagem, serviços de adaptação ou venda de módulos integrados já representa um potencial comercial relevante. Um produto derivado desta solução poderia gerar receita por meio da venda do kit completo, licenciamento do software ou serviços de customização para aplicações específicas, atendendo empresas e centros que buscam validar ideias de forma rápida e acessível.

## 1.5 ANÁLISE DE VIABILIDADE

A solução proposta mostra-se viável dentro do prazo da disciplina porque utiliza componentes acessíveis, amplamente documentados e já consolidados na comunidade de sistemas embarcados. O ESP32-CAM e a ESP32 dedicada ao controle dos motores possuem bibliotecas estáveis, exemplos oficiais e suporte extenso, o que reduz significativamente o tempo necessário para configuração, testes e integração dos módulos. Além disso, o processamento visual ocorre em um computador externo, o que elimina a necessidade de treinar ou executar modelos complexos diretamente no dispositivo embarcado. Isso diminui a carga computacional do protótipo e torna o desenvolvimento mais rápido e previsível.

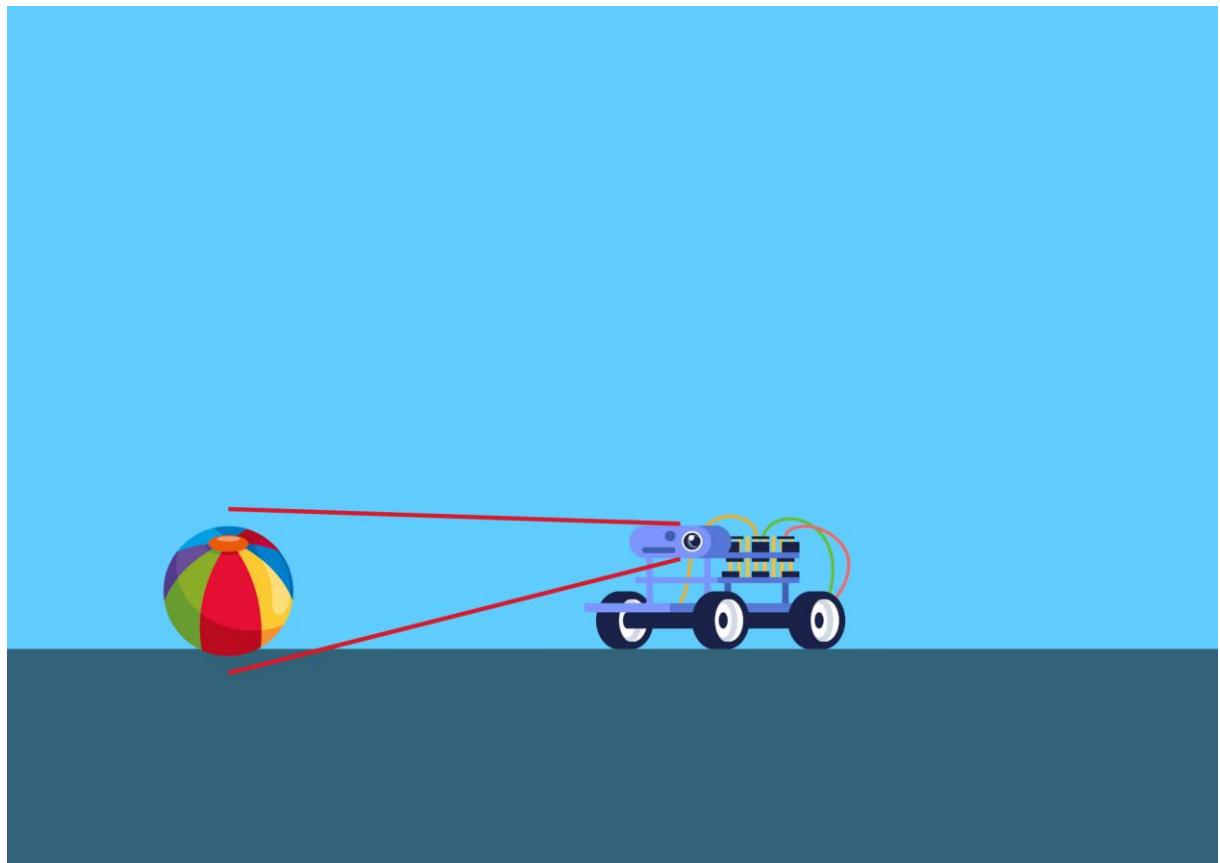
A arquitetura proposta também facilita a implementação incremental. O projeto pode evoluir em etapas independentes, como: captura de imagens, transmissão via Wi-Fi, detecção de objetos, processamento no computador, envio de comandos e controle dos motores

## 2 PROJETO

### 2.1 VISÃO GERAL

A primeira imagem mostra o carrinho equipado com duas câmeras localizadas na extremidade frontal. As câmeras estão orientadas para um objeto à frente, representado por uma bola colorida. Os feixes vermelhos simbolizam o campo de visão e a linha de detecção utilizada pelo algoritmo.

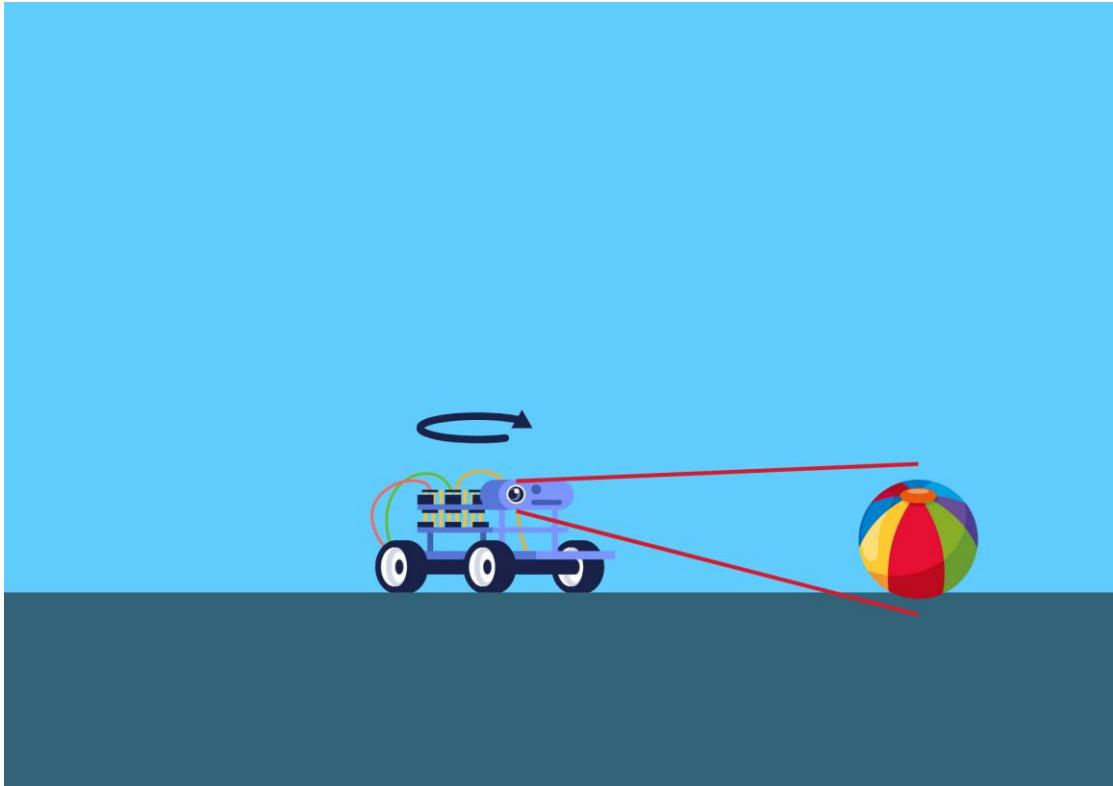
Figura 4 – Representação da detecção do objeto



A segunda imagem mostra o momento em que a câmera realiza um movimento de correção angular (rotacionando para a esquerda ou direita) para manter o objeto centralizado no campo de visão. A seta circular indica o giro do módulo onde as câmeras estão instaladas, enquanto as linhas vermelhas novamente representam o alinhamento entre o sistema de visão e o objeto.

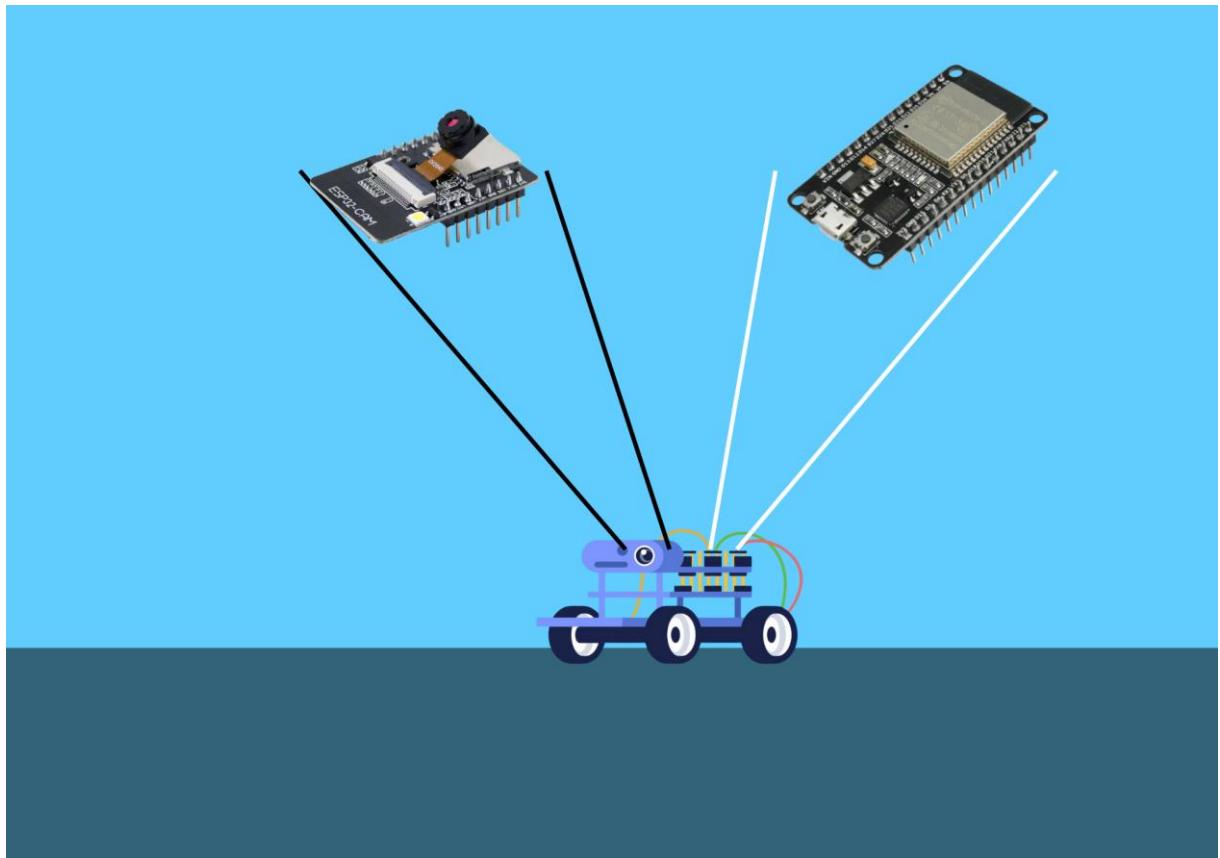
Nesta etapa, o sistema identifica que o alvo está deslocado lateralmente e inicia o ajuste automático da inclinação e orientação da câmera, conforme descrito nos requisitos funcionais. Esse processo garante que o alvo permaneça no centro da imagem.

Figura 5 – Movimentação do robô



Nesta terceira imagem, é ilustrada a disposição dos principais componentes embarcados utilizados no sistema. À esquerda, observa-se o módulo ESP32-CAM, responsável pela captura das imagens do ambiente por meio da câmera OV2640 e pelo envio do fluxo de vídeo via Wi-Fi para o computador, onde ocorre o processamento e a detecção do alvo. À direita, encontra-se o módulo ESP32, responsável por receber, também via Wi-Fi, os comandos gerados pelo algoritmo e controlar os motores da base móvel, além de ajustar o servo que altera a inclinação da câmera para manter o objeto centralizado. Ambos os módulos estão conectados à estrutura do carrinho, que abriga os motores e os fios necessários para a alimentação e comunicação entre os componentes.

Figura 6 – Robô com os componentes



## 2.2 PREMISSAS

As premissas definem o conjunto de condições que se espera encontrar durante a execução e operação do projeto. Elas servem como base para as decisões técnicas e para o planejamento das etapas, permitindo prever o comportamento do sistema em um cenário idealizado. A seguir, descrevem-se os principais pontos assumidos como verdadeiros para o desenvolvimento deste trabalho.

1. **Conectividade adequada:** Considera-se que o ambiente disponibilizará uma rede Wi-Fi estável e suficiente para suportar o envio e recebimento contínuo de dados entre o computador e o ESP32, evitando interrupções na comunicação.
2. **Iluminação mínima garantida:** Assume-se que o local onde o sistema será utilizado contará com um nível básico de iluminação, suficiente para que a câmera consiga capturar imagens com nitidez e identificar corretamente elementos visuais.

3. **Configuração fixa das câmeras:** Parte-se da ideia de que a posição relativa das câmeras e suas características ópticas não sofrerão alterações durante o uso, já que variações nesse arranjo comprometeriam a estimativa de profundidade.
4. **Integridade dos equipamentos:** Supõe-se que todos os dispositivos envolvidos — câmeras, microcontroladores e demais módulos — operarão conforme seus padrões de fábrica, sem falhas que prejudiquem a coleta ou o processamento dos dados.
5. **Modelos de detecção prontos para uso:** Assume-se que os algoritmos de detecção envolvidos já terão sido treinados, validados e ajustados antes de sua aplicação no hardware, garantindo que a integração se concentre apenas na execução.
6. **Computador com desempenho adequado:** Considera-se que o equipamento utilizado para o processamento das informações terá recursos suficientes para executar as tarefas propostas sem comprometer o desempenho geral do sistema.

## 2.3 ANÁLISE DE REQUISITOS

A análise de requisitos tem como objetivo estabelecer, de forma organizada e clara, tudo aquilo que o sistema deve realizar e as condições sob as quais essas funcionalidades serão executadas. Essa etapa orienta tanto o desenvolvimento do software embarcado quanto a aplicação de alto nível responsável pelo processamento e pela interface de controle. Os requisitos apresentados a seguir definem o comportamento esperado do sistema, dividindo-o entre funcionalidades essenciais e limitações técnicas que influenciam sua implementação.

### 2.3.1 Requisitos funcionais

Os requisitos funcionais representam aquilo que o sistema precisa fazer para cumprir seus objetivos. Eles descrevem as ações, respostas e comportamentos esperados durante a operação. Ao longo do desenvolvimento, cada um desses requisitos servirá de base para validações, testes e verificações, assegurando que o sistema entregue exatamente o que foi planejado.

#### 2.3.1.1 Requisitos Funcionais — Sistema Embarcado

- **RF01:** O sistema deve receber as posições cartesianas do objeto.

- **RF02:** O sistema deve realizar os cálculos de controle com os dados recebidos.
- **RF03:** O sistema deve utilizar os valores de controle para deixar o objeto centralizado.
- **RF04:** O sistema deve utilizar os valores de controle para manter sempre uma distância do objeto.
- **RF05:** O sistema deve parar caso a conexão seja perdida.
- **RF06:** O sistema deverá realizar transmissão remota de imagens.
- **RF07:** O sistema deve estar montado em uma estrutura única.
- **RF08:** O sistema deve ter alimentação própria.

### 2.3.1.2 Requisitos Funcionais — Alto Nível

- **RF01:** O programa deve receber as imagens enviadas remotamente.
- **RF02:** O programa deve detectar o objeto a ser reconhecido.
- **RF03:** O programa deve calcular as posições cartesianas do objeto.
- **RF04:** O programa deve mostrar as posições calculadas.
- **RF05:** O programa deve enviar uma mensagem caso o alvo não seja encontrado.
- **RF06:** O programa deve realizar a transmissão das posições calculados.
- **RF07:** O programa deve receber as imagens enviadas remotamente.
- **RF08:** O programa deve exibir as imagens recebidas.

## 2.3.2 Requisitos não funcionais

Os requisitos não funcionais definem restrições, padrões de qualidade e condições técnicas que o sistema deve obedecer. Eles não descrevem ações diretas, mas estabelecem limites, tecnologias adotadas, tempo de resposta, desempenho esperado, confiabilidade da comunicação e características de segurança. Esses fatores são fundamentais para garantir não apenas que o sistema funcione, mas que funcione bem e dentro das expectativas de usabilidade e eficiência.

### 2.3.2.1 Requisitos Não Funcionais — Sistema Embarcado

- **RNF01:** O sistema deve ser implementado em linguagem C.
- **RNF02:** O sistema deve ser implementado em ESP32 DEVKIT1
- **RNF03:** O sistema deverá utilizar ESPCAMs para capturar as imagens.

- **RNF04:** O sistema deve estar em uma estrutura impressa em 3D.
- **RNF05:** O recebimento das posições deve ser via Wi-Fi e utilizar o protocolo TCP.
- **RNF06:** O sistema deve utilizar protocolo HTTP para streaming de imagens.
- **RNF07:** O sistema deve utilizar motores DC para movimentação.
- **RNF08:** O sistema deve utilizar servo motor para controle do ângulo das câmeras.
- **RNF09:** Os motores devem ser controlados por uma Ponte-H.
- **RNF10:** O sistema deve ser alimentado por duas baterias LiPO 18650 3.7v.

### **2.3.2.2 Requisitos Não Funcionais — Alto Nível**

- **RNF01:** O sistema deve ser implementado em Python utilizando Numpy e OpenCV.
- **RNF02:** O processamento de detecção não deve exceder o tempo estabelecido pela captura dos frames.
- **RNF03:** A comunicação do sistema por Wi-Fi terá protocolo TCP.
- **RNF04:** O sistema deve utilizar processamento estereoscópico para identificar a distância do objeto para o sistema embarcado.

### **2.3.3 Regras de negócio**

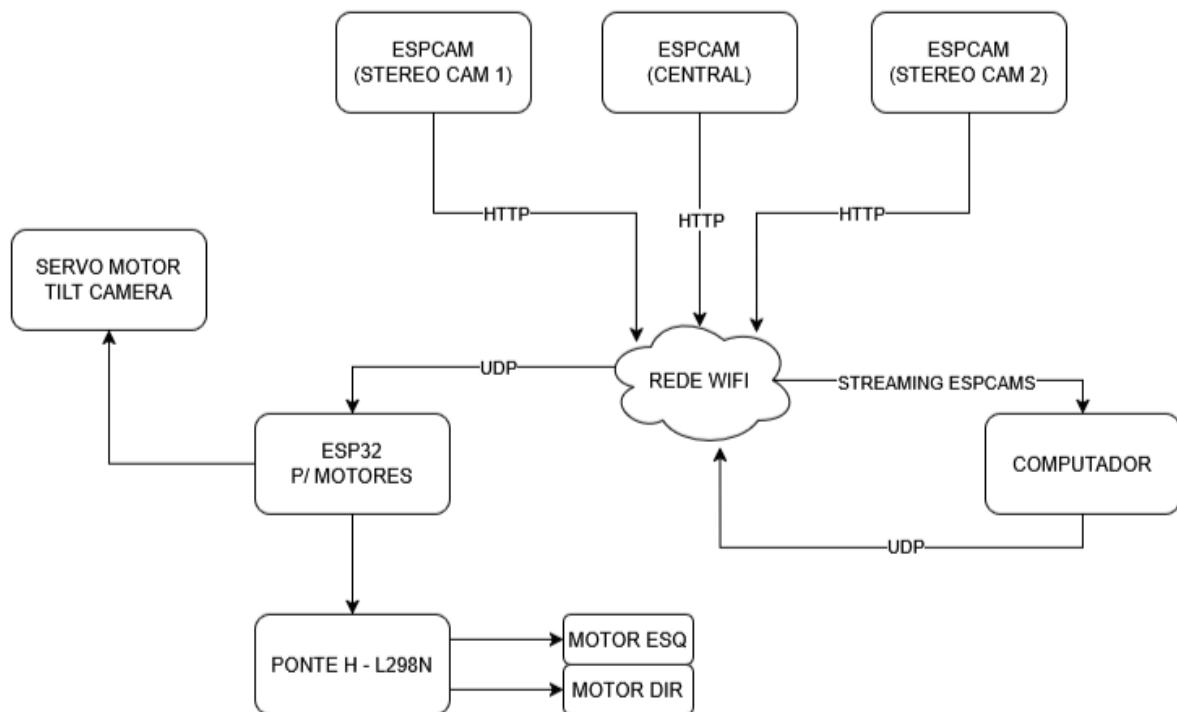
Nesta seção, apresente as regras de negócio do sistema a ser desenvolvido.

- **RN01:** Se o alvo desaparecer do campo de visão o carrinho deve enviar uma mensagem ao operador e aguardar um novo alvo, ou comando externo
- **RN02:** O sistema terá sua comunicação interrompida em caso de risco
- **RN03:** O sistema deve ajustar automaticamente a orientação da câmera para manter o objeto centralizado, quando detectado.
- **RN04:** Em caso de perca temporária de pacotes o sistema deve ser reconectado.
- **RN05:** O cálculo da distância será considerado se somente ambas as câmeras estiverem funcionando corretamente.
- **RN06:** O carrinho só pode iniciar o movimento após a conexão ser estabelecida.
- **RN07:** Os modelos de detecção devem ser leves para operar dentro das restrições de hardware;
- **RN08:** Em caso de falha de detecção ou inconsistência dos dados, o sistema deve priorizar parar.

## 2.4 ARQUITETURA DE HARDWARE

A arquitetura de hardware do sistema foi planejada para integrar, de forma eficiente, os módulos responsáveis pela captura de imagens, pela comunicação sem fio e pelo controle da base móvel. O conjunto é composto por quatro blocos principais: o subsistema de visão (três módulos ESP32-CAM), o microcontrolador responsável pela atuação (ESP32 DevKit), o estágio de potência (ponte H L298N) e os elementos mecânicos de movimentação (motores DC e servo motor). A Figura 7 apresenta o diagrama de blocos que resume essa estrutura.

Figura 7 – Diagrama de blocos do hardware



### 2.4.1 Visão Geral da Arquitetura

O sistema utiliza três ESP32-CAMs posicionadas na parte frontal do robô: duas câmeras laterais destinadas à captura estereoscópica e uma câmera central dedicada ao fluxo de visão principal. Todas as câmeras conectam-se à rede Wi-Fi local e enviam seus fluxos de vídeo para o computador, responsável por processar os frames, realizar a detecção de objetos e calcular a profundidade da cena.

O computador, após processar os dados das câmeras, envia comandos ao módulo ESP32 responsável pelo controle da base móvel. Esse módulo interpreta as instruções recebidas via UDP e altera o estado dos motores e do servo motor. Toda a comunicação do sistema ocorre pela mesma rede Wi-Fi, que serve como infraestrutura central de dados.

## **2.4.2 VISÃO GERAL DA ARQUITETURA**

### **a. Módulos ESP32-CAM**

As três ESP32-CAM são responsáveis pela aquisição de imagens. Cada módulo incorpora:

- Sensor fotográfico OV2640
- Conexão Wi-Fi para streaming de vídeo via HTTP
- Capacidade de configuração remota via API interna

A câmera central fornece a visão primária utilizada para detecção e rastreamento do alvo.

As **duas câmeras laterais** são posicionadas de forma simétrica para permitir cálculo de profundidade utilizando visão estereoscópica, contribuindo para o requisito funcional RF04.

Todas as ESP32-CAM são alimentadas com 5 V, com aterramento comum ao restante do circuito.

### **b. ESP32 DevKit – Módulo de Controle da Base Móvel**

Um ESP32 dedicado é utilizado exclusivamente para:

- Recebimento de comandos do computador via UDP
- Controle dos motores DC por meio da ponte H L298N
- Controle do servo que ajusta o tilt da câmera principal
- Gerenciamento de conexão e fail-safe em caso de perda de comunicação

Esse módulo cumpre os requisitos RF03, RF04 e RNF05.

A utilização de um ESP32 separado evita sobrecarga nos módulos de captura de imagem, garantindo maior estabilidade no controle do robô.

### c. Servo Motor (Tilt da Câmera)

O servo motor é conectado ao ESP32 DevKit e tem a função de ajustar o ângulo vertical da câmera central, permitindo que o sistema mantenha o objeto detectado dentro do campo de visão mesmo quando a altura relativa do alvo varia.

O sinal PWM de controle é gerado diretamente pelo ESP32, enquanto a alimentação (5 V) provém da mesma fonte das câmeras.

### d. Ponte H L298N – Estágio de Potência

A ponte H L298N é utilizada para o controle dos dois motores DC responsáveis pela tração do carrinho. Ela recebe:

- Alimentação de 7,4 V proveniente da bateria
- Sinais lógicos de direção e PWM enviados pelo ESP32 DevKit
- Aterramento comum ao restante do sistema

A ponte H fornece a corrente necessária aos motores e permite o controle independente de cada lado, possibilitando movimentos como avanço, ré, curvas e rotações.

### e. Motores DC da Base Móvel

Os motores DC de 6 V são acoplados às rodas e formam o sistema de locomoção principal. O controle individual de cada motor permite manobras precisas conforme os comandos do processamento de visão.

### f. Rede Wi-Fi

A rede Wi-Fi atua como o barramento lógico que integra todos os módulos. Por meio dela:

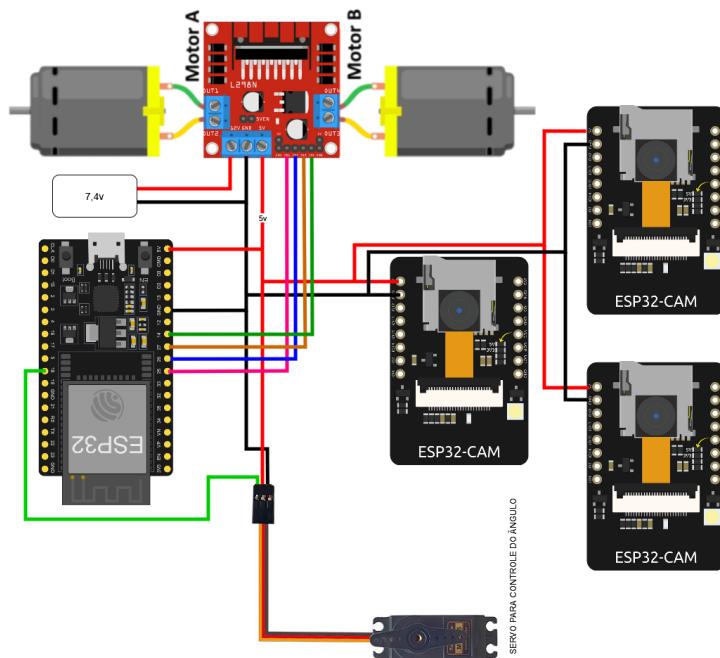
- As ESP32-CAM enviam streaming por HTTP ao computador;
- O computador processa os frames e envia comandos via UDP ao ESP32;
- O ESP32 executa os comandos e atualiza o movimento do robô.

A estabilidade da rede é uma premissa (Premissa 1), fundamental para o funcionamento do sistema em tempo real.

### 2.4.3 DIAGRAMA ESQUEMÁTICO

A Figura 8 apresenta o diagrama elétrico simplificado do sistema, incluindo conexões de alimentação, sinais de controle e interligação dos módulos.:

Figura 8 – Diagrama esquemático das conexões



O diagrama mostra:

- Conexão das ESP32-CAM ao barramento de 5 V
- Ligação do ESP32 DevKit ao servo motor e à ponte H
- Alimentação dos motores com 7,4 V

- Aterramento unificado entre todos os módulos
- Cabos de sinal entre ESP32 e L298N

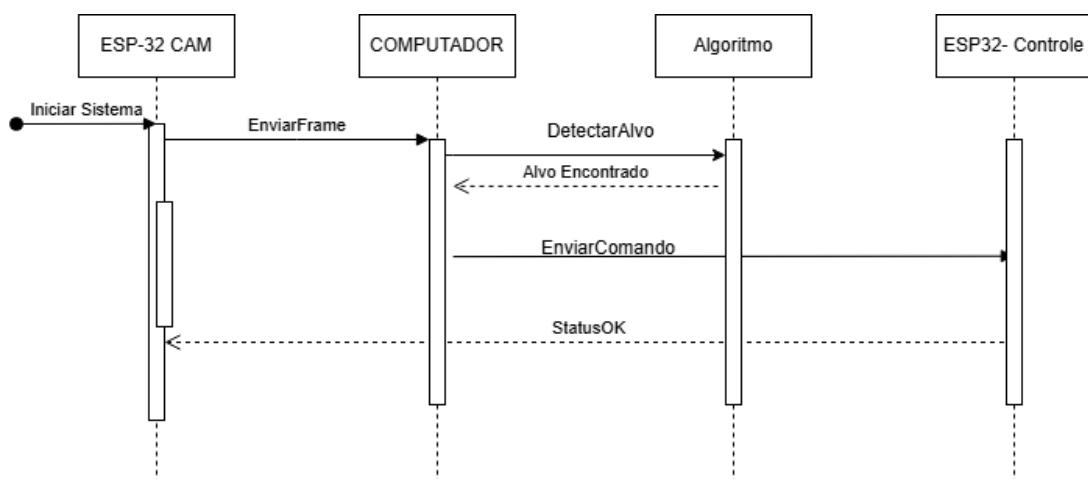
O diagrama completo, incluindo variações de alimentação e referências de componentes, será apresentado nos apêndices do relatório.

## 2.5 ARQUITETURA DE SOFTWARE

Nesta esta seção apresenta-se a arquitetura de software do sistema, mostrando como os componentes se relacionam para capturar imagens, processá-las e controlar o carrinho. Primeiro, exibe-se o fluxograma geral, que representa o fluxo dos dados desde a captura até a ação realizada. Em seguida, apresenta-se o diagrama de casos de uso, que identifica os atores do sistema e as principais funcionalidades, permitindo visualizar claramente as operações executadas.

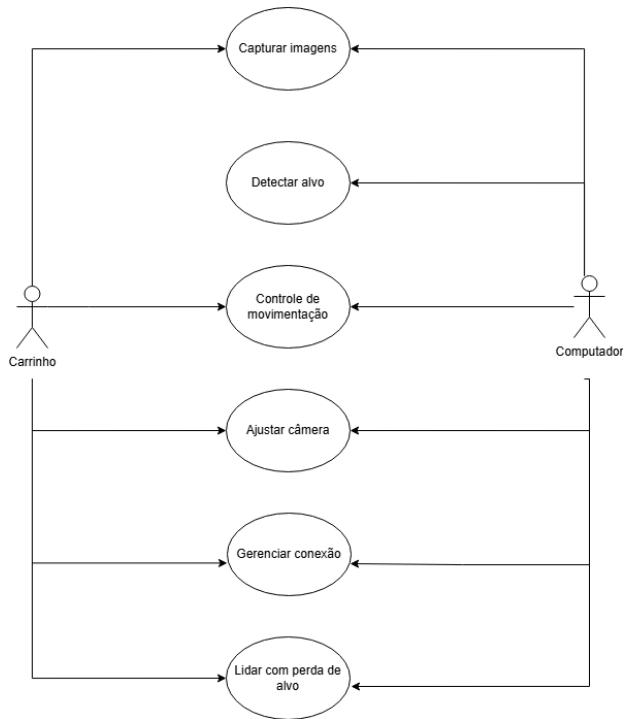
O diagrama de sequência ilustra a troca de mensagens entre os componentes do sistema durante a operação. A ESP32-CAM inicia o processo enviando os frames capturados ao computador. O computador delega o processamento ao módulo de detecção, que identifica a presença do alvo e retorna essa informação. Em seguida, o computador envia comandos ao ESP32-Controle, responsável por acionar os motores e o servo conforme a decisão tomada. Após executar a ação, o módulo de controle retorna uma confirmação de status, permitindo a continuidade do ciclo de rastreamento

Figura 8 – Diagrama de sequência



O diagrama de casos de uso mostra como Computador e Carrinho interagem durante a operação do sistema. O Computador captura as imagens, detecta o alvo e decide os comandos de movimentação. O Carrinho executa esses comandos, ajusta a câmera para manter o alvo centralizado e gerencia a conexão para garantir comunicação contínua. Quando o alvo deixa de ser detectado, o sistema aciona o caso de uso Lidar com Perda de Alvo, colocando o carrinho em um estado seguro até que novas instruções sejam enviadas. O quadro a seguir apresenta a justificativa de cada caso de uso.

Figura 9 – Diagrama de casos de uso



Quadro 2 - Justificativa para os Casos de Uso

Caso de Uso	Atores	Justificativa
Capturar Imagens	Carrinho e Computador	O Carrinho captura a imagem, mas o Computador é quem a recebe via Wi-Fi para processamento
Detectar Alvo	Computador	O processamento e a detecção do alvo ocorrem no computador externo.
Controle de Movimentação	Carrinho e Computador	O Computador gera os comandos de movimento e o Carrinho recebe para controlar os motores da base.
Ajustar Câmera	Carrinho e Computador	O Computador envia o comando de ajuste, e o Carrinho o recebe para controlar o servo que regula a inclinação da câmera.
Gerenciar conexão	Carrinho e computador	Ambos os módulos estão envolvidos na comunicação via Wi-Fi
Lidar com perda de Alvo	Carrinho e computador	O Computador detecta a ausência do alvo, e o Carrinho deve aguardar o novo comando ou alvo

## 2.6 LISTA DE MATERIAIS E ORÇAMENTO

A seguir apresenta-se a lista dos materiais necessários para a construção do protótipo, incluindo a disponibilidade atual dos itens e os responsáveis por sua aquisição. Todos os componentes utilizados encontram-se acessíveis comercialmente e compatíveis com as restrições de custo e tempo da disciplina.

Quadro 3 – Lista de materiais

Item	Qtd	Disponível	Responsável(is)
Kit de desenvolvimento ESP32	01	Sim	Alunos
ESPCAM	03	Sim	Alunos
Filamento para Impressão 3D	01	Sim	Alunos
Servo Motor	01	Sim	Alunos
Motor DC 6V	02	Sim	Alunos
Bateria 3.7V	02	Sim	Alunos
Jumpers	20	Sim	Alunos
Ponte H L298N	01	Sim	Alunos

Todos os itens encontram-se disponíveis pelos integrantes do grupo, não havendo necessidade de aquisição adicional para a construção do protótipo.

Quadro 4 – Orçamento

<b>Item</b>	<b>Custo unitário</b>	<b>Qtd</b>	<b>Custo parcial</b>	<b>Fornecedor de referência</b>
Kit de desenvolvimento ESP32	R\$ 40,00	01	R\$ 40,00	Mercado Livre
ESPCAM	R\$ 70,00	03	R\$ 210,00	Mercado Livre
Filamento para Impressão 3D	R\$ 90,00	01	R\$ 90,00	Mercado Livre
Servo Motor	R\$ 20,00	01	R\$ 20,00	Mercado Livre
Motor DC 6V	R\$ 7,50	02	R\$ 15,00	RoboCore
Bateria 3.7V	R\$ 25,00	02	R\$ 50,00	Mercado Livre
Jumpers	R\$ 0,25	20	R\$ 10,00	Mercado Livre
Ponte H L298N	R\$ 20,00	01	R\$ 70,00	Mercado Livre

## 2.7 PLANEJAMENTO

A validação do sistema foi conduzida através de testes unitários e de integração, confrontando o funcionamento prático do protótipo com os requisitos levantados na etapa de concepção. A abordagem adotada consistiu na verificação por inspeção (para aspectos físicos e estruturais) e testes de execução dinâmica (para validação de lógica e controle).

A seguir, são apresentados os roteiros de testes utilizados. O Quadro 5 detalha a verificação do Sistema Embarcado, abrangendo desde a alimentação elétrica até a atuação dos motores via microcontrolador.

Quadro 5 – Plano de verificação – Sistema Embarcado

<b>Requisito</b>	<b>Procedimento de verificação/teste</b>	<b>Resultado esperado</b>
RF01	Enviar coordenadas de teste (X, Y, Z) via protocolo de comunicação (TCP) e monitorar o log serial do microcontrolador.	O sistema recebe os pacotes de dados e imprime no monitor serial as mesmas coordenadas enviadas, sem corrupção.
RF02	Simular um erro de posição (input) e verificar as variáveis de saída do algoritmo de controle (ex: PWM) via debug.	O sistema gera valores de atuação (saída do PID ou similar) proporcionais ao erro inserido, tentando corrigir a divergência.
RF03	Posicionar o objeto-alvo fora do centro da imagem (horizontalmente ou verticalmente).	O sistema aciona os motores (ou servo da câmera) na direção correta até que a coordenada do objeto tenda a (0,0) ou centro do frame.
RF04	Mover o objeto para uma distância diferente do setpoint (mais perto ou mais longe).	O sistema aciona os motores de tração para avançar ou recuar, estabilizando quando a distância alvo for atingida.
RF05	Interromper propositalmente a conexão de rede ou o envio de dados enquanto os motores estiverem ativos.	O sistema detecta o timeout ou queda da conexão e imediatamente zera o PWM dos motores, parando o movimento.
RF06	Acessar o endereço IP do ESP32 através de um navegador web ou software visualizador.	O fluxo de vídeo (stream) é carregado e exibido na tela com latência aceitável para o monitoramento.
RF07	Realizar inspeção visual da montagem mecânica e teste de integridade física ao mover o robô.	Todos os componentes (placas, bateria, câmera, motores) estão fixados solidamente no chassi, sem partes soltas penduradas.
RF08	Desconectar o cabo de programação/alimentação USB e ligar o sistema pela chave geral (bateria).	O sistema inicializa (LEDs acendem, conexão é estabelecida) e opera normalmente utilizando apenas a energia da bateria interna.
RNF01	Inspecionar o código-fonte.	Todo o código principal está escrito em C.
RNF02	Verificar a placa utilizada no hardware.	O firmware está carregado e executando em um ESP32 DEVKIT1.
RNF03	Conferir hardware conectado e testar reconhecimento da câmera.	O sistema reconhece a ESPCAM e consegue capturar imagens.
RNF04	Verificar fisicamente a estrutura do sistema.	A estrutura mecânica é totalmente fabricada em impressão 3D conforme projeto.
RNF05	Inspecionar o código de comunicação e monitorar pacotes.	Comunicação entre computador e ESP32 ocorre via TCP.
RNF06	Verificar protocolo utilizado pela ESP32-CAM durante o streaming.	O vídeo é transmitido corretamente via protocolo HTTP.
RNF07	Conferir motores utilizados para movimentação do sistema.	Os motores são motores DC de 6V
RNF08	Conferir motor utilizado para controle do ângulo da câmera.	É utilizado servo motor para controle do ângulo da camera
RNF09	Verificar se os motores estão sendo controlados por uma Ponte-H	Os motores são controlados por ponte-H

Simultaneamente, o módulo de visão computacional foi submetido aos testes descritos no Quadro 6, visando validar a precisão da estereoscopia, a latência do processamento e a estabilidade da comunicação TCP.

Quadro 6 – Plano de verificação – Controle da câmera

<b>Requisito</b>	<b>Procedimento de verificação/teste</b>	<b>Resultado esperado</b>
RF01	Iniciar o script Python e verificar o estabelecimento do <i>socket</i> ou conexão com o stream de vídeo.	O sistema deve processar as informações capturadas para identificar alvos e realizar desvios de obstáculos em tempo real.
RF02	Apresentar o objeto-alvo diante das câmeras em diferentes orientações e iluminações.	O carrinho recebe comandos para girar ou avançar de acordo com a posição do alvo detectado, ou parar ao identificar um obstáculo.
RF03	Posicionar o objeto em distâncias conhecidas e fixas (ex: 30cm, 50cm, 100cm) medidas com trena.	O console do computador exibe o log "Alvo Perdido" e o carrinho interrompe a movimentação imediatamente.
RF04	Executar o programa e observar o terminal (console) ou a interface gráfica (GUI).	A distância é calculada a partir de um ponto central calculado pelo lado do quadrado dividido por dois. Assim obtém-se os eixos e é realizado o deslocamento.
RF05	Retirar o objeto do campo de visão das câmeras.	O software exibe a mensagem "Objeto não encontrado" ou para de enviar comandos de movimento (envia flags de espera).
RF06	Utilizar um software de monitoramento de rede (como Wireshark ou Packet Sender) ou o log do receptor.	Os pacotes TCP contendo as coordenadas chegam ao destino (ESP32) com os mesmos valores calculados no Python.
RF07	Verificar as propriedades do frame recebido (resolução e canais de cor) via código.	As matrizes de imagem (frames) possuem a resolução esperada (ex: 640x480) e não estão corrompidas (bytes vazios).
RF08	Executar o programa e olhar para o monitor do computador.	Uma janela (cv2.imshow) abre e exibe o fluxo de vídeo em tempo real para o usuário.
RNF01	Inspeccionar o código-fonte e as bibliotecas importadas.	O arquivo tem extensão .py e contém import cv2, import numpy.
RNF02	Verificar a configuração do objeto socket no código Python.	O tempo de processamento de cada frame é menor que o intervalo de chegada de novos frames (ex: > 15 FPS para garantir fluidez).
RNF03	O sistema deve lidar com a perda temporária de pacotes	O socket é instanciado como socket.SOCK_STREAM, caracterizando o protocolo TCP.
RNF04	Testar o sistema com uma única câmera (monocular) versus duas câmeras (estéreo) e verificar o cálculo de profundidade (Z).	O sistema utiliza disparidade entre dois frames (triangulação) para obter o valor Z, e não apenas estimativa por tamanho de pixel.

## 2.8 CRONOGRAMA

Defina o cronograma de execução estruturado sem semanas e para cada atividade defina o entregável esperado para aquela atividade no prazo definido.

Quadro 2 – Cronograma de execução

Atividade	Sem. 1	Sem. 2	Sem. 3	Sem. 4	Entregável
Projeto	X	X			Documento de projeto
Implementação		X	X	X	Código e protótipo funcional
Verificação			X	X	Resultados de testes
Avaliação			X	X	Análise dos Resultados
Documentação	X	X	X	X	Relatório Final completo

## 2.9 ANÁLISE DE RISCOS

Nesta seção, procure identificar riscos que ameacem o atendimento dos requisitos do projeto.

Quadro 3 – Análise de riscos

Risco	Probabilidade	Impacto	Gatilho	Plano de contingência
Atraso ou queda no processamento das imagens	Média	Alto	Interrupção na comunicação TCP, ausência de resposta do carrinho, ou <i>timeout</i> no recebimento de comandos.	Reducir resolução da imagem, simplificar o modelo de detecção ou ajustar a taxa de captura

## 3 DESENVOLVIMENTO

Este capítulo descreve o processo de construção e validação do sistema, detalhando a integração entre os componentes físicos e os algoritmos de software desenvolvidos. O link com o repositório está disponível em: <https://github.com/fetrentin7/SelfDrivingCar>

### 3.1 IMPLEMENTAÇÃO

A implementação do sistema foi dividida em módulos funcionais para facilitar a depuração e a integração: o subsistema de captura de imagens, o subsistema de processamento remoto e o subsistema de controle embarcado. A arquitetura geral adota uma abordagem de processamento distribuído, onde a carga computacional pesada (visão computacional) é alocada em um computador remoto, enquanto o controle de atuadores permanece no microcontrolador local.

#### 3.1.1 Hardware

A arquitetura de hardware do projeto foi desenvolvida utilizando uma abordagem modular e distribuída, baseada no ecossistema da Espressif. O sistema embarcado foi dividido em dois subsistemas principais: o módulo de captação de imagens e o módulo de controle de atuadores.

**Microcontroladores e Captação de Imagens** Para a captura visual, foram utilizados módulos ESP32-CAM. O projeto original previa a utilização de três câmeras simultâneas (duas para visão estéreo e uma central). No entanto, durante a fase de validação, constatou-se que o tráfego de dados de três fluxos de vídeo MJPEG simultâneos congestionava a largura de banda da rede Wi-Fi, elevando a latência a níveis inaceitáveis para um sistema de tempo real. Dessa forma, a implementação final consolidada utilizou dois módulos ESP32-CAM, operando exclusivamente como streamers de vídeo via protocolo HTTP, configurados para formar o sistema de visão estereoscópica.

**Controle e Atuação** Para o gerenciamento da locomoção e da lógica de controle, optou-se por utilizar um ESP32 DevKit V1 dedicado, separando o processamento de controle do processamento de imagem. Este microcontrolador atua como um servidor TCP, recebendo as coordenadas (X, Y, Z) processadas remotamente e convertendo-as em sinais de atuação.

O sistema de atuação é composto por:

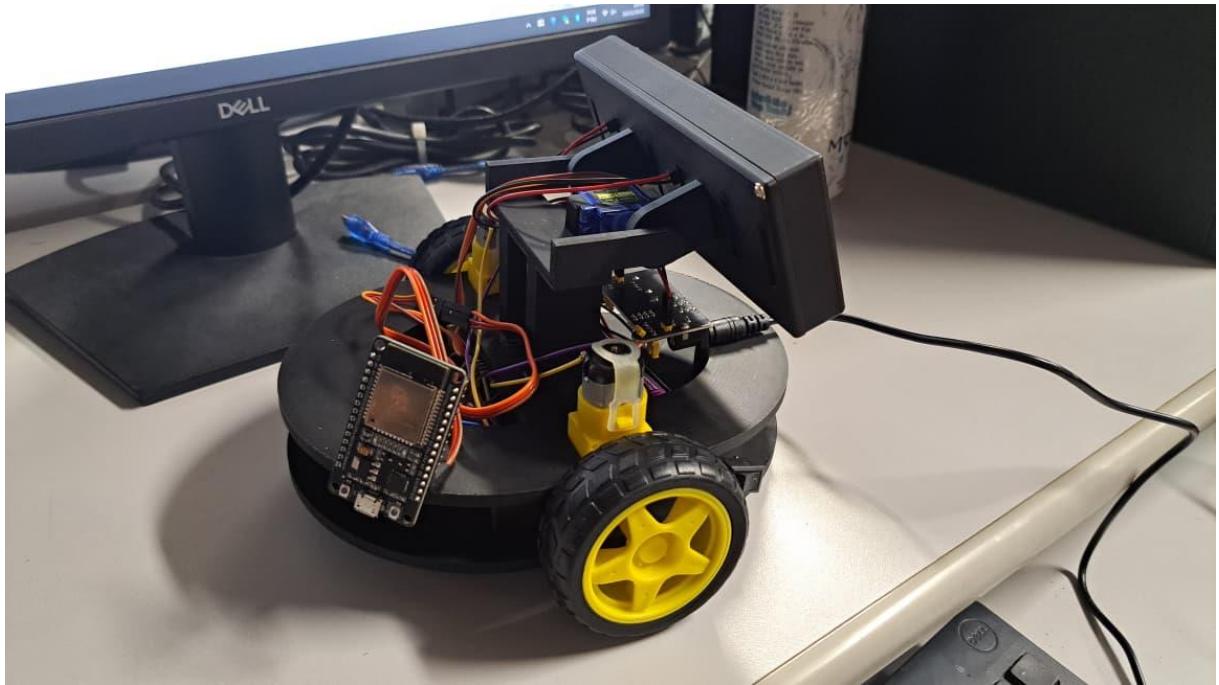
- Locomoção: Dois motores DC acoplados a uma caixa de redução, controlados por um driver Ponte-H L298N. Durante os testes de integração, identificou-se uma "zona morta" nos motores, onde sinais de PWM inferiores a 40% não eram suficientes para vencer a inércia e o atrito do sistema, exigindo ajustes na curva de aceleração do firmware.
- Rastreamento Vertical (Tilt): Um servomotor foi acoplado à estrutura das câmeras para controlar o ângulo de inclinação, permitindo que o robô mantenha o alvo centralizado verticalmente independentemente da distância.

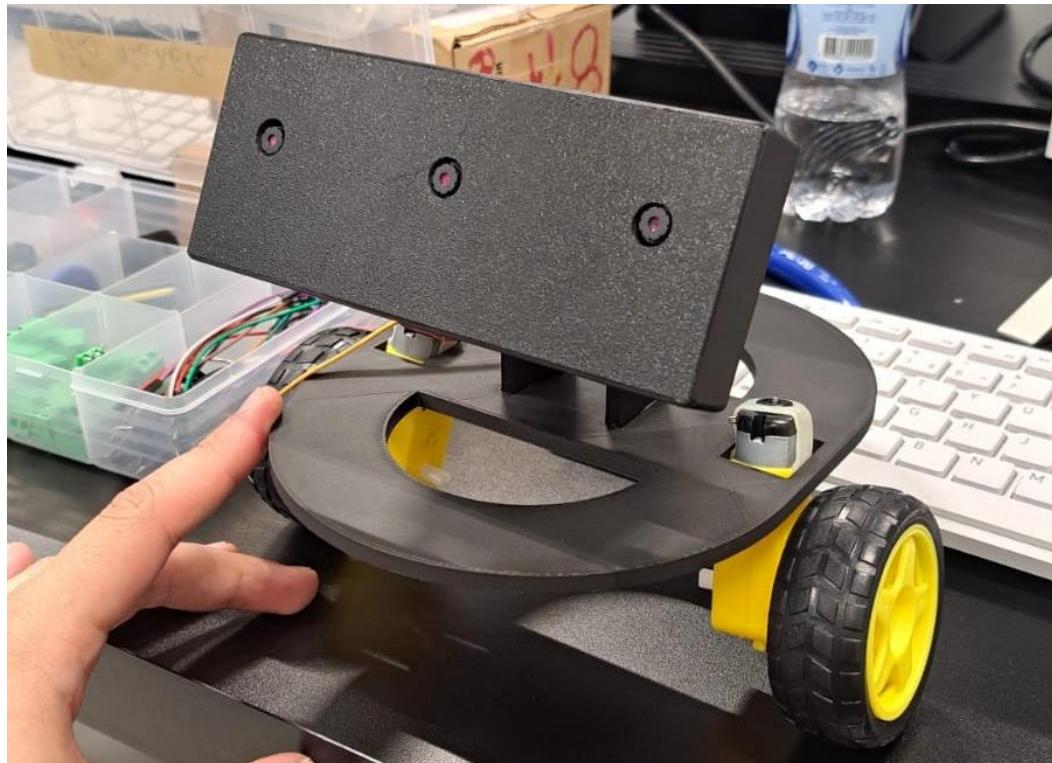
Alimentação e Estrutura A alimentação do sistema é provida por uma fonte de 7.4V, constituída por duas baterias LiPo 18650 de 3.7V ligadas em série. Esta configuração foi necessária para suprir a demanda de corrente dos motores e, simultaneamente, o alto consumo observado nos módulos ESP32-CAM durante a transmissão Wi-Fi.

Todos os componentes foram integrados em uma estrutura desenvolvida em modelagem 3D, projetada para manter o alinhamento fixo entre as câmeras, requisito essencial para a precisão da estereoscopia.

Figura 10 – Foto do carrinho







### A. Função app\_main (Inicialização e Orquestração)

```
void app_main(void)
{
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret ==
ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ESP_ERROR_CHECK(nvs_flash_init());
    }

    newDataSemaphore = xSemaphoreCreateBinary();
    if (newDataSemaphore == NULL) {
        ESP_LOGE("MAIN", "Falha ao criar newDataSemaphore");
        esp_restart();
    }

    posMutex = xSemaphoreCreateMutex();
    if (posMutex == NULL) {
        ESP_LOGE("MAIN", "Falha ao criar posMutex");
        esp_restart();
    }

    ESP_LOGI("MAIN", "Inicializando Wi-Fi...");
    wifi_init_sta();
```

```

BaseType_t ok;
ok = xTaskCreate(tcp_server_task, "tcp_server", 8192, NULL, 5, NULL);
if (ok != pdPASS) {
    ESP_LOGE("MAIN", "Falha ao criar tcp_server_task");
}

ok = xTaskCreate(control_task, "control", 8192, NULL, 6, NULL);
if (ok != pdPASS) {
    ESP_LOGE("MAIN", "Falha ao criar control_task");
}

while (1) {
    //ESP_LOGI("MAIN", "Sistema rodando...");
    vTaskDelay(pdMS_TO_TICKS(5000));
}
}

```

A função app\_main atua como o ponto de entrada do firmware. Sua responsabilidade é configurar o hardware, inicializar as estruturas de dados do sistema operacional e lançar as tarefas paralelas.

- **Inicialização do NVS:** O código inicia verificando a memória não volátil (NVS), necessária para o funcionamento da pilha Wi-Fi. Há um tratamento de erro robusto que apaga e reinicia a partição caso esteja corrompida ou sem espaço.
- **Mecanismos de Sincronização:** São criados dois objetos fundamentais para a segurança do sistema multitarefa:
  1. newDataSemaphore: Um semáforo binário utilizado para sinalizar à tarefa de controle que novos dados chegaram.
  2. posMutex: Um Mutex (exclusão mútua) para garantir que as variáveis globais de posição (posX, posY, posZ) não sejam acessadas simultaneamente por tarefas diferentes (evitando *Race Conditions*).
- **Criação de Tarefas:** A função instancia as duas tarefas principais (tcp\_server\_task e control\_task) no Scheduler do FreeRTOS, definindo suas prioridades. A verificação pdPASS garante que houve memória suficiente no Heap para alocá-las.

## B. Tarefa tcp\_server\_task (Comunicação e Recepção)

```

void tcp_server_task(void *pvParameters) {
    char rx_buffer[128];
    char addr_str[128];
    int addr_family = AF_INET;
    int ip_protocol = IPPROTO_IP;

    struct sockaddr_in dest_addr;
    dest_addr.sin_addr.s_addr = htonl(INADDR_ANY);
    dest_addr.sin_family = AF_INET;
    dest_addr.sin_port = htons(PORT);

    int listen_sock = socket(addr_family, SOCK_STREAM, ip_protocol);
    if (listen_sock < 0) {
        ESP_LOGE(TAGWIFI, "Unable to create socket: errno %d", errno);
        vTaskDelete(NULL);
        return;
    }

    bind(listen_sock, (struct sockaddr *)&dest_addr, sizeof(dest_addr));
    listen(listen_sock, 1);

    ESP_LOGI(TAGWIFI, "Servidor TCP escutando na porta %d", PORT);

    while (1) {
        struct sockaddr_in source_addr;
        socklen_t addr_len = sizeof(source_addr);

        int sock = accept(listen_sock, (struct sockaddr *)&source_addr,
&addr_len);
        if (sock < 0) {
            ESP_LOGE(TAGWIFI, "Erro no accept: errno %d", errno);
            continue;
        }

        inet_ntoa_r(((struct sockaddr_in *)&source_addr)->sin_addr.s_addr,
                    addr_str, sizeof(addr_str) - 1);
        ESP_LOGI(TAGWIFI, "Cliente conectado: %s", addr_str);

        send(sock, "Conexao estabelecida!\n", 23, 0);

        while (1) {
            int len = recv(sock, rx_buffer, sizeof(rx_buffer) - 1, 0);
            if (len <= 0) break;

            rx_buffer[len] = 0;
            // ESP_LOGI(TAGWIFI, "Recebido: %s", rx_buffer);
        }
    }
}

```

```

        float x, y, z;

        // Tenta ler com ou sem espaços
        if (sscanf(rx_buffer, "posX:%f, posY:%f, posZ:%f", &x, &y, &z)
== 3 ||

        sscanf(rx_buffer, "posX:%f , posY:%f , posZ:%f", &x, &y, &z)
== 3) {

            // Protege com mutex
            if(posMutex != NULL) {
                xSemaphoreTake(posMutex, portMAX_DELAY);
                posX = x;
                posY = y;
                posZ = z;
                xSemaphoreGive(posMutex);
            }

            // Notifica a task de controle
            if(newDataSemaphore != NULL) {
                xSemaphoreGive(newDataSemaphore);
            }

            ESP_LOGI(TAGWIFI, "POS -> X=%.2f Y=%.2f Z=%.2f", posX, posY,
posZ);
            send(sock, "OK\n", 3, 0);
        }
        else {
            send(sock, "Formato invalido\n", 17, 0);
        }
    }

    ESP_LOGW(TAGWIFI, "Cliente desconectado");
    close(sock);
}
vTaskDelete(NULL);
}

```

Esta tarefa é responsável pela conectividade e atua como o **Servidor** da aplicação.

- **Configuração de Sockets:** Utiliza a API de Sockets BSD (lwip/sockets.h) para criar um socket TCP, fazer o *bind* na porta definida e ficar em modo de escuta (*listen*).

- **Loop de Recepção:** O loop interno aceita conexões de clientes (o script Python no PC). A função recv é bloqueante, ou seja, a tarefa fica suspensa aguardando pacotes, economizando CPU.
- **Parsing e Tratamento de Dados:** Ao receber uma string (ex: "posX:10, posY:5, posZ:1.2"), utiliza a função sscanf para extrair os valores numéricos.
- **Região Crítica (Mutex):** Antes de escrever nas variáveis globais, a tarefa solicita o posMutex. Isso garante que ela não altere os valores enquanto a tarefa de controle os está lendo.
- **Sinalização:** Após atualizar os dados com sucesso, ela libera o semáforo newDataSemaphore. Isso "acorda" imediatamente a tarefa de controle para processar a nova posição.

### C. Tarefa control\_task (Lógica de Controle e Atuação)

```
void control_task(void *pvParameters)
{
    motors_init();
    servo_init();

    ESP_LOGI("CONTROL", "Teste inicial: servo 90, motores 0");
    servo_set_angle(90.0f);
    motors_set(0, 0);
    vTaskDelay(pdMS_TO_TICKS(500));

    PID_t pid_x, pid_z;
    PID_Init(&pid_x, 1, 0.01, 0.10, 0.10);
    PID_Init(&pid_z, 0.60, 0.00, 0.20, 0.10);

    PID_SetOutputLimits(&pid_x, -255, 255);
    PID_SetOutputLimits(&pid_z, -255, 255);

    const float Z_target = 1.00f;

    while (1)
    {
        // aguarda dado novo do Wi-Fi (bloqueante). Se quiser ver logs
        periódicos,
        if (xSemaphoreTake(newDataSemaphore, portMAX_DELAY) == pdTRUE) {
            float x, y, z;
```

```

    // leitura segura das variáveis globais
    if (xSemaphoreTake(posMutex, pdMS_TO_TICKS(1000)) == pdTRUE) {
        x = posX;
        y = posY;
        z = posZ;
        xSemaphoreGive(posMutex);
    } else {
        ESP_LOGW("CONTROL", "Falha ao tomar posMutex");
        continue;
    }

    ESP_LOGI("CONTROL", "Pos recebida: x=%.3f y=%.3f z=%.3f", x, y,
z);

    float err_x = x;
    float err_z = Z_target - z;

    //float out_x = PID_Update(&pid_x, err_x);
    float out_x = err_x; // só proporcional para teste
    float out_z = PID_Update(&pid_z, err_z);

    //float motor_esq = out_z - out_x;
    //float motor_dir = out_z + out_x;

    float motor_esq = -out_x;
    float motor_dir = out_x;

    // saturação
    motor_esq = fminf(fmaxf(motor_esq, -255.0f), 255.0f);
    motor_dir = fminf(fmaxf(motor_dir, -255.0f), 255.0f);

    // log antes de enviar aos motores
    ESP_LOGI("CONTROL", "Motors raw: L=%.2f R=%.2f", motor_esq,
motor_dir);
    motors_set((int)motor_esq, (int)motor_dir);

    float angulo = angulo_atual + (y * 0.03);
    if (angulo < 0.0f) angulo = 0.0f;
    if (angulo > 180.0f) angulo = 180.0f;

    angulo_atual = angulo;

    ESP_LOGI("CONTROL", "Servo angulo pedido: %.2f", angulo);
    servo_set_angle(angulo);
} else {
    // raro se usar portMAX_DELAY, mas deixa para robustez
    ESP_LOGW("CONTROL", "xSemaphoreTake(newDataSemaphore)
falhou/timeout");
}

```

```
[    }
}
```

Esta tarefa executa o processamento dos dados e o comando dos atuadores. Ela permanece em estado de bloqueio (*blocked*) na maior parte do tempo, executando apenas quando há novos dados.

- **Sincronização:** A linha `xSemaphoreTake(newDataSemaphore, portMAX_DELAY)` faz com que a tarefa aguarde indefinidamente até que a tarefa TCP sinalize a chegada de um novo pacote. Isso elimina a necessidade de *polling* (ficar verificando variáveis em loop), tornando o sistema extremamente eficiente energeticamente.
- **Cópia Segura:** Ao acordar, a tarefa usa o `posMutex` para fazer uma cópia local das variáveis globais. Isso libera o Mutex rapidamente para que a tarefa TCP possa receber novos dados sem atraso.
- **Algoritmo de Controle:**
  - **Eixo X (Direção):** Calcula o erro horizontal. O código mostra uma implementação híbrida onde o PID pode ser ativado, mas atualmente está configurado como proporcional direto (`out_x = err_x`) para testes de validação.
  - **Eixo Z (Distância):** Calcula o erro de profundidade em relação ao *setpoint* (`Z_target = 1.00f`). Utiliza o algoritmo PID para calcular a velocidade necessária para avançar ou recuar.
- **Mixagem (Differential Drive):** Combina as saídas de controle linear e angular para gerar os comandos individuais dos motores esquerdo e direito.
- **Saturação (Clamping):** As funções `fminf` e `fmaxf` limitam os valores de PWM entre -255 e 255, protegendo a lógica da ponte H contra valores inválidos.
- **Controle do Servo:** Implementa um controle proporcional simples para o eixo Y (Tilt), ajustando o ângulo da câmera para manter o rosto centralizado verticalmente, com limitadores de segurança entre 0° e 180°.

### 3.1.2 Implementação do código usando OpenCV

Nesta seção, apresente o máximo de detalhes referentes ao que foi implementado. Descreva o fluxo de ferramentas utilizados e apresente trechos representativos do código fonte (não precisa incluir todo o código). Também apresente fotos do protótipo outras e evidências de implementação.

#### 3.1.2.1 Gerenciamento da leitura da câmera

A leitura da câmera foi realizada por meio de uma classe dedicada (VideoStream), responsável por abrir o dispositivo, capturar os frames e atualizar a imagem em uma *thread* separada. Caso a captura falhe, o módulo tenta reconectar automaticamente, evitando interrupções no fluxo. Essa abordagem garante leitura contínua e independente do restante do processamento.

classe.py

```
class VideoStream:
    def __init__(self, src=0):
        self.src = src
        self.stream = cv2.VideoCapture(src)
        self.ret, self.frame = self.stream.read()
        self.stopped = False
        if not self.ret:
            self.frame = None

    def start(self):
        Thread(target=self.update, args=()).start()
        return self

    def update(self):
        while True:
            if self.stopped:
                self.stream.release()
                return

            # Tenta ler o frame
            grabbed, frame = self.stream.read()
```

```

        if grabbed:
            self.frame = frame
            self.ret = True
        else:
            # Se falhar, marca como erro e tenta reconectar
            self.ret = False
            # Não printa toda hora para não poluir, mas tenta reabrir
            try:
                self.stream.release()
                time.sleep(1) # Espera um pouco antes de tentar de
novo
                self.stream = cv2.VideoCapture(self.src)
                print(f"Tentando reconectar a {self.src}...")
            except:
                pass

    def read(self):
        return self.ret, self.frame

    def stop(self):
        self.stopped = True

```

### 3.1.2.2 Detecção do rosto

A detecção de rostos foi realizada utilizando classificadores Haar do OpenCV por meio da função detectMultiScale, que varre a imagem em múltiplas escalas para identificar rostos próximos ou distantes da câmera. O método é aplicado de forma sequencial para três casos: rosto frontal, perfil e perfil invertido (obtido espelhando a imagem, já que o classificador de perfil reconhece apenas um dos lados). Quando uma detecção é realizada, a função retorna as coordenadas da região do rosto; caso contrário, retorna None. Essa abordagem garante robustez e flexibilidade na identificação do rosto, independentemente da orientação do usuário.

Face\_deteccao.py

```

def detect_one_face(img):
    if img is None: return None
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

```

```

# 1. Frontal
faces = face_cascade.detectMultiScale(gray, 1.3, 5)
if len(faces) > 0: return faces[0]

# 2. Perfil
faces_prof = profile_cascade.detectMultiScale(gray, 1.3, 5)
if len(faces_prof) > 0: return faces_prof[0]

# 3. Perfil Invertido
flipped = cv2.flip(gray, 1)
faces_flip = profile_cascade.detectMultiScale(flipped, 1.3, 5)
if len(faces_flip) > 0:
    (x, y, w, h) = faces_flip[0]
    x = img.shape[1] - x - w
    return (x, y, w, h)

return None

```

### 3.1.2.3 Inicialização do vídeo

O código inicia as duas câmeras e lê continuamente os frames de cada uma pelo IP. Se alguma câmera falhar, o sistema apenas espera e continua o loop. A imagem da câmera esquerda é tratada como referência (“mestre”), servindo como base para centralização, marcação do ponto central e cálculo do deslocamento. Todas as comparações e ajustes são feitos baseadas a essa câmera como ponto fixo.

#### Leitura\_face.py

```

cam_left = VideoStream("http://172.20.10.6").start()
cam_right = VideoStream("http://172.20.10.5").start()
time.sleep(2.0)

last_send_time = 0

while True:
    retL, frameL = cam_left.read()
    retR, frameR = cam_right.read()

```

```

# Se alguma câmera caiu, mostra aviso mas NÃO TRAVA O LOOP
if frameL is None or frameR is None:
    print("Aguardando sinal das câmeras...")
    time.sleep(0.5)
    continue

# Setup da imagem (Esquerda é a Mestre)
h, w, _ = frameL.shape
frame_center = (w // 2, h // 2)
cv2.circle(frameL, frame_center, 5, (0, 255, 0), -1) # Centro da tela
(Verde)

# Detecção
rectL = detect_one_face(frameL)
rectR = detect_one_face(frameR)

text_info = "Procurando..."

```

### 3.1.3 Posicionamento e distância

Quando o sistema encontra o rosto nas duas câmeras, ele calcula sua posição no espaço. Primeiro, obtém as coordenadas do rosto em cada imagem e define o centro da detecção na câmera esquerda, que funciona como referência. Em seguida, calcula os erros horizontal e vertical ( $dx$  e  $dy$ ) ao comparar o centro do rosto com o centro do frame, o que indica a direção necessária para manter o alinhamento. Por fim, determina a profundidade ( $Z$ ) a partir da diferença entre as posições horizontais do rosto nas duas imagens (disparidade). Essa disparidade alimenta a fórmula da visão estéreo:  $Z = \frac{f \cdot B}{\text{disparidade}}$ , onde  $f$  é a distância focal e  $B$  é a distância entre as câmeras.

#### Leitura\_face.py

```

# === LÓGICA PRINCIPAL (Só se achar rosto nas duas) ===
if rectL is not None and rectR is not None:

    # Coordenadas Esquerda
    (xL, yL, wL, hL) = rectL
    face_center_L = (xL + wL//2, yL + hL//2)

```

```

# Coordenadas Direita
(xR, yR, wR, hR) = rectR
cxR = xR + wR//2

# 1. CÁLCULO DE ERRO (X, Y)
dx = face_center_L[0] - frame_center[0]
dy = -(face_center_L[1] - frame_center[1])

# 2. CÁLCULO DE PROFUNDIDADE (Z)
cxL = face_center_L[0]
disparity = abs(cxL - cxR)

if disparity > 2: # Filtro de ruído
    Z = (FOCAL_LENGTH * BASELINE) / disparity
else:
    Z = 0.0

# Desenho Visual

```

### 3.1.4 Envio para ESP32

Após localizar o rosto, o sistema desenha o retângulo e marca o centro da detecção na câmera esquerda para facilitar a visualização. Em seguida, monta a mensagem contendo os valores de posição em X, Y e Z e envia esses dados ao ESP32 em intervalos definidos, sem bloquear a execução caso o dispositivo não responda. O sistema também tenta receber uma resposta e a imprime quando disponível. Por fim, reduz o tamanho das imagens, combina as duas visões lado a lado e exibe a janela do sistema estéreo para acompanhamento em tempo real.

Envio\_esp.py

```

cv2.circle(frameL, face_center_L, 5, (0, 0, 255), -1) # Centro do
rosto (Vermelho)
cv2.rectangle(frameL, (xL, yL), (xL+wL, yL+hL), (255, 0, 0), 2)

text_info = f"X:{dx} Y:{dy} Z:{Z:.2f}"

```

```
# 3. ENVIO TEMPORIZADO
current_time = time.time()
if (current_time - last_send_time) > SEND_INTERVAL:

    try:
        msg = f"posX:{int(dx)}, posY:{int(dy)}, posZ:{Z:.2f}"
        s.send(msg.encode())

    # Ler resposta do ESP32 (não trava se não vier)
    try:
        resp = s.recv(1024).decode()
        print("ESP32:", resp)
    except:
        pass

    print(f"☒ ENVIADO! >> {msg}")
    last_send_time = current_time

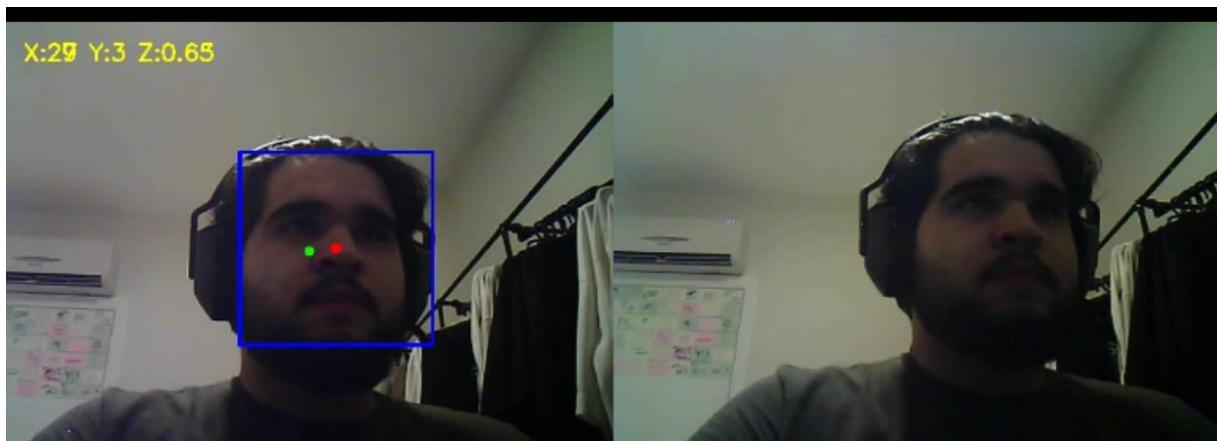
except Exception as e:
    print("Erro ao enviar TCP:", e)
# === EXIBIÇÃO ===
cv2.putText(frameL, text_info, (20, 40), cv2.FONT_HERSHEY_SIMPLEX,
0.8, (0,255,255), 2)

try:
    frameL_s = cv2.resize(frameL, (0,0), fx=0.6, fy=0.6)
    frameR_s = cv2.resize(frameR, (0,0), fx=0.6, fy=0.6)
    combined = cv2.hconcat([frameL_s, frameR_s])
    cv2.imshow("Sistema Stereo XYZ", combined)
except Exception as e:
    print(f" ERRO ENVIO: O ESP32 não respondeu. {e}")
    pass
```

## 3.2 VERIFICAÇÃO

### 3.2.1 Opencv

A imagem exibe o resultado do sistema de visão estéreo durante a detecção facial. Na câmera da esquerda, o rosto é identificado por meio de um retângulo azul, e o ponto central da face é marcado em vermelho. No canto superior esquerdo, são exibidos os valores calculados: o deslocamento horizontal (**X = 29**), o deslocamento vertical (**Y = 3**) e a distância estimada até o rosto (**Z = 0.65**). Esses parâmetros representam, respectivamente, quanto o rosto está deslocado do centro da imagem e sua profundidade relativa em relação ao par de câmeras.



O sistema detecta o rosto, calcula posX, posY e posZ, e depois aguarda 1 segundo antes de enviar os valores a ESP32.

```
PS C:\OpenCVEsp\OpenCV> & C:/Users/Fernando/AppData/Local/Programs/Python/Python312/python.exe c:/OpenCVEsp/OpenCV/teste.py
Conectado!
==INICIANDO SISTEMA==
ESP32: Conexao estabelecida!

    ✓ ENVIADO! >> posX:-252, posY:47, posZ:0.38
ESP32: OK

    ✓ ENVIADO! >> posX:-246, posY:40, posZ:0.38
ESP32: OK

    ✓ ENVIADO! >> posX:-226, posY:9, posZ:0.36
ESP32: OK

    ✓ ENVIADO! >> posX:-230, posY:-9, posZ:0.34
ESP32: OK

    ✓ ENVIADO! >> posX:-202, posY:13, posZ:0.36
ESP32: OK

    ✓ ENVIADO! >> posX:-240, posY:18, posZ:0.80
ESP32: OK

    ✓ ENVIADO! >> posX:-249, posY:-3, posZ:0.41
ESP32: OK

    ✓ ENVIADO! >> posX:-251, posY:-22, posZ:0.47
ESP32: OK
```

### 3.2.2

Apresente os procedimentos de verificação utilizados e os resultados obtidos com a aplicação do plano de verificação. Transcreva o quadro apresentado na seção anterior e acrescente uma coluna para descrever os resultados obtidos. É importante analisar o grau de atendimento de cada requisito. Também apresente imagens de experimentos de verificação, tais como diagramas de formas de onda de simulação, capturas de tela de console e equipamentos de instrumentação, fotos da operação do protótipo. Procure descrever como a imagem evidencia o cumprimento dos requisitos.

## 3.3 RESULTADOS

Apresente os resultados experimentais com as métricas de avaliação do protótipo, discutindo ao máximo os resultados obtidos. Procure também resumir esses resultados em tabelas.

## 4 CONCLUSÃO

Até esta etapa, o sistema apresenta uma arquitetura bem definida, com a separação clara das responsabilidades entre o Computador e o Sistema Embarcado. As funcionalidades de captura e processamento de imagem serão implementadas utilizando a biblioteca OpenCV, responsável pela leitura dos frames, detecção do alvo e extração das informações visuais que orientarão o controle.

O próximo passo consiste na integração completa com o sistema embarcado, permitindo que os comandos gerados a partir da análise das imagens sejam transmitidos em tempo real. Essa fase inclui o envio dos sinais de controle, o ajuste automático da câmera e a verificação da comunicação contínua entre os dispositivos. Após essa integração, serão executados testes para avaliar a precisão da detecção, o tempo de resposta e o comportamento do sistema em cenários com e sem perda do alvo.

## **REFERÊNCIAS**

Apresente as referências citadas no documento seguindo as normas da ABNT para elaboração de referências (NBR 6023 de agosto de 2002, ou posterior).