



Definição de trajetória robótica usando visão computacional: um modelo baseado por identificação de marcadores de sinalização com aplicação em Teoria dos Grafos

Lucas da Silva Carvalho
Orientador - Prof. Glauber Balthazar

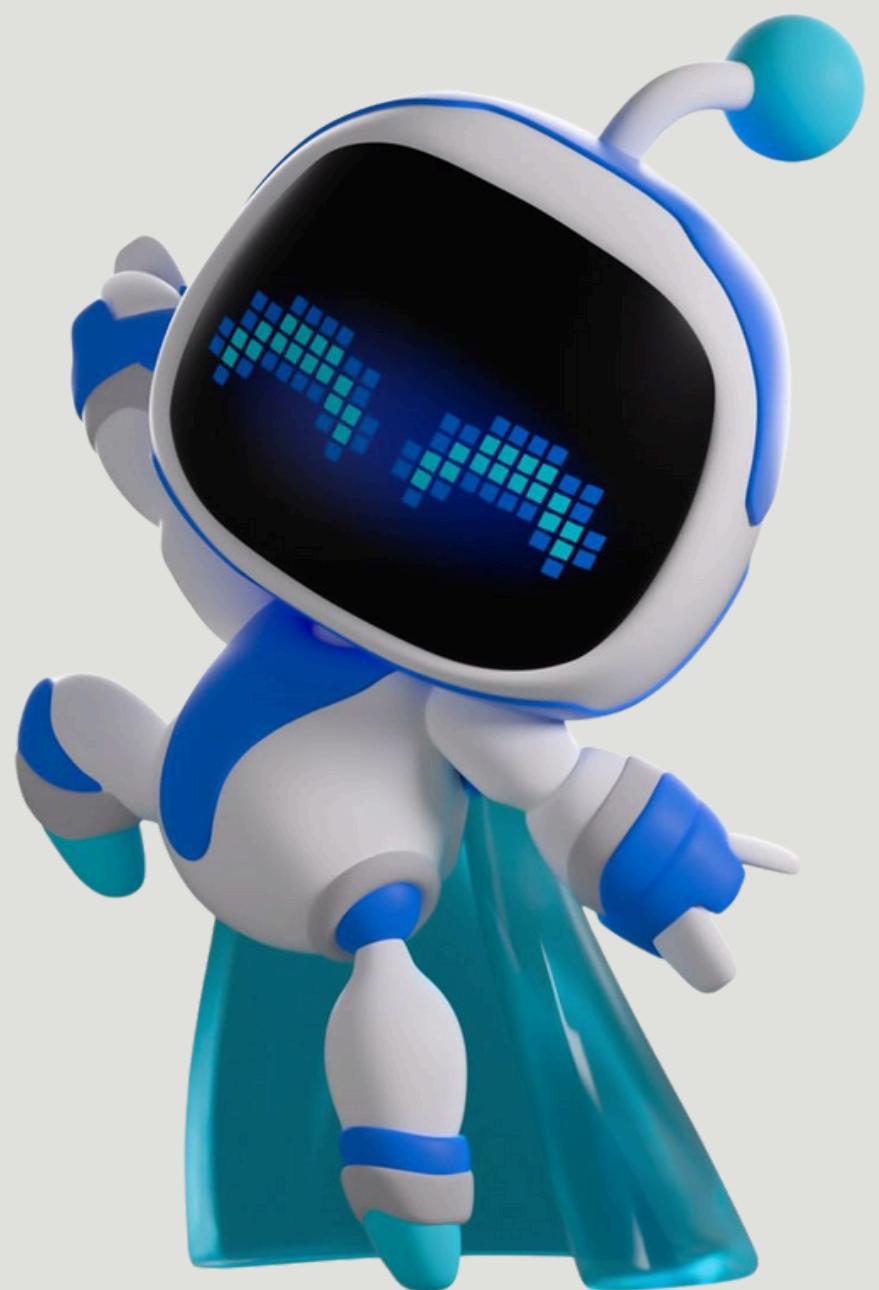


Introdução

Ideia Inicial

A partir de uma vontade de conhecer melhor o campo da visão computacional surgiu a ideia

Desenvolver um robô que possa se deslocar de forma autônoma, utilizando-se de marcadores visuais, para alcançar seu objetivo





Introdução **Motivação**

Grande possibilidade de estudos com a robótica móvel como: autonomia, navegação, otimização, aprendizagem e cooperação

A visão computacional permite que robôs e humanos compartilhem informações visuais ajudando robôs a entenderem e interagirem com o mundo



Introdução **Motivação**

Aplicação de um sistema de navegação robótico utilizando a tecnologia de visão computacional

Pretende-se explorar como essa tecnologia pode ser utilizada na navegabilidade autônoma com ajuda de modelos matemáticos da Teoria dos Grafos.



Introdução **Justificativa**

A popularização da visão computacional tem expandido significativamente as capacidades dos robôs industriais

Esta tecnologia tornou-se essencial para facilitar a navegação autônoma, reconhecimento de padrões e precisão em ambientes dinâmicos e com maior grau de complexidade



Introdução

Hipótese

H0 o uso de visão computacional para identificação de marcadores, combinados com um modelo matemático de navegação são suficientes para determinar uma trajetória de deslocamento para um robô

H1 afirma que o uso destes marcadores não são suficientes para determinar trajetórias de deslocamento para um robô



Planejamento **Como foi feito**

Foi utilizado um elemento robótico móvel previamente existente que possui recursos de processamento computacional e suporte a câmeras

O robô permitiu rodar um programa de navegação por marcadores visuais utilizando a linguagem de programação Python

Sua execução foi realizada em ambiente controlado indoor gerando assim dados experimentais para uso em treinamento e teste da identificação das placas marcadores



Planejamento **Objetivo Geral**

Treinar um robô para navegação autônoma com o uso de marcadores de sinalização obtidos pelo mapeamento de um conjunto de placas de marcação (placas de sinalização)



Planejamento Objetivos Específicos

- Construir um dataset de imagens de placas de sinalização para definir os marcadores de tomada de decisão
- Treinar um modelo algorítmico de visão computacional para identificar os marcadores de sinalização
- Implementar e testar o modelo em uma estrutura robótica já existente
- Implementar o modelo matemático de Teoria dos Grafos para determinação da tomada de decisão de movimentação baseada nos marcadores identificados



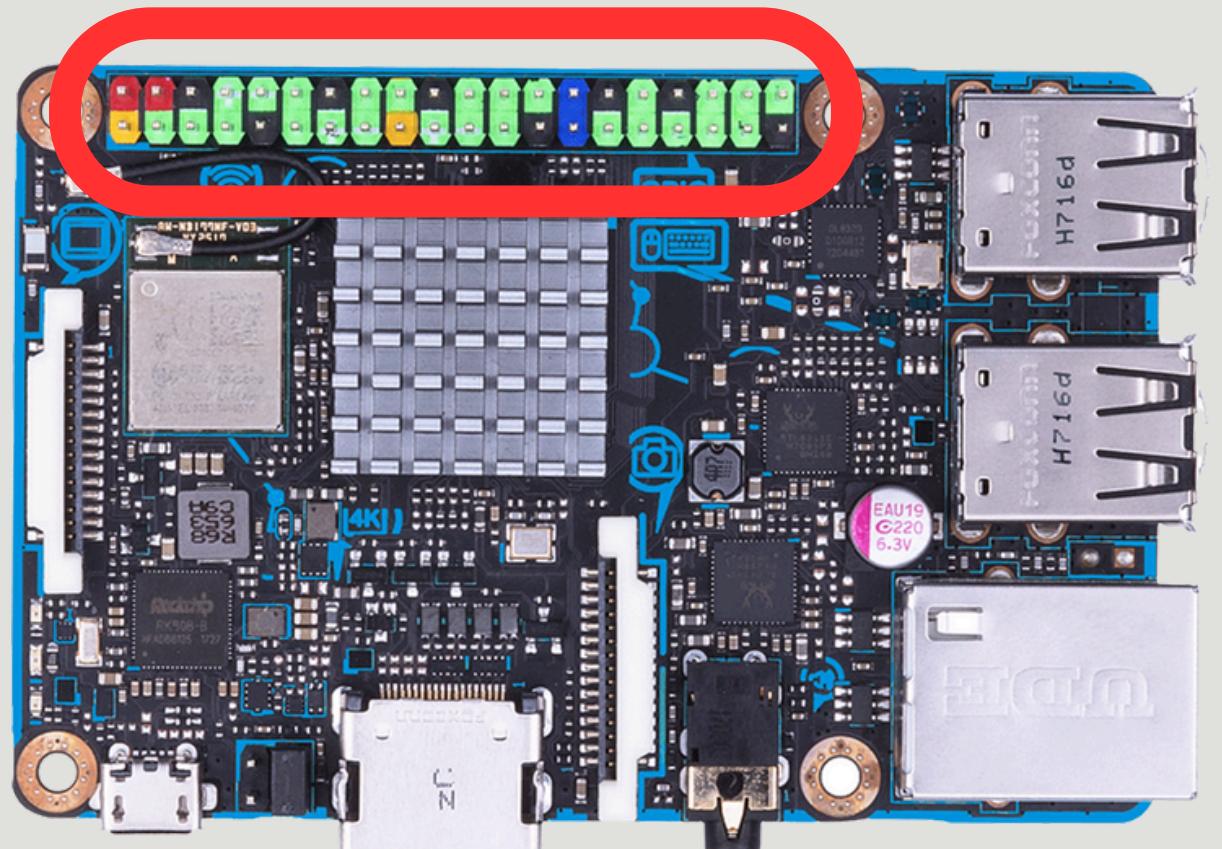
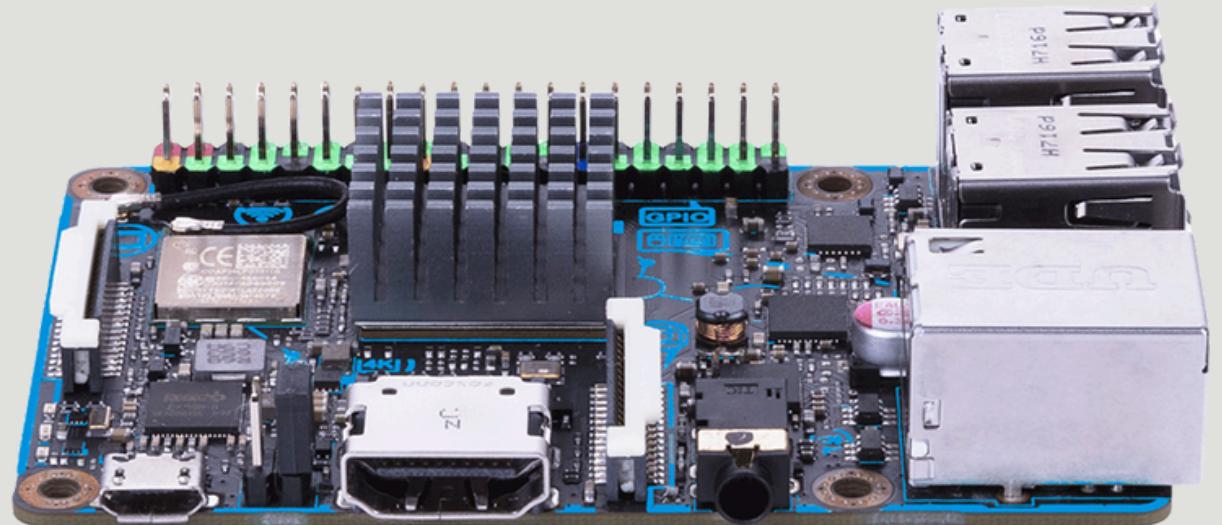
Material e Métodos

Asus Tinkerboard

Computador de placa única (SBC) baseado em ARM

- SO TinkerOS (Debian 10)
- Processador Quad-Core RK3288
- 4x USB + Lan
- Placa gráfica integrada - ARM® Mali™-T764 GPU*1
- 2 gb ram
- Até 28 GPIOs configuráveis como entrada ou saída digital

GPIO - General Purpose Input/Output



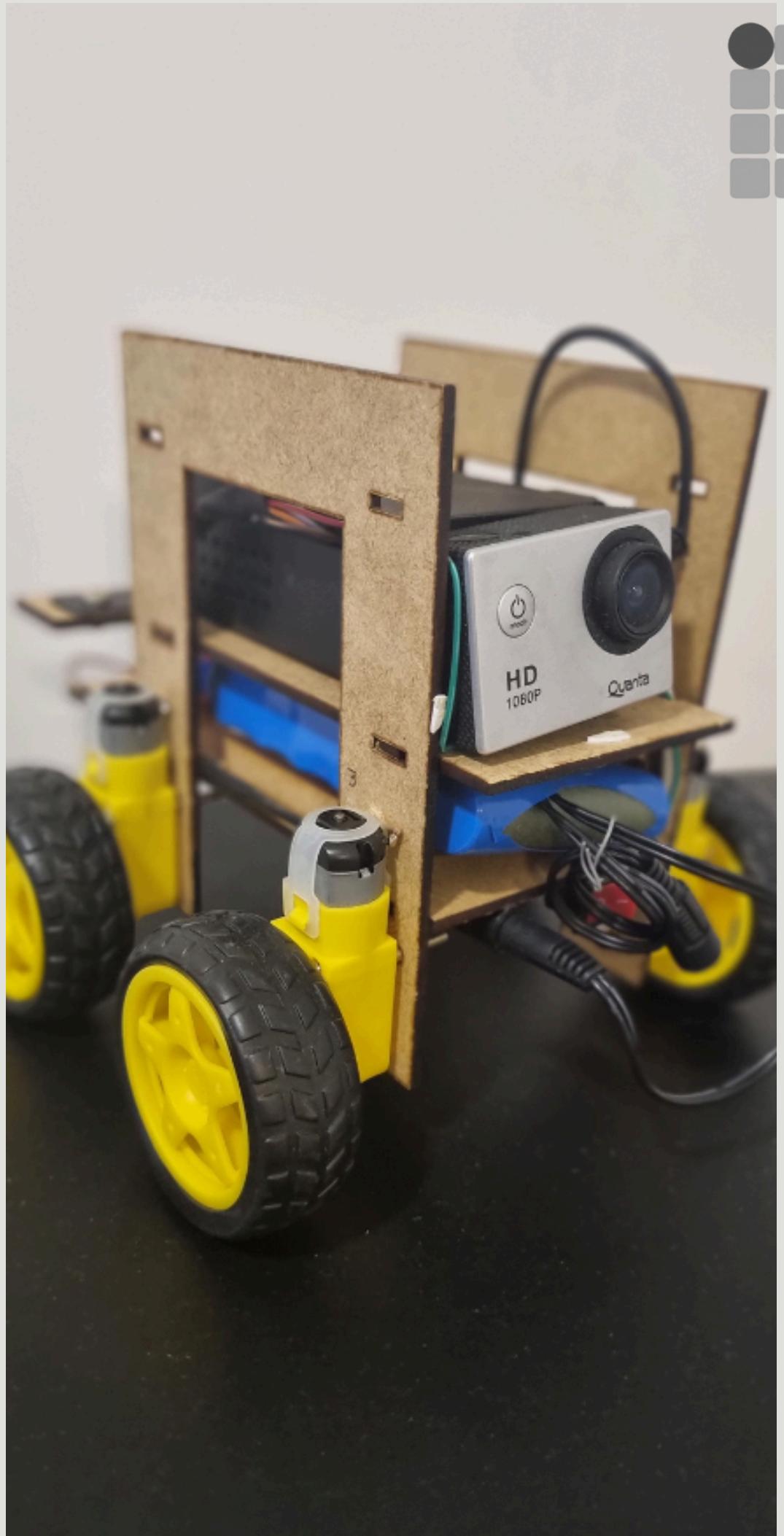


Material e Métodos

Estrutura Robótica

Estrutura em madeira com suporte a:

- SBC Asus Tinkerboard
- bateria
- 4 Rodas com motores
- Botão para ligar e desligar sistema





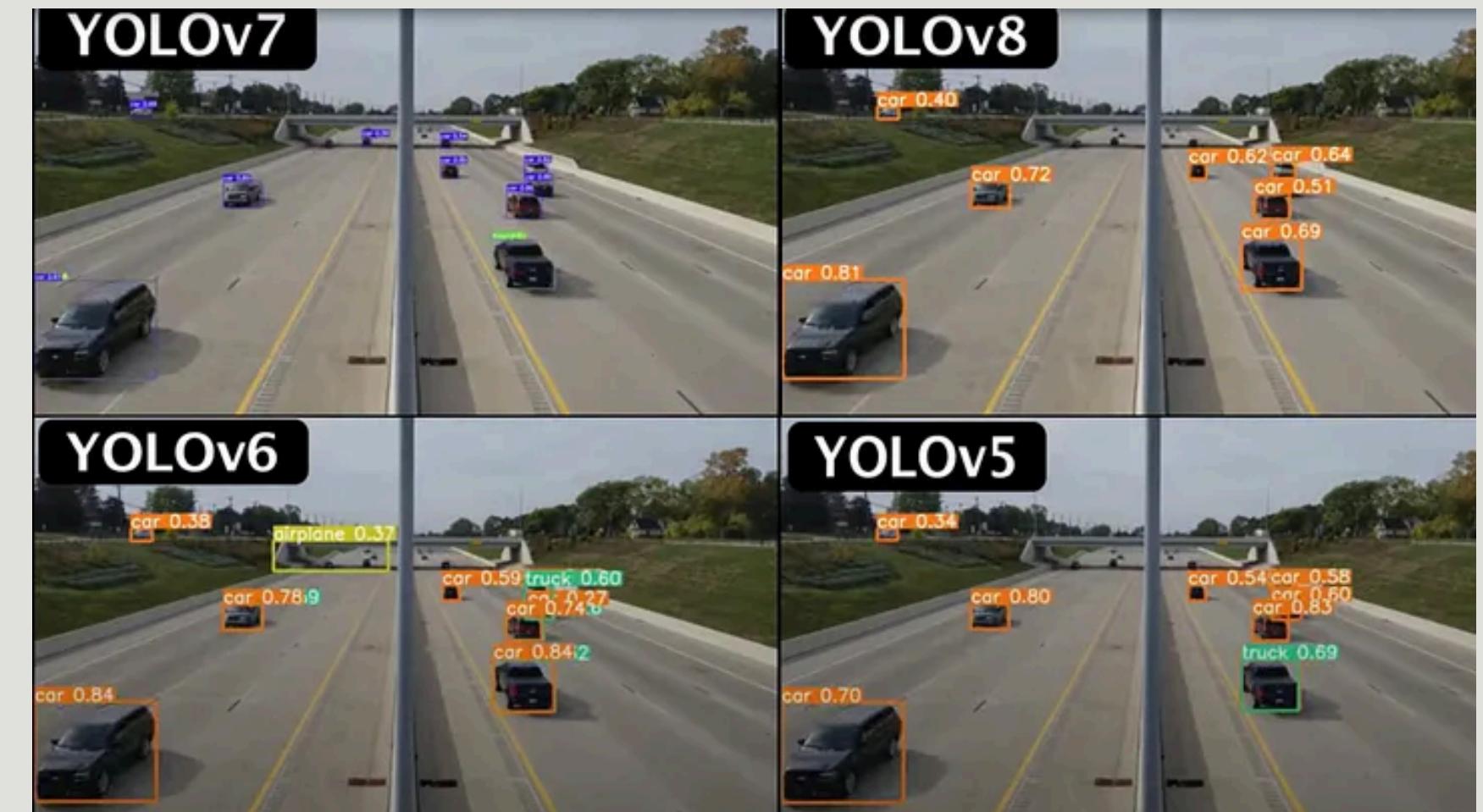
Material e Métodos

YoloV8

modelo avançado de detecção de objetos que pode ser treinado com um dataset próprio para aplicações

YOLOv8 funciona como uma plataforma de integração tecnológica

Conecta APIs para várias tarefas de pós processamento



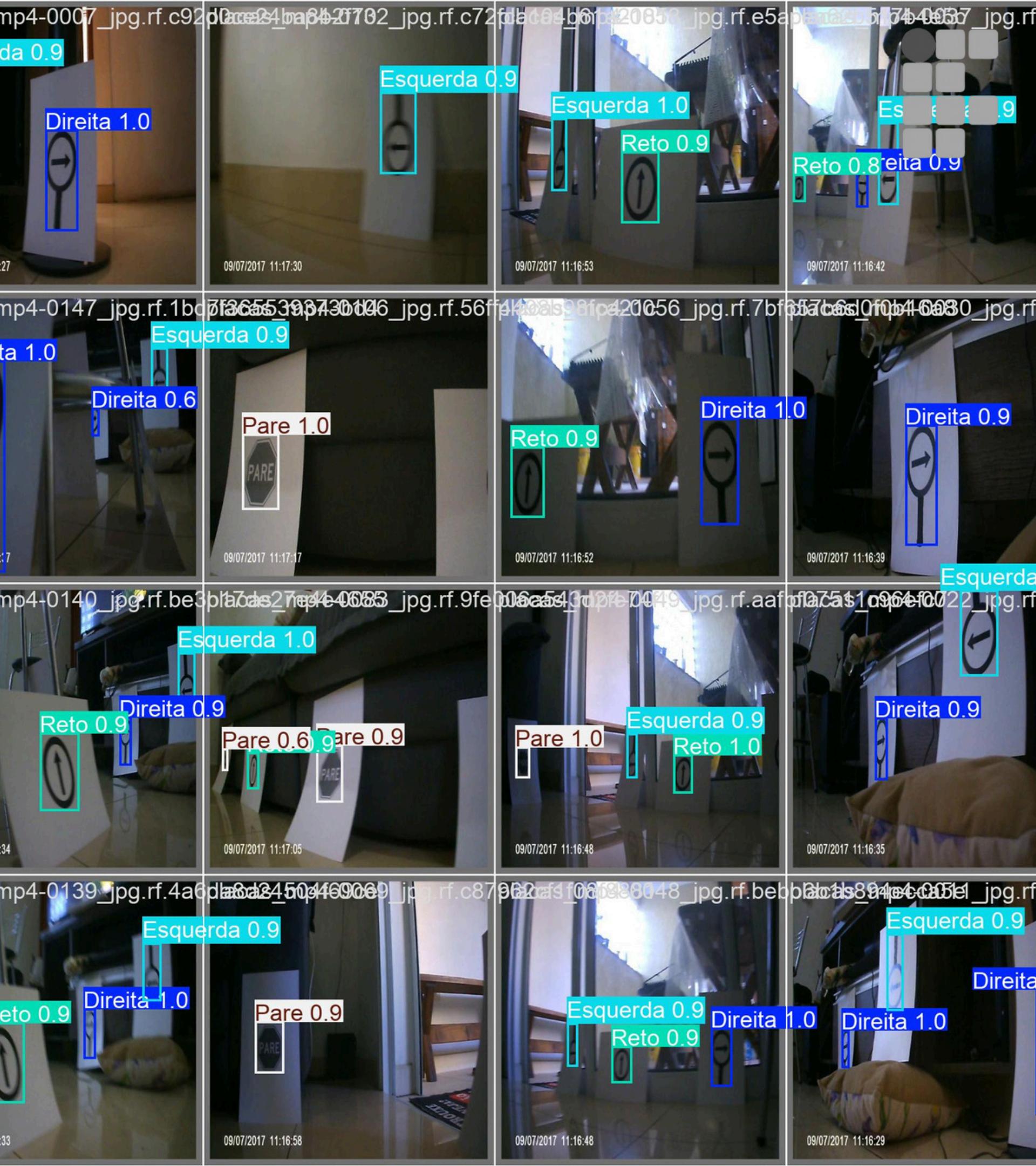
Material e Métodos

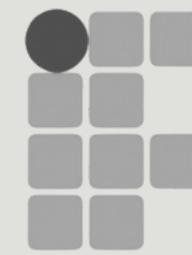
Dataset

É preciso mapear (fazer anotações - labels) manualmente em vários frames de um arquivo de vídeo

As anotações servirão de base para realizar o treinamento com YoloV8

Para realizar as anotações foi utilizado o software **ROBOFLOW**





Material e Métodos

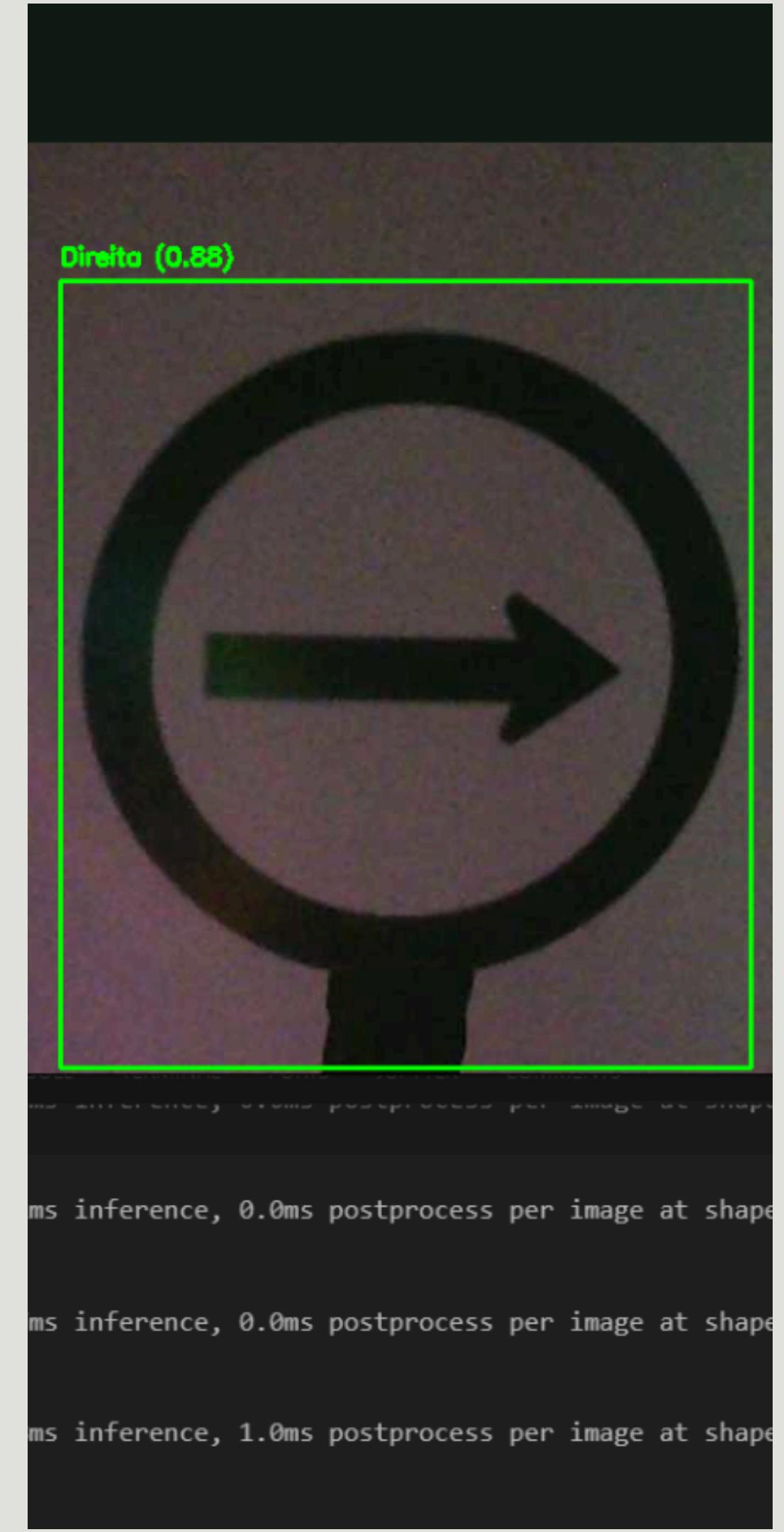
Treinamento

Para realizar o treinamento é preciso:

- Configurar arquivo data.yaml, especificando caminhos, número de classes e rótulos
- Tipo do modelo de treinamento (detecção de objetos)
- ajuste de hiperparâmetros como número de épocas e tamanho das imagens.

O treinamento gera um arquivo **.pt**

No programa em python esse arquivo, junto à biblioteca ultralytics, servirá como referencia para que o programa encontre referências as classes de treinamento





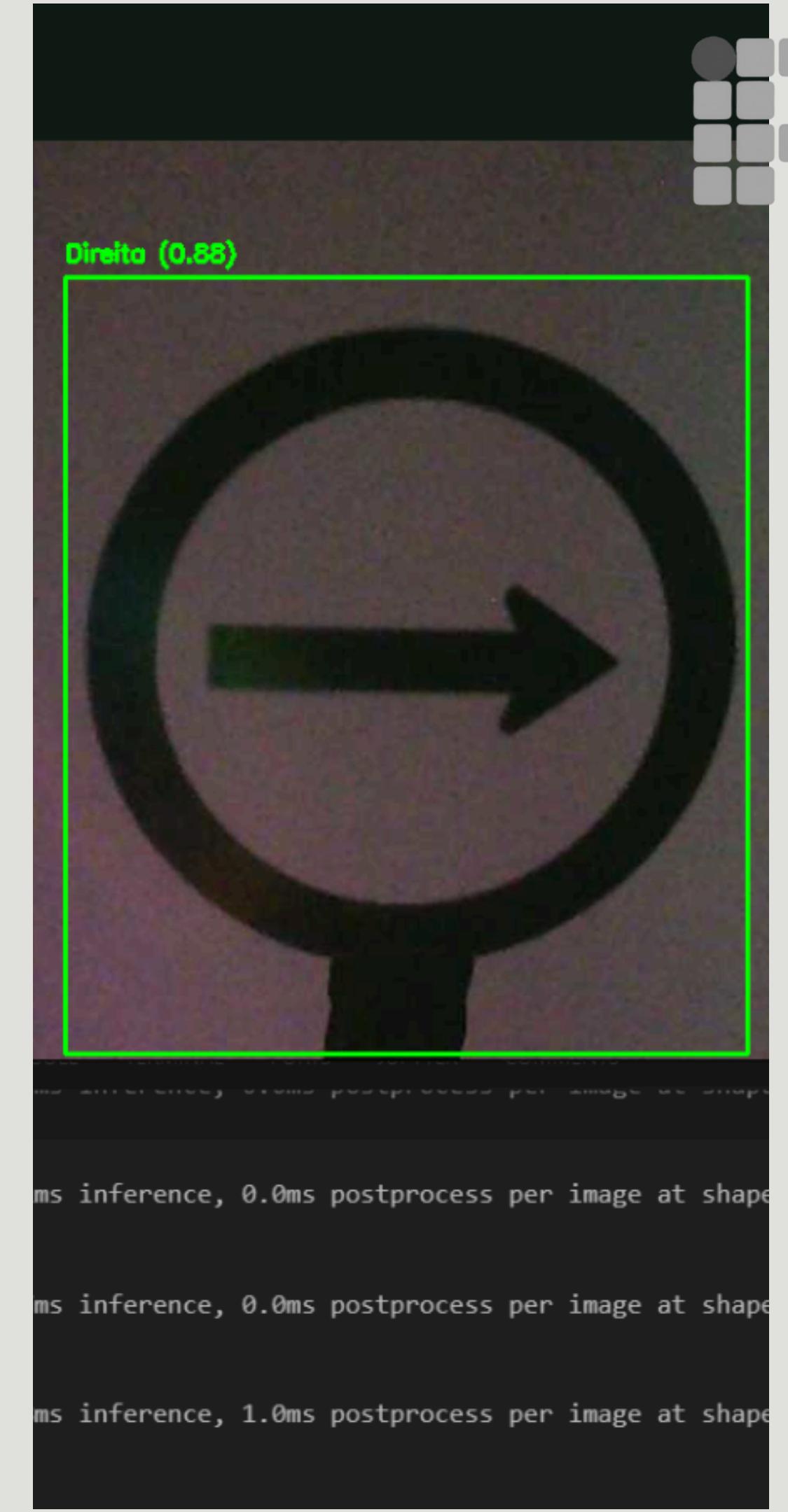
Material e Métodos

Testes

Nessa etapa pode-se observar pontos a ser melhorado no treinamento para aprimorar os resultados

Um programa para testes que mostra em tempo real o resultado do treinamento é crucial para criação de um arquivo **best.pt** mais fiel ao resultado esperado

O treinamento deve ser realizado novamente até encontrar um resultado satisfatório



Material e Métodos

Integração ao robô

Com o arquivo **best.pt** em mãos e testado

Podemos desenvolver o algoritmo que irá associar a detecção das classes ao movimento do robô.

A biblioteca **ASUS.GPIO** permite o controle de ativação de desativação dos módulos conectados às portas GPIO

A limitação do sistema torna essa etapa muito complexa, é importante que todos pacotes sejam compatíveis um com o outro





Material e Métodos

Desafios da implementação

Dificuldades com incompatibilidade de pacotes

Limitações do sistema exigem compilações e downloads de pacotes com cuidado para reduzir consumo de recursos do sistema

Sobrecarregamento do sistema levando a um reboot



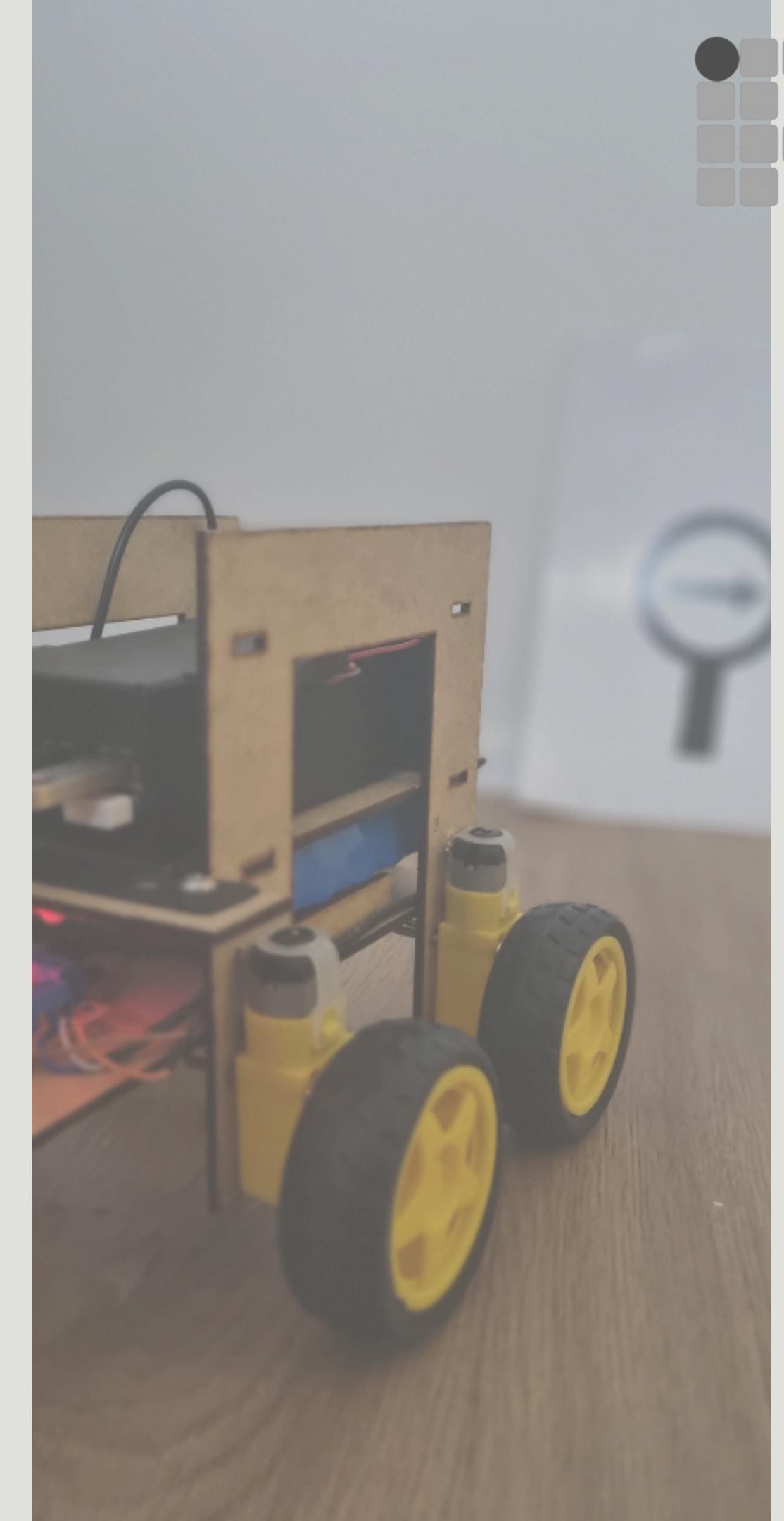
Material e Métodos

Soluções de otimização

Define o número de threads usadas ao compilar pacotes

Uso de pacotes pré-compilados sempre que possível

Configuração de swap memory para mitigar limitações de RAM





Material e Métodos

Grafos e algorítmo de Dijkstra

Foi adicionado ao código a funcionalidade do algorítmo de Dijkstra

Ele é capaz de encontrar o caminho mais curto entre dois pontos em uma rede

Para isso foi utilizado o peso real de tempo, a distância entre placas foi realizada utilizando o tempo como métrica



Material e Métodos

Grafos e algoritmo de Dijkstra

Início

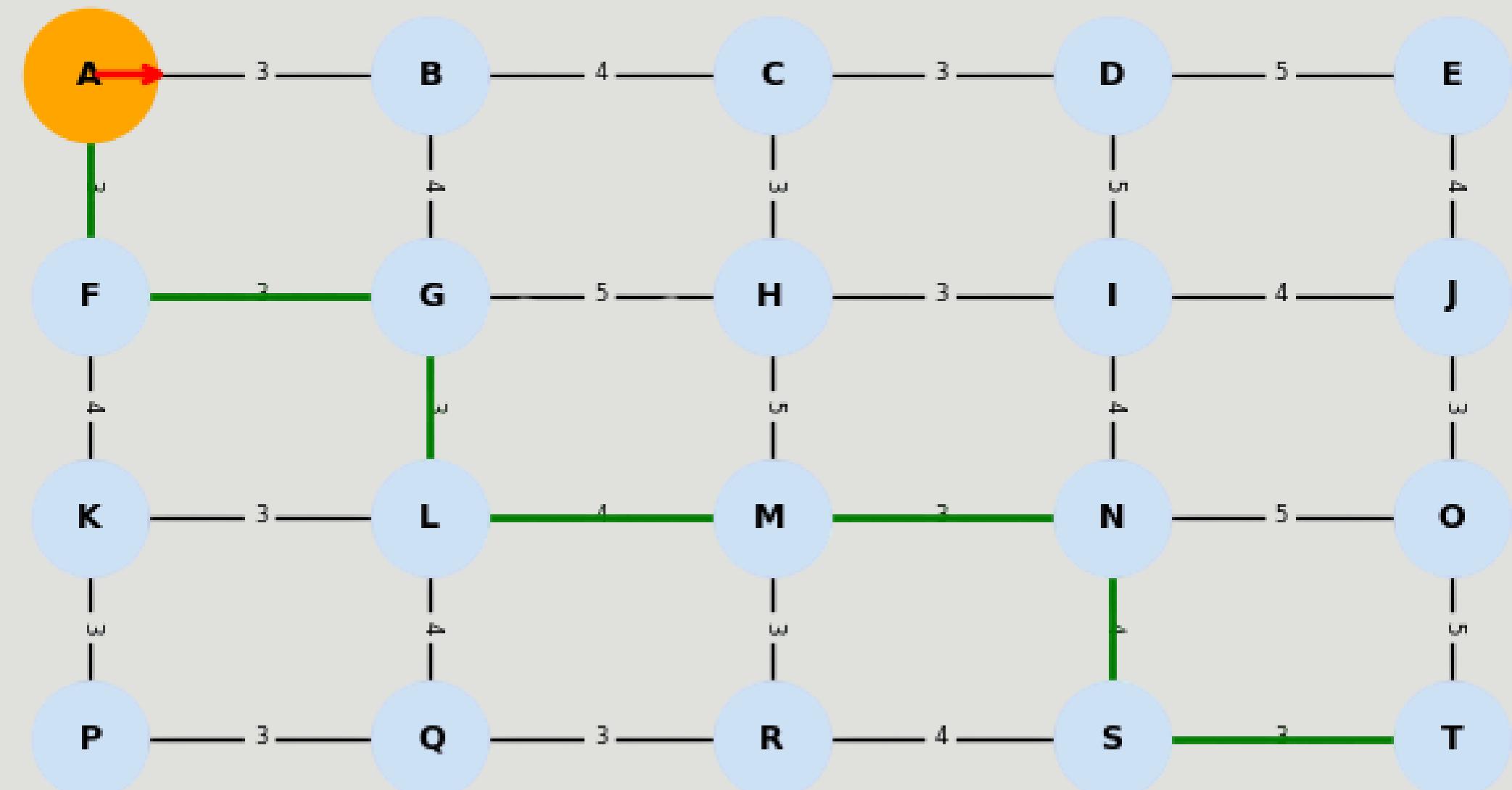
['A']

Destino

['T']

Caminho ideal

['A', 'F', 'G', 'L', 'M', 'N', 'S', 'T']





```
def dijkstra(origem, destino):  
    ...  
    while fila:  
        custo, atual = heapq.heappop(fila)  
        ...  
        for vizinho, peso in grafo[atual].items():  
            novo_custo = custo + peso  
            if vizinho not in caminhos or novo_custo < caminhos[vizinho][1]:  
                caminhos[vizinho] = (atual, novo_custo)  
                heapq.heappush(fila, (novo_custo, vizinho))
```

...



Material e Métodos

Grafos e algoritmo de Dijkstra

Nesse ponto do trabalho estava sendo difícil executar o programa no Asus Tinkerboa

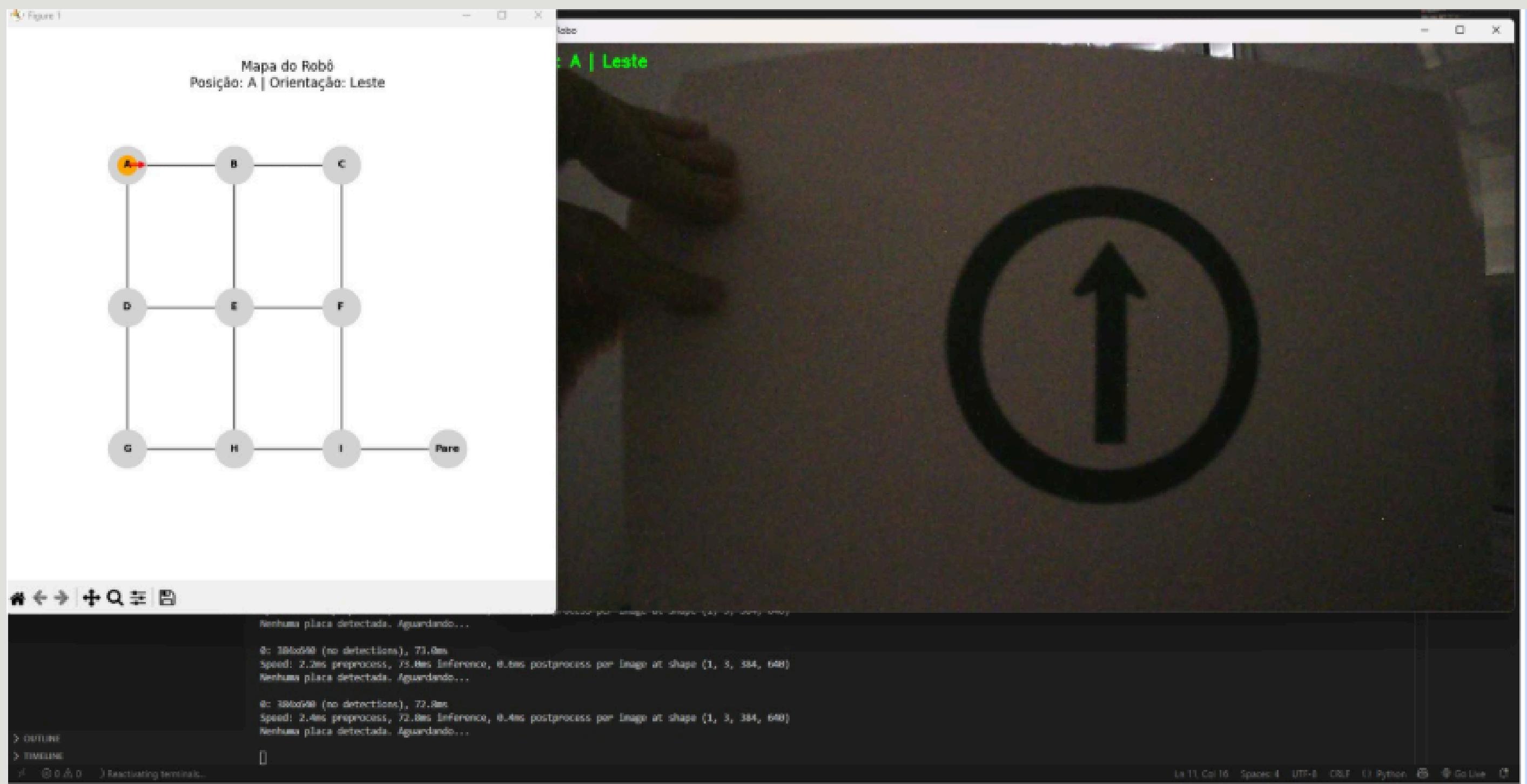
Dijkstra possibilitou a criação de um ambiente de teste

O algoritmo deve simular exatamente o comportamento do robô ao usar o tempo como métrica



Resultados

Software para testes





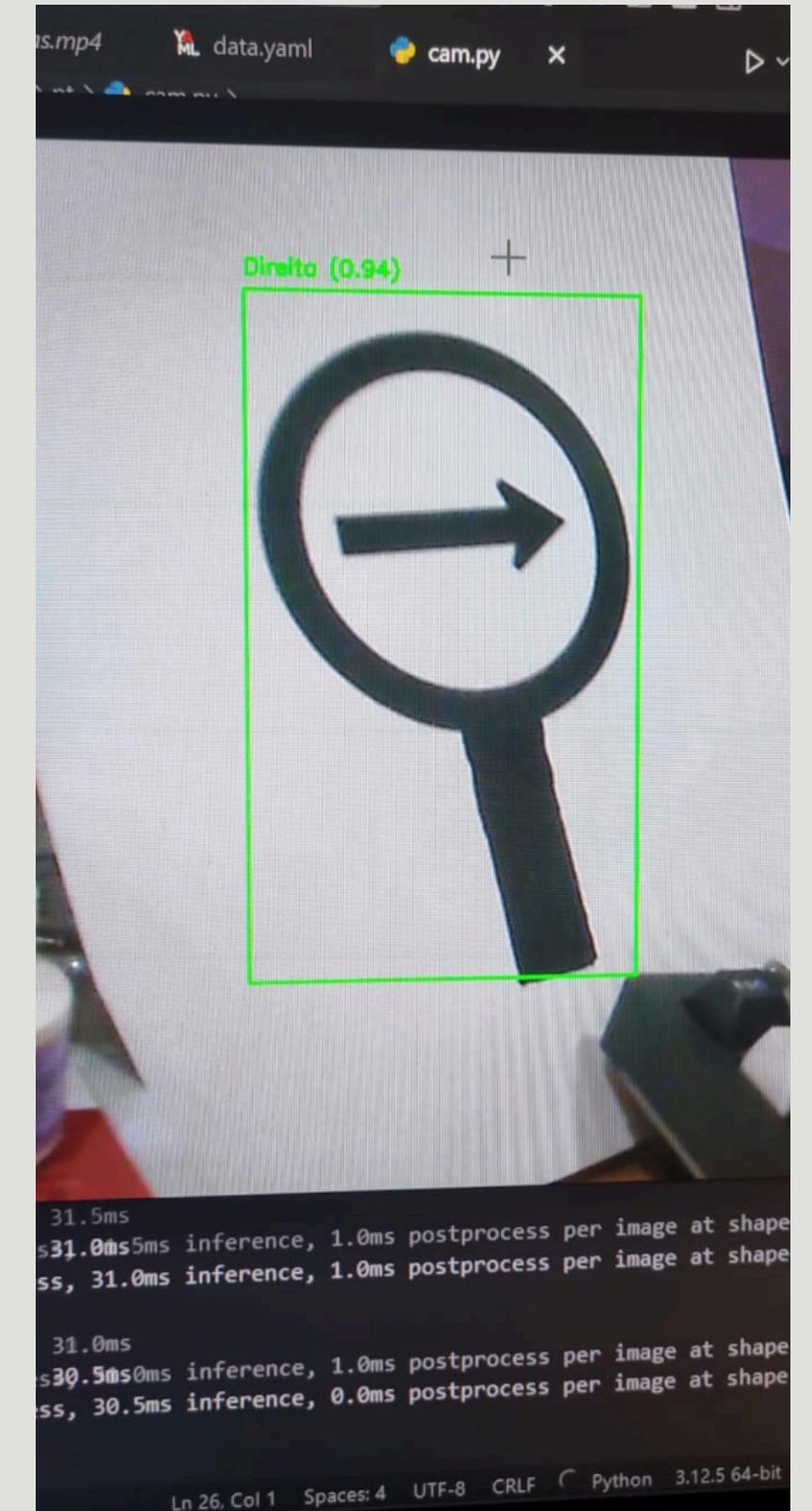
Resultados

Dataset e treinamento

O dataset foi recriado quanto à necessidade com base nos resultados do primeiro treinamento

Os resultados do treinamento tornavam nítidas a necessidade de melhorias

Após a realização de 3 treinamentos, melhorando os datasets. O treinamento apresentou um resultado satisfatório na precisão da detecção de placas





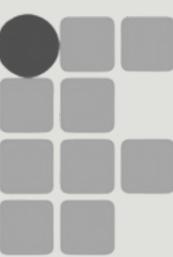
Reto 0.92



Pare 0.91



09/07/2017 11:17:00



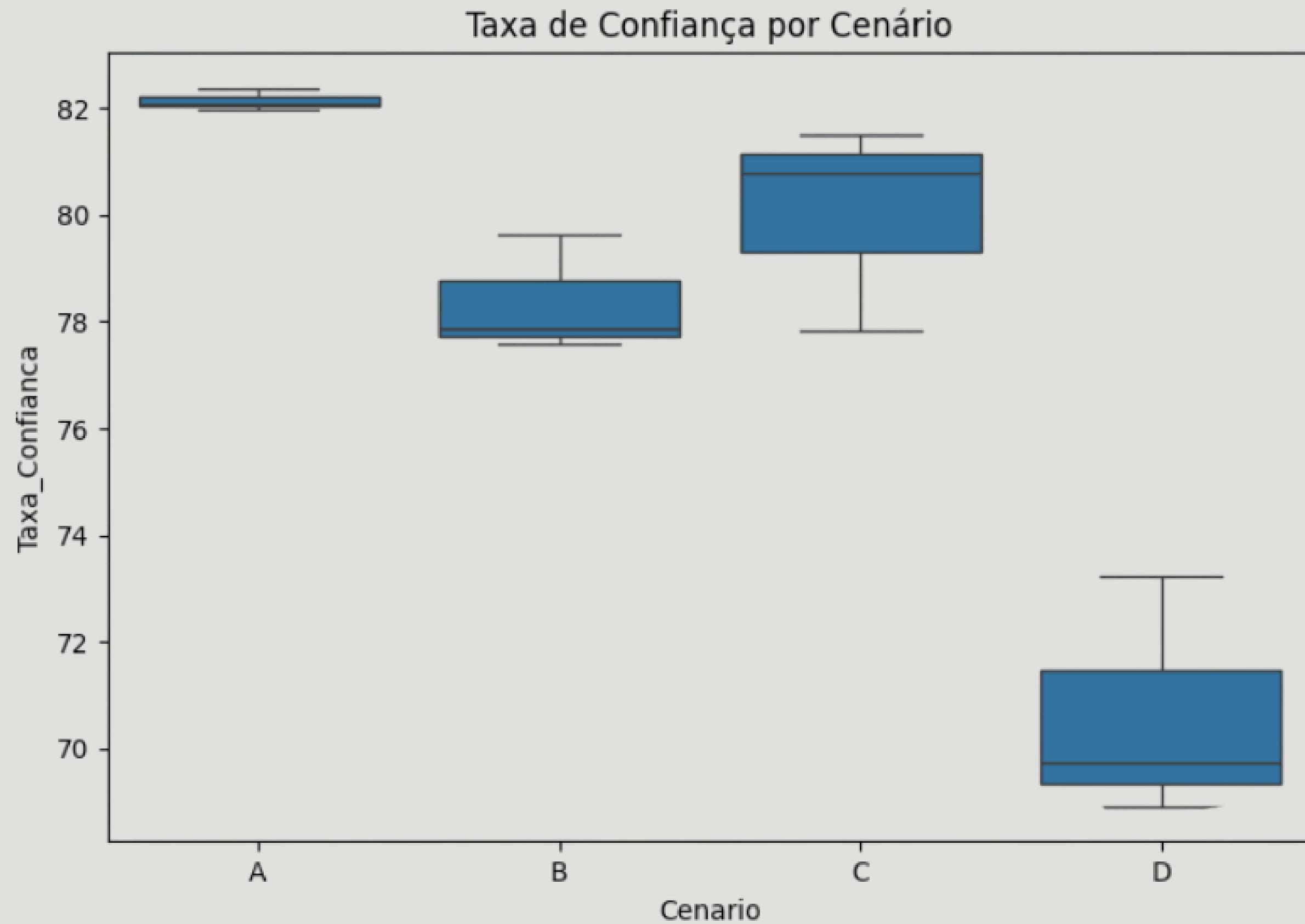
Resultados

Cenários

Cenário	Distância da placa	Condições Visuais	Grau de confiança
A	100cm	Cenário Iluminado	0.7
B	200cm	Cenário Iluminado	0.7
C	100cm	Cenário Escuro	0.7
D	200cm	Cenário Escuro	0.6



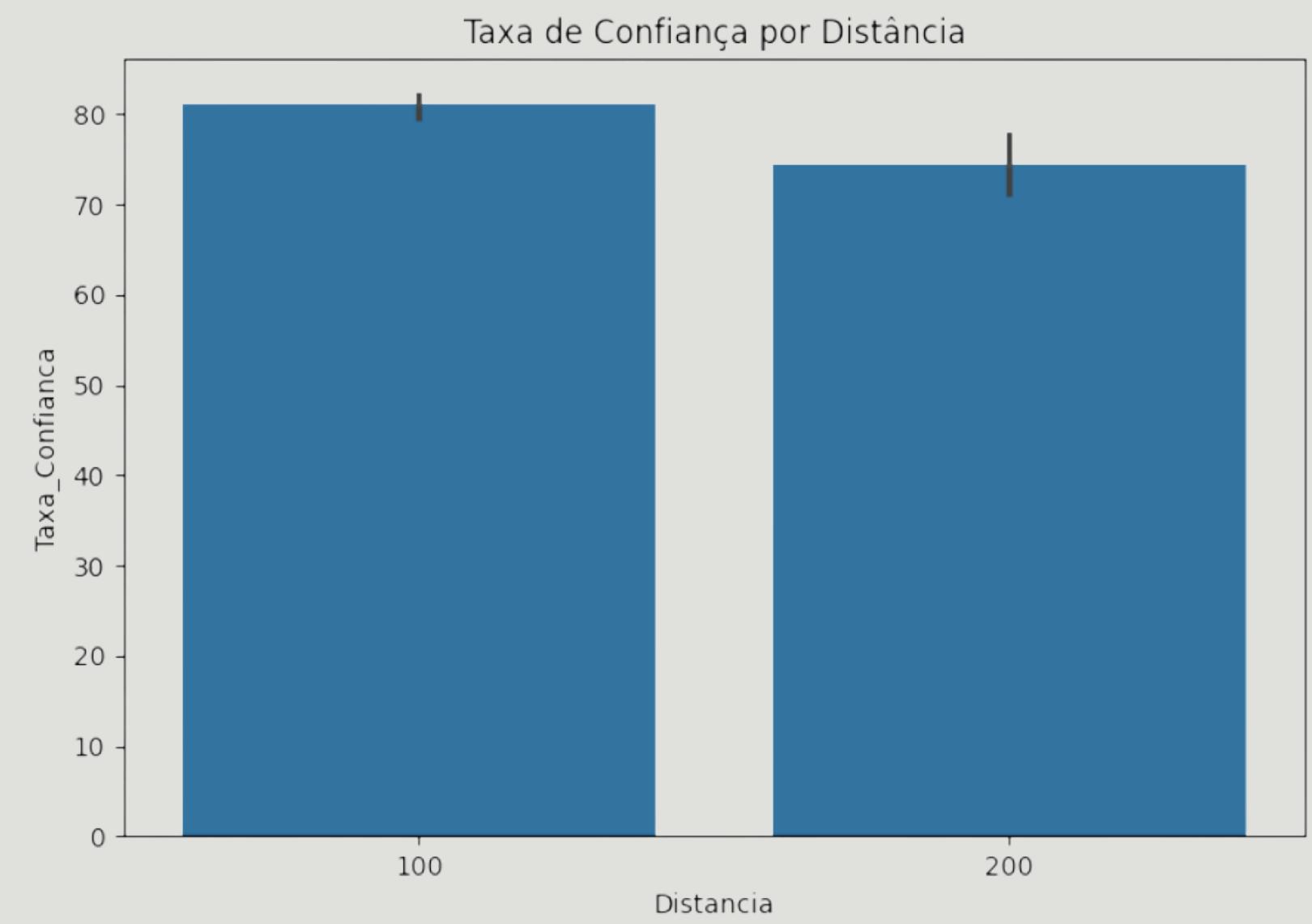
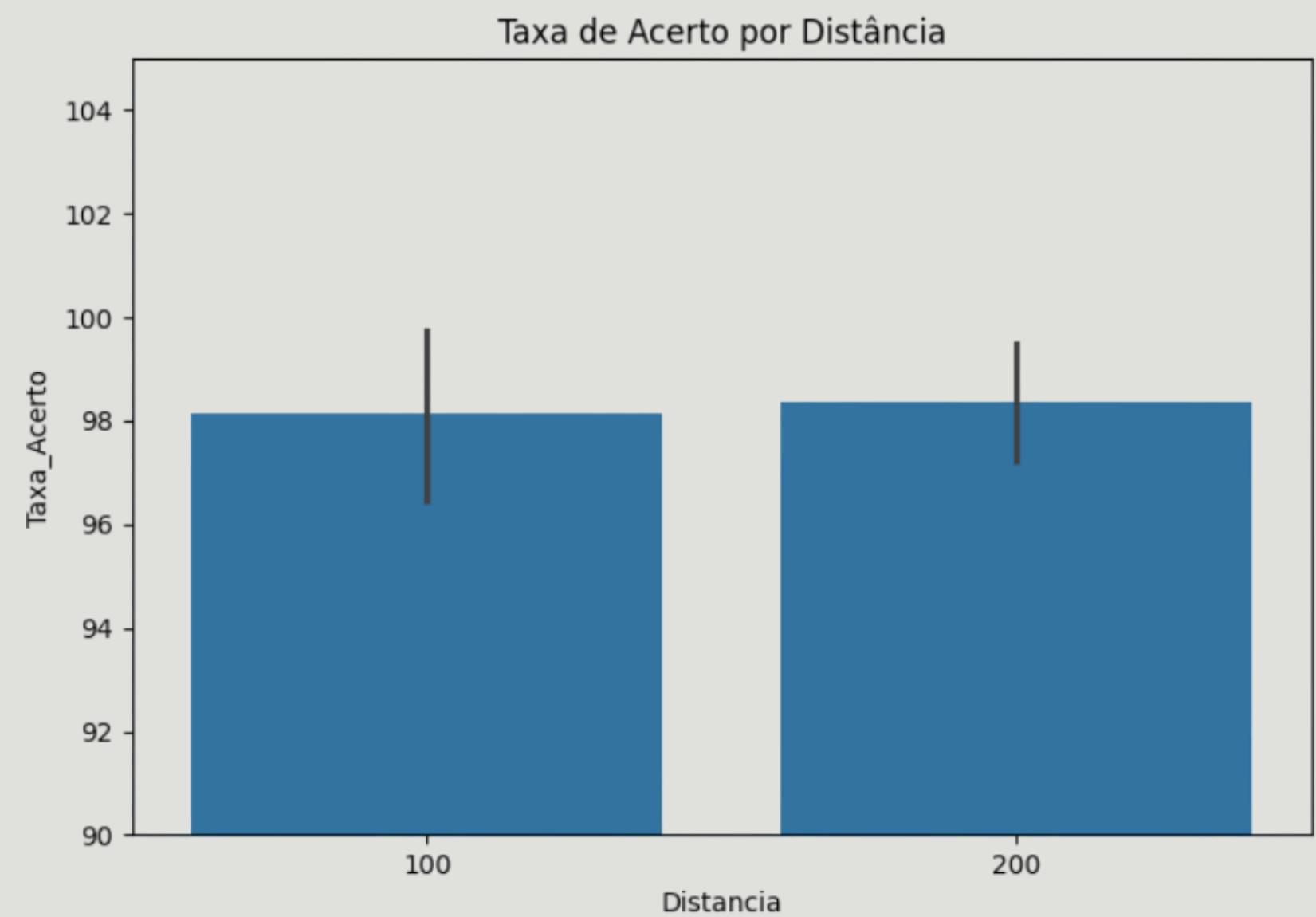
Resultados Análise





Resultados

Análise





Resultados

Conclusões da análise estatística

Taxa de Acerto

p-valores:

- Cenário: 0,4237
- Distância: 0,8399
- Condição Visual: 0,2333
- Caminho: 0,7636

Taxa de Confiança

p-valores:

- Cenário: 0,0001
- Distância: 0,0076
- Condição Visual: 0,0685
- Caminho: 0,9828



Resultados

Implementação

O sistema demonstra comportamento como o esperado na detecção de placas e adaptação de movimento

Entretanto, sobrecarregamentos ocorrem devido ao peso de um programa YoloV8 rodando no sistema. Isso leva a um reboot inesperado

Ainda é preciso configurar todo o sistema pela interface para rodar o programa.



Resultados

Conclusões finais

O sistema apresentou resultados satisfatórios à hipótese H0.

Apesar de suas limitações, foi possível fazer com que a estrutura robótica funcionasse com sucesso rodando o programa de algoritmo que permite seu deslocamento autônomo.

Além disso, a possibilidade de expansão e melhoria torna o trabalho cada vez mais interessante



Resultados

Trabalhos futuros

Melhorias no Software e Hardware com otimizações para aprimorar o desempenho do modelo construído

Atualmente o sistema utiliza interface gráfica, limitando bastante o desempenho



Referências Bibliográficas

BAHRIN, M., OTHMAN, M., AZLI, N., & TALIB, M. (2016). INDUSTRY 4.0: A REVIEW ON INDUSTRIAL AUTOMATION AND ROBOTIC. , 78, 137-143.
<https://doi.org/10.11113/JT.V78.9285>.

Luo, Y., Ci, Y., Jiang, S. et al. A novel lightweight real-time traffic sign detection method based on an embedded device and YOLOv8. J Real-Time Image Proc 21, 24 (2024).
<https://doi.org/10.1007/s11554-023-01403-7>.

WANG, Chien-Yao; LIAO, Hong-Yuan Mark. YOLOv1 to YOLOv10: The fastest and most accurate real-time object detection systems. Taiwan: Academia Sinica; National Taipei University of Technology; <https://arxiv.org/abs/2408.09332>.

SANTOS, Daniel Rodrigues dos; KHOSHELHAM, Kourosh. MAPEAMENTO 3D DE AMBIENTES INTERNOS USANDO DADOS RGB-D. Boletim de Ciências Geodésicas, [S.L.], 21, n. 3, p. 442-464, set. <http://dx.doi.org/10.1590/s1982-21702015000300025>.

Majeed, A., & Rauf, I. (2020). Graph Theory: A Comprehensive Survey about Graph Theory Applications in Computer Science and Social Networks. Inventions.
<https://doi.org/10.3390/inventions5010010>



Obrigado