**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ**

**ANDERSON SOARES DA SILVA**

# SISTEMA AUTÔNOMO DE PREDIÇÃO DE ROTAS PARA VEÍCULOS TERRESTRES

**APUCARANA**

**2023**

**ANDERSON SOARES DA SILVA**

# SISTEMA AUTÔNOMO DE PREDIÇÃO DE ROTAS PARA VEÍCULOS TERRESTRES

## Autonomous Route Prediction System For Land Vehicles

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia da Computação do Curso Engenharia da Computação da Universidade Tecnológica Federal do Paraná.

Orientador: Dr. Fábio Irigon Pereira

Coorientador: Dr. Rafael Gomes Mantovani

**APUCARANA**

**2023**

**ANDERSON SOARES DA SILVA**

# SISTEMA AUTÔNOMO DE PREDIÇÃO DE ROTAS PARA VEÍCULOS TERRESTRES

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia da Computação do Curso Engenharia da Computação da Universidade Tecnológica Federal do Paraná.

Data de aprovação: 16/novembro/2023

---

Fabio Irigon Pereira
Título (doutorado)
Universidade Tecnológica Federal do Paraná

---

Mauricio Eiji Nakai
Título (doutorado)
Universidade Tecnológica Federal do Paraná

---

Muriel De Souza Godoi
Título (mestrado)
Universidade Tecnológica Federal do Paraná

**APUCARANA**
**2023**

Dedico este trabalho com carinho aos meus pais, Luciana e Ailson, ao meu irmão Alison, aos amigos e ao meu amor Mariana. Seu apoio e carinho foram fundamentais.

## AGRADECIMENTOS

Primeiramente, quero expressar minha profunda gratidão a Deus por ter me concedido a oportunidade de alcançar este ponto na minha jornada. Sem Sua orientação e bênçãos, nada disso seria possível.

Em segundo lugar, desejo dedicar um agradecimento especial e do fundo do meu coração à minha amada família, que ao longo desse caminho infindável de desafios, provações e conquistas, esteve ao meu lado de forma incondicional. Seu apoio inabalável, amor e encorajamento foram a força propulsora que me impulsionou a seguir em frente e perseguir meus sonhos. A vocês, minha gratidão é eterna.

Também não posso deixar de expressar minha profunda apreciação ao meu amor, a pessoa que compartilhou cada alegria e cada desafio comigo durante essa emocionante jornada. Seu apoio inquebrantável, compreensão e amor incondicional foram meu porto seguro em meio às tempestades da vida acadêmica.

Aos meus colegas, que se tornaram mais do que amigos; eles se tornaram uma extensão da minha família durante esses anos. Juntos, enfrentamos desafios acadêmicos, celebramos conquistas e nos ajudamos mutuamente a superar os obstáculos. A amizade que compartilhamos fez toda a diferença e transformou os desafios da graduação em experiências valiosas e inesquecíveis.

Ao meu orientador, Fábio, sou profundamente grato pela orientação, conhecimento e apoio que me forneceu ao longo dessa jornada acadêmica. Seus ensinamentos sábios e orientação experta moldaram meu desenvolvimento como engenheiro e me permitiram alcançar um nível de competência que eu nunca imaginei ser possível. Sua dedicação em me auxiliar a atingir todo o meu potencial é algo que guardarei para sempre em meu coração.

Cada um de vocês teve um papel crucial nessa realização e sou profundamente grato por isso. Obrigado, do fundo do meu coração!

"Eu disse essas coisas para que em mim vocês tenham paz. Neste mundo vocês terão aflições; contudo, tenham ânimo! Eu venci o mundo". - João 16:33

## RESUMO

O presente trabalho possui como principal propósito desenvolver um sistema que utilize de visão computacional para detectar obstáculos no ambiente e traçar rotas que os contorne afim de encontrar um objeto objetivo na cena. Com o crescente aumento da frota de veículos, torna-se imprescindível o desenvolvimento de soluções que facilitem a locomoção nas vias, otimizando o tempo de deslocamento e garantindo a segurança de condutores e pedestres. O desenvolvimento é constituído de um conjunto de etapas, incluindo a detecção de profundidade em uma cena usando inteligência artificial, a extração da distância e posição de obstáculos, o mapeamento aéreo do ambiente e predição de rotas. O uso de algoritmos de visão computacional permite uma análise mais precisa do ambiente em que o veículo está inserido, possibilitando o reconhecimento de obstáculos e a escolha do melhor caminho a ser seguido. A pesquisa contribui para a área de navegação veicular, bem como para o desenvolvimento de tecnologias aplicadas à segurança no trânsito.



**Palavras-chave:** robótica; processamento de imagens; visão de robô; veiculos autônomos.

**ABSTRACT**

The present work has as its main purpose to develop a system that utilizes computer vision to detect obstacles in the environment and plan routes to navigate around them in order to find a target object in the scene. With the increasing growth of vehicle fleets, it becomes essential to develop solutions that facilitate transportation on roads, optimizing travel time and ensuring the safety of drivers and pedestrians. The methodology of the study consists of a series of steps, including depth detection in a scene, analysis of the pixel intensity spectrum, aerial mapping of the environment, and route prediction. The use of computer vision algorithms allows for a more precise analysis of the vehicle's surroundings, enabling the recognition of obstacles and the selection of the best path to follow. The research contributes to the field of vehicular navigation, as well as to the development of technologies applied to traffic safety.

**Keywords:** robotics; image processing; robot vision; autonomous vehicles.

## LISTA DE FIGURAS

**SUMÁRIO**

# 1 INTRODUÇÃO

A navegação de veículos terrestres é um problema complexo que tem sido objeto de estudo em diversas áreas do conhecimento, como a engenharia e a ciência da computação. Com o avanço da tecnologia de visão computacional, tornou-se possível utilizar algoritmos para determinar rotas de navegação com precisão e eficiência a partir da manipulação de dados de imagem.

A tarefa da navegação envolve a determinação de uma trajetória segura e eficiente para um robô móvel a partir de informações sensoriais, como dados de sensores de visão, lidar e odometria. Essa tarefa pode ser formalizada como a estimativa da posição e orientação do robô móvel em relação ao ambiente, a detecção e identificação de obstáculos e a tomada de decisões sobre ações a serem realizadas para alcançar um determinado objetivo. Essas decisões devem levar em conta incertezas nas informações sensoriais e nos modelos do ambiente, e podem ser baseadas em técnicas de planejamento e controle de movimento. Nesse sentido, a navegação é vista como um problema de inferência probabilística e tomada de decisão em ambientes incertos e dinâmicos (THRUN; BURGARD; FOX, 2005).

Este trabalho tem como objetivo desenvolver um sistema que utilize de algoritmos de visão computacional para o mapeamento de obstáculos no ambiente e que determine rotas de navegação para contorna-los e encontrar um objetivo na cena, como uma caixa vermelha. Para isso, serão realizadas etapas como a estimação da profundidade das imagens capturadas pela câmera, a análise do espectro de intensidade do pixel nas imagens com profundidade, determinação do mapa aéreo com os obstáculos estimados e a predição de rotas utilizando o Algoritmo A* (A-Estrela).

Com a realização deste trabalho, espera-se contribuir para o avanço da tecnologia de navegação de veículos terrestres, promovendo a utilização de algoritmos de visão computacional nos sistemas de navegação.

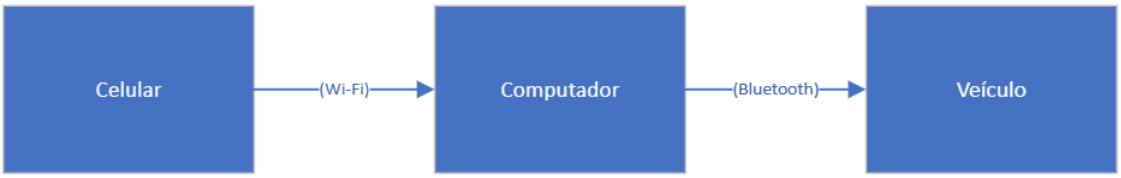
## 1.1 Descrição do projeto

A navegação autônoma busca criar sistemas independentes de intervenção humana, capazes desviar de obstáculos e decidir trajetos de forma automática. No entanto, a navegação autônoma enfrenta desafios significativos, como a capacidade de tomar decisões em tempo real em um ambiente dinâmico e imprevisível, a segurança em todas as condições, incluindo a detecção e correção de falhas, e a necessidade de estabelecer padrões e regulamentações claras.

O problema da localização consiste em estimar a posição do robô em relação ao ambiente, com isso, é possível gerar dados que irão realizar um modelo de mundo (OLSON, 2000). Para isso é necessário considerar alguns paradigmas que serão detalhados posteriormente, como métodos locais e globais, ambientes estáticos e dinâmicos, métodos passivos e ativos e

um robô em comparação à múltiplos robôs (THRUN *et al.*, 2000). Nesse contexto, é imperativo que analisemos o fluxograma apresentado na figura 1, o qual ilustra de maneira simplificada o andamento do projeto.

**Figura 1 – Fluxograma simplificado do projeto**




**Fonte: Autoria Própria.**

Além disso, um outro grande desafio na navegação autônoma é a escolha da melhor rota a ser seguida pelo veículo, levando em consideração as condições do ambiente, a segurança e a eficiência do percurso. Para resolver essa questão, é comum o uso de sistemas de navegação baseados em GPS e mapas digitais, no entanto, esses sistemas possuem algumas limitações, como a dificuldade de operar em ambientes com pouca visibilidade, como em áreas com muitos obstáculos.

Nesse contexto, a aplicação de algoritmos de visão computacional surge como uma alternativa para melhorar a precisão e eficiência da navegação autônoma, permitindo que o veículo obtenha informações detalhadas do ambiente e possa tomar decisões mais precisas e seguras. Os algoritmos de visão computacional podem ser usados para identificar obstáculos, marcas de trânsito, placas de sinalização e outras informações relevantes para a navegação do veículo.

Assim, o objetivo deste projeto é aplicar algoritmos de visão computacional para determinação de rotas de navegação de veículos terrestres. Para isso, serão realizadas as etapas de detecção de profundidade em uma cena, a analise do espectro de intensidade do pixel da imagem processada, o mapeamento aéreo do ambiente e a predição de rotas no mapa aéreo. A contribuição deste trabalho está em desenvolver uma solução eficiente para o problema da navegação autônoma em ambientes urbanos, melhorando a segurança e eficiência do transporte terrestre.

Vale ressaltar que, a modelagem cinemática do veículo terrestre é realizada como complemento neste projeto. Está sendo considerado o trabalho de conclusão de curso de um outro membro do curso de engenharia de computação da UTFPR Apucarana, Mariana Gonçalves Rodrigues (RODRIGUES, 2023). Sendo assim, este trabalho consiste em realizar todos os procedimentos de cálculo de rota a partir de imagens para fornecer informações de ângulo e distância da rota para um veículo que realiza sua própria modelagem cinemática.

## 1.2 Motivação

A navegação autônoma tem se mostrado uma área promissora em diversos setores, como transporte, como é mostrado na figura 4, agricultura, como em colheitadeiras autônomas conforme a figura 2, mineração como em caminhões de transporte de minérios conforme é evidenciado na figura 3 e socorro em ambientes inacessíveis para o humano. A capacidade de um veículo se deslocar de forma autônoma, identificando obstáculos e determinando rotas, pode trazer inúmeros benefícios, como redução de custos, aumento da produtividade e maior segurança. Nesse sentido, os algoritmos de visão computacional combinados com a inteligência artificial para planejamento de rotas, são uma ferramenta fundamental para a implementação de sistemas autônomos.

**Figura 2 – Colheita Automatizada**

**Fonte: (GREENAWAY, 2020) .**

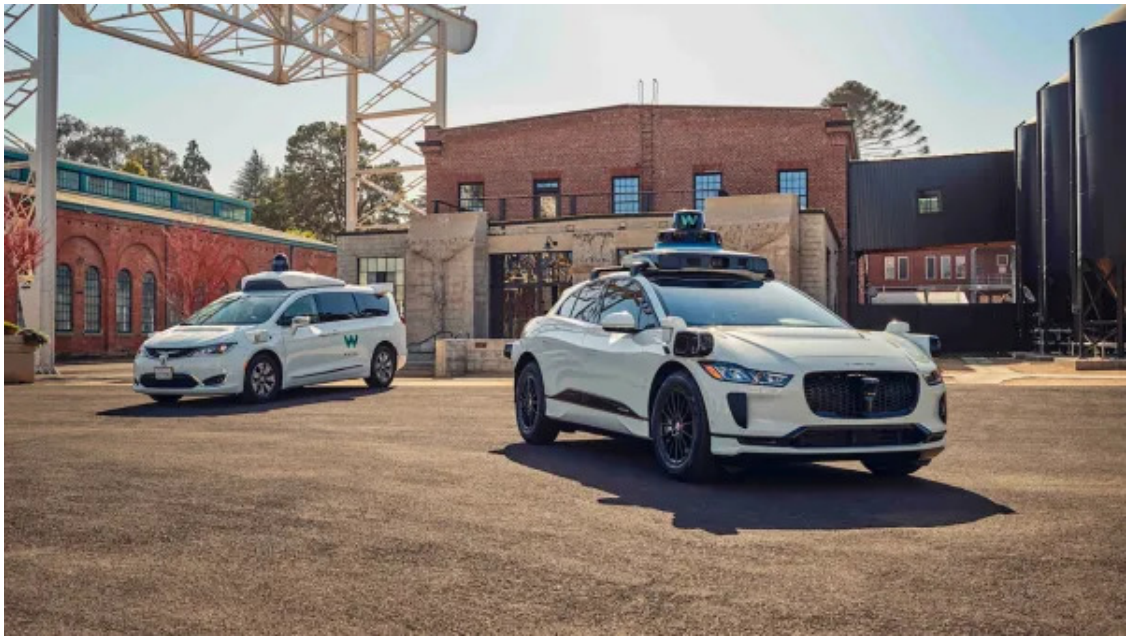
Além disso, recentemente houve uma significativa evolução no campo da condução autônoma. Em um artigo publicado em 28 de maio de 2023, Kristin Houser relata a parceria entre Uber e Waymo para oferecer corridas em carros totalmente autônomos em Phoenix, Arizona (HOUSER, 2023).

A visão computacional permite que um veículo autônomo "enxergue"o ambiente ao seu redor e identifique obstáculos, como outros veículos, pedestres, animais e objetos estáticos. Com essa informação, o sistema pode determinar a melhor rota a seguir, evitando colisões e garantindo um deslocamento seguro e eficiente.

**Figura 3 – Caminhão autônomo Cat 793F em teste na Mina Navajo.**


**Fonte: (MORELL, 2017) .**

**Figura 4 – Waymo - Carro autônomo da Google**

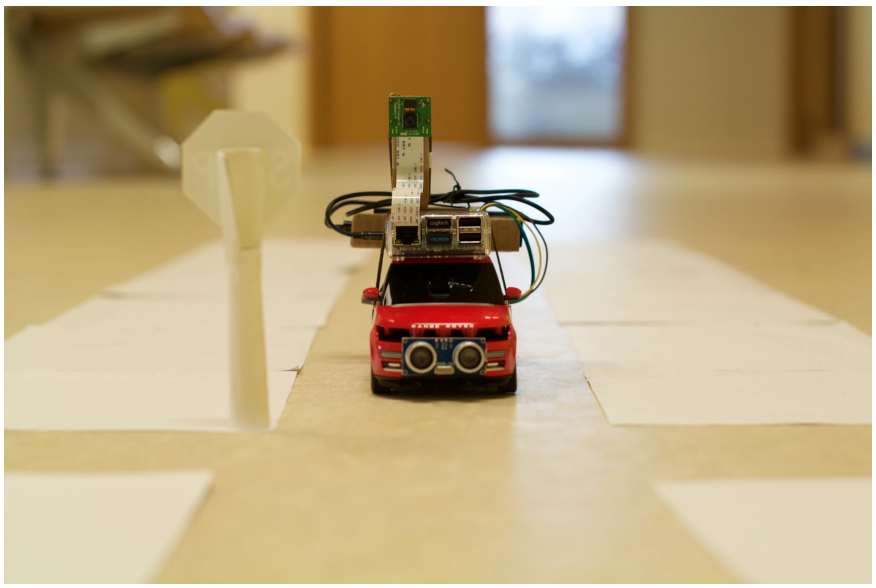


**Fonte: (HOUSER, 2023) .**

## 1.3 Carros de Controle Remoto com Condução Autônoma

Para abordar a questão da navegação autônoma, é importante destacar os avanços tecnológicos que têm permitido a realização de pesquisas e desenvolvimentos inovadores nessa área. Uma das vertentes interessantes e acessíveis para experimentação e prototipagem de sistemas de navegação autônoma são os veículos controlados remotamente (RC cars), especialmente quando equipados com tecnologias de automação. Esses veículos, quando adaptados com sensores e algoritmos adequados, podem ser transformados em veículos autônomos de pequena escala.

**Figura 5 – Self driving RC car**



**Fonte: (WANG, 2015) .**

Os self-driving RC cars, ou carros de controle remoto autônomos, têm ganhado destaque como plataformas de teste e desenvolvimento para algoritmos de navegação autônoma. Eles oferecem uma maneira acessível e segura de explorar e aprimorar tecnologias relacionadas à condução autônoma. Esses veículos podem ser equipados com sensores como câmeras, LiDAR (Light Detection and Ranging), ultrassom, giroscópios e acelerômetros para coletar dados do ambiente ao seu redor.

A utilização de self-driving RC cars como plataforma de pesquisa e desenvolvimento apresenta diversas vantagens:

## 1.4 Objetivos

O presente trabalho tem como objeto de pesquisa a aplicação de algoritmos de visão computacional para definição de rotas de navegação de veículos terrestres utilizando inteligência artificial. Deseja-se desenvolver um sistema capaz de identificar e reconhecer obstáculos

em tempo real, traçar rotas seguras, e controlar a navegação do veículo terrestre de forma autônoma.

### 1.4.1 Objetivo geral

Desenvolver um sistema autônomo de navegação de veículos terrestres utilizando algoritmos de visão computacional para determinação de rotas seguras em ambientes complexos e dinâmicos.

### 1.4.2 Objetivos específicos

1. Para empregar técnicas de visão computacional com o propósito de identificar e reconhecer objetos relevantes para a locomoção do veículo terrestre, é essencial, antes de tudo, compreender as técnicas disponíveis (ALPAYDIN, 2010). Além disso, é importante desenvolver um sistema de localização probabilística que possa estimar a posição do veículo em tempo real, utilizando uma câmera e técnicas eficazes de processamento de imagens, a fim de melhorar a precisão e confiabilidade da navegação autônoma (FOX; BURGARD; THRUN, 1999; MONTEMERLO *et al.*, 2002).

2. Uma vez estabelecido o sistema de localização, é possível implementar algoritmos de planejamento de trajetórias capazes de considerar informações do ambiente, restrições de movimento do veículo e objetivos de navegação definidos no sistema. Nesse sentido, técnicas de aprendizado de máquina e otimização têm se mostrado bastante efetivas e o algoritmo A Estrela foi utilizado para estimar a melhor rota (BERG; PENNY, 2011; KARAMAN; FRAZZOLI, 2011).

3. Para avaliar o desempenho do sistema proposto, é necessário testá-lo em diferentes ambientes, incluindo ambientes estáticos e dinâmicos, a fim de verificar sua robustez e eficiência em situações adversas (THRUN; BURGARD; FOX, 2005; THRUN *et al.*, 2008). Além disso, é importante integrar o sistema de navegação autônoma em um veículo terrestre real e realizar testes práticos em um ambiente controlado para validar a efetividade e aplicabilidade da solução proposta (FOX; BURGARD; THRUN, 1999; THRUN *et al.*, 2008).

## 1.5 Estrutura do trabalho

A estrutura deste trabalho é composta por cinco seções, cada uma desempenhando um papel fundamental na abordagem do projeto:

1. **Introdução**: Nesta seção, apresentamos de maneira clara e concisa a ideia principal do projeto, juntamente com os objetivos a serem alcançados durante o desenvolvimento do trabalho. Ela serve como uma visão geral do que será discutido no trabalho.

2. **Revisão bibliográfica**: A seção de revisão bibliográfica explora a pesquisa existente relacionada ao tema do projeto. Ela fornece um contexto, destacando as teorias e estudos relevantes que embasam o trabalho. Isso ajuda a estabelecer a base teórica e a justificativa para o projeto.

3. **Materiais e métodos**: Aqui detalhamos os materiais, equipamentos e abordagens metodológicas utilizados para a execução do projeto. Isso inclui a definição do problema, a coleta de dados, as técnicas empregadas e a descrição de qualquer algoritmo desenvolvido.

4. **Resultados**: A seção de resultados destaca o que foi alcançado ao longo do projeto. Ela inclui a apresentação dos resultados obtidos e outras informações relevantes que demonstram o progresso e as descobertas.

5. **Conclusão**: Na seção de conclusão, resumimos os principais resultados e descobertas do projeto, relacionando-os aos objetivos iniciais. Além disso, discutimos a importância dos resultados e suas possíveis implicações. Esta seção encerra o trabalho de forma abrangente.

# 2 REVISÃO BIBLIOGRÁFICA

Este capítulo oferece uma revisão abrangente da literatura relevante para o projeto em questão. A revisão bibliográfica é uma etapa crucial para compreender o contexto e o estado da arte das tecnologias e conceitos que serão explorados ao longo deste trabalho. Nesta seção inicial, apresentaremos uma visão geral das principais abordagens e métodos que desempenham um papel fundamental na execução deste projeto. Começaremos discutindo o Modelo de Depth Estimation and Super-Resolution (MiDaS), uma solução inovadora para a estimativa de profundidade a partir de imagens monoscópicas. Em seguida, exploraremos técnicas de análise de espectro de intensidade do pixel, mapeamento aéreo, o algoritmo A* e a busca de objetos específicos, todos os quais desempenham um papel vital na aplicação de veículos autônomos e sistemas de visão computacional. Por fim, discutiremos as possíveis aplicações desses conceitos em veículos autônomos, delineando a relevância e o potencial dessas tecnologias na criação de sistemas de navegação autônoma avançados.

## 2.1 Redes Neurais Convolucionais (CNNs) em Visão Computacional

As Redes Neurais Convolucionais (CNNs) têm desempenhado um papel revolucionário em tarefas de visão computacional nas últimas décadas. Elas são uma classe de redes neurais profundas projetadas especificamente para processar dados de imagem de maneira eficaz e aprender representações hierárquicas.

As CNNs são compostas por camadas convolucionais, de pooling e totalmente conectadas, que trabalham juntas para extrair características relevantes das imagens. As camadas convolucionais aplicam filtros convolucionais para detectar padrões, como bordas e texturas, em várias regiões da imagem. As camadas de pooling reduzem a dimensionalidade dos mapas de características, mantendo as informações mais importantes. As camadas totalmente conectadas realizam a classificação ou regressão com base nas características extraídas (GOODFELLOW; BENGIO; COURVILLE, 2016).

Na detecção de objetos, as CNNs se destacam na localização e identificação de objetos em uma cena. Elas são frequentemente treinadas em grandes conjuntos de dados rotulados para aprender a reconhecer objetos de interesse. Uma vez treinadas, essas redes podem identificar objetos em novas imagens com alta precisão.

Além disso, as CNNs também são amplamente utilizadas em tarefas de segmentação semântica, onde a meta é atribuir um rótulo a cada pixel da imagem, identificando a que objeto ou classe ele pertence. Isso é essencial para a compreensão detalhada do conteúdo das imagens.

As CNNs têm sido empregadas em uma variedade de aplicações de veículos autônomos, incluindo detecção de obstáculos, reconhecimento de sinais de trânsito, classificação de

objetos na estrada e muito mais. Sua capacidade de aprender representações complexas diretamente dos dados as torna uma ferramenta poderosa em sistemas de visão computacional.

### 2.1.1 MiDaS (estimativa de profundidade monocular)

A estimação de profundidade a partir de uma única imagem é um desafio complexo, uma vez que não há informações estéreo disponíveis (ou seja, duas câmeras) para comparação de disparidade. O Modelo de Depth Estimation and Super-Resolution (MiDaS) aborda esse problema de forma altamente eficaz, utilizando uma abordagem baseada em aprendizado profundo (RANFTL *et al.*, 2022).

O MiDaS é uma rede neural convolucional profunda desenvolvida pela Intel Technology para estimar a profundidade de cenas a partir de imagens monoscópicas. Sua capacidade de fornecer estimativas de profundidade precisas é notável, mesmo em condições desafiadoras, como imagens capturadas em ambientes externos com variações de iluminação e texturas complexas, isso pode ser observado na figura 6.

**Figura 6 – MiDaS**

**Fonte: (RANFTL *et al.*, 2022) .**

Este modelo foi treinado em até 12 conjuntos de dados diferentes, incluindo ReDWeb, DIML, Movies, MegaDepth, WSVD, TartanAir, HRWSI, ApolloScape, BlendedMVS, IRS, KITTI e NYU Depth V2, com otimização multiobjetivo. Ele é altamente versátil, oferecendo vários modelos com diferentes níveis de qualidade e desempenho, adequados para diferentes aplicativos e recursos de hardware.

O MiDaS é capaz de lidar com múltiplas escalas de profundidade, o que é crucial para capturar objetos próximos e distantes em uma cena, cada um com sua própria escala de profundidade. Para alcançar isso, o MiDaS utiliza uma abordagem de orientação adaptativa em várias escalas durante a inferência.

A precisão do MiDaS é geralmente avaliada em termos de métricas de erro de profundidade em conjuntos de dados de teste. Essas métricas ajudam a determinar o quão bem o

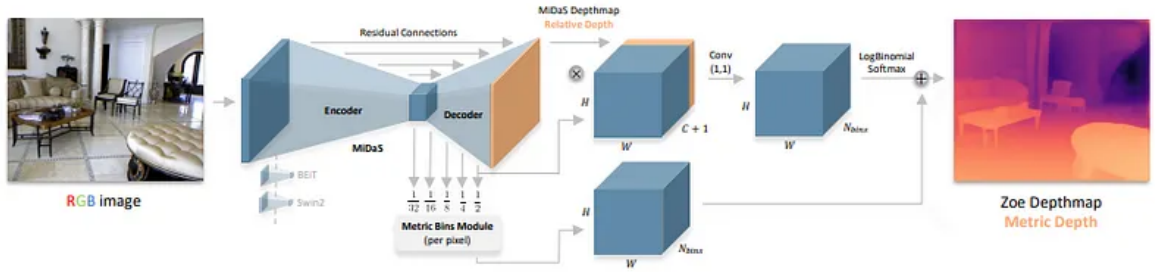
modelo realiza a tarefa de estimação de profundidade em comparação com outros métodos existentes.

Para usar o MiDaS, você pode escolher um modelo específico de acordo com suas necessidades e fazer o download dos pesos correspondentes. Depois, basta executar o modelo em suas imagens de entrada para gerar mapas de profundidade correspondentes.

Além disso, o MiDaS é parte integrante de diversos projetos da Intel Labs, como o ZoeDepth e o LDM3D, ampliando seu uso em várias aplicações, incluindo visão computacional avançada e geração de dados de profundidade a partir de imagens.

O ZoeDepth refere-se a um modelo de estimativa de profundidade que combina a estimativa de profundidade relativa e métrica. Em essência, ZoeDepth é um modelo que tenta unir esses dois aspectos da estimativa de profundidade, o que leva a um desempenho de generalização excelente, mantendo a escala métrica. O modelo ZoeD-M12-NK é pré-treinado em 12 conjuntos de dados diferentes usando a estimativa de profundidade relativa e, posteriormente, é ajustado em dois conjuntos de dados específicos usando a estimativa de profundidade métrica. Para realizar essa tarefa, o ZoeDepth utiliza um novo módulo de cabeçote chamado "Metric Bins Module"para calcular a profundidade métrica. Durante a inferência, o modelo ZoeDepth é capaz de direcionar automaticamente a imagem de entrada para o cabeçote apropriado, com base em um classificador latente. Isso faz do ZoeDepth uma solução versátil e eficaz para a estimativa de profundidade, abordando tanto a profundidade relativa quanto a métrica.

**Figura 7 – Arquitetura ZoeDepth**

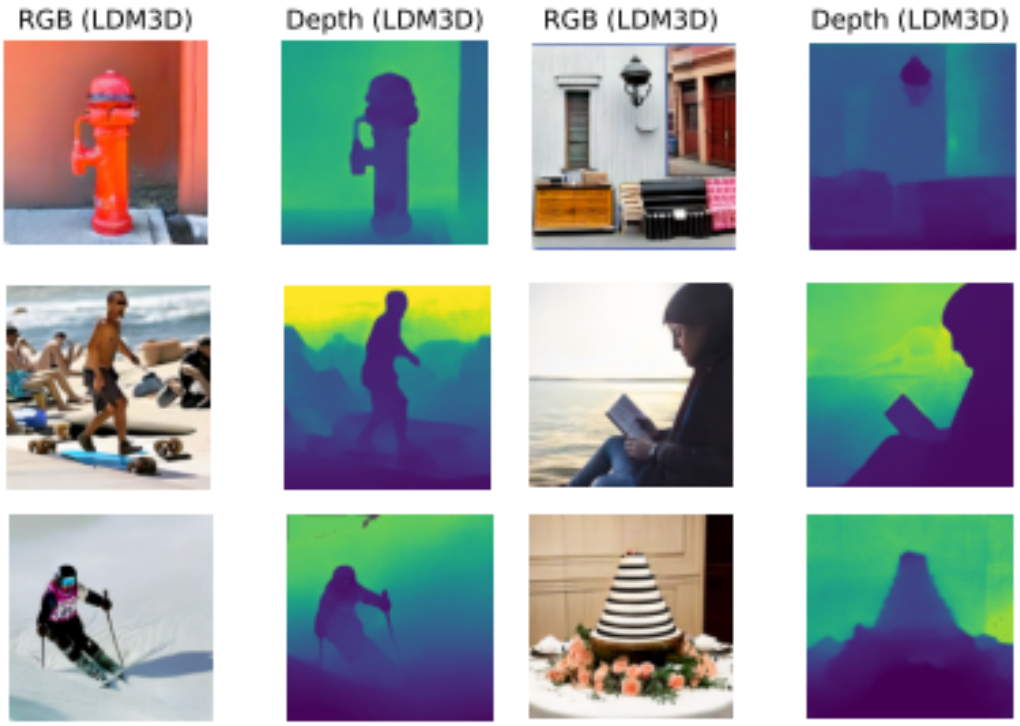



**Fonte: (BHAT *et al.*, 2023) .**

Quanto ao LDM3D (Modelo de Difusão Latente para 3D), ele é um modelo que gera dados de imagem e mapas de profundidade a partir de um prompt de texto específico. Esse modelo é treinado em um conjunto de dados contendo imagens RGB, mapas de profundidade e legendas associadas. O LDM3D é capaz de criar imagens RGBD a partir de prompts de texto, o que o torna uma ferramenta poderosa para gerar conteúdo visual baseado em descrições textuais. Além disso, os criadores desenvolveram um aplicativo chamado "DepthFusion"que utiliza as imagens RGB geradas e os mapas de profundidade para criar experiências imersivas e interativas de visão de 360 graus. Esse modelo tem o potencial de impactar diversas indústrias, como entretenimento, jogos, arquitetura e design, ao facilitar a geração de conteúdo visual a

partir de texto, ampliando as possibilidades criativas e visuais. O LDM3D representa uma contribuição significativa para o campo da IA generativa e da visão computacional, demonstrando seu potencial para contribuir significativamente a criação de conteúdo e experiências digitais.

**Figura 8 – Visualização de profundidade com LDM3D**




**Fonte: (STAN *et al.*, 2023) .**

O MiDaS está em constante evolução, com lançamentos recentes introduzindo novos modelos e melhorias significativas em sua precisão. Cada modelo é acompanhado de informações detalhadas sobre seu desempenho em diferentes conjuntos de dados, para que você possa escolher o mais adequado para sua aplicação específica.

## 2.2 Variação da Intensidade luminosa dos Pixels

A análise da intensidade luminosa dos pixels é uma etapa fundamental na percepção visual, particularmente em sistemas de visão computacional voltados para a detecção e evasão de obstáculos. Esta técnica é usada para extrair informações essenciais das imagens capturadas, permitindo a identificação de objetos e obstáculos com base em variações de intensidade luminosa (PEDRINI; SCHWARTZ, 2008).

Essa técnica parte do princípio de que os obstáculos em uma cena são representados com variações na intensidade luminosa dos pixels na imagem com profundidade. Essas variações são causadas porque o midas retorna uma máscara em que a intensidade do pixel é inversamente proporcional à sua profundidade como consta na figura 18.

A abordagem começa com a seleção de uma linha ou região de interesse na imagem, comumente denominada de "linha central", onde a análise será realizada. Essa linha é escolhida com base em considerações práticas e na geometria da câmera.

Uma das principais tarefas na análise é a detecção de picos de intensidade por meio do cálculo da derivada na linha central. Picos positivos indicam regiões de alta intensidade, enquanto picos negativos indicam regiões de baixa intensidade. Os picos positivos geralmente correspondem a bordas ou limites de objetos, enquanto os picos negativos podem representar depressões ou vazios.

Após a detecção dos picos de intensidade, eles são emparelhados para criar uma representação dos obstáculos na cena. Cada par de picos indica a largura do obstáculo, enquanto a intensidade máxima entre eles fornece uma estimativa da altura do obstáculo. Essa representação é fundamental para a identificação e localização de obstáculos no ambiente.

## 2.3 Mapeamento Aéreo e Representação de Obstáculos

O mapeamento aéreo e a representação de obstáculos são elementos fundamentais em sistemas de navegação autônoma. Nesta seção, exploramos como o ambiente circundante é mapeado em um plano bidimensional (mapa aéreo) e como os obstáculos são representados nesse contexto.

O mapeamento aéreo é uma técnica que cria uma representação abstrata do ambiente em que um veículo autônomo está operando. Essa representação permite que o veículo planeje rotas seguras e evite colisões com obstáculos. Existem várias abordagens para o mapeamento, incluindo mapas 2D e 3D, mas o que será utilizado é apenas o mapeamento 2D conforme é exemplificado na figura 9.

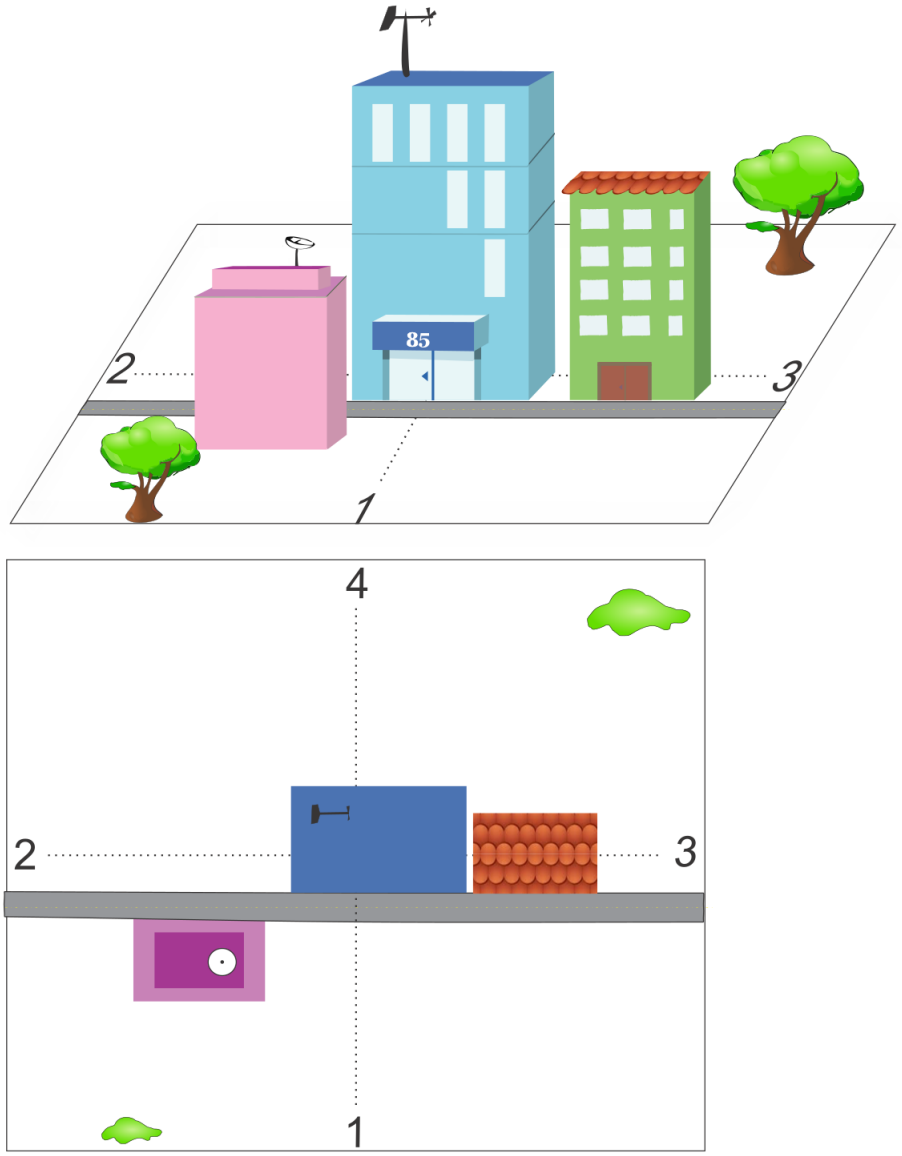
A criação do mapa aéreo envolve a identificação e a representação de obstáculos significativos detectados na cena. Os obstáculos filtrados, que atendem aos critérios especificados, são considerados na criação do mapa. Para cada obstáculo, as coordenadas correspondentes no mapa aéreo são marcadas com o valor "1,"indicando a presença de um obstáculo naquela posição.

A representação de obstáculos no mapa aéreo é essencial para a navegação autônoma, pois fornece informações sobre a distribuição espacial dos obstáculos no ambiente. Essa representação é usada para evitar colisões e planejar trajetórias livres de obstáculos. Quanto mais precisa e atualizada for a representação, mais segura será a navegação do veículo autônomo.

## 2.4 Busca de Objetos Específicos (Objeto Vermelho)

A identificação e categorização de objetos em uma foto colorida comum é uma tarefa complexa que envolve uma série de conceitos e etapas fundamentais em visão computacional

**Figura 9 – Visão perpendicular e aérea**




**Fonte: (BREDA, 2017) .**

e processamento de imagem. Essa abordagem visa reconhecer objetos de interesse em uma cena, determinar sua posição e, em alguns casos, classificá-los em categorias específicas, como no caso da busca por um objeto vermelho em uma imagem.

Para realizar essa tarefa, é necessário seguir uma série de passos interligados. Primeiro, a imagem é segmentada, ou seja, é dividida em regiões que correspondem a diferentes objetos ou partes da cena. Isso é feito com base em propriedades como cor, textura e bordas. A segmentação ajuda a isolar as áreas de interesse na imagem. É possível observar isso conforme o exemplo 10.

Uma vez que a imagem está segmentada, as características relevantes de cada região são extraídas. No contexto da busca por um objeto vermelho, a característica principal a ser

**Figura 10 – Exemplo segmentação por cores**

**Fonte: (MAULION, 2021) .**

considerada é a cor. Isso envolve analisar os valores dos canais de cores RGB para determinar se uma região é predominantemente vermelha (REN *et al.*, 2015).

Como as imagens podem conter ruído, como variações na iluminação ou pequenas flutuações de cor, é comum aplicar técnicas de filtragem para reduzir o impacto do ruído na detecção de objetos.

Para identificar os limites dos objetos e separá-los do fundo da imagem, a detecção de contornos é frequentemente utilizada. Isso ajuda a delinear a forma dos objetos.

Se for necessário categorizar objetos, como diferenciar um objeto vermelho de outros objetos na cena, pode ser necessário treinar um modelo de classificação. Esse modelo aprenderá a reconhecer padrões específicos associados a cada categoria de objeto.

Após a detecção e categorização dos objetos, podem ser aplicadas técnicas de pós-processamento para refinar os resultados, eliminar falsos positivos ou realizar operações adicionais, como rastreamento de objetos ao longo do tempo.

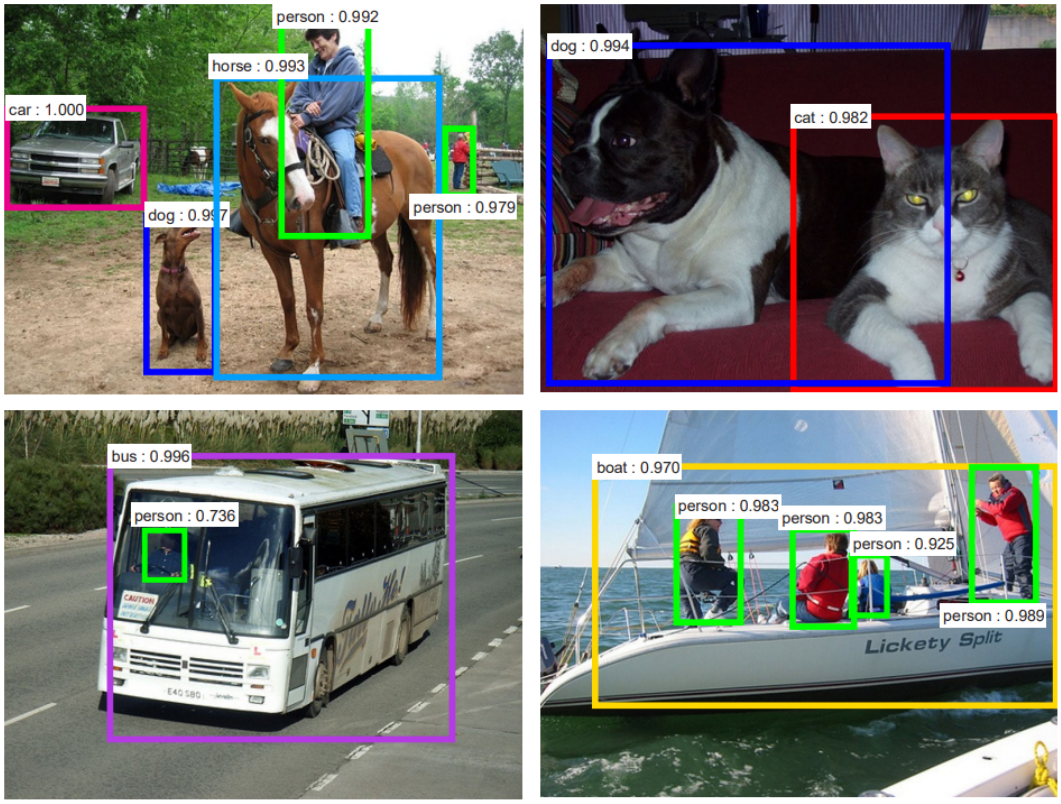
A escolha de algoritmos e técnicas específicas dependerá do contexto do projeto e da natureza dos objetos a serem detectados. Existem abordagens baseadas em redes neurais convolucionais (CNNs) para tarefas de detecção de objetos, bem como métodos mais tradicionais que utilizam técnicas de processamento de imagem.

A teoria por trás da identificação e categorização de objetos em uma imagem RGB é essencial para desenvolver sistemas de visão computacional eficazes, como os utilizados em veículos autônomos e em uma variedade de outras aplicações de processamento de imagem. A combinação de técnicas avançadas de processamento de imagem e aprendizado de máquina desempenha um papel fundamental na realização dessas tarefas de forma precisa e eficiente.

## 2.5 Algoritmo A * (A-Estrela)

O algoritmo A* é um método de busca de caminho que visa encontrar a rota mais curta entre dois pontos em um grafo ponderado. Sua eficácia é baseada em uma combinação inteligente de informações de custo e heurística. A abordagem de busca é guiada por uma função de

**Figura 11 – Exemplo de detecções em imagens coloridas**




**Fonte: (REN *et al.*, 2015) .**

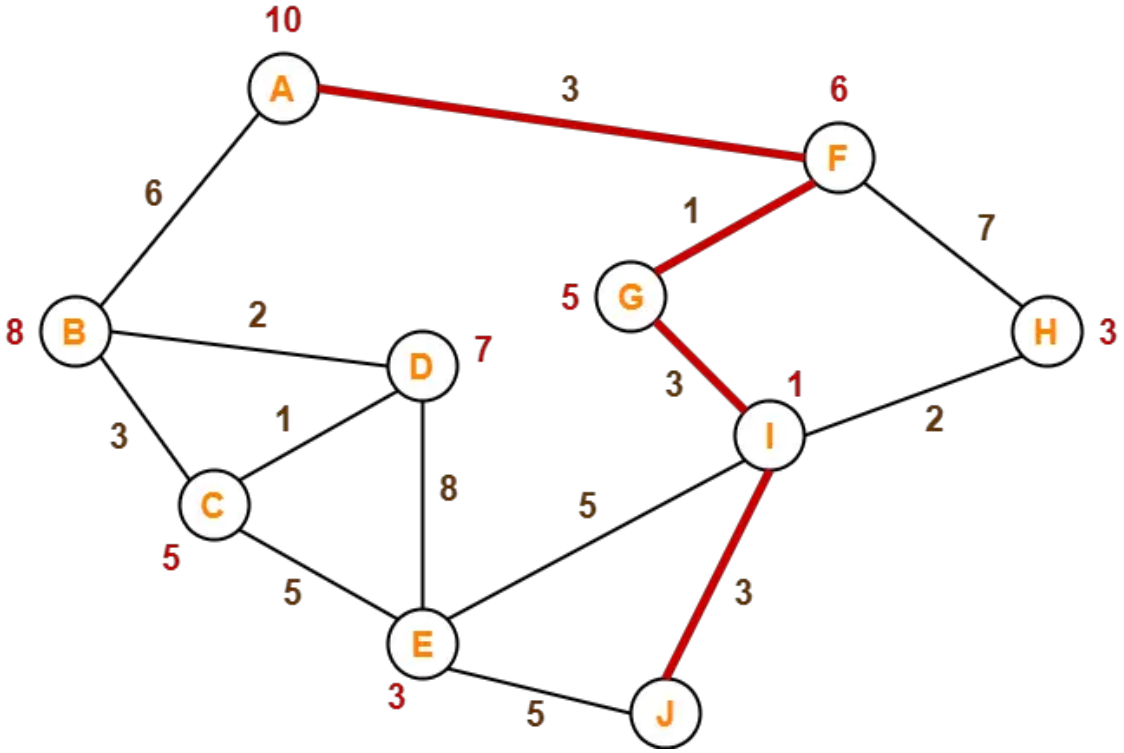
custo que considera o custo acumulado para alcançar um determinado ponto e uma estimativa do custo restante para alcançar o destino (HART; NILSSON; RAPHAEL, 1968).

Esse algoritmo é amplamente aplicado em sistemas de navegação autônoma, onde é usado para planejar trajetórias que evitem obstáculos e levem o veículo autônomo de sua posição atual ao destino desejado. A representação do ambiente, muitas vezes na forma de um mapa aéreo, é fundamental para a aplicação do A*. Os obstáculos identificados na representação são tratados como áreas intransitáveis, enquanto as áreas livres são acessíveis.

Uma característica importante do algoritmo A* é o uso de uma heurística que fornece uma estimativa do custo restante para alcançar o destino a partir de um determinado ponto. A escolha de uma boa heurística pode afetar significativamente o desempenho do algoritmo. Heurísticas comuns incluem a distância euclidiana até o destino (heurística Euclidiana) e a distância de Manhattan. A heurística é usada para priorizar a expansão dos nós que parecem promissores, acelerando a busca. Um exemplo pode ser observado na figura 12.

O algoritmo A* é conhecido por sua eficiência e capacidade de encontrar rotas ótimas. No entanto, sua eficiência depende da escolha adequada da heurística e de estruturas de dados eficientes, como filas de prioridade, para armazenar e gerenciar os nós a serem explorados. Em aplicações de navegação autônoma, a otimização é fundamental para garantir que o planejamento de trajetórias seja realizado em tempo real.

**Figura 12 – Exemplo do funcionamento do algoritmo A Estrela**




**Fonte: (SINGHAL, 2018) .**

Ademais, ele é aplicado em conjunto com a representação de obstáculos criada na seção anterior. Os obstáculos identificados no ambiente são tratados como nós intransitáveis no grafo de busca, enquanto os espaços livres são considerados nós transitáveis. O A* então encontra um caminho que leva em consideração a topografia do ambiente, evitando obstáculos e buscando a rota mais curta.

O algoritmo A* desempenha um papel crucial na navegação autônoma, permitindo que veículos autônomos planejem trajetórias seguras e eficientes em ambientes complexos e dinâmicos. Sua capacidade de considerar informações de custo e heurística o torna uma ferramenta poderosa para tomar decisões de navegação em tempo real.

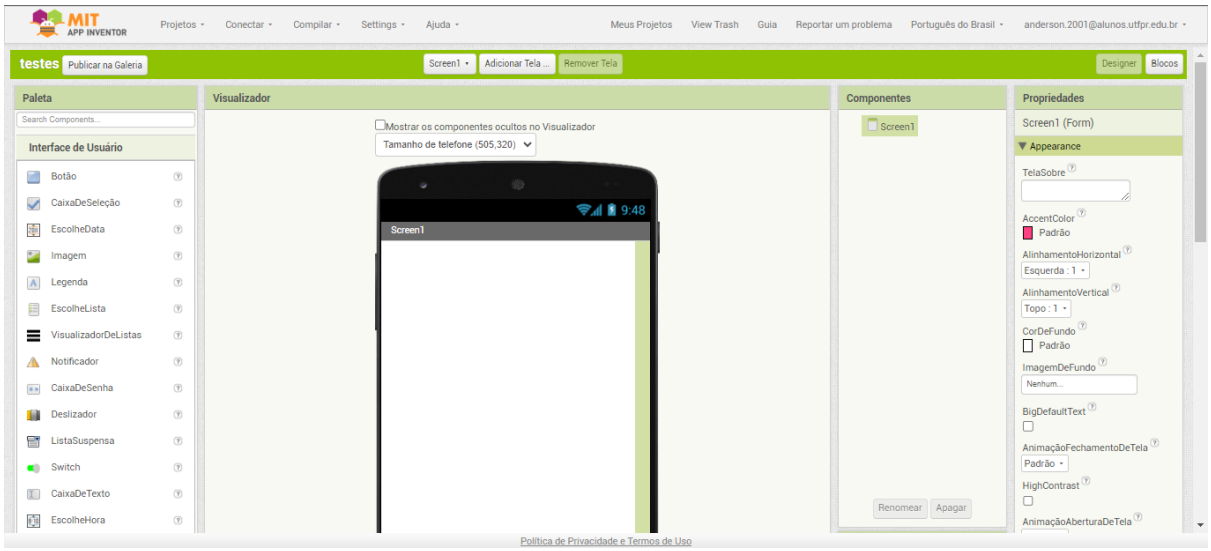
## 2.6 MIT App Inventor

O MIT App Inventor é uma plataforma de desenvolvimento de aplicativos móveis visual e intuitiva que permite que indivíduos, mesmo sem experiência em programação, criem aplicativos personalizados para dispositivos Android. A plataforma foi criada pelo MIT (Massachusetts Institute of Technology) em colaboração com o Google e fornece uma abordagem baseada em blocos para a criação de aplicativos.

### 2.6.1 Visão Geral do MIT App Inventor

O MIT App Inventor foi projetado para democratizar o desenvolvimento de aplicativos, tornando-o acessível a uma ampla gama de usuários, incluindo educadores, estudantes, empreendedores e entusiastas de tecnologia. A plataforma oferece uma abordagem de desenvolvimento de aplicativos semelhante a um quebra-cabeça, onde os desenvolvedores criam lógica por meio da combinação de blocos de código predefinidos em uma interface gráfica como é mostrado na figura 13.

**Figura 13 – Interface do MIT App Inventor**




**Fonte: Autoria Própria .**

A abordagem baseada em blocos do MIT App Inventor simplifica a criação de aplicativos, pois os desenvolvedores não precisam escrever código tradicionalmente. Em vez disso, eles montam blocos lógicos que representam ações, eventos e comportamentos do aplicativo.

### 2.6.2 Criação de Aplicativos no MIT App Inventor

Para criar um aplicativo no MIT App Inventor, os desenvolvedores seguem as seguintes etapas:

1. **Prototipação Visual:** Os desenvolvedores projetam a interface do aplicativo arrastando e soltando componentes visuais, como botões, caixas de texto e imagens, na tela de design. Eles podem personalizar a aparência e o comportamento desses componentes usando propriedades e eventos.

2. **Programação de Blocos:** Para adicionar funcionalidade ao aplicativo, os desenvolvedores programam blocos em uma tela de programação. Os blocos representam ações, como capturar imagens da câmera ou enviar dados para um servidor.

3. **Teste e Depuração:** Os desenvolvedores podem testar o aplicativo em um emulador Android ou em um dispositivo físico conectado. O MIT App Inventor fornece ferramentas de depuração para identificar e corrigir erros.

4. **Publicação:** Após a conclusão do aplicativo, os desenvolvedores podem empacotá-lo e publicá-lo na Google Play Store ou disponibilizá-lo para outros usuários por meio de distribuição de arquivos APK.

### 2.6.3 Uso do MIT App Inventor em Aplicações de Navegação Autônoma

O MIT App Inventor pode ser uma ferramenta valiosa para desenvolver aplicativos personalizados que suportam sistemas de navegação autônoma, como o sistema de planejamento de rotas mencionado. Aqui estão alguns cenários em que o MIT App Inventor pode ser útil:

1. **Interface de Controle de Navegação:** Os desenvolvedores podem criar uma interface de controle de navegação personalizada que permite aos usuários interagir com o sistema de planejamento de rotas, iniciar e interromper a navegação e visualizar informações sobre o trajeto.

2. **Integração de Funcionalidades Específicas:** O MIT App Inventor pode ser usado para integrar funcionalidades específicas, como a captura de imagens da câmera do dispositivo móvel e o envio dessas imagens para o servidor do sistema de planejamento de rotas. Para isso, a extensão KIO4_Base641 pode ser empregada, permitindo a transformação da imagem em base64 antes de enviar via método POST.

3. **Feedback em Tempo Real:** Os aplicativos criados com o MIT App Inventor podem fornecer feedback em tempo real aos usuários, como informações sobre a posição atual do veículo autônomo, atualizações de tráfego e alertas de obstáculos.

4. **Personalização do Sistema de Navegação:** Os desenvolvedores podem personalizar a interface e o comportamento do sistema de navegação de acordo com as necessidades específicas do projeto, adaptando-o para diferentes tipos de veículos ou cenários de navegação.

### 2.6.4 Comunidade e Recursos do MIT App Inventor

O MIT App Inventor possui uma comunidade ativa de desenvolvedores e educadores que compartilham conhecimentos e recursos. Além disso, a plataforma oferece tutoriais, documentação e exemplos de aplicativos para ajudar os usuários a aprender a criar aplicativos de forma eficaz.

## 2.7 HTTP Request e Comunicação com Servidores

A comunicação com servidores é essencial em muitos aplicativos móveis, especialmente em sistemas de navegação autônoma. Para interagir com um servidor, o MIT App Inventor oferece a funcionalidade de HTTP Request.

O HTTP Request permite que os aplicativos enviem solicitações HTTP, como GET, POST e PUT, para servidores web. Essas solicitações podem ser usadas para obter dados do servidor, enviar informações, fazer o upload de imagens ou arquivos e muito mais (FONSECA, 2018).

Os principais componentes envolvidos na comunicação com servidores via HTTP Request no MIT App Inventor incluem:

- **HTTP Request Component:** Este componente é usado para configurar as propriedades da solicitação, como o método HTTP (GET, POST, PUT, etc.), a URL do servidor e os parâmetros da solicitação.
- **Web Component:** O Web Component é usado para lidar com as respostas do servidor. Ele pode processar os dados recebidos, como texto, JSON ou XML, e acionar eventos com base nas respostas do servidor.
- **Tratamento de Erros:** É importante incluir tratamento de erros ao lidar com solicitações HTTP, pois a comunicação com o servidor pode falhar devido a problemas de rede ou outras razões. Os aplicativos podem incluir lógica para lidar com erros e fornecer feedback adequado aos usuários.
- **Segurança:** A segurança é uma consideração fundamental ao lidar com comunicação com servidores. Os aplicativos devem garantir que os dados sejam transmitidos de forma segura e que medidas adequadas sejam tomadas para proteger a privacidade e a integridade dos dados.

Em um contexto de sistemas de navegação autônoma, o HTTP Request pode ser usado para enviar informações de localização, receber atualizações de tráfego em tempo real e comunicar-se com servidores de planejamento de rotas.

# 3 MATERIAIS E MÉTODOS

Neste capítulo, é descrito detalhadamente as etapas e técnicas empregadas na implementação do sistema. Isso abrange o entendimento das técnicas de visão computacional e o desenvolvimento do sistema de localização probabilística, bem como a implementação dos algoritmos de planejamento de trajetória e os testes realizados em ambientes diversos. Além disso, abordaremos as etapas específicas da metodologia, como o desenvolvimento do aplicativo para aquisição e transmissão de imagens ao sistema, detecção de profundidade em uma cena, análise do espectro de intensidade do pixel, mapeamento aéreo do ambiente, predição de rotas, identificação de um objeto vermelho na cena como o alvo a ser alcançado e transmissão dos dados via bluetooth. O objetivo é fornecer uma descrição completa e detalhada de todas as etapas e técnicas utilizadas no projeto, garantindo assim a replicabilidade e validade dos resultados obtidos.

## 3.1 Materiais

Para a realização deste projeto, a utilização de um dispositivo de captura de imagens foi imperativa, desempenhando a função de registrar as informações visuais do ambiente. Esse registro é fundamental para discernir obstáculos e outros elementos de interesse que influenciam diretamente a estratégia de navegação adotada pelo veículo terrestre autônomo. Adicionalmente, nas fases preliminares do projeto, realizou-se a simulação da aquisição de imagens, emulando a perspectiva que um veículo terrestre comum experimentaria.

No que diz respeito à implementação dos algoritmos voltados à visão computacional e prognóstico de rotas, optou-se pela linguagem de programação Python. A escolha se justifica pela notoriedade dessa linguagem como uma plataforma abrangente e condescendente, provida de bibliotecas especializadas voltadas ao processamento de imagens e visão computacional. Destacam-se as bibliotecas pertinentes ao escopo de visão computacional, como o OpenCV, Scikit-Image e MiDaS, as quais foram convocadas para conduzir a análise das imagens obtidas pela câmera, identificando com meticulosidade obstáculos e demais componentes do cenário circundante.

Além disso, recorremos ao MIT App Inventor, uma plataforma de desenvolvimento de aplicativos móveis visual e intuitiva, para criar um aplicativo capaz de capturar imagens e transmiti-las ao sistema de detecção de obstáculos e planejamento de rotas. Esse aplicativo desempenhou um papel crucial na integração do sistema, permitindo a aquisição contínua de imagens em tempo real durante o funcionamento do veículo terrestre autônomo.

Com o objetivo de agilizar e facilitar as iterações inerentes ao ciclo de desenvolvimento e testes dos algoritmos, adotou-se a plataforma Jupyter Notebook. Essa plataforma oferece um ambiente interativo singular que possibilita a execução fluida e dinâmica de códigos Python.

## 3.2 Métodos

As fases subsequente, as quais serão detalhadamente abordadas adiante, englobam os passos iniciais preconizados para a execução do projeto.

### 3.2.1 Diagrama de blocos da arquitetura do sistema proposto

Aqui, é apresentado um diagrama de blocos que ilustra a arquitetura do sistema proposto. O diagrama de blocos é uma representação visual que ajuda a compreender a interconexão e as relações entre os componentes-chave do sistema de navegação autônoma para veículos terrestres. O diagrama apresenta uma visão geral das etapas e dos componentes envolvidos na implementação bem-sucedida da navegação autônoma.

O sistema é composto por vários módulos essenciais, incluindo técnicas de visão computacional, um sistema de localização probabilística, algoritmos de planejamento de trajetória e integração em um veículo terrestre real. Cada um desses componentes desempenha um papel fundamental na capacidade do veículo de perceber seu ambiente, tomar decisões autônomas e seguir trajetórias seguras e eficientes.

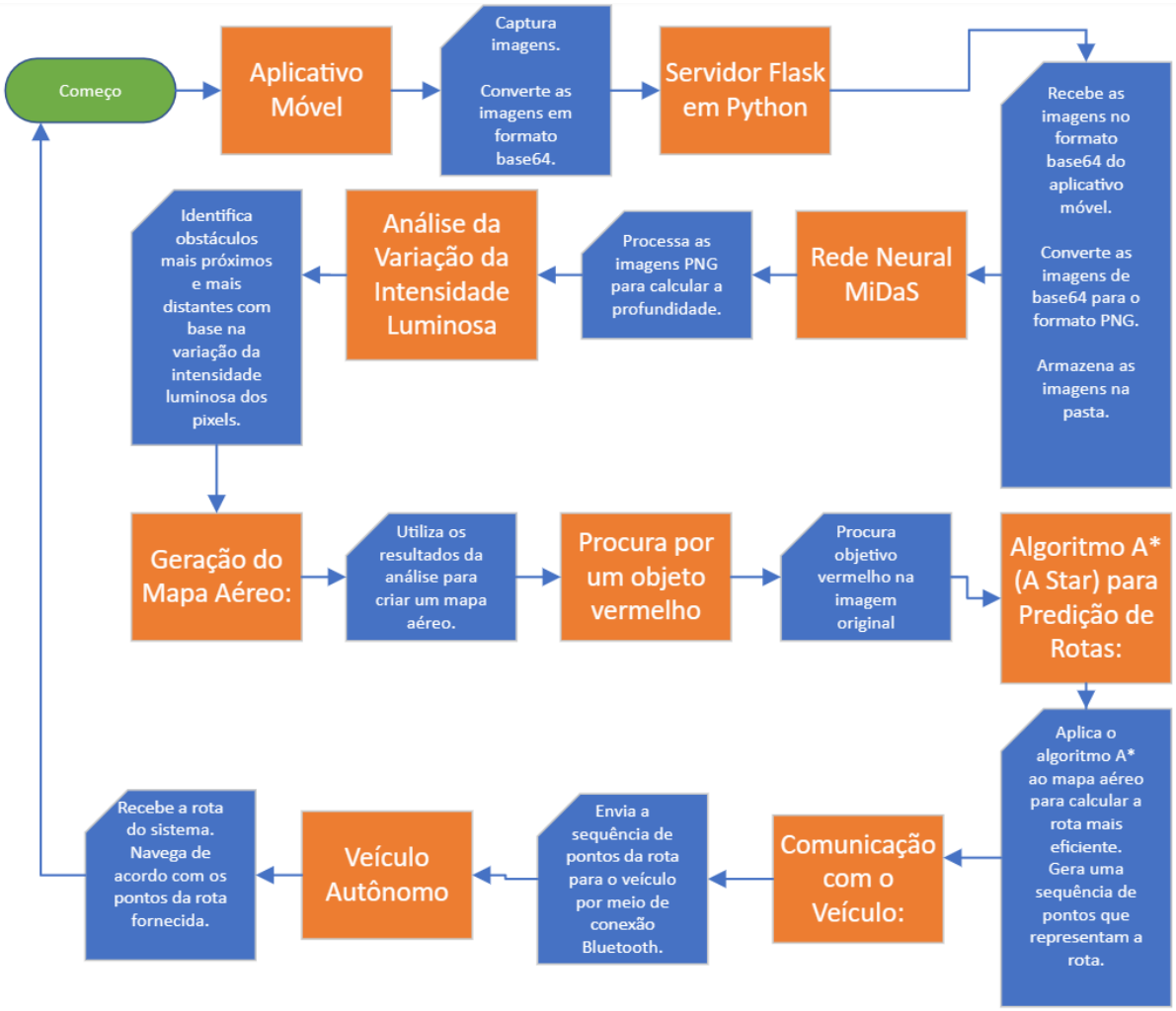
No diagrama da figura 14, e apresentada setas que representam o fluxo de informações entre os diferentes componentes. Essas setas indicam como os dados são transmitidos e processados ao longo do sistema, garantindo uma operação fluida e coordenada. Este diagrama é uma ferramenta importante para compreender o contexto do sistema e será referenciado à medida que exploramos os detalhes e resultados da implementação do sistema de navegação autônoma para veículos terrestres.

### 3.2.2 Aplicativo de Captura e Envio de Imagens e Servidor

Como parte integrante das capacidades de aquisição de dados visuais necessárias para este projeto de navegação autônoma, a aplicação móvel desempenhou um papel crucial. Esta aplicação foi desenvolvida utilizando a plataforma MIT App Inventor, uma ferramenta visual e intuitiva para a criação de aplicativos móveis.

O aplicativo foi projetado para operar em um dispositivo móvel, permitindo que o usuário capture imagens do ambiente circundante em tempo real. Para isso, o veículo estará equipado com um celular, que funcionará como a câmera de captura de imagens. As imagens capturadas pelo celular são posteriormente convertidas em strings utilizando a extensão KIO4 Base64 e enviadas ao sistema de detecção de obstáculos e planejamento de rotas por meio de solicitações HTTP (como será explicado em subseções posteriores). Isso fornecerá uma fonte contínua de informações visuais para a tomada de decisões do veículo terrestre autônomo.

**Figura 14 – Diagrama de blocos da representação do sistema**



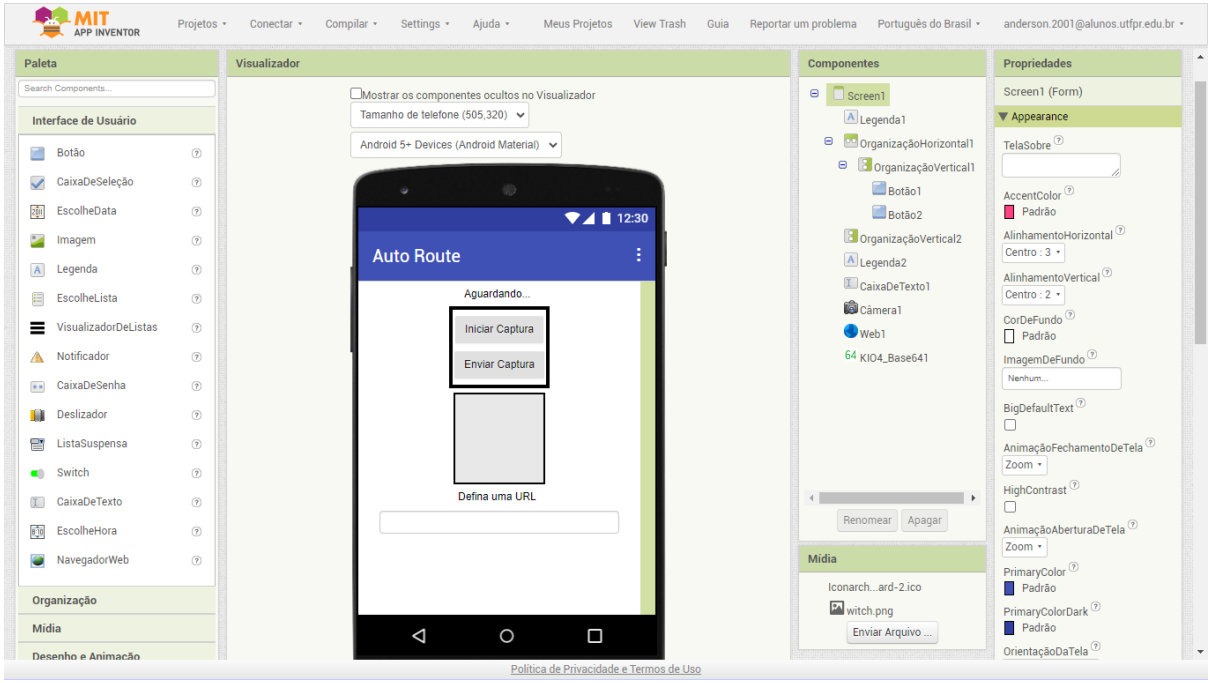

**Fonte: Autoria Própria .**

A aplicação móvel foi desenvolvida com base em uma linguagem de blocos no MIT App Inventor, como exemplificado na figura 16. Essa abordagem permitiu estruturar a lógica do programa de forma eficiente.

Essa integração entre o aplicativo e o sistema principal permitiu que o sistema reagisse dinamicamente às mudanças no ambiente, tornando-o mais adaptável e capaz de lidar com cenários em constante evolução. A aplicação móvel funcionou como uma interface eficaz para a coleta de dados visuais, contribuindo para a eficácia global do projeto.

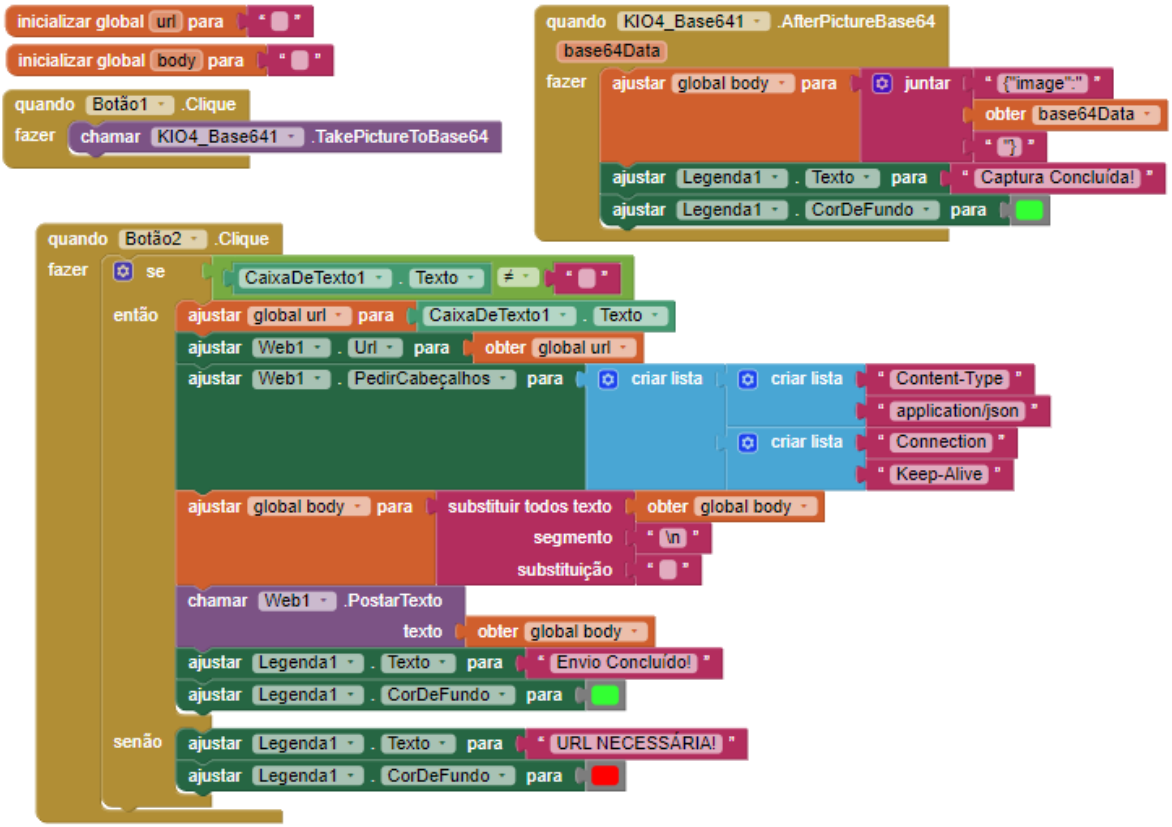
A escolha de utilizar o MIT App Inventor demonstrou ser vantajosa para simplificar o processo de captura e envio de imagens, garantindo uma integração tranquila entre o dispositivo móvel (celular) e o sistema de navegação autônoma. Essa abordagem, combinada com as outras técnicas e tecnologias mencionadas, fortaleceu a base do projeto e o tornou capaz de enfrentar os desafios da navegação autônoma de forma mais abrangente e eficiente.

**Figura 15 – Interface do aplicativo desenvolvido no MIT App Inventor**




**Fonte: Autoria Própria .**

**Figura 16 – Linguagem de blocos no MIT App Inventor**




**Fonte: Autoria Própria .**

### 3.2.3 Servidor de Recebimento de Imagens

Nesta seção, detalharemos o servidor responsável por receber as imagens do aplicativo móvel. O servidor foi implementado em Python utilizando a biblioteca Flask e tem a função de receber as imagens em formato base64 e salvá-las em uma pasta local. Essas imagens serão posteriormente utilizadas para a análise de obstáculos e o planejamento de rotas no contexto da navegação autônoma.

#### 3.2.3.1 Implementação em Python

O servidor foi desenvolvido utilizando a biblioteca Flask, um framework leve para a criação de aplicativos web em Python, ela está exemplificada conforme segue o apêndice A. O Flask funciona com base em rotas (URLs) e pode processar requisições HTTP, o que é essencial para a comunicação entre o aplicativo móvel e o servidor.

A rota `/upload` é um ponto de entrada crucial no servidor. Ela é configurada para aceitar requisições HTTP do tipo POST, onde os dados são esperados em formato JSON. A função `upload_image` é a manipuladora dessa rota. Quando recebe uma requisição POST, verifica se contém o campo `image` no formato base64, que representa a imagem a ser transmitida.

A função `save_base64_image` é responsável por processar a imagem em base64 recebida e armazená-la em um diretório local (`uploads`). Primeiro, a imagem é decodificada a partir do formato base64 para sua representação binária. Em seguida, é salva no formato PNG, o que é útil para manter a qualidade da imagem e garantir que ela seja facilmente manipulável para futuras análises.

Este servidor atua como uma interface entre o aplicativo móvel e o sistema de processamento de imagens e navegação autônoma. Ele permite o envio contínuo de imagens capturadas pelo aplicativo para análise e tomada de decisões em tempo real. Essa funcionalidade é crucial para a integração bem-sucedida do sistema, pois facilita a troca de informações entre os componentes do sistema, contribuindo para a eficiência da navegação autônoma.

### 3.2.4 Estimação de Profundidade

A fim de efetuar a estimativa de profundidade neste projeto voltado à navegação autônoma, adotou-se o modelo MiDaS (Modelo de Aprendizado de Máquina da Intel Labs para Estimação de Profundidade Monocular). O MiDaS configura-se como um modelo de aprendizado profundo desenvolvido com foco na inferência de profundidade a partir de imagens monocularmente adquiridas, conferindo-lhe um status de instrumento altamente eficaz para a percepção do ambiente. Este modelo encontra-se disponível através da biblioteca de código aberto dedicada ao aprendizado de máquina e aprendizado profundo, PyTorch. É amplamente reconhecido e adotado na pesquisa e desenvolvimento de modelos de aprendizado profundo, abar-

cando uma variedade de arquiteturas de redes neurais, tais como redes neurais convolucionais (CNNs), redes neurais recorrentes (RNNs) e várias outras estruturas de redes neurais, conforme documentado em (RANFTL *et al.*, 2020).

Inicialmente, implementou-se um arranjo estratégico de câmeras em um ambiente simulado, as quais operaram com o propósito de registrar imagens emulando a perspectiva de um veículo terrestre conforme é demonstrado na figura 17. Estas câmeras meticulosamente posicionadas capturaram dados visuais que se mostram imprescindíveis para a tarefa de estimação de profundidade.

**Figura 17 – Emulação da perspectiva do veículo terrestre**


**Fonte: Autoria Própria .**

O modelo MiDaS emprega estas imagens como entrada e, mediante o emprego de redes neurais convolucionais profundas, realiza inferências a respeito da profundidade de cada pixel presente na cena. Importa destacar que o MiDaS desempenha tal função com maestria em uma diversificada gama de cenários, contemplando ambientes tanto externos quanto internos, prescindindo de sensores LiDAR ou câmeras estereoscópicas. O êxito ao cumprir sua função é resultado da minuciosa capacitação da arquitetura da rede neural, que foi treinada em um conjunto diversificado de dados, o que possibilita a obtenção de estimativas de profundidade precisas mesmo em cenários de elevada variabilidade.

Os dados relativos à profundidade adquiridos desempenham um papel multifacetado em diversas etapas do projeto, compreendendo desde o planejamento de trajetória até a detecção de obstáculos. Eles emergem como elemento central nas tomadas de decisão do veículo autônomo, conferindo-lhe a habilidade de percorrer seu itinerário de forma segura e evitar potenciais colisões. A execução dessa poderosa ferramenta consta no apêndice B.

**Figura 18 – Estimativa de profundidade**

**Fonte: Autoria Própria.**

### 3.2.5 Análise da variação da intensidade luminosa dos pixels

A análise da variação da intensidade luminosa dos pixels desempenha um papel crucial na percepção visual, particularmente em sistemas de visão computacional destinados à detecção e esquiva de obstáculos. Esta técnica é instrumental na extração de informações críticas a partir das imagens capturadas, permitindo a identificação de objetos e obstáculos com base nas variações na intensidade luminosa.

Essa abordagem pressupõe que objetos e obstáculos em uma cena frequentemente se traduzem em flutuações na intensidade dos pixels na imagem. Essas flutuações são resultado das interações de luz com os objetos, gerando regiões de alta e baixa intensidade luminosa na imagem, conforme previamente discutido na revisão bibliográfica.

Após a etapa de estimação de profundidade, é gerada uma imagem que destaca os objetos mais próximos da câmera com pixeis mais luminosos, enquanto os objetos mais distantes ficam com pixeis menos luminosos conforme a Figura 18. O processo inicia com a seleção de uma linha ou região de interesse na imagem, muitas vezes denominada de "linha central", onde a análise será conduzida. A escolha dessa linha é pautada por considerações práticas e pela geometria da câmera utilizada no sistema, e então é possível observar um gráfico como o da Figura 28.

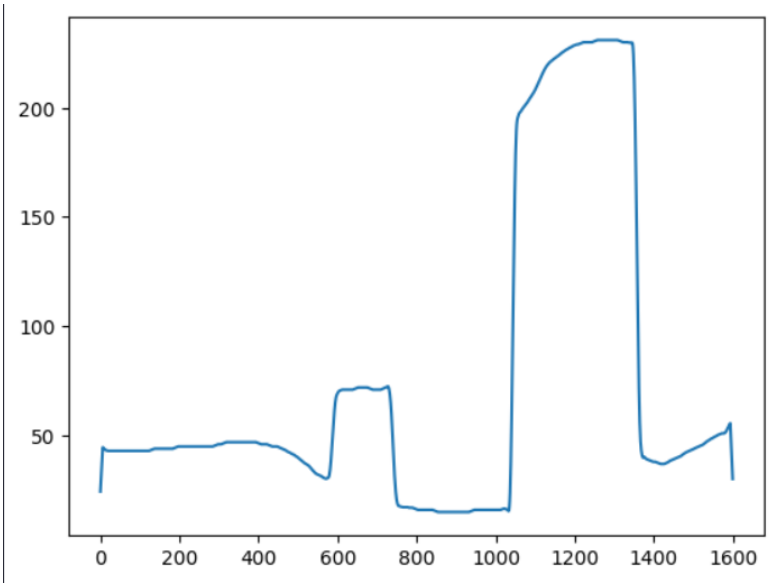
Na análise da variação da intensidade luminosa dos pixels, uma das tarefas fundamentais é a detecção de "picos de intensidade". Esses picos são pontos notáveis no perfil de intensidade de uma linha da imagem que indicam variações significativas na intensidade luminosa. Esses picos são identificados através do cálculo da derivada da linha central e podem ser de dois tipos:

1. "Picos positivos"são identificados quando há um aumento acentuado na intensidade luminosa em relação aos pixels vizinhos ao longo da linha. Geralmente, esses picos correspondem a áreas de alta intensidade na imagem. Em muitos casos, picos positivos estão associados a bordas ou contornos de objetos na cena. Por exemplo, quando uma linha brilhante de um objeto contrasta com seu fundo mais escuro, isso pode resultar em um pico positivo na derivada da intensidade.

2. "Picos negativos"são detectados quando há uma diminuição acentuada na intensidade luminosa em relação aos pixels adjacentes. Esses picos indicam regiões de baixa intensidade na imagem. Em algumas situações, os picos negativos podem representar depressões ou espaços vazios na cena. Por exemplo, se houver uma sombra projetada sobre uma superfície, isso pode resultar em um pico negativo na derivada da intensidade.

Portanto, ao analisar o variação da intensidade luminosa dos pixels, a identificação de picos positivos e negativos desempenha um papel crítico na detecção de características visuais importantes na imagem. Os picos positivos muitas vezes destacam o começo dos objetos na horizontal, enquanto os picos negativos podem indicar o término desses objetos, e isso pode ser observado na Figura 21.

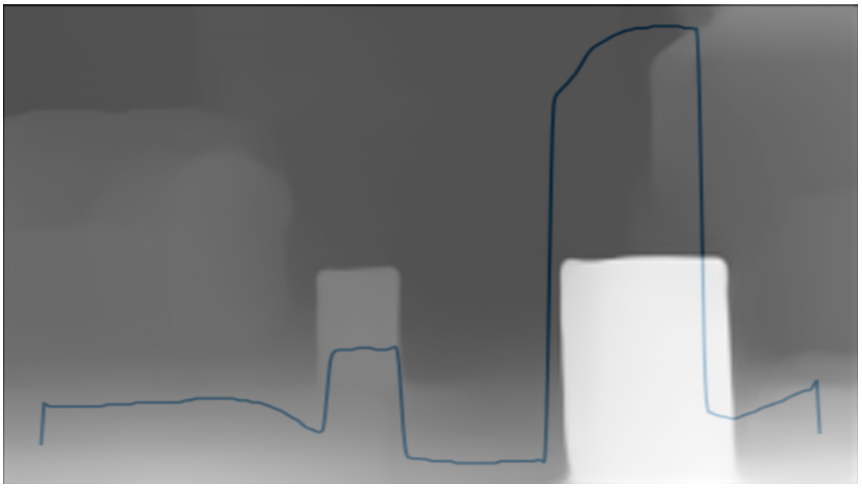
**Figura 19 – Linha Central analisada**




**Fonte: Autoria Própria.**

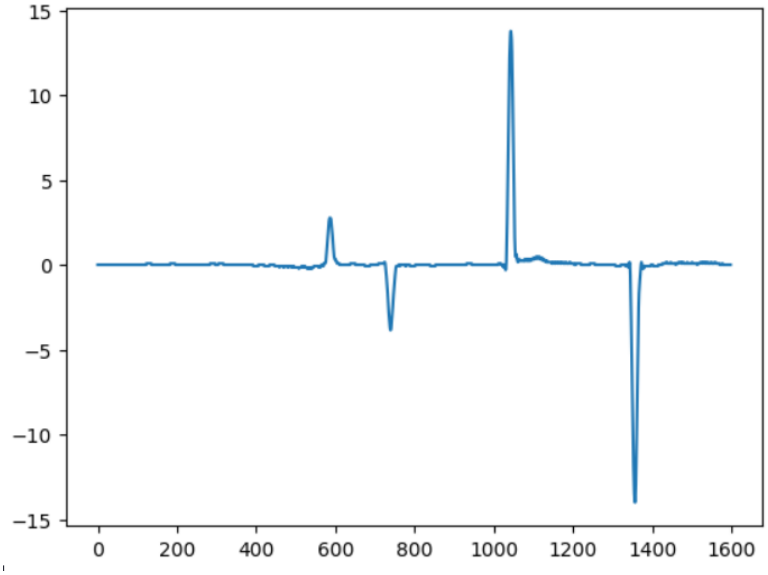
Conforme ilustrado na figura 20, a capacidade do MiDaS de representar objetos mais próximos à câmera com uma maior intensidade de pixels na imagem permite uma análise detalhada da linha central do mapa de profundidade, possibilitando uma extração rica de informações sobre as distâncias entre os objetos. Esse fenômeno de representação luminosa dos objetos próximos na imagem oferece uma oportunidade valiosa para a avaliação das relações de distância entre os elementos do cenário.

**Figura 20 – Linha Central sobreposta no mapa de profundidade**



**Fonte: Autoria Própria.**

**Figura 21 – Picos positivos e negativos**




**Fonte: Autoria Própria.**

Após a detecção dos picos de intensidade, procede-se ao emparelhamento destes para construir uma representação dos obstáculos presentes na cena. Cada par de picos sugere a largura do obstáculo, e a intensidade máxima entre eles fornece uma estimativa da altura do obstáculo. Essa representação é de importância primordial para a identificação e localização de obstáculos no ambiente. O código que realiza essa analise consta nos apêndices C e D.

### 3.2.6 Mapeamento Aéreo do Ambiente

Essa é uma técnica crucial que visa criar uma representação abstrata do ambiente em que um veículo autônomo opera. Essa representação é fundamental para que o veículo possa

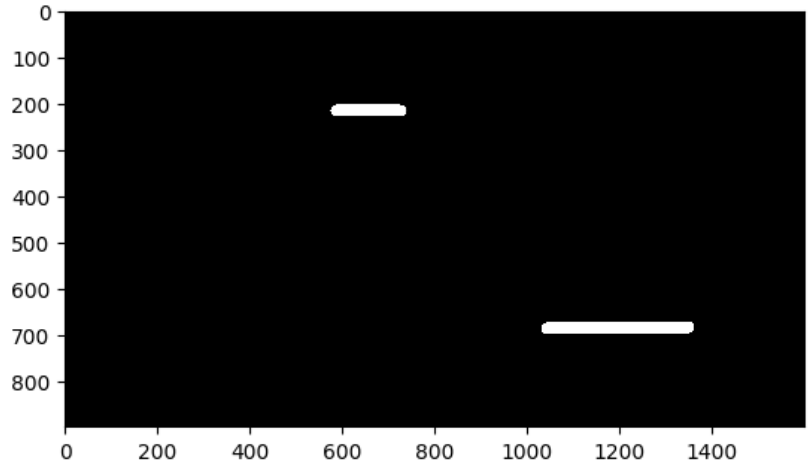
planejar trajetórias seguras e evitar colisões com obstáculos durante sua navegação. No contexto deste projeto, concentramos nossos esforços na criação de um mapa aéreo bidimensional.

A criação do mapa aéreo implica na identificação e representação de obstáculos significativos detectados na cena. Os obstáculos filtrados, que atendem a critérios específicos, são considerados na construção do mapa. Cada obstáculo é representado por um conjunto de coordenadas no mapa aéreo, onde cada ponto é marcado com o valor "1,"indicando a presença de um obstáculo naquela posição. Esse processo é demonstrado no apêndice E.

A representação de obstáculos no mapa aéreo desempenha um papel central na navegação autônoma, fornecendo informações importantes sobre a distribuição espacial dos obstáculos no ambiente. Essa representação é usada para evitar colisões e planejar trajetórias que estejam livres de obstáculos. Quanto mais precisa e atualizada for a representação, maior será a segurança na navegação do veículo autônomo.

Para criar o mapa aéreo, utilizamos uma matriz que representa as dimensões do mapa, conforme definido nos parâmetros do projeto. A partir dos dados de obstáculos filtrados, preenchemos o mapa aéreo, marcando as posições relevantes com "1". O ponto de visão do veículo no mapa aéreo está representado em x = 800 e y = 900 na figura 22. Conforme os obstáculos possuíssem uma intensidade de pixel maior, estes foram representados no mapa aéreo como obstáculos mais próximos do ponto de origem da captura da imagem original. A visualização do mapa aéreo resultante é útil para a análise e validação das informações de mapeamento.

**Figura 22 – Exemplo de Mapa Aéreo do Ambiente**




**Fonte: Autoria Própria.**

A Figura 22 apresenta um exemplo de mapa aéreo do ambiente gerado pelo sistema. Esse mapa é uma representação visual das informações de mapeamento e é essencial para o funcionamento seguro e eficaz do veículo autônomo.

### 3.2.7 Predição de Rotas

A predição de rotas é uma etapa crítica em sistemas de navegação autônoma, onde o objetivo é planejar trajetórias seguras e eficientes para um veículo autônomo, evitando obstáculos e levando-o de sua posição atual ao destino desejado. Nesta seção, exploraremos o uso do algoritmo A* (A-Estrela) como uma ferramenta poderosa para a predição de rotas.

#### 3.2.7.1 Algoritmo A* (A-Estrela)

O algoritmo A* é uma técnica de busca de caminho amplamente empregada que visa encontrar a rota mais curta entre dois pontos em um grafo ponderado. Sua eficácia é resultado de uma abordagem inteligente que combina informações de custo e heurística. O A* é particularmente útil em aplicações de navegação autônoma, onde é utilizado para planejar trajetórias que contornam obstáculos e alcançam o destino desejado.

O funcionamento do algoritmo A* baseia-se na busca orientada por uma função de custo, que leva em consideração o custo acumulado para alcançar um ponto específico e uma estimativa do custo restante para atingir o destino. Essa função de custo é essencial para priorizar a expansão dos nós mais promissores durante a busca. O A* é conhecido por sua capacidade de encontrar rotas ótimas, desde que uma boa heurística seja escolhida.

Uma característica fundamental do A* é a utilização de uma heurística que fornece uma estimativa do custo restante para atingir o destino a partir de um ponto dado. Heurísticas comuns incluem a distância euclidiana até o destino (heurística Euclidiana) e a distância de Manhattan. A escolha de uma heurística adequada pode impactar significativamente o desempenho do algoritmo.

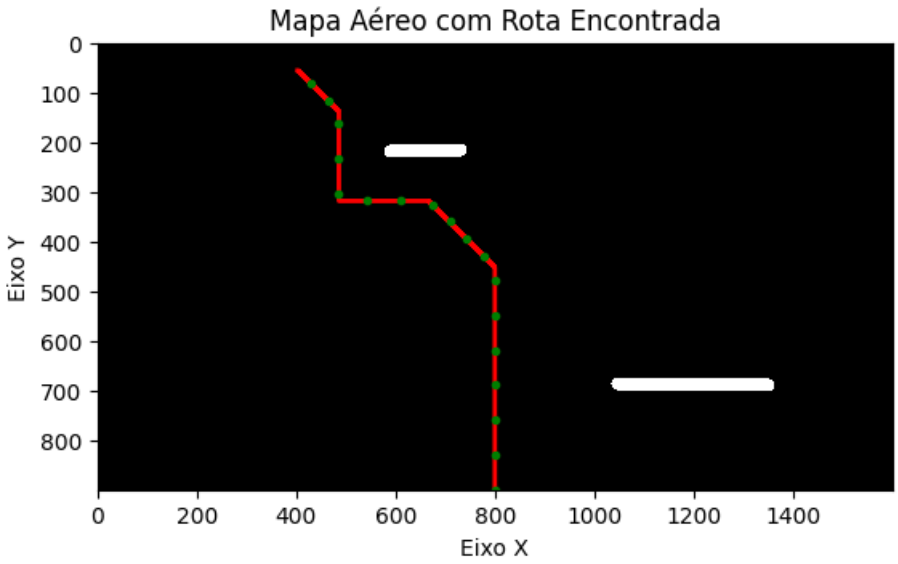
Para aplicar o A* na navegação autônoma, é essencial uma representação precisa do ambiente, muitas vezes na forma de um mapa aéreo. Os obstáculos detectados no ambiente são tratados como áreas intransitáveis, enquanto os espaços livres são considerados acessíveis. O A* utiliza essa representação para encontrar um caminho que leva em consideração a topografia do ambiente e evita colisões. O A* foi aplicado no apêndice F.

A Figura 23 apresenta um exemplo de mapa aéreo com uma rota encontrada pelo algoritmo A*. A rota é representada em vermelho, destacando o caminho planejado pelo sistema autônomo. O A* desempenha um papel crucial na navegação autônoma, permitindo que veículos autônomos tomem decisões em tempo real para alcançar seus destinos com eficácia e segurança.

### 3.2.8 Identificação do objeto vermelho na cena

A identificação do objeto vermelho na cena é uma etapa importante para sistemas de processamento de imagem e visão computacional. Nesta seção, apresentaremos um algoritmo

**Figura 23 – Exemplo de Mapa Aéreo com Rota Encontrada pelo A***




**Fonte: Autoria Própria.**

que utiliza técnicas de processamento de imagem para detectar e identificar objetos de cor vermelha em uma imagem.

No apêndice G, uma imagem é carregada e convertida para o espaço de cores HSV para facilitar a detecção de pixels vermelhos. Em seguida, é aplicada uma máscara para identificar os pixels na faixa de cor vermelha. Os contornos são encontrados na máscara para determinar a posição do objeto vermelho na imagem. As coordenadas do centro do objeto vermelho são então utilizadas como informações para a predição de rotas.

A identificação do objeto vermelho é crucial para muitas aplicações, incluindo robótica e automação, onde a localização de objetos específicos é essencial para o planejamento de trajetórias e tomada de decisões autônomas.

### 3.2.9 Envio de dados via Bluetooth

O envio de dados via Bluetooth é uma operação que permite a transmissão de informações entre dispositivos utilizando a tecnologia Bluetooth, que é uma forma de comunicação sem fio de curto alcance. O código a seguir exemplifica como implementar essa funcionalidade em Python.

A funcionalidade de envio de dados via Bluetooth é implementada através da função `enviar_dados_via_bluetooth(destino, dados)`. Vamos analisar o código para entender a teoria por trás dessa implementação:

1. **Criação do Socket Bluetooth**: O código começa criando um socket Bluetooth utilizando a biblioteca `socket` do Python. A função `socket.socket()` é usada com

os parâmetros `AF_BLUETOOTH`, `SOCK_STREAM` e `BTPROTO_RFCOMM`. Isso configura um socket Bluetooth para comunicação.

2. **Conexão ao Dispositivo Bluetooth de Destino**: Após a criação do socket, o código tenta se conectar ao dispositivo de destino usando o endereço MAC do dispositivo Bluetooth e a porta 1 (um canal padrão para comunicação Bluetooth).

3. **Codificação e Envio dos Dados**: Os dados a serem enviados são codificados para uma sequência de bytes usando `dados.encode('utf-8')`. Isso é necessário para garantir que os dados possam ser transmitidos pelo socket.

4. **Envio dos Dados**: Os dados codificados são enviados pelo socket Bluetooth usando o método `send()`. Isso implica a transmissão dos dados para o dispositivo Bluetooth de destino.

5. **Fechamento do Socket**: Após o envio dos dados, o socket é fechado para liberar os recursos.

6. **Tratamento de Exceções**: O código está envolto em um bloco `try-except` para capturar possíveis exceções que podem ocorrer durante a criação do socket, a conexão ou o envio dos dados. Caso ocorra uma exceção, é exibida uma mensagem de erro.

A função `enviar_dados_via_bluetooth` encapsula esse processo de envio de dados via Bluetooth, facilitando a reutilização em outros pontos do código.

Além disso, o código também inclui a preparação dos dados a serem enviados. Os pontos selecionados são convertidos em uma string formatada antes de serem enviados pelo Bluetooth. Isso é feito para garantir que os dados estejam no formato desejado antes da transmissão.

O código apresentado no apêndice G facilita o envio de dados para um dispositivo, como o Arduino, que possua um módulo Bluetooth específico por meio da criação de um socket Bluetooth, conexão com o dispositivo, codificação e envio dos dados, e, por fim, o fechamento do socket.

# 4 RESULTADOS

Neste capítulo, é apresentado os resultados obtidos na implementação e experimentação do sistema de navegação autônoma proposto. Este sistema é baseado em técnicas avançadas de visão computacional e inteligência artificial, destinado a capacitar um veículo terrestre autônomo a perceber o ambiente circundante, planejar trajetórias seguras e tomar decisões autônomas em tempo real. Os resultados apresentados aqui representam a culminação de extensos esforços de pesquisa, implementação e testes, e são fundamentais para a avaliação do desempenho e da viabilidade do sistema.

O capítulo está organizado em seções que abordam os principais componentes do sistema, começando pelo aplicativo e servidor, que enviam e recebem imagens capturadas na perspectiva do veículo, a estimação de profundidade, seguida da detecção de obstáculos, planejamento de trajetória, identificação do objeto vermelho na cena e, por fim, a transmissão de dados via Bluetooth. Cada seção apresenta uma análise detalhada dos resultados obtidos e uma discussão sobre suas implicações em relação aos objetivos estabelecidos no início deste projeto.

É importante ressaltar que este projeto tem como objetivo criar uma plataforma versátil e eficaz para a navegação autônoma de veículos terrestres em ambientes variados. Portanto, os resultados apresentados têm a finalidade de avaliar a capacidade do sistema em lidar com diferentes cenários e desafios, fornecendo uma visão abrangente de seu desempenho.

Neste capítulo, não apenas será destacado as realizações notáveis do sistema, mas também será reconhecida as limitações e os desafios encontrados ao longo do caminho. Essas informações são fundamentais para direcionar futuros desenvolvimentos e melhorias no sistema, além de contribuir para o avanço da pesquisa em navegação autônoma e visão computacional.

## 4.1 Escopo do sistema

O sistema de navegação autônoma tem como objetivo permitir que um veículo autônomo navegue com segurança em um ambiente desconhecido, evitando obstáculos. O sistema se baseia em imagens capturadas a partir de uma perspectiva de veículo, que são processadas e analisadas para a detecção de obstáculos. As funcionalidades e características principais do sistema incluem:

1. **Captura de Imagens**: O sistema é capaz de capturar imagens em tempo real da perspectiva do veículo.

2. **Processamento de Imagens**: As imagens capturadas são processadas para extrair informações relevantes, como a detecção de obstáculos, pontos de interesse e informações de rota.

3. **Detecção de Obstáculos**: O sistema utiliza algoritmos de visão computacional para detectar obstáculos.

4. **Planejamento de Rotas**: Com base nas informações de detecção de obstáculos, o sistema calcula rotas seguras para o veículo, evitando colisões.

5. **Feedback Visual**: O sistema fornece feedback visual em tempo real ao veículo, mostrando a rota planejada e os obstáculos detectados.

6. **Integração de Hardware**: O sistema é projetado para se integrar câmeras de controle do veículo.

7. **Comunicação com o Veículo**: As informações da rota planejada são fornecidas ao veículo autônomo para controle e navegação seguros.

De início, foi planejado executar esse sistema em uma RaspberryPi, contudo, tivemos incompatibilidades de hardware que impossibilitou o uso dessa ferramenta. Portanto, o sistema está sendo executado em um computador com acesso à mesma rede do dispositivo mobile que captura as imagens do veículo e envia os dados de rota via bluetooth para o veículo.

Ademais existem alguns pontos negativos desse sistema, como:

1. O sistema não realiza o controle direto do veículo; ele apenas fornece informações de planejamento de rotas.

2. Não inclui aquisição de imagens fora da perspectiva do veículo, como imagens aéreas.

3. Não aborda questões regulatórias no que diz respeito à comunicação entre o veículo e o sistema.

## 4.2 Implementação do sistema

Nesta seção, abordaremos em detalhes todas as etapas que constituem o funcionamento do sistema de navegação autônoma. Este sistema incorpora diversas etapas críticas que trabalham de maneira coordenada para permitir que um veículo autônomo navegue de forma segura e eficaz em ambientes complexos.

O coração desse sistema reside na eficácia da captura e transmissão de imagens em tempo real, o processamento dessas imagens para identificar obstáculos, a estimação de profundidade para avaliar a distância desses obstáculos e, por fim, a geração de rotas seguras para o veículo seguir. Cada um desses componentes desempenha um papel crucial na realização da navegação autônoma.

Nesta subseção, nos concentraremos na primeira etapa desse processo, que envolve a "Captura e Envio de Imagens pelo Aplicativo."Abordaremos em detalhes como o aplicativo móvel

é projetado para capturar imagens em tempo real, a qualidade dessas imagens, a transmissão bem-sucedida para o sistema de detecção de obstáculos e planejamento de rotas, e a interface do aplicativo que permite aos usuários interagirem com o sistema.

A captura e transmissão de imagens são a base de todo o sistema, uma vez que permitem ao veículo autônomo perceber o ambiente circundante. Portanto, é essencial entender como essa etapa é executada com eficácia para garantir a segurança e o desempenho do veículo.

Na sequência, abordaremos em subseções subsequentes as demais etapas do sistema, incluindo o servidor Flask de recebimento de imagens, a estimação de profundidade, a análise da variação da intensidade luminosa dos pixels, o mapeamento aéreo do ambiente, a predição de rotas e a identificação do objeto vermelho. Cada uma dessas etapas desempenha um papel crítico na navegação autônoma, contribuindo para o sucesso geral do sistema.

### 4.2.1 Captura e Envio de Imagens pelo Aplicativo

Os resultados obtidos por meio do aplicativo desenvolvido para captura e envio de imagens ao sistema de detecção de obstáculos e planejamento de rotas abrangem a eficácia da captura de imagens em tempo real, a qualidade das imagens obtidas e a transmissão bem-sucedida dessas imagens ao sistema.

O aplicativo foi projetado para capturar imagens em tempo real a partir do dispositivo móvel, que atua como a câmera de captura de imagens. Durante os testes, avaliamos a eficiência da captura de imagens em cenários de ambientes internos com variedades de obstáculos.

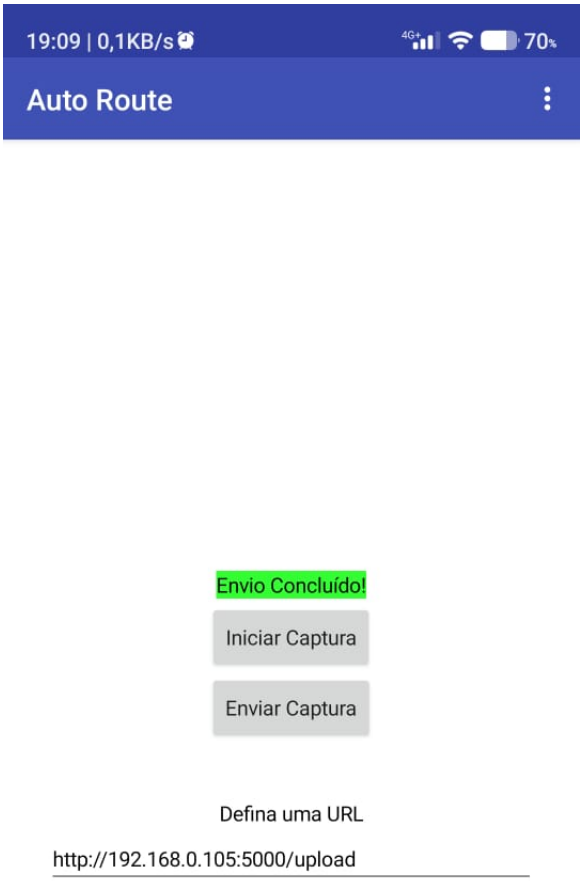
A interface do aplicativo, conforme a figura 24 apresenta dois botões e uma caixa de texto. O primeiro botão é responsável por iniciar o processo de captura de imagem, converter essa imagem em uma representação no formato base64 e armazená-la em uma variável. Enquanto isso, o segundo botão tem a finalidade de enviar a imagem, agora em formato de string, para um servidor cujo endereço é especificado na caixa de texto. Caso a caixa de texto permaneça em branco, a aplicação exibirá uma mensagem de erro na tela inicial.

Acima dos botões, encontra-se uma área informativa que mantém os usuários atualizados sobre o status das tarefas executadas pelo aplicativo. Em caso de falha em qualquer tarefa, um aviso de erro será prontamente exibido nesta área, bem como a confirmação de sucesso quando a tarefa for concluída com êxito.

Os resultados indicam que o aplicativo é capaz de capturar imagens de forma contínua e com baixa latência, permitindo uma aquisição de dados eficaz para a percepção do ambiente circundante.

Além da eficiência, a qualidade das imagens desempenha um papel crítico na precisão das análises subsequentes, tais como a estimação de profundidade e a detecção de obstáculos. Durante os testes, avaliamos o nível de qualidade das imagens capturadas, levando em consideração aspectos como resolução e nitidez.

**Figura 24 – Envio de uma imagem concluído no aplicativo móvel**




**Fonte: Autoria Própria.**

Os resultados evidenciam que as imagens capturadas pelo aplicativo mantêm uma resolução apropriada, mesmo que esta seja modesta, com dimensões de apenas 240 x 135 pixels, suficientes para a análise de obstáculos. Além disso, elas se apresentam nítidas o bastante para permitir a identificação de detalhes cruciais do ambiente.

A transmissão das imagens do aplicativo ao sistema de detecção de obstáculos e planejamento de rotas foi avaliada quanto à confiabilidade e integridade dos dados transmitidos.

Os resultados mostram que a transmissão das imagens ocorre de forma bem-sucedida, sem perda de dados. Isso é essencial para a percepção em tempo real do ambiente pelo sistema de navegação autônoma.

### 4.2.2 Servidor Flask de recebimento das imagens do aplicativo

O servidor, construído utilizando o framework Flask, desempenha um papel fundamental na integração do sistema e na viabilização da comunicação contínua entre o aplicativo móvel e o sistema de processamento de imagens e navegação autônoma.

Como mencionado na seção de metodologia, o servidor utiliza rotas (URLs) para processar as requisições HTTP. A rota `/upload` é o ponto de entrada principal para o servidor, e ela é configurada para aceitar requisições do tipo POST, nas quais os dados são fornecidos no formato JSON. A função `upload_image` é responsável por gerenciar essa rota e realizar as operações necessárias. O código responsável por esses processos está evidenciado no algoritmo A.

Um aspecto crítico do servidor é a capacidade de processar imagens enviadas pelo aplicativo móvel. A função `save_base64_image` é responsável por decodificar as imagens no formato base64 e armazená-las localmente no diretório `/uploads` como consta na figura 25. A imagem resultante consta na figura 26. Essas imagens são salvas no formato PNG, garantindo que a qualidade seja mantida e que elas estejam prontas para análises futuras.

### 4.2.3 Análise da Estimativa de Profundidade

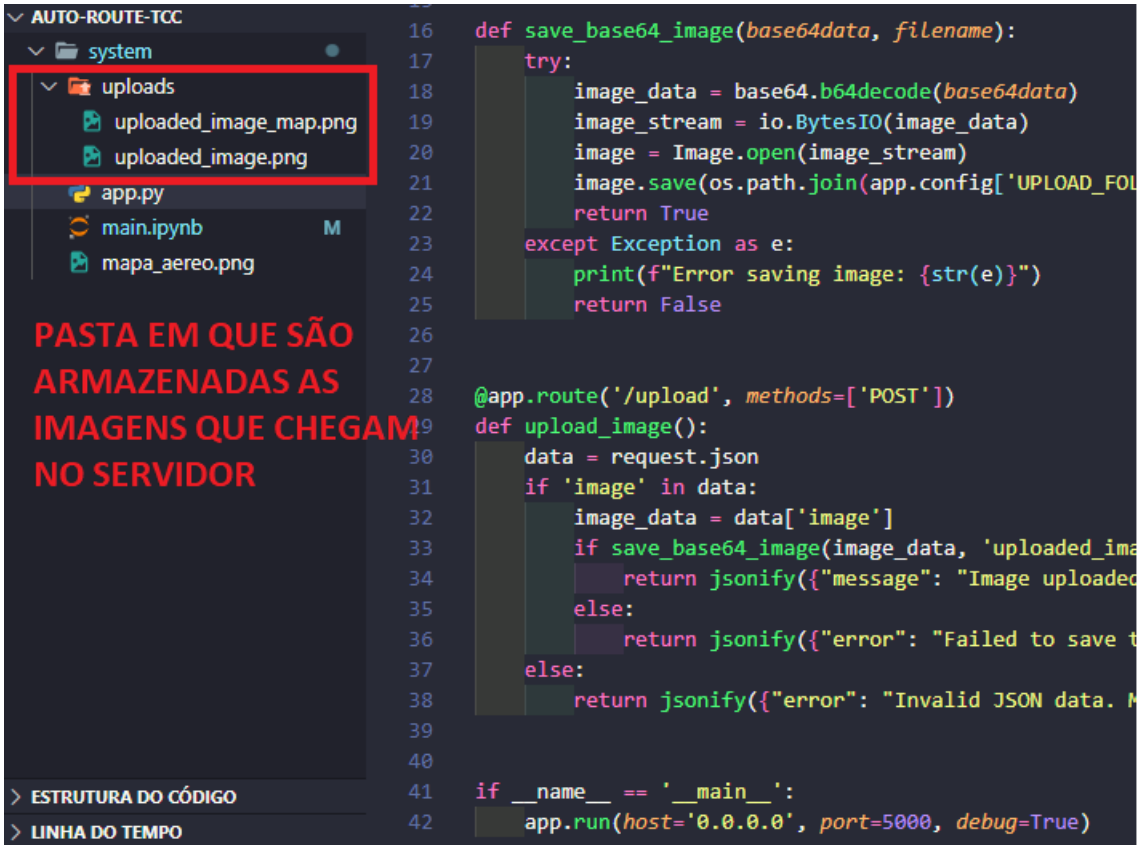
A estimação de profundidade desempenha um papel essencial na percepção do ambiente, fornecendo informações cruciais sobre a distância dos objetos em relação ao veículo autônomo. A precisão e a confiabilidade dessas estimativas são fundamentais para o planejamento de trajetórias seguras e a detecção de obstáculos no caminho do veículo.

Para avaliar a estimação de profundidade, utilizamos o modelo MiDaS (*Modelo de Aprendizado de Máquina da Intel Labs para Estimação de Profundidade Monocular*), que foi descrito na seção de metodologia. O modelo MiDaS é uma ferramenta poderosa para inferir a profundidade a partir de imagens capturadas pela câmera do veículo. A seguir, descrevemos a metodologia de teste adotada:

1. **Carregando o Modelo:** Inicialmente, carregamos o modelo MiDaS e o configuramos para o processamento na CPU. O modelo está pronto para realizar estimações de profundidade.

2. **Processo de Estimação:** Para estimar a profundidade, alimentamos o modelo com as imagens capturadas pelo veículo. O modelo utiliza redes neurais convolucionais profundas para inferir a profundidade de cada pixel na cena.

**Figura 25 – Pasta uploads no projeto com as imagens que chegaram no servidor**




**Fonte: Autoria Própria.**

3. **Interpolação e Normalização:** A saída do modelo é uma imagem de profundidade inicial. Realizamos uma interpolação para corresponder às dimensões da imagem original e, em seguida, normalizamos a imagem de profundidade resultante para uma representação útil no intervalo [0, 255].

Além das métricas quantitativas, é importante analisar os resultados qualitativos. A imagem de profundidade resultantes são uma ferramenta valiosa para avaliar visualmente o desempenho do sistema.

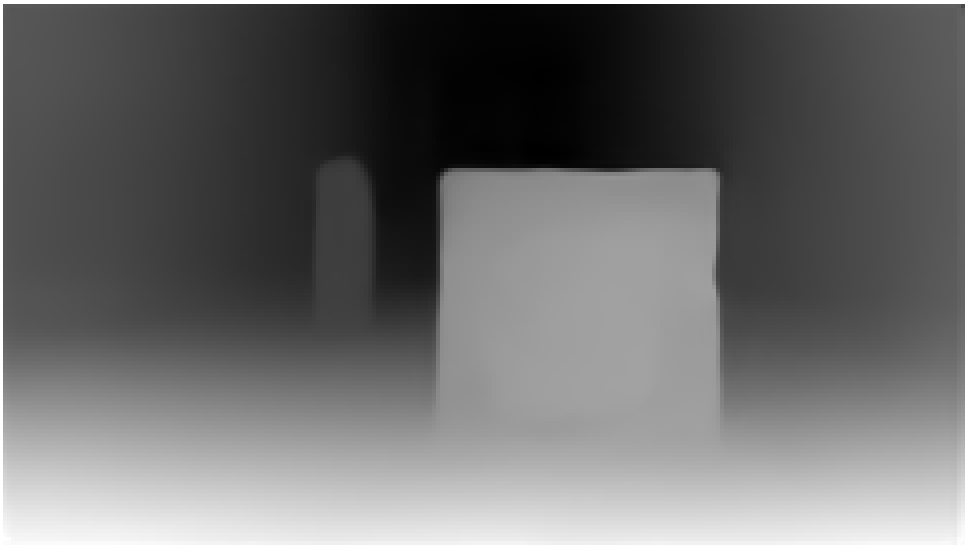
A figura 27 representa uma imagem de profundidade resultante a partir da figura 26. Essa imagem oferece uma representação visual das estimativas de profundidade e é crucial para o planejamento de trajetórias seguras e a detecção de obstáculos.

Os resultados da estimação de profundidade são promissores, indicando que o sistema é capaz de inferir a profundidade com precisão em uma variedade de cenários. No entanto, é importante notar que existem desafios e limitações, como a influência de condições de iluminação e textura do ambiente. Esses fatores podem afetar o desempenho da estimação de profundidade.

**Figura 26 – Imagem recebida no servidor e decodificada**

**Fonte: Autoria Própria.**

**Figura 27 – Imagem de Profundidade Resultante**



**Fonte: Autoria Própria.**

### 4.2.4 Análise da Variação da Intensidade Luminosa dos Pixels

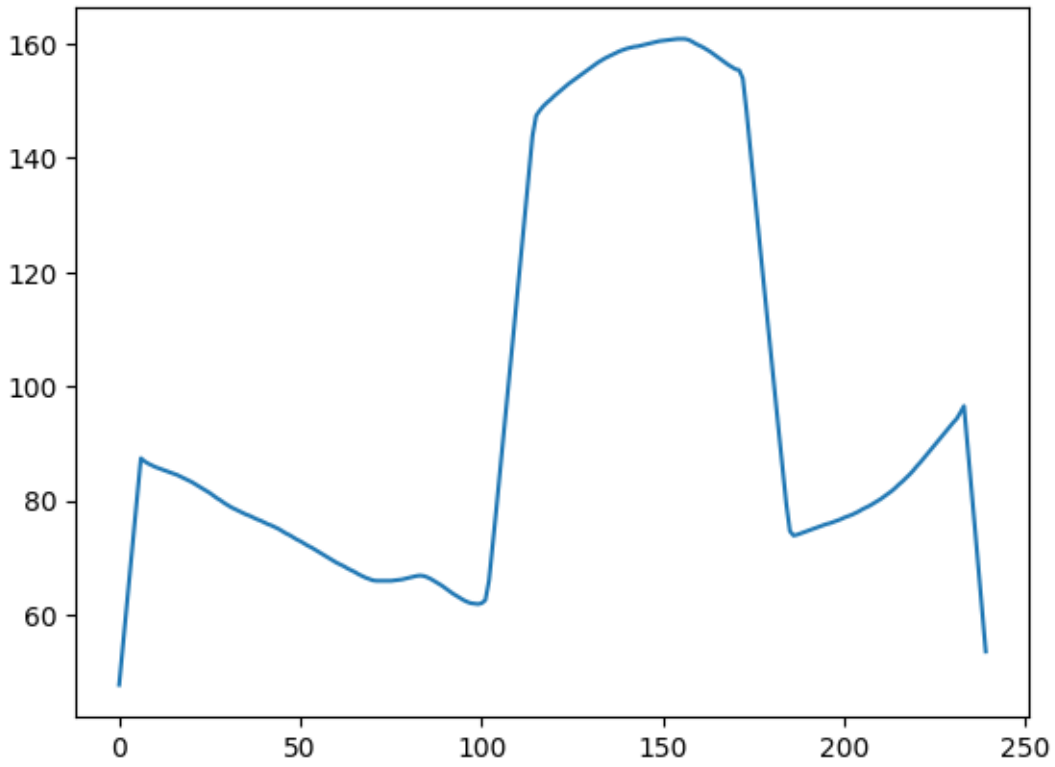
A análise da variação da intensidade luminosa dos pixels desempenha um papel crítico na percepção visual, especialmente em sistemas de visão computacional voltados para a detecção e esquiva de obstáculos. Essa técnica desempenha um papel instrumental na extração de informações cruciais a partir das imagens capturadas, permitindo a identificação de objetos e obstáculos com base nas variações na intensidade luminosa.

A premissa fundamental dessa abordagem é que objetos e obstáculos em uma cena frequentemente se traduzem em flutuações na intensidade dos pixels na imagem. Essas flutu-

ações são consequência das interações da luz com os objetos, criando regiões de alta e baixa intensidade luminosa na imagem, conforme discutido anteriormente na revisão bibliográfica.

Após a etapa de estimação de profundidade, é gerada uma imagem que realça os objetos mais próximos da câmera com pixels mais luminosos, enquanto os objetos mais distantes são representados por pixels menos luminosos, como ilustrado na Figura 27. O processo começa com a seleção de uma linha ou região de interesse na imagem, frequentemente denominada "linha central", em que a análise será conduzida. A escolha dessa linha é fundamentada em considerações práticas e na geometria da câmera utilizada no sistema, e isso resulta em um gráfico semelhante ao da Figura 28.

**Figura 28 – Linha central analisada**




**Fonte: Autoria Própria.**

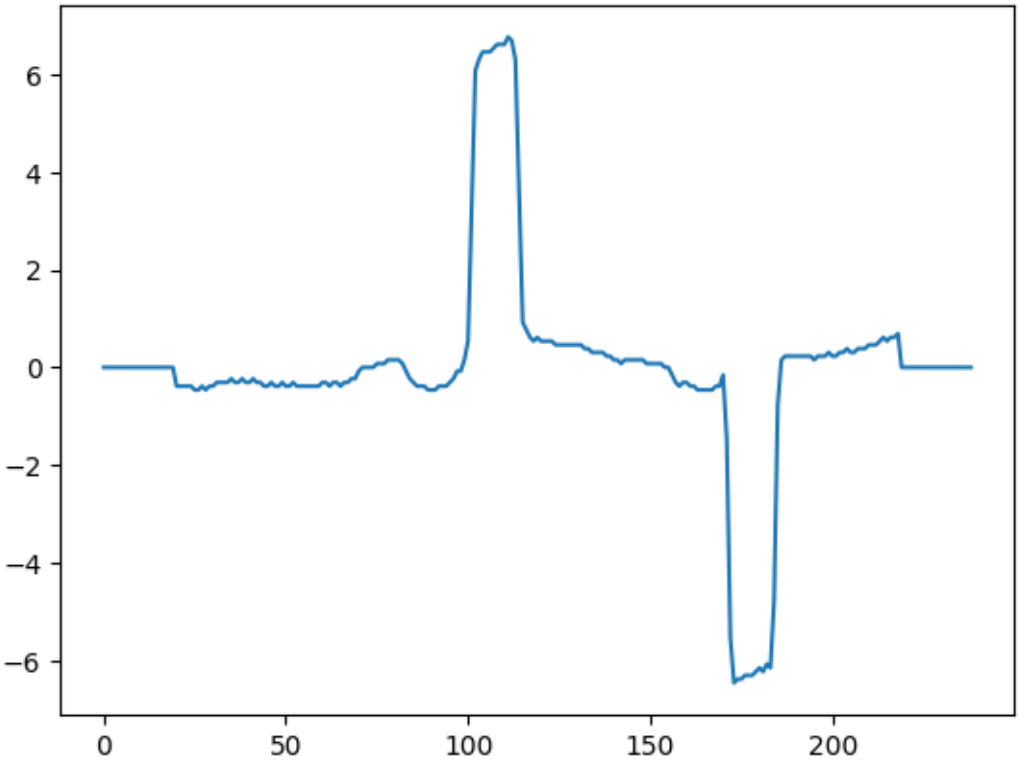
A análise da variação da intensidade luminosa dos pixels envolve a detecção de "picos de intensidade"ao longo da linha central da imagem, que são identificados a partir da derivada dessa linha central. Esses picos são pontos notáveis no perfil de intensidade da linha, indicando variações significativas na intensidade luminosa. Os picos de intensidade podem ser de dois tipos principais:

1. "Picos Positivos": São identificados quando ocorre um aumento acentuado na intensidade luminosa em comparação com os pixels vizinhos ao longo da linha. Geralmente, esses picos correspondem a áreas de alta intensidade na imagem e, em muitos casos, estão associados a bordas ou contornos de objetos na cena.

2. "Picos Negativos": São detectados quando há uma diminuição acentuada na intensidade luminosa em relação aos pixels adjacentes. Esses picos indicam regiões de baixa intensidade na imagem, e podem representar depressões ou espaços vazios na cena.

A identificação e análise desses picos de intensidade desempenham um papel crucial na detecção de características visuais significativas na imagem. Os picos positivos frequentemente destacam o início dos objetos na horizontal, enquanto os picos negativos podem indicar o término desses objetos, como ilustrado na Figura 29.

**Figura 29 – Picos positivos e negativos**




**Fonte: Autoria Própria.**

Após a detecção dos picos de intensidade, o próximo passo é o emparelhamento desses picos para construir uma representação dos obstáculos presentes na cena. Cada par de picos sugere a largura do obstáculo, enquanto a intensidade máxima entre eles fornece uma estimativa da altura do obstáculo e da proximidade em relação ao veículo. Essa representação é de importância primordial para a identificação e localização de obstáculos no ambiente, como indicado no apêndice D.

### 4.2.5 Mapeamento Aéreo do Ambiente

A criação de um mapa aéreo do ambiente é um componente fundamental para o sucesso da navegação autônoma, pois fornece uma representação abstrata do ambiente em que

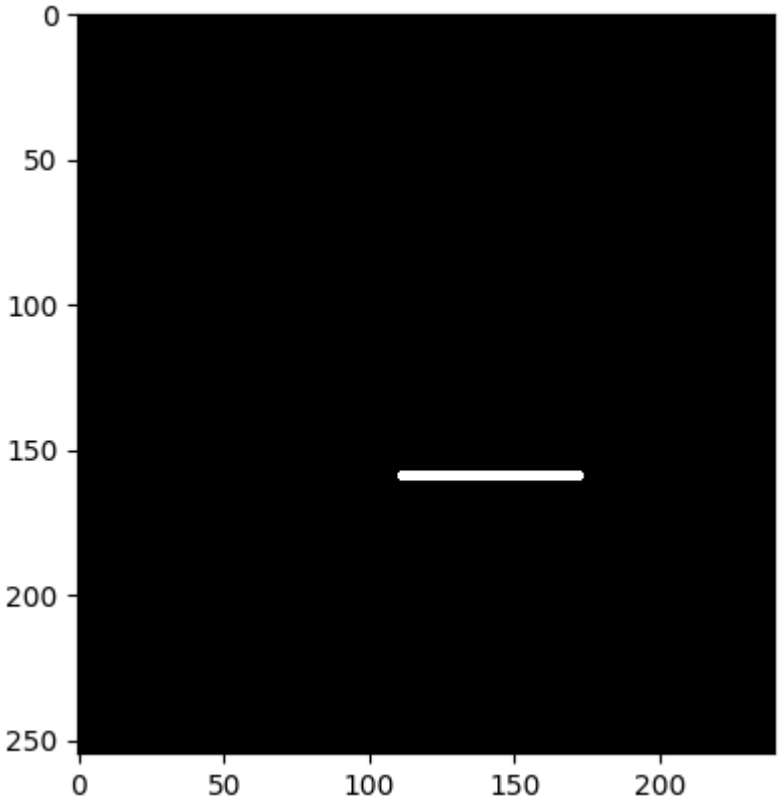
um veículo autônomo opera. Nesse contexto, o objetivo principal é identificar obstáculos significativos e representá-los em um mapa aéreo bidimensional, a fim de planejar trajetórias seguras e evitar colisões.

O processo de criação do mapa aéreo é iniciado após a etapa de estimação de profundidade, que destaca os objetos mais próximos da câmera com pixels mais luminosos na imagem resultante. Os obstáculos filtrados, que atendem a critérios específicos, são considerados na construção do mapa. Cada obstáculo é representado por um conjunto de coordenadas no mapa aéreo, onde cada ponto é marcado com o valor "1", indicando a presença de um obstáculo naquela posição. Esse procedimento é ilustrado no apêndice E.

A representação de obstáculos no mapa aéreo desempenha um papel crucial na navegação autônoma, uma vez que fornece informações valiosas sobre a distribuição espacial dos obstáculos no ambiente. Essa representação é usada para evitar colisões e planejar trajetórias que estejam livres de obstáculos, tornando a navegação mais segura e precisa.

No mapa aéreo criado, cada obstáculo é refletido como uma área preenchida com "1,"o que indica que aquela região representa um obstáculo na cena. Quanto mais intenso o pixel da imagem original associado a um obstáculo, mais próximo ele é representado do ponto de origem da captura da imagem, que corresponde ao ponto de visão do veículo autônomo no mapa aéreo.

**Figura 30 – Resultado do Mapa Aéreo**




**Fonte: Autoria Própria.**

A Figura 30 apresenta o resultado do mapa aéreo do ambiente gerado pelo sistema. Nessa imagem, é possível visualizar uma representação visual das informações de mapeamento. O mapa aéreo demonstra de maneira clara e concisa a localização e extensão dos obstáculos no ambiente, permitindo que o veículo autônomo tome decisões informadas para evitar colisões e planejar trajetórias seguras.

O sucesso na criação do mapa aéreo é um marco significativo para o sistema de navegação autônoma, uma vez que a qualidade e precisão dessa representação têm um impacto direto na segurança e eficácia da navegação.

### 4.2.6 Predição de Rotas

A predição de rotas é uma fase crítica em sistemas de navegação autônoma, onde o principal objetivo é planejar trajetórias seguras e eficientes para um veículo autônomo, permitindo que ele navegue de sua posição atual ao seu destino enquanto evita obstáculos. Nesta seção, exploramos o uso do algoritmo A* (A-Estrela) como uma ferramenta poderosa para a predição de rotas.

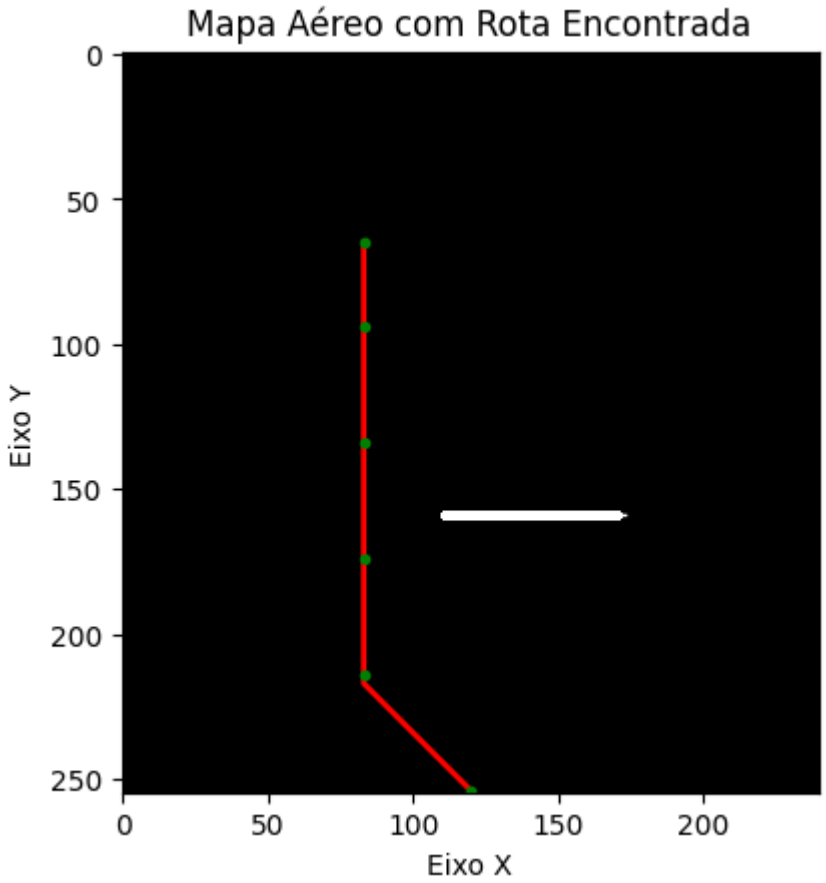
O algoritmo A* é amplamente reconhecido como uma técnica de busca de caminho eficaz e é aplicado com sucesso em cenários de navegação autônoma. Ele opera com base em uma função de custo que considera o custo acumulado para alcançar um ponto específico e uma estimativa do custo restante para chegar ao destino. Essa função de custo é fundamental para priorizar a expansão dos nós mais promissores durante a busca, permitindo que o A* encontre rotas ótimas.

Uma característica distintiva do A* é a utilização de uma heurística que fornece uma estimativa do custo restante para atingir o destino a partir de um ponto dado. A escolha da heurística é crítica e pode afetar significativamente o desempenho do algoritmo. No contexto da navegação autônoma, a representação precisa do ambiente é essencial. Os obstáculos detectados no ambiente são tratados como áreas intransitáveis, enquanto os espaços livres são considerados acessíveis. O A* utiliza essa representação para encontrar um caminho que leva em consideração a topografia do ambiente e evita colisões.

Na implementação específica do algoritmo A*, um mapa aéreo é utilizado como representação do ambiente. Os obstáculos detectados são representados como áreas intransitáveis, e os espaços livres são considerados acessíveis para o veículo autônomo. O processo envolve a dilatação dos obstáculos no mapa aéreo, que cria margens de segurança ao redor dos obstáculos, garantindo que o veículo tenha espaço suficiente para passar com segurança.

A rota planejada pelo algoritmo A* é visualizada no mapa aéreo, destacando o caminho em vermelho, como ilustrado na Figura 31. Essa rota representa a trajetória ideal para o veículo autônomo seguir, evitando colisões com os obstáculos detectados no ambiente e com objetivo definido no objeto vermelho da cena que é representado por uma caixinha vermelha. O processo de localização e definição de trajeto para o objeto vermelho será relatado na seção a seguir.

**Figura 31 – Rota gerada pelo algoritmo A Estrela**




**Fonte: Autoria Própria.**

O sucesso na predição de rotas é fundamental para a navegação segura e eficiente do veículo autônomo. O algoritmo A* desempenha um papel crucial, permitindo que o veículo tome decisões em tempo real e siga uma trajetória segura em direção ao seu destino. A qualidade dessa predição é um fator determinante para a segurança e eficácia da navegação autônoma.

### 4.2.7 Identificação do Objeto Vermelho

A identificação do objeto vermelho na cena é uma etapa de destaque em sistemas de processamento de imagem e visão computacional, desempenhando um papel crucial no contexto da navegação autônoma. Nesta seção, exploramos os resultados da identificação do objeto vermelho na imagem e como suas coordenadas são utilizadas como um objetivo a ser alcançado na navegação autônoma.

No algoritmo apresentado, a imagem original, capturada e enviada pelo aplicativo mobile, conforme é mostrado na figura 26, é carregada e, em seguida, é convertida para o espaço de cores HSV (Matiz, Saturação e Valor). A conversão para o espaço de cores HSV é uma técnica comum que facilita a detecção de objetos com base em sua cor. No caso em questão, o objetivo é detectar objetos de cor vermelha na imagem.

Uma máscara é aplicada à imagem no espaço de cores HSV para identificar os pixels que se encontram na faixa de cor vermelha. Essa máscara isola eficazmente os pixels vermelhos, destacando o objeto de interesse.

Os contornos do objeto vermelho são então encontrados na máscara, permitindo determinar a posição e a forma desse objeto na imagem. Dentre todos os contornos detectados, o algoritmo identifica o maior, que é considerado o objeto vermelho principal na cena. Isso é realizado calculando a área de cada contorno e selecionando o maior.

Uma vez que o maior contorno, representando o objeto vermelho, é identificado, são calculadas as coordenadas do seu centro. Essas coordenadas representam o centro do objeto vermelho na imagem e são utilizadas como o objetivo a ser alcançado na navegação autônoma. O objetivo é mover o veículo autônomo em direção a esse ponto, evitando obstáculos ao longo do caminho.

A identificação do objeto vermelho é uma etapa essencial em muitas aplicações de automação e robótica, permitindo que sistemas autônomos localizem e sigam objetos de interesse em seu ambiente. No contexto da navegação autônoma, a identificação do objeto vermelho desempenha um papel vital na definição do destino e na tomada de decisões autônomas, garantindo que o veículo alcance seu objetivo com precisão e eficácia.

### 4.2.8 Envio de Dados via Bluetooth

A funcionalidade de envio de dados via Bluetooth é uma parte crucial do sistema de navegação autônoma, permitindo a comunicação entre o veículo autônomo e outros dispositivos, como o Arduino. A seguir, é descrito os resultados do processo de envio de dados via Bluetooth no contexto do sistema:

1. **Extração de Dados de Rota**: Inicialmente, o sistema extrai os dados de rota que descrevem o caminho planejado pelo algoritmo A*, conforme consta no apêndice I. Esses dados incluem informações sobre distância percorrida e mudanças de direção, representadas como ângulos. Com as informações de ângulo e distância, o veículo pode realizar a modelagem cinemática para executar seu movimento que se trata do tabalho.

2. **Codificação dos Dados**: Os dados de rota extraídos são convertidos em uma sequência de caracteres formatados, que são essenciais para a transmissão via Bluetooth. Essa codificação é feita para garantir que os dados sejam transmitidos de maneira organizada e possam ser interpretados corretamente pelo dispositivo receptor.

3. **Envio dos Dados**: A função `enviar_dados_via_bluetooth` é acionada para enviar os dados via Bluetooth. O sistema cria um socket Bluetooth, estabelece uma co-

nexão com o dispositivo de destino (Arduino) usando o endereço MAC correspondente e a porta padrão 1, e envia os dados codificados.

4. **Tratamento de Exceções**: O código está envolto em um bloco `try-except` para lidar com possíveis exceções que podem ocorrer durante o envio dos dados via Bluetooth. Qualquer erro é devidamente tratado e exibido como mensagem de erro.

Além disso, o código lida com a preparação dos dados para envio, incluindo o cálculo das distâncias percorridas e das mudanças de direção, bem como a conversão de ângulos em radianos. Todos esses detalhes são essenciais para garantir que os dados enviados sejam interpretados corretamente pelo dispositivo de destino, que no caso é o Arduino.

O resultado final é a transmissão bem-sucedida dos dados da rota para o Arduino via Bluetooth, fornecendo as informações necessárias para a navegação autônoma. Essa funcionalidade é vital para permitir que o veículo autônomo siga a rota planejada e tome decisões com base nas informações recebidas.

## 4.3 Testes em ambientes reais

Esta seção tem como objetivo analisar o desempenho do sistema em testes práticos. O sistema foi desenvolvido para processar as imagens capturadas pela câmera, a fim de identificar obstáculos e evitar colisões, bem como identificar um ponto de destino na rota, que neste caso é um objeto vermelho. O ambiente prático é mostrado na figura 32. A seguir, apresentaremos o comportamento do sistema à medida que a rota é gerada.

**Figura 32 – Ambiente real de ação do sistema de gerenciamento de rotas em um veículo terrestre**


**Fonte: Autoria Própria.**

Na perspectiva do veículo, a visão no ponto inicial do trajeto é representada na figura 33. Nesta cena, o objetivo ainda não está evidenciado na figura. Portanto, o sistema está programado para avançar o veículo, desviando dos obstáculos, até que o objetivo seja identificado.

**Figura 33 – Perspectiva do veículo em sua posição inicial**


**Fonte: Autoria Própria.**

Após receber informações relacionadas aos ângulos e distâncias dos pontos ao longo da rota, geradas pelo algoritmo A*, o veículo realiza uma modelagem cinemática para conduzir sua trajetória, culminando na sua chegada ao estado representado na figura 34. É importante destacar que, de acordo com a descrição deste projeto, a modelagem cinemática do veículo autônomo faz parte do trabalho de conclusão de curso da graduanda em Engenharia de Computação, Mariana Gonçalves Rodrigues (RODRIGUES, 2023).

Neste teste, com a terceira etapa realizada pelo veículo autônomo, o sistema de planejamento de rotas já consegue identificar o objetivo da rota, que corresponde ao objeto vermelho presente na cena conforme consta na figura 35. Como resultado, uma rota é traçada até o objeto vermelho, e informações detalhadas sobre os ângulos e distâncias necessários para alcançar o destino são transmitidas ao veículo.

Com o sucesso na consecução do objetivo, sem qualquer incidente de colisão do veículo com obstáculos, chegamos ao término desta seção. Consequentemente, não há mais necessidade de efetuar novas capturas do ambiente.

**Figura 34 – Perspectiva do veículo após executar a primeira instrução de rota**


**Fonte: Autoria Própria.**

**Figura 35 – Perspectiva do veículo após executar a segunda instrução de rota**


**Fonte: Autoria Própria.**

# 5 CONCLUSÃO

No âmbito deste projeto de conclusão de curso, foi empreendido um esforço significativo na criação de um sistema que se utilizasse de recursos mínimos para o planejamento de rotas de um veículo autônomo. A meta principal era demonstrar que apenas a câmera de um smartphone, uma rede e um computador seriam suficientes para realizar essa tarefa de maneira eficaz.

Os objetivos específicos deste projeto foram centrados na identificação de objetos cruciais para a locomoção do veículo terrestre e na estimativa da posição do veículo em tempo real, utilizando exclusivamente os dados obtidos pela câmera. Nesse contexto, o algoritmo de estimativa de profundidade monocular, MiDaS, destacou-se de maneira notável, alcançando resultados extremamente satisfatórios na estimativa de profundidade. Adicionalmente, a análise das imagens geradas pela rede neural revelou-se de importância crucial na identificação de obstáculos próximos e distantes, desempenhando um papel determinante na estimação da localização do veículo na cena. Essa abordagem permitiu que a implementação do algoritmo A* desempenhasse um papel fundamental ao prever rotas com base na detecção de obstáculos e na localização do veículo no ambiente circundante.

Com a rota determinada pelo algoritmo A*, tornou-se necessário extrair informações intrínsecas da rota, como mudanças de direção, ângulos e distâncias. Isso porquê apenas essas informações devem ser enviadas para o veículo, visto que a modelagem cinemática é um complemento desse trabalho, que foi realizado pela Mariana (RODRIGUES, 2023).

Inicialmente, tínhamos planejado implementar o sistema de planejamento de rotas e a identificação de obstáculos em uma Raspberry Pi, devido à sua praticidade e mobilidade, considerando a possibilidade de instalá-la no veículo. No entanto, nos deparamos com desafios significativos, já que esse dispositivo não conseguiu desempenhar as tarefas necessárias devido a incompatibilidades com as bibliotecas do Python, especialmente com o PyTorch, que é uma biblioteca de código aberto amplamente utilizada para machine learning e deep learning e inclui o MiDaS para fins de estimativa de profundidade.

Como resultado dessas limitações, foi necessário adaptar o escopo do projeto. Optamos por executar o sistema de planejamento de rotas em um computador com acesso à rede, que também possuía recursos de comunicação bluetooth para transmitir os dados processados ao veículo. Essa mudança permitiu superar os desafios técnicos iniciais e as incompatibilidades de hardware, garantindo assim a continuidade e eficácia do projeto.

A avaliação do desempenho do sistema envolveu testes em diversos ambientes, tanto internos quanto externos. No entanto, em ambientes externos, o sistema enfrentou desafios significativos, principalmente devido à presença de um grande número de obstáculos na maioria dos cenários e à variação na incidência de luz, o que comprometeu a precisão na determinação da profundidade das imagens. Como perspectiva para melhorias futuras, é evidente a necessi-

dade de tornar o cálculo de profundidade mais robusto, algo que pode ser abordado em versões subsequentes do software.

Por outro lado, em ambientes internos, em que a iluminação pode ser controlada, o sistema obteve um desempenho notável. Conseguiu identificar obstáculos de forma consistente e planejar rotas eficazes com base nas informações disponíveis. Este projeto, portanto, representou um importante passo em direção à criação de um sistema de navegação autônoma eficiente e acessível, evidenciando o potencial e os desafios a serem enfrentados nesse campo em constante evolução.

Ainda assim, o projeto cumpre com o objetivo de criar um sistema de navegação autônoma, aplicando algoritmos de visão computacional para a definição de rotas de navegação de veículos terrestres utilizando inteligência artificial, identificando e reconhecendo obstáculos em tempo real, traçando rotas seguras que permitem a locomoção do veículo de forma autônoma.

Em suma, o presente trabalho contribui para o avanço da visão computacional, especialmente na área de navegação autônoma. O processamento de imagens desempenha um papel fundamental na otimização dos processos de navegação, permitindo que veículos se movam de forma eficiente e segura em ambientes complexos utilizando poucos recursos. Espera-se que este trabalho seja útil para instituições científicas e empresas que buscam desenvolver tecnologias acessíveis e avançadas nessa área.

# REFERÊNCIAS


ALPAYDIN, E. **Introdução à aprendizagem computacional**. 2. ed. [*S.l.*]: LTC, 2010.

BERG, J.; PENNY, G. Dijkstra versus a* revisited. **International Journal of Computer Science Issues (IJCSI)**, v. 8, n. 6, p. 188–195, 2011.

BHAT, S. F. *et al.* Zoedepth: Zero-shot transfer by combining relative and metric depth. **Cornell University, arXiv preprint**, 2023. Disponível em: https://arxiv.org/abs/2302.12288.

BREDA, T. V. **"Por que eu tenho que trabalhar lateralidade?": Experiências formativas com professoras dos anos iniciais**. 2017, 2017.

FONSECA, E. **O que é HTTP, Request, GET, POST, Response, 200, 404?** 2018. Disponível em: https://www.treinaweb.com.br/blog/o-que-e-http-request-get-post-response-200-404.

FOX, D.; BURGARD, W.; THRUN, S. Markov localization for mobile robots in dynamic environments. **Journal of Artificial Intelligence Research**, v. 11, p. 391–427, 1999.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [*S.l.*]: MIT Press, 2016.

GREENAWAY, T. Automated harvest is coming. what will it mean for farmworkers and rural communities? **Civil Eats**, 2020. Accessed on October 17, 2023. Disponível em: https://civileats.com/2020/09/29/automated-harvest-is-coming-what-will-it-mean-for-farmworkers-and-rural-communities/.

HART, P. E.; NILSSON, N. J.; RAPHAEL, B. A formal basis for the heuristic determination of minimum cost paths. **IEEE Transactions on Systems Science and Cybernetics**, IEEE, v. 4, n. 2, p. 100–107, 1968.

HOUSER, K. **Fully self-driving cars are finally available on Uber**. 2023. Disponível em: https://www.freethink.com/transportation/autonomous-cars.

KARAMAN, S.; FRAZZOLI, E. Sampling-based algorithms for optimal motion planning. **International Journal of Robotics Research**, SAGE Publications Sage UK: London, England, v. 30, n. 7, p. 846–894, 2011.

MAULION, M. Color image segmentation — image processing. **Medium**, 2021. Disponível em: https://mattmaulion.medium.com/color-image-segmentation-image-processing-4a04eca25c0.

MONTEMERLO, M. *et al.* Fastslam: A factored solution to the simultaneous localization and mapping problem. **AAAI/IAAI**, Citeseer, v. 593, p. 593–598, 2002.

MORELL, J. **Self-Driving Mining Trucks**. 2017. Published on March 16, 2017. Disponível em: https://www.asme.org/topics-resources/content/selfdriving-mining-trucks.

OLSON, C. F. Probabilistic self-localization for mobile robots. **IEEE Transactions on Robotics and Automation**, v. 16, n. 1, p. 1–12, fev. 2000.

PEDRINI, H.; SCHWARTZ, W. R. **Análise de Imagens Digitais**. São Paulo: Thomson, 2008.

RANFTL, R. *et al.* Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. **IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)**, 2020.

RANFTL, R. *et al.* Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. **IEEE Transactions on Pattern Analysis and Machine Intelligence**, v. 44, n. 3, 2022.

REN, S. *et al.* Faster r-cnn: Towards real-time object detection with region proposal networks. **arXiv preprint arXiv:1506.01497**, 2015.

RODRIGUES, M. G. **DESENVOLVIMENTO DE UM MODELO CINEMÁTICO PARA UM VEÍCULO TERRESTRE DE BAIXO CUSTO QUE UTILIZA DE SISTEMAS INTELIGENTES PARA NAVEGAÇÃO AUTÔNOMA**. Out 2023 — Universidade Tecnológica Federal do Paraná, Apucarana, Out 2023.

SINGHAL, A. A* algorithm | a* algorithm example in ai. **Gate Vidyalay**, 2018. Disponível em: https://www.gatevidyalay.com/a-algorithm-a-algorithm-example-in-ai/.

STAN, G. B. M. *et al.* Ldm3d: Latent diffusion model for 3d. **Nome da revista ou conferência onde a obra foi publicada**, 2023. Disponível em: https://paperswithcode.com/paper/ldm3d-latent-diffusion-model-for-3d.

THRUN, S. *et al.* Probabilistic algorithms and the interactive museum tour-guide robot minerva. **International Journal of Robotics Research**, 2000. To appear.

THRUN, S.; BURGARD, W.; FOX, D. **Probabilistic Robotics**. [*S.l.*]: MIT Press, 2005.

THRUN, S. *et al.* Probabilistic algorithms and the interactive museum tour-guide robot minerva. **International Journal of Robotics Research**, v. 27, n. 5, p. 523–545, 2008.

WANG, Z. **Self Driving RC Car**. 2015. Disponível em: https://zhengludwig.wordpress.com/projects/self-driving-rc-car/.

# APÊNDICE A – Servidor Flask para receber as imagens do aplicativo

```
 from flask import Flask, request, jsonify
import os
import base64
import io
from PIL import Image

app = Flask(__name__)

UPLOAD_FOLDER = 'uploads'
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

if not os.path.exists(UPLOAD_FOLDER):
    os.makedirs(UPLOAD_FOLDER)


def save_base64_image(base64data, filename):
    try:
        image_data = base64.b64decode(base64data)
        image_stream = io.BytesIO(image_data)
        image = Image.open(image_stream)
        image.save(os.path.join(app.config['UPLOAD_FOLDER'], filename
   ), 'PNG')
        return True
    except Exception as e:
        print(f"Error saving image: {str(e)}")
        return False


@app.route('/upload', methods=['POST'])
def upload_image():
    data = request.json
    if 'image' in data:
        image_data = data['image']
        if save_base64_image(image_data, 'uploaded_image.png'):
```

```
34              return jsonify({"message": "Image uploaded and saved
    successfully."}), 200
35          else:
36              return jsonify({"error": "Failed to save the image."}),
    500
37      else:
38          return jsonify({"error": "Invalid JSON data. Make sure it
    contains 'image' field."}), 400
39
40
41  if __name__ == '__main__':
42      app.run(host='0.0.0.0', port=5000, debug=True)
```

# APÊNDICE B – Código para achar a profundidade das imagens

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
from queue import PriorityQueue
import torch
import os
import socket

midas = torch.hub.load('intel-isl/MiDaS', 'MiDaS_small')
midas.to('cpu')
midas.eval()

transforms = torch.hub.load('intel-isl/MiDaS', 'transforms')
transform = transforms.small_transform

image_folder = 'uploads'
image_filename = 'uploaded_image.png'

image_path = os.path.join(image_folder, image_filename)
frame = cv2.imread(image_path)

img = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
imgbatch = transform(img).to('cpu')

with torch.no_grad():
    prediction = midas(imgbatch)
    prediction = torch.nn.functional.interpolate(
        prediction.unsqueeze(1),
        size=img.shape[:2],
        mode='bicubic',
        align_corners=False
    ).squeeze()

    output = prediction.cpu().numpy()
```

```
77      output = (output - output.min()) / (output.max() - output.min())
78      output = (255 * output).astype(np.uint8)
79
80  plt.imshow(output, cmap='gray')
81  plt.pause(0.00001)
82
83  output_filename = os.path.splitext(image_filename)[0] + "_map.png"
84  output_path = os.path.join(image_folder, output_filename)
85  cv2.imwrite(output_path, output)
86  print(f'Saved {output_path}')
87  print(image_filename)
```

# APÊNDICE C – Código para análise da variação da intensidade luminosa dos pixels

```
 img = cv2.imread('uploads/uploaded_image_map.png',
                  cv2.IMREAD_GRAYSCALE)

deubom = False

if img is None:
    print("Imagem não carregada.")
else:
    deubom = True
    print("Imagem carregada.")

    linha_central = 80 #linha principal do espectro de intensidade do
    pixel

if deubom:
    x = img[linha_central, :]

    window_size = 13
    x = np.convolve(x, np.ones(window_size) / window_size, 'same')
    plt.plot(x)

    # segunda derivada da linha central da imagem
    dx = np.diff(x)
    dx[:20] = 0
    dx[-20:]= 0

    plt.figure()
    plt.plot(dx)

    # picos positivos
    pks_positivos = []
    y_values_positivos = []
    while (np.max(dx) > 2.5) and len(pks_positivos) < 10:
        pos = np.argmax(dx)
```

```
121         pks_positivos.append(pos)
122         y_values_positivos.append(dx[pos])
123         lo, hi = max(0, pos - 50), min(len(x), pos + 50)
124         dx[lo:hi] = 0
125     print("Picos Positivos:", pks_positivos)
126 
127     # picos negativos
128     dx_negativo = -dx
129     pks_negativos = []
```

# APÊNDICE D – Análise dos picos positivos e negativos da derivada da linha central.

```
130 while (np.max(dx_negativo) > 2.5) and len(pks_negativos) < 10:
131         pos = np.argmax(dx_negativo)
132         pks_negativos.append(pos)
133         lo, hi = max(0, pos - 50), min(len(x), pos + 50)
134         dx_negativo[lo:hi] = 0
135     print("Picos Negativos:", pks_negativos)
136 
137     # Crie a tupla de obstáculos
138     if len(pks_positivos) > 0 or len(pks_negativos) > 0:
139 
140         if len(pks_positivos) == 0:
141             pks_positivos.append(0)
142 
143         if len(pks_negativos) == 0:
144             pks_negativos.append(len(dx))
145 
146         if pks_negativos[0] < pks_positivos[0]:
147             pks_positivos = [0] + pks_positivos
148 
149         if pks_negativos[-1] < pks_positivos[-1]:
150             pks_negativos = pks_negativos + [len(dx)]
151 
152 
153         if len(pks_positivos) != len(pks_negativos):
154             print("erro", pks_positivos, pks_negativos, len(dx))
155 
156     mean_x = [(a+b)//2 for a, b in zip(pks_positivos, pks_negativos)]
157     y_values_positivos = [x[c] for c in mean_x]
158     obstaculos = list(
159         zip(pks_positivos, pks_negativos, y_values_positivos))
160     print("Obstáculos:", obstaculos)
161 
162     # Filtrar obstáculos com base nos critérios
163     obstaculos_filtrados = [
```

```
164         obstaculo for obstaculo in obstaculos if obstaculo[0] >= 5 or
    obstaculo[1] <= 235]
165     print("Obstáculos filtrados:", obstaculos_filtrados)
166 
167     # Define limites mínimos para altura e diferença entre início e
    fim de obstáculos
168     limite_altura_minima = 126
169     limite_diferenca_x = 10
```

# APÊNDICE E – Criação do mapa aéreo

```
 # Define as dimensões do mapa aéreo (ajuste conforme necessário)
largura_mapa = 240
altura_mapa = 255

# Cria uma matriz para representar o mapa aéreo
mapa_aereo = np.zeros((altura_mapa, largura_mapa))

# Preenche o mapa aéreo com os obstáculos significativos
for obstaculo in obstaculos_filtrados:
    inicio_x, fim_x, altura_y = obstáculo
    altura_y = int(altura_y)
    inicio_x = max(0, inicio_x)
    fim_x = min(largura_mapa - 1, fim_x)
    if altura_y >= altura_mapa:
        altura_y = altura_mapa - 1
    for x in range(inicio_x, fim_x + 1):
        mapa_aereo[altura_y, x] = 1

plt.figure()
plt.imshow(mapa_aereo, cmap='gray')
plt.show()

nome_arquivo_saida = 'mapa_aereo.png'
cv2.imwrite(nome_arquivo_saida, mapa_aereo * 255)
print(f"Mapa aéreo salvo como '{nome_arquivo_saida}'")
```

# APÊNDICE F – Implementação do algoritmo A* para predição de rotas

```
195
196 def heuristic(node, goal, img):  # dist ncia euclidiana
197     penalty = 1.0
198     distance = np.sqrt((node[0] - goal[0]) ** 2 + (node[1] - goal[1])
    ** 2)
199     penalty_term = penalty * img[node[0], node[1]] / 255.0
200     return distance + penalty_term
201
202 def is_valid(node, img):  # verifica limites do mapa e se não é um
   obstáculo
203     if (
204         0 <= node[0] < img.shape[0] and
205         0 <= node[1] < img.shape[1] and
206         img[node[0], node[1]] == 0
207     ):
208         return True
209     return False
210
211 def astar(img, start, goal):
212     open_set = PriorityQueue()
213     open_set.put((0, start))
214     came_from = {}
215     g_score = {node: float('inf') for node in np.ndindex(img.shape)}
216     g_score[start] = 0
217     f_score = {node: float('inf') for node in np.ndindex(img.shape)}
218     f_score[start] = heuristic(start, goal, img)
219
220     #directions = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1,
   -1), (1, 0), (1, 1), (-2, -2), (-2, 0), (-2, 2), (0, -2), (0, 2),
   (2, -2), (2, 0), (2, 2)]
221
222     directions = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1,
   -1), (1, 0), (1, 1)]
223
```

```
224
225     while not open_set.empty():
226         _, current = open_set.get()
227
228         if current == goal:
229             path = []
230             while current in came_from:
231                 path.append(current)
232                 current = came_from[current]
233             path.append(start)
234             return path[::-1]
235
236         for direction in directions:
237             neighbor = (current[0] + direction[0], current[1] +
    direction[1])
238
239             if is_valid(neighbor, img):
240                 tentative_g_score = g_score[current] + 1
241                 if tentative_g_score < g_score[neighbor]:
242                     came_from[neighbor] = current
243                     g_score[neighbor] = tentative_g_score
244                     f_score[neighbor] = tentative_g_score + heuristic
    (neighbor, goal, img)
245                     open_set.put((f_score[neighbor], neighbor))
246
247     return None
248
```

# APÊNDICE G – Identificação do objeto vermelho na cena

```
249  image_path = 'uploads/uploaded_image.png'
250
251 imgColor = cv2.imread(image_path)
252
253 hsv_img = cv2.cvtColor(imgColor, cv2.COLOR_BGR2HSV) # Converte a
       imagem em HSV
254
255 hsv_img[:, :, 0] = (hsv_img[:, :, 0] + 20) % 180 # Ajusta o
       componente H (Matiz) em HSV
256
257 # Limites para o vermelho em HSV
258 lower_red = np.array([15, 120, 120])
259 upper_red = np.array([25, 255, 255])
260
261 mask = cv2.inRange(hsv_img, lower_red, upper_red) # Máscara para
       pixels vermelhos
262
263 contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.
       CHAIN_APPROX_SIMPLE) # Contornos na máscara
264
265 center_x, center_y = -1, -1
266
267 if len(contours) > 0:
268     largest_contour = max(contours, key=cv2.contourArea) # Maior
       contorno
269     M = cv2.moments(largest_contour)
270
271     # Coordenadas do centro do objeto vermelho
272     if M["m00"] != 0:
273         center_x = int(M["m10"] / M["m00"])
274         center_y = int(M["m01"] / M["m00"])
275
276 goal_node = (center_y, center_x)
```

## APÊNDICE H – Código para enviar os dados via bluetooth.

```
277  import socket
278
279 def enviar_dados_via_bluetooth(destino, dados):
280     try:
281         sock = socket.socket(socket.AF_BLUETOOTH, socket.SOCK_STREAM,
        socket.BTPROTO_RFCOMM)
282         sock.connect((destino, 1))
283
284         sock.send(dados.encode('utf-8'))
285
286         sock.close()
287         print("Dados enviados com sucesso via Bluetooth.")
288     except Exception as e:
289         print(f"Erro ao enviar dados via Bluetooth: {e}")
290
291 endereco_destino = 'XX:XX:XX:XX:XX:XX'  # Endereço MAC do dispositivo
        Bluetooth
292
293 selected_points = np.array(selected_points)
294
295 if selected_points is not None and len(selected_points) > 0:
296     dados_a_enviar = ",".join([f"{x},{y}" for x, y in selected_points
       ])  # Converte os pontos selecionados em uma string para envio via
        Bluetooth
297
298     enviar_dados_via_bluetooth(endereco_destino, dados_a_enviar)
299 else:
300     print("Nenhum ponto selecionado para enviar via Bluetooth.")
```

# APÊNDICE I – Código para a extração dos dados de rotas para envio via Bluetooth.

```
302  import math
303
304 def calculate_distance(point1, point2, scale):
305     delta_y = point1[0] - point2[0]
306     delta_x = point1[1] - point2[1]
307     distance_cm = math.sqrt(delta_y**2 + delta_x**2) * scale
308     return distance_cm
309
310 def calculate_angle(point1, point2):
311     delta_y = point2[0] - point1[0]
312     delta_x = point2[1] - point1[1]
313     angle_rad = math.atan2(delta_y, delta_x)
314     angle_deg = math.degrees(angle_rad)
315     return -(angle_deg + 90)
316
317 pixels_per_cm = 0.1  # escala real
318
319 previous_point = path[0]
320 previous_angle = None
321 distance_accumulated = 0
322 direction_changes = 0
323 start_point = path[0]
324 direction_distance = 0
325 direction_angle = 0
326 arrayEnvio = []
327
328 print("Caminho:")
329 for i, point in enumerate(path):
330     if i > 0:
331         distance = calculate_distance(previous_point, point,
    pixels_per_cm)
332         angle = calculate_angle(previous_point, point)
333
334         if i == 1:
```

```
335             previous_angle = angle
336 
337         angle_change = previous_angle - angle
338 
339         if angle_change != 0:
340             if i > 1:
341                 print(f"Mudança de direção: início {start_point}, fim
     {previous_point}, "
342                       f"dist ncia {direction_distance:.2f} cm,
     ngulo  {direction_angle:.2f} graus")
343                 direction_angle_rad =  (direction_angle * math.pi)
    /180
344                 arrayEnvio.append(f"{direction_angle_rad:.2f}, {
    direction_distance:.2f}")
345                 lastAngle = direction_angle
346 
347                 start_point = previous_point
348                 direction_distance = 0
349                 direction_angle = 0
350                 direction_changes += 1
351 
352             direction_distance += distance
353             direction_angle = angle
354 
355             previous_point = point
356             previous_angle = angle
357             distance_accumulated += distance
358 
359 print(f"Mudança de direçãookok: início {start_point}, fim {
    previous_point}, "
360       f"dist ncia {direction_distance:.2f} cm,  ngulo  {
    direction_angle:.2f} graus")
361 print(f"Dist ncia total percorrida: {distance_accumulated:.2f} cm")
362 print(f"Total de mudanças de direção: {direction_changes}")
```

```
363 lastAngle = direction_angle - lastAngle
364 lastAngleRad = (lastAngle * math.pi)/180
365 arrayEnvio.append(f"{lastAngleRad:.2f}, {direction_distance:.2f}")
366 print(arrayEnvio)
367
368 """RESULTADOS
369 Caminho:
370 Mudança de direção: início (254, 120), fim (208, 74), dist ncia 6.51
        cm,  ngulo  45.00 graus
371 Mudança de direçãookok: início (208, 74), fim (44, 74), dist ncia
       16.40 cm,  ngulo  -0.00 graus
372 Dist ncia total percorrida: 22.91 cm
373 Total de mudanças de direção: 1
374 ['0.79, 6.51', '-0.79, 16.40']
375 """
```