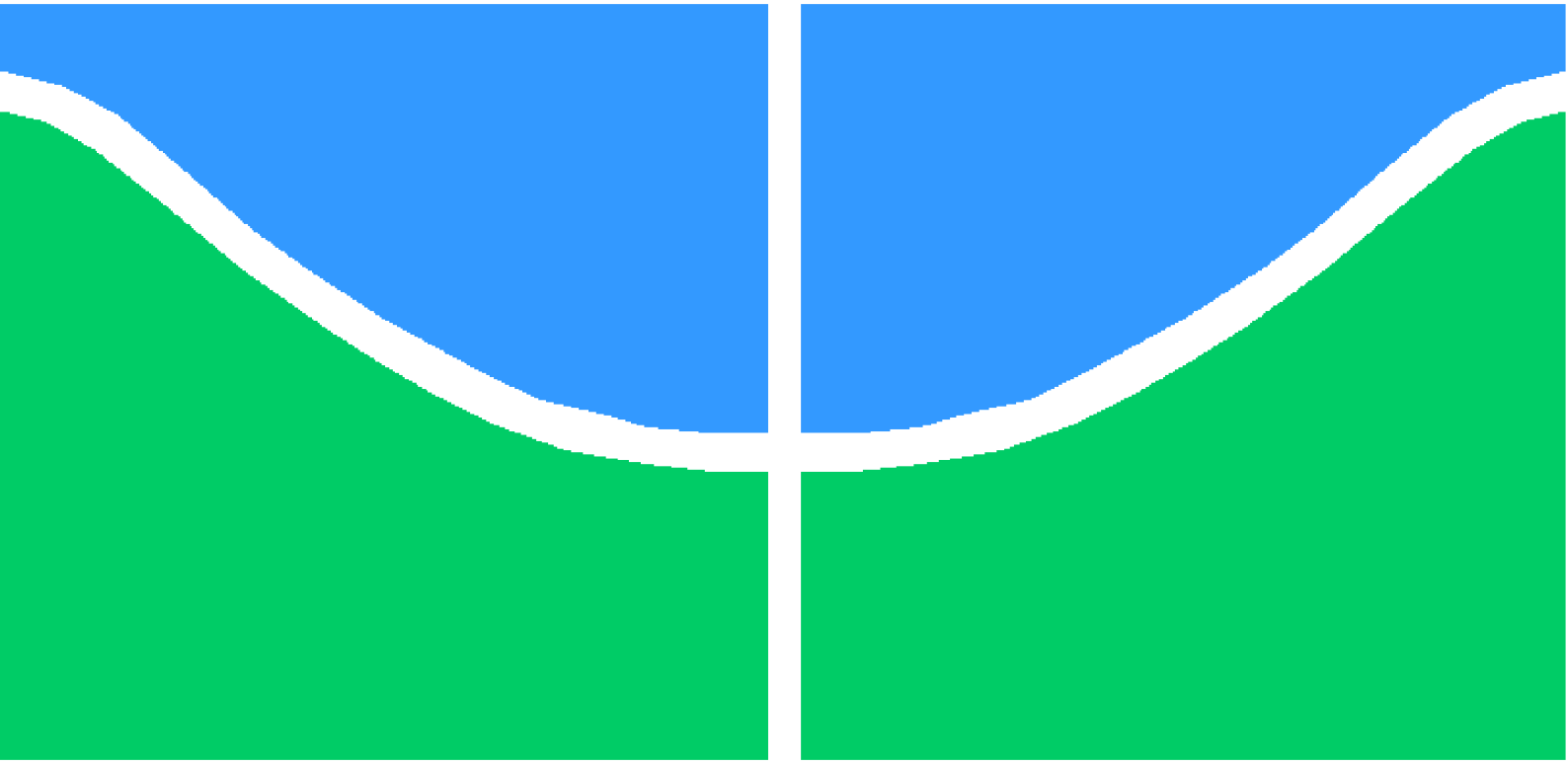


Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Engenharia de Software

Uma Proposta para Detecção de Objetos e Estimação de Distância em um Simulador de Veículos Autônomos

Autor: Guilherme Guy de Andrade

Orientador: Prof. Giovanni Almeida Santos

Brasília, DF

2022

Guilherme Guy de Andrade

Uma Proposta para Detecção de Objetos e Estimação de Distância em um Simulador de Veículos Autônomos

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Universidade de Brasília - UnB

Faculdade UnB Gama - FGA

Orientador: Prof. Giovanni Almeida Santos

Brasília, DF

2022

Guilherme Guy de Andrade

Uma Proposta para Detecção de Objetos e Estimação de Distância em um Simulador de Veículos Autônomos/ Guilherme Guy de Andrade. – Brasília, DF, 2022-

72 p. : il. (algumas color.) ; 30 cm.

Orientador: Prof. Giovanni Almeida Santos

Trabalho de Conclusão de Curso – Universidade de Brasília - UnB
Faculdade UnB Gama - FGA , 2022.

1. Veículos Autônomos. 2. Detecção de Objetos. I. Prof. Giovanni Almeida Santos. II. Universidade de Brasília. III. Faculdade UnB Gama. IV. Uma Proposta para Detecção de Objetos e Estimação de Distância em um Simulador de Veículos Autônomos

CDU 02:141:005.6

Guilherme Guy de Andrade

Uma Proposta para Detecção de Objetos e Estimação de Distância em um Simulador de Veículos Autônomos

Monografia submetida ao curso de graduação em Engenharia de Software da Universidade de Brasília, como requisito parcial para obtenção do Título de Bacharel em Engenharia de Software.

Trabalho aprovado. Brasília, DF, 03 de meio de 2022:

Prof. Giovanni Almeida Santos
Orientador

Prof. Dr. Maurício Serrano
Convidado 1

Profª. Dra. Milene Serrano
Convidado 2

Brasília, DF
2022

Agradecimentos

Agradeço a minha família e meus amigos pelo apoio e ajuda durante esta etapa da minha vida, em especial as pessoas que me ajudaram a me manter motivado e a ir em frente durante os anos da pandemia de COVID-19. Agradeço também ao meu orientador, professor Giovanni, por ter me proporcionado diversas oportunidades e me auxiliado, não só com conhecimentos acadêmicos, mas também com conselhos que levarei para a vida.

Resumo

Veículos autônomos serão ferramentas muito úteis para as cidades do futuro. Ainda há desafios pela frente, mas o investimento nestas linhas de pesquisa tem crescido. Veículos autônomos dependem de sensores para enxergarem o mundo à sua , e também de algoritmos de visão computacional para compreender estes dados. Um dado útil para um veículo autônomo é a informação da distância até outro veículo. Calcular este valor depende de técnicas de detecção de objetos e de estimação de distância. Este trabalho, utilizando simulador CARLA, apresenta o resultado da implementação de um sensor que combina a detecção de objetos do YOLO com cálculos de distância utilizando IPM e dois métodos de cálculo estereoscópico. Os resultados são comparados utilizando o cálculo da raiz do erro médio quadrático em relação ao valor real obtido no simulador. O melhor método teve erro médio de 0,759 metros (erro relativo de 5,77%) e o pior de 4,695 metros (erro relativo de 33,84%).

Palavras-chaves: Veículos Autônomos. CARLA. simulador. Visão computacional. YOLO. Detecção de objetos. Estimação de distância. IPM. Estereoscopia.

Abstract

Autonomous vehicles will be very useful for the cities of the future. There are still challenges ahead, but the investment in autonomous vehicles research has been growing. They use sensors to see the world around them and also computer vision algorithms to understand incoming data. A useful metric for an autonomous vehicle is the distance to another vehicle. To obtain this metric a combination of object detection and distance estimation is used. This work, using the CARLA Simulator, presents the result of the implementation of a sensor that combines YOLO's object detection with distance estimation using IPM and two stereoscopy based methods. The results are compared with the real measurement taken from the simulator and the root mean squared error is calculated. The best method had an error of 0,759 meters (5,77% relative error) and the worst of 4,695 meters (33,84% relative error).

Key-words: Autonomous Vehicles. CARLA. Simulator. Computer vision. YOLO. Object Detection. Distance Estimation. IPM. Stereoscopy.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	*Application Programming Interface*
CARLA	*Car Learning to Act*
GPS	*Global Positioning System* (Sistema de Posicionamento Global)
IPM	*Inverse Perspective Mapping* (Mapeamento Inverso de Perspectiva)
LiDAR	*Light Detection And Ranging*
OpenCV	*Open Source Computer Vision Library*
RADAR	*Radio Detection and Ranging*
RMSE	*Root Mean Square Error* (Erro Médio Quadrático)
TCC	Trabalho de Conclusão de Curso
UnB	Universidade de Brasília
YOLO	*You Only Look Once*

Sumário

1 Introdução

Faisal et al. (2019), em sua revisão de literatura, conclui que os veículos autônomos são uma ferramenta muito útil para as cidades do futuro, e possuem o potencial de trazer melhorias e inteligência à mobilidade urbana, em um mundo que está cada vez mais urbanizado. Também cita que apenas um número limitado de estudos conseguem observar o efeito disruptivo que a adoção de veículos autônomos pode causar à sociedade, seja no comportamento das pessoas ou no *design* das cidades. Rojas-Rueda et al. (2020) analisa os impactos, positivos e negativos, da adoção em larga escala dessa tecnologia em relação à saúde pública, sendo que, um dos impactos diretos mais notáveis ocorre na segurança do trânsito. Estima-se que 90% dos acidentes de trânsito nos Estados Unidos são causados por falhas humanas (NHTSA, 2015), e os acidentes de trânsito são a 8.º maior causa de morte para pessoas de todas as idades e a maior causa de morte para a faixa de 5 à 29 anos (WHO, 2018). A adoção de veículos autônomos pode evitar, pelo menos, 585.000 mortes em um período de 10 anos a partir de 2035, quando se estima que haverá uma efetiva utilização de veículos autônomos (LANCTOT, 2017). Entretanto, para chegar em um maior nível de automação, existem diversos desafios em áreas como inteligência artificial, processamento de imagens e sinais, tecnologias de comunicação e outros.

Tecnologias relacionadas a veículos autônomos e suas aplicações têm ganhado destaque ao longo do tempo. O que era um sonho distante, que apenas ocorria em obras de ficção, tem se tornado cada vez mais próximo do real. De acordo com Hopkins e Schwanen (2021), o interesse pela área de veículos autônomos tem aumentado nos últimos anos, o que gera um aumento das pesquisas e novas descobertas na área. Um desafio para a pesquisa de veículos autônomos em ambientes urbanos é a execução de testes, devido aos custos de infraestrutura e dificuldades logísticas para treinar e testar estes sistemas no mundo real (DOSOVITSKIY et al., 2017).

Para Wachenfeld e Winner (2016), substituir motoristas humanos por veículos autônomos deve ser feito com atenção especial aos requisitos de segurança, para que o uso de automação não reduza a segurança no trânsito. Para que isso se realize, múltiplas validações e mecanismos de segurança são aplicados, dependendo do nível de automação presente no veículo. Estes elementos podem ser garantias mínimas quanto a taxas de falha dos componentes ou a presença de mecanismos de *fallback* para que o motorista assuma o controle do veículo em casos específicos. Além disso, uma métrica importante para o teste das funções de um veículo autônomo é a análise de seu comportamento em diversos cenários, o que pode ser obtido ao submeter o veículo a diversos testes, chegando na casa de milhões de quilômetros rodados.

Wachenfeld e Winner (2016) também observa que, uma etapa que auxilia no processo de desenvolvimento de tecnologias de automação é que os testes sejam feitos, desde a concepção do produto, de forma virtual, e por meio de simulações. Testar em ambientes virtuais não remove a necessidade de testes no mundo real, mas pode auxiliar na evolução do produto e na melhora de sua segurança em um primeiro momento do ciclo de vida do produto. As simulações de computador para veículos autônomos também possuem um papel de democratizar o acesso a ferramentas de desenvolvimento para novas tecnologias. Algumas áreas na pesquisa em veículos autônomos necessitam de um investimento financeiro que pode inviabilizar a participação de novos indivíduos.

Visão computacional é um conceito de extrema importância para veículos autônomos, permitindo que os sistemas destes veículos possam obter informações sobre o mundo real utilizando algoritmos e sensores para interpretar estes dados. Algumas das áreas que este conceito abrange são: detecção de objetos, visão estéreo, segmentação semântica, reconstrução de cenas tridimensionais, entre outros. Um dos fatores atribuídos por Janai et al. (2020) para que, apesar dos avanços tecnológicos, os veículos autônomos ainda não sejam amplamente adotados é a precisão da percepção do mundo, que, em muitos modelos de visão computacional, ainda é inferior ao nível de precisão da visão humana. Em uma simulação, é possível testar e desenvolver algoritmos que melhorem a capacidade de percepção do veículo antes de implementar em um veículo real.

1.1 Objetivos

O objetivo geral deste trabalho é propor uma estrutura básica para a detecção de objetos e estimação de localização de veículos no simulador CARLA. Além disso, possui por objetivos específicos:

1. Explorar o funcionamento e as aplicações do simulador CARLA;
2. Integrar o algoritmo YOLO ao simulador CARLA para detecção de objetos no formato de um sensor no veículo virtual;
3. Implementar algoritmos de detecção de distância baseados em câmeras;
4. Implementar prova de conceito que demonstre o funcionamento da detecção de objetos em conjunto com estimação de distância.

1.2 Organização do Trabalho

O Capítulo 1 contém disposições gerais sobre o trabalho. O Capítulo 2 aborda o referencial teórico deste trabalho. comentando sobre o simulador CARLA, a rede neural

YOLO, e métodos para a estimação de distância baseados em imagens. O Capítulo 3 discorre sobre as etapas de desenvolvimento para o trabalho. O Capítulo 4 mostra o que foi desenvolvido e os dados obtidos. Por fim, o Capítulo 5 conclui o trabalho, e relaciona os resultados aos objetivos.

2 Referencial Teórico

Este capítulo aborda alguns aspectos gerais sobre veículos autônomos, a utilização de algoritmos de detecção de objetos, estimação de distância e também sobre simuladores de veículos autônomos.

2.1 Aspectos Gerais

SAE (2021) classifica 6 níveis de automação em veículos autônomos. Estas classificações são realizadas com base na capacidade do veículo de realizar tarefas de direção dinâmicas, definidas como todas as funções operacionais e táticas necessárias para operar um veículo na pista, com a exclusão de tarefas de planejamento estratégico. Algumas das tarefas de direção dinâmicas são: controle lateral e longitudinal do veículo, monitoramento do ambiente, planejamento e execução de resposta a mudanças do ambiente, planejamento de manobras, e execução de ações que melhorem a visibilidade do veículo para outros motoristas por meio da utilização de farol, buzinas e outros métodos. Conforme o veículo autônomo consegue realizar mais tarefas de direção dinâmica, maior é sua classificação de automação. Outro fator que afeta a classificação é o quanto se espera que um motorista humano esteja disponível e atento para corrigir a execução de uma tarefa de direção dinâmica. A Tabela 2.1 traz, em tradução livre, uma adaptação do que é proposto nestes níveis de automação.

Segundo Kocić, Jovičić e Drndarević (2018), sensores são componentes-chave para veículos autônomos. Eles são responsáveis por coletar dados sobre o mundo, que serão analisados pelo veículo. Muitas vezes, os dados destes sensores passam por um processo de fusão para criar dados que possuam maior grau de certeza. Um exemplo deste processo de fusão é Meng, Wang e Liu (2017), que realizou a fusão dos dados de sensores para melhorar a precisão da localização global do veículo, devido às imprecisões de obter este dado apenas através de um único sensor. Os veículos são equipados com múltiplos e diversos sensores para melhorar a robustez dos dados obtidos (JANAI et al., 2020).

Varghese e Boone (2015) separa os sensores em internos ao veículo e sensores para o mundo externo. Sensores internos podem medir informações como a rotação das rodas, inclinação do veículo, velocidade e outros. Já sensores para o mundo externo lidam com a percepção de diferentes aspectos do mundo. Sensor de GPS, que realiza o mapeamento global a partir da triangulação do sinal de satélites para obter a posição absoluta do veículo, é um sensor de mundo externo, assim como os sensores de RADAR, LiDAR e câmeras.

Tabela 2.1 – Níveis de automação de direção

Nível	Nome	Descrição
0	*Nenhuma automação*	O veiculo é controlado completamente pelo motorista, mesmo quando sistemas de segurança ativa forem utilizados
1	*Assistência ao motorista*	Execução de ação específica pelo sistema de automação que exerça ou controle lateral ou controle longitudinal do veículo contando com o motorista faça o controle da outra direção
2	*Automação parcial da direção*	Execução, pelo sistema de automação, do controle lateral e longitudinal do veículo com a expectativa de que o motorista faça apenas a etapa final da ação ou fique como supervisor das ações do veículo
3	*Automação condicional da direção*	Execução automatizada da completude de uma ação com controle lateral e longitudinal do veículo, ativada pelo motorista com a expectativa de que o motorista seja apenas supervisor e esteja pronto para assumir o comando em caso de erro, risco à segurança ou quando solicitado
4	*Automação da direção*	Execução completamente automatizada do veiculo, ativada pelo motorista, sem a expectativa de que este assuma o controle em algum momento mas permitindo a intervenção
5	*Automação completa da direção*	Execução completamente automatizada do veiculo a todos os momentos, sem a expectativa de que o motorista assuma o controle em algum momento

Fonte: Adaptada de SAE (2021)

Sensores do tipo RADAR utilizam ondas eletromagnéticas para detectar a distância e a velocidade de um objeto. Uma vantagem específica do RADAR é que ele pode detectar a velocidade de um objeto de forma direta, utilizando o efeito Doppler. Radares que operam em alta frequência podem ter um alcance de 200 metros, mas um campo de visão limitado (KOCIĆ; JOVIČIĆ; DRNDAREVIĆ, 2018). O sensor LiDAR utiliza o tempo que um feixe de luz leva para rebater em algum objeto e retornar ao sensor para obter a distância deste ponto. O LiDAR possui uma precisão maior que o RADAR, porém, um custo mais alto (JANAI et al., 2020).

Câmeras também são muito utilizadas, principalmente, por possuírem um preço mais acessível que outros sensores, permitindo uma visualização do ambiente ao seu redor, e fornecendo informações visuais que os outros sensores não possuem. Entretanto, trazem desafios, pois o poder computacional necessário para processar as imagens geradas é grande, sendo necessário processar milhões de pixels para cada imagem, em múltiplas imagens por segundo (KOCIĆ; JOVIČIĆ; DRNDAREVIĆ, 2018). Duas câmeras podem ser colocadas lado a lado, com uma distância conhecida, para permitir a recuperação de alguns dados da profundidade e distância de objetos, além de outros métodos que serão discutidos na Seção 2.4.

2.2 Simulação de Veículos Autônomos

Ao desenvolver algoritmos e soluções para veículos autônomos, surge a necessidade de executar testes para garantir a eficiência, segurança e qualidade destas soluções (WACHENFELD; WINNER, 2016). Entretanto, realizar estes testes no mundo real pode ser muito difícil devido ao custo ou à segurança dos envolvidos, principalmente, em etapas iniciais de desenvolvimento, ou devido ao teste que deve ser feito. Veículos autônomos necessitam de passar por uma abundância de situações antes de serem disponibilizados para uso geral. Além disso, testes com veículos autônomos costumam possuir grande complexidade, já que o sistema deste veículo deve conseguir responder a múltiplos agentes e seguir um conjunto complexo de regras de trânsito.

Devido a esses desafios, uma proposta é a utilização de simulações de computador para que, em um primeiro momento, se possa validar uma ideia ou um algoritmo, antes de se gastar recursos com testes de mundo real. Utilizar uma simulação auxilia na democratização do acesso às ferramentas de pesquisa.

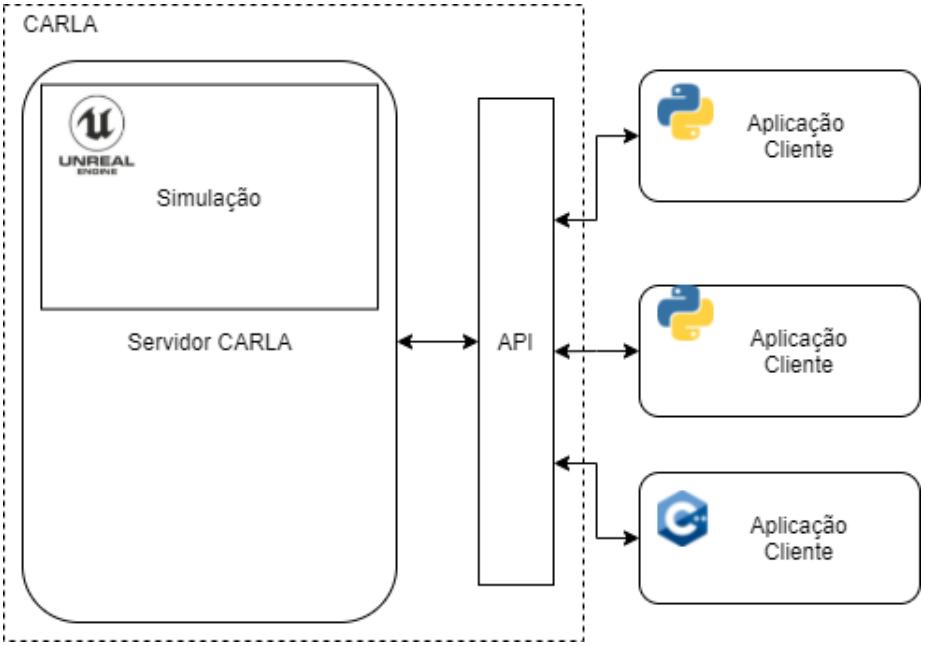
Com o objetivo de criar uma ferramenta de acesso aberto, que seja capaz de auxiliar no desenvolvimento de tecnologias para veículos autônomos, existe o simulador CARLA (DOSOVITSKIY et al., 2017), o qual é uma plataforma aberta, flexível e gratuita, que inclui um conjunto de ferramentas como: simulação de física, de tráfego, modelos de objetos virtuais, simulação de clima e outros. O acesso a estas ferramentas é disponibilizado por uma API que simplifica a interação de um código com a simulação.

A Figura 2.1 apresenta a arquitetura do simulador CARLA e como outras aplicações interagem com ele. Sua estrutura é de uma aplicação cliente-servidor. O servidor CARLA é responsável por executar o processamento visual e de física da simulação, a aplicação do servidor é criada com a *Unreal Engine 4* (Epic Games, 2019), um motor gráfico e de física para jogos, utilizado comercialmente. Já as aplicações cliente conectam-se, e podem enviar comandos para controlar essa simulação por uma API.

Conforme visto na Seção 2.1, sensores são uma parte essencial para um veículo autônomo. O CARLA permite a criação de sensores virtuais para simular o comportamento de sensores que estariam presentes em um veículo autônomo. É possível criar sensores baseados em imagens, gerados com diferentes e customizáveis tipos de câmeras. Adicionalmente, é possível utilizar sensores como LiDAR e de posicionamento. Pode-se ainda implementar um modelo de sensor customizado por meio das APIs disponíveis.

Deve-se notar que os sensores virtuais, muitas vezes, retornam dados perfeitos. Métodos para adicionar ruído e falhas a estes dados podem ser aplicados para aumentar seu realismo. Entretanto, uma recriação exata do que um sensor real replicaria é uma tarefa muito árdua, fazendo com que o nível de realismo desejado tenha que ser escolhido considerando a aplicação sendo desenvolvida (ELMQUIST; NEGRUT, 2020).

Figura 2.1 – Visão geral da arquitetura para o servidor CARLA



Fonte: Elaborada pelo autor.

A utilidade do CARLA como uma fonte de dados é evidenciada pelo trabalho de Dworak et al. (2019), que utilizou o simulador para treinar uma rede neural para reconhecimento de objetos em nuvens de pontos, que foram coletados por um sensor LiDAR virtual. Ao realizar o treinamento supervisionado de um algoritmo de aprendizado de máquina, a qualidade dos dados iniciais e sua separação em categorias afeta a qualidade final do modelo. Nesse trabalho, por meio do CARLA e dos objetos virtuais presentes no simulador, os autores conseguiram extrair a categorização real de cada objeto, facilitando a geração de dados para inserir no aprendizado de máquina. Obtiveram sucesso com a utilização de uma base de dados híbrida, misturando dados reais e dados do simulador.

É possível popular o mundo da simulação com outros agentes, como carros, caminhões, motocicletas ou pedestres. Isso permite criar um ambiente de trânsito com grande diversidade, e observar como um agente de um veículo autônomo se comportaria em um ambiente de trânsito com outros agentes, além de programar as ações destes outros agentes, para formar um cenário, que permite que uma determinada situação seja investigada de diversas formas, ou que, um agente de aprendizado de máquina possa ser treinado para resolver um tipo específico de problema. Um cenário pode ser um evento fictício ou baseado em algo que ocorreu no mundo real. Devido à importância dos cenários para o desenvolvimento de veículos autônomos, e na busca de criar cenários mais realistas, Osiński et al. (2021) cria uma coleção de mais de 60 mil cenários baseados em dados extraídos do mundo real.

Por ser um projeto de código aberto, há uma comunidade de desenvolvedores au-

xiliando no desenvolvimento e na criação de documentação e materiais de apoio. O repositório oficial do CARLA conta com uma coleção de exemplos que servem para demonstrar como utilizar a API, e também como ponto de partida para novas criações, possibilitando que desenvolvedores novos não precisem criar código para acessar funcionalidades comuns, como salvar dados em disco, ou desenhar na tela a saída de uma câmera virtual.

Não há uma linguagem de programação específica para a escrita destes programas, desde que consigam utilizar a API do simulador. O CARLA possui exemplos em *C++* e *Python*, sendo a maioria nesta segunda linguagem. Os exemplos demonstram muitas das funcionalidades disponíveis no CARLA, e fornecem uma estrutura básica para a criação de novo código e novas integrações. Geralmente, quando se exibe a aplicação cliente na tela, nos exemplos em *Python*, é utilizada a biblioteca *PyGame* (SHINNERS, 2011). A Figura 2.2 exibe a interface padrão para alguns dos exemplos criados com a *PyGame*. As imagens são geradas por meio dos sensores na simulação, e convertidas para formatos compatíveis com esta biblioteca.

Figura 2.2 – Execução de um cliente do CARLA em um dos exemplos de código

Fonte: Elaborada pelo autor.

2.3 Detecção de Objetos Utilizando Redes Neurais

Em 2016, Redmon et al. (2016) propôs a arquitetura de rede neural *You Only Look Once* (YOLO) para reconhecimento e classificação de objetos em imagens. O foco desta arquitetura é a velocidade de detecção, sendo capaz de fornecer reconhecimento em

tempo real. Em um dos testes realizados pelos autores, o YOLO, na sua primeira versão, era o único algoritmo de detecção capaz de executar em tempo real.

Alguns algoritmos de detecção e classificação de objetos, primeiramente, encontram áreas de interesse e, em outro passo, fazem a classificação destas áreas em objetos. Muitas vezes existem passos intermediários, como eliminar detecções duplicadas ou refinar a área de interesse. Estes passos podem ser complexos e computacionalmente caros de se executar. O YOLO propõe uma mudança na forma de se lidar com este problema, pensando na detecção de objetos como um único problema de regressão, unificando a detecção de área de interesse e a classificação de objetos (REDMON et al., 2016). Dessa forma, diz-se que o algoritmo tenta olhar a imagem apenas uma vez.

O processo de detecção do YOLO pode ser dividido em três etapas. A primeira é redimensionar a imagem para o tamanho esperado pela rede neural. Depois disso, é utilizada uma rede neural de convolução que observa a imagem e gera áreas de interesse na imagem e suas classificações. Por fim, é feito um filtro removendo os valores não máximos dos resultados obtidos. Este processo pode ser visto na Figura 2.3. Uma das diferenças-chave do YOLO para outros algoritmos encontra-se na rede neural de convolução que é aplicada.

O passo de se encontrar áreas de interesse e realizar a classificação é unificado, sendo que, quando uma área de interesse é encontrada, já é obtida sua classificação. Ao combinar a detecção de áreas de interesse e classificação, o YOLO consegue ser um algoritmo muito rápido, mesmo que comprometa um pouco sua acurácia.

Figura 2.3 – Funcionamento do YOLO

Fonte: Adaptado/Tradução livre de Redmon et al. (2016)

Outra vantagem em relação a outros algoritmos da época, que também é permitida por este processo, é que, ao realizar a classificação, consideram-se os dados globais da imagem, fazendo com que o YOLO consiga diferenciar o que é o plano de fundo e o que é o objeto.

A Figura 2.4 demonstra uma forma de se interpretar a classificação feita pelo YOLO, cercando os objetos com os valores das bordas das áreas de interesse e categorizando com o resultado provido pelo YOLO.

Até o presente momento, expôs-se sobre a primeira versão (REDMON et al., 2016)

Figura 2.4 – Exemplo de classificação feito pelo Yolo em uma imagem

Fonte: Elaborada pelo autor.

da arquitetura YOLO. O YOLO possui mais duas versões. Uma desvantagem do YOLO é que, quanto mais classes ele tem para classificar objetos, mais lento ele se torna. Melhorias foram alcançadas com a sua segunda versão (REDMON; FARHADI, 2016), a qual trouxe a possibilidade de um aumento na quantidade de classes disponíveis para classificação, e também melhorias nos resultados obtidos da rede neural. Por meio dessas melhorias na arquitetura do YOLOv2, foi possível criar um modelo capaz de classificar objetos em uma de 9000 classes de objetos possíveis. Além disso, para a época, o YOLOv2 continua como uma das alternativas mais rápidas para a detecção de objetos. A terceira versão do YOLO (REDMON; FARHADI, 2018) continua a tendência de fornecer detecção de objetos de forma rápida, quando comparado a outros algoritmos da época. A terceira versão aprimorou a precisão da rede neural, com melhorias na detecção de áreas de interesse e na detecção de objetos pequenos. Bochkovskiy, Wang e Liao (2020) com a quarta, e mais recente versão, YOLOv4, trouxe mais otimizações para a arquitetura da rede neural, melhorando o desempenho e a precisão dos resultados, incluindo mais melhorias para a detecção de objetos pequenos.

A integração de visão computacional e simuladores pode ser vista no trabalho de Valeja et al. (2021), onde o YOLOv3 é utilizado em uma simulação de veículos autônomos para desenvolver um mecanismo de controle de velocidade do veículo que observa a velocidade escrita nas placas de trânsito, alterando a velocidade máxima do veículo na simulação. Já Juanola (2019) fez a detecção de sinais de trânsito e semáforos para enviar comandos para o veículo autônomo, conforme o que era detectado, para realizar a frenagem ou o controle de velocidade. Além disso, fez uma otimização nas categorias utilizadas para a classificação do YOLOv3 para acelerar a detecção e limitar os resultados possíveis. Isso foi possível porque foi gerado um novo modelo para a rede neural do YOLO, contando

com imagens geradas no simulador.

Os métodos de detecção baseados em redes neurais não são perfeitos, possuem uma taxa de erro nos modelos e são suscetíveis a ataques direcionados, como o trabalho de Wu et al. (2020), que conseguiu desenvolver um método para elaborar uma camuflagem para os veículos, de forma que o veículo não seja detectado ou apresente detecção imprecisa quando analisado por redes neurais.

2.4 Estimação de Distância

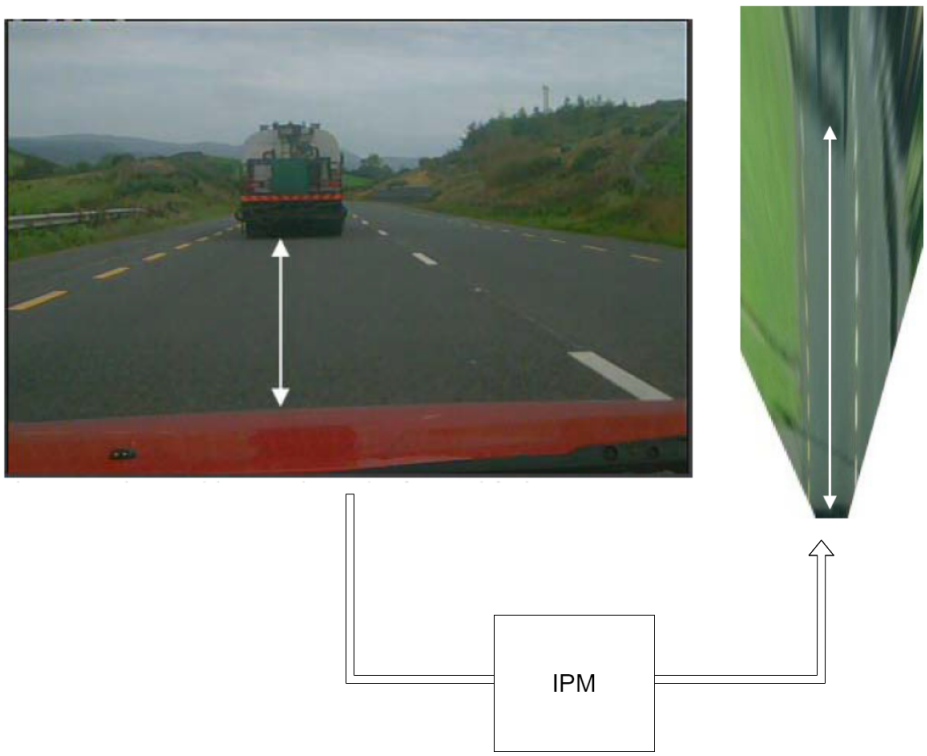
Ao se capturar uma imagem por uma câmera, o resultado é influenciado pelas características desta câmera. Alguns fatores como a distância focal, o tamanho do sensor e o tipo de lente afetam, por exemplo, o campo de visão aparente na imagem. Quando uma fotografia é gerada, é feito o processo de conversão de um espaço tridimensional, no mundo real, para um espaço de duas dimensões, na imagem. Ao fazer isso, perdemos os dados da distância dos objetos. Entretanto, existem formas de se recuperar, de forma aproximada, esses dados. Em visão computacional, a criação de imagens segue princípios parecidos com a de câmeras reais, tentando criar uma transformação que converta o espaço tridimensional com objetos para a tela do computador. Essa pode ser feita por meio de uma matriz de projeção (BLOOMENTHAL; ROKNE, 1994). Ela utiliza coordenadas homogêneas para possibilitar a aplicação de operações de translação, rotação e escala. Com coordenadas homogêneas é possível definir, conforme a Equação 2.1, um ponto que pode ser expressado no sistema homogêneo como (x', y', z', w), de forma que $x = x'/w$, $y = y'/w$ e $z = z'/w$, sendo w um valor constante.

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = P \begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} \tag{2.1}$$

Na Equação 2.1, é possível observar a matriz $4x4$ de projeção P. Essa matriz define quais transformações serão feitas aos pontos. Um conjunto de operações que pode ser realizado com coordenadas homogêneas é o mapeamento inverso de perspectiva (IPM). Esta operação consiste em tentar reverter o processo que uma câmera faz ao capturar a imagem e recuperar alguma informação de profundidade. Tanveer e Sgorbissa (2018) utilizou esta técnica para mapear o solo em frente a um robô, Muad et al. (2004) para mapear as faixas na pista em frente a um veículo.

Aplicar IPM resulta em uma imagem em que a perspectiva represente uma visão de cima da cena. Porém, quanto mais distante do centro óptico da imagem, mais distorcida

Figura 2.5 – Exemplo de resultado da aplicação de IPM



Fonte: Adaptado de Tuohy et al. (2010)

se torna a imagem, como pode ser observado na Figura 2.5. Para elaborar a matriz P, que aplica o IPM, são necessárias algumas informações.

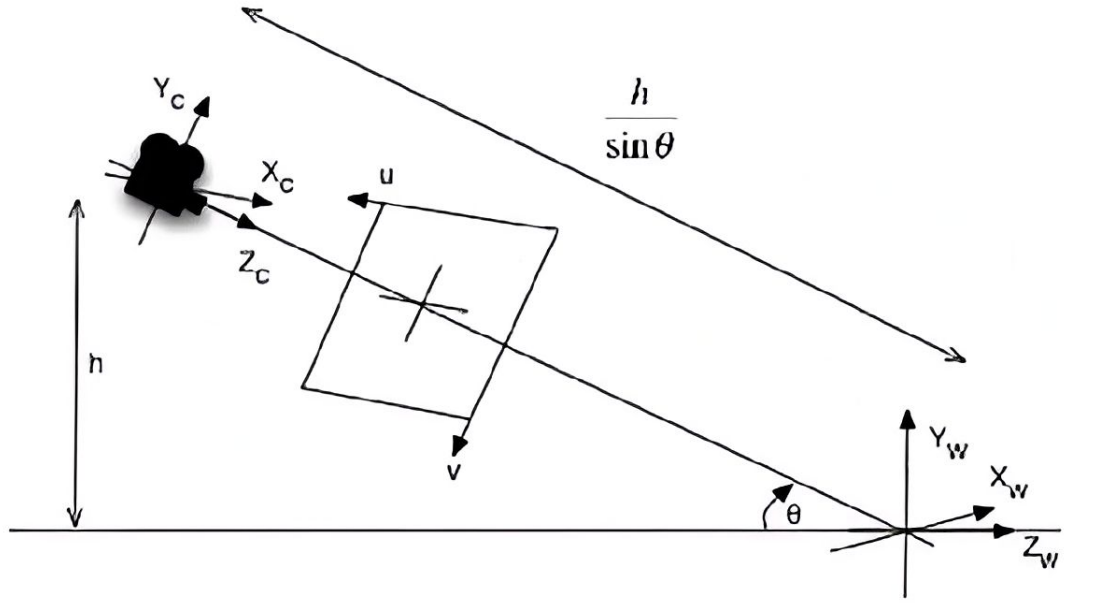
$$K = \begin{bmatrix} f & s & u0 & 0 \\ 0 & f & v0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{2.2}$$

A matriz K, definida na Equação 2.2, contém os parâmetros da câmera. Estes parâmetros são: f, a distância focal. E $u0$, $v0$, que representam as coordenadas do centro óptico da imagem. O parâmetro s é a viés dos pixeis da câmera.

$$T = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -h/sen(\theta) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

$$R = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta) & -sen(\theta) & 0 \\ 0 & sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

Figura 2.6 – Diagramação da conversão do sistema de coordenadas do IPM



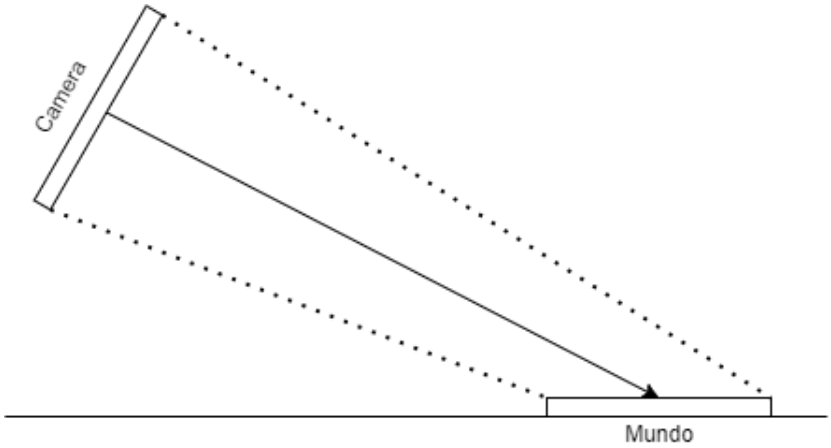
Fonte: Adaptado de Tuohy et al. (2010)

A Figura 2.6 demonstra o objetivo da matriz P, converter o plano da imagem definido por (u, v) que está alinhado com a câmera de centro (Yc, Xc, Zc) para o plano da imagem com centro em (Yw, Xw, Zw). Para isso, é necessário fazer uma translação das coordenadas pela distância entre a câmera e o ponto, definida pela Equação 2.3, e também a rotação das coordenadas segundo o ângulo, definido na Equação 2.4. Pode-se combinar as matrizes T e R por meio de multiplicação, para formar a matriz de parâmetros extrínsecos à câmera, e a matriz K representa os parâmetros intrínsecos da câmera. A aplicação dos parâmetros extrínsecos resulta no que é exibido na Figura 2.7. A matriz K faz a conversão de coordenadas do plano da imagem para coordenadas da câmera. Já a matriz extrínseca, TR, é responsável por converter as coordenadas a partir da câmera para coordenadas no mundo. Ao combinar estas transformações, obtêm-se uma matriz que converte um ponto na imagem para um ponto no mundo.

$$P = K * T * R \tag{2.5}$$

A matriz P pode ser definida pela multiplicação da matriz intrínseca pela matriz extrínseca, conforme a Equação 2.5. Expandindo os valores de P na Equação 2.1, obtêm-se a Equação 2.6. Caso seja considerado que a imagem será convertida para um plano, pode-se fixar o valor da variável y na Equação 2.6 como 0, o que resulta na Equação 2.7. Essa conversão transforma um ponto da imagem para um ponto no plano do mundo.

Figura 2.7 – Resultado da aplicação da matriz extrínseca



Fonte: Elaborada pelo autor.

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} P11 & P12 & P13 & P14 \\ P21 & P22 & P23 & P24 \\ P31 & P32 & P33 & P34 \end{bmatrix} \begin{bmatrix} x' \\ y' \\ z' \\ w \end{bmatrix} \tag{2.6}$$

$$\begin{bmatrix} x \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} P11 & P13 & P14 \\ P21 & P23 & P24 \\ P31 & P33 & P34 \end{bmatrix} \begin{bmatrix} x' \\ z' \\ w \end{bmatrix} \tag{2.7}$$

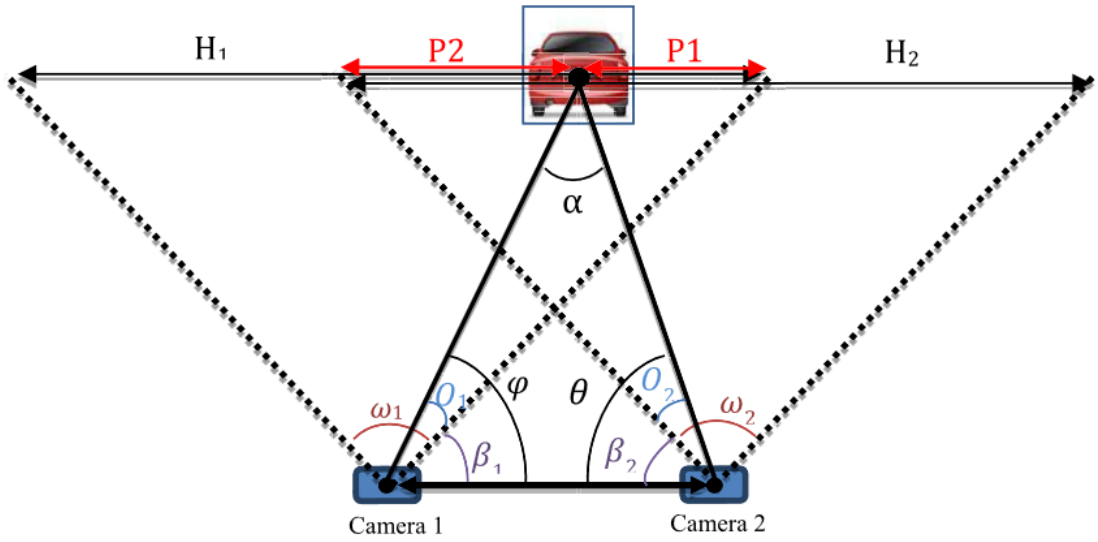
Este método foi utilizado por Praciano (2020) em conjunto com um sistema multi-câmera para fazer o cálculo de distância entre as câmeras e um veículo. Uma das limitações deste método é quando se possui um plano inclinado. Para isso, Nieto et al. (2007) utilizou o ângulo da pista e outros parâmetros para modificar o IPM. Outra distorção que pode ocorrer é a gerada pelo movimento de um veículo, a qual o método de Jeong e Kim (2016) tenta resolver.

O cálculo de distância pode ser feito de outras formas, como no proposto por Vajgl, Hurtik e Nejezchleba (2022), que faz adaptações à estrutura da rede neural YOLO, e treina um novo modelo, baseado nos dados de Geiger et al. (2013), de forma a realizar a previsão da distância junto da classificação objeto. Esta implementação não apresentou perda de desempenho significativo em relação à implementação original do YOLO, sendo revelado erro médio relativo de 11% e um erro médio absoluto de 2,5 metros.

Até o momento, as técnicas apresentadas consideram apenas uma câmera, mas, em sistemas de duas câmeras, é possível utilizar a estereoscopia. Caso se saiba a distância entre duas câmeras idênticas, que estejam alinhadas, é possível calcular a distância de um ponto baseado na imagem resultante. Usualmente, o sensor com duas câmeras é chamado

câmera estéreo. A geometria epipolar faz a relação entre duas imagens que observam a mesma cena. Zhang (1998) fez a revisão de diversos métodos para determinar a relação geométrica de duas imagens. Já Zaarane et al. (2020), utilizando conceitos de geometria epipolar, desenvolveu um método para o cálculo de distância até um veículo utilizando duas câmeras. A Figura 2.8 demonstra os ângulos e distâncias envolvidas no cálculo da Equação 2.8, onde h é a distância até o objeto.

Figura 2.8 – Diagramação do cálculo de distância com cãmera stereo



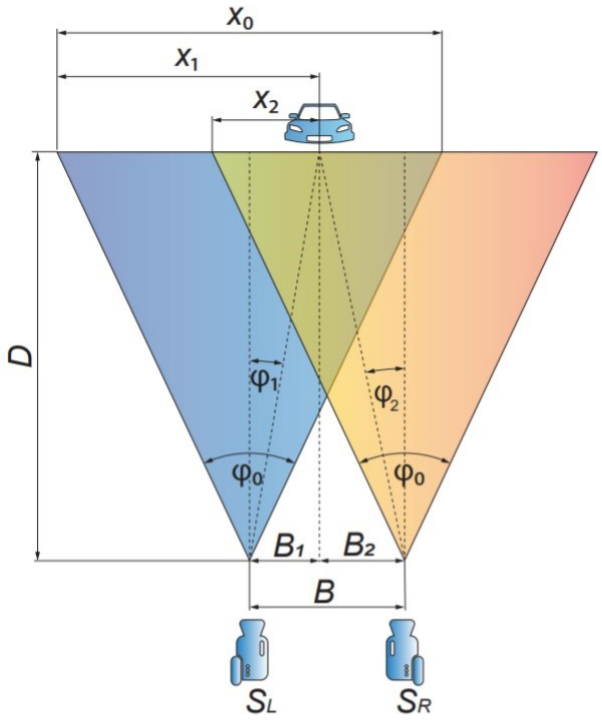
Fonte: Zaarane et al. (2020)

$$h = \frac{Asin(P2.\frac{\omega_2}{H_2} + \beta_2)sin(P1.\frac{\omega_1}{H_1} + \beta_1))}{sin(180 - (P2.\frac{\omega_2}{H_2} + \beta_2 + P1.\frac{\omega_1}{H_1} + \beta_1))} \tag{2.8}$$

Uma etapa importante para este método é identificar o mesmo objeto nas duas imagens. Zaarane et al. (2020) utiliza o algoritmo de Zaarane et al. (2019) para detectar o que é um veículo em uma imagem, e depois é feita uma comparação para achar a área similar na segunda imagem. A Figura 2.9, em conjunto com a Equação 2.9, demonstra outra forma de fazer este cálculo, pelo método proposto por Strbac et al. (2020), que utiliza a rede neural YOLO para fazer detecções de objetos em ambas imagens, e depois um algoritmo para descobrir quais das detecções são relacionadas. Como demonstrado por Bertozz, Broggi e Fascioli (1998), é possível adaptar o método de IPM para utilizar a visão estéreo e detectar obstáculos no caminho de um veículo. Neste método são utilizados procedimentos de manipulação de imagem para detectar uma área de interesse, ao invés de uma rede neural.

$$D = \frac{B * X_0}{2 * tan(\frac{\varphi_0}{2}) * (X_1 - X_2)} \tag{2.9}$$

Figura 2.9 – Diagramação para o cálculo da distância a partir de câmera estéreo



Fonte: Strbac et al. (2020)

3 Metodologia

Para Gil et al. (2002), a metodologia é "um conjunto de procedimentos a serem seguidos na realização da pesquisa". Há a necessidade de definir alguns dos aspectos gerais que caracterizam esta pesquisa, sendo esses abordados neste capítulo. Na Seção 3.1, serão definidos os aspectos desta pesquisa; na Seção 3.2, serão definidos os procedimentos a serem seguidos na realização da pesquisa, e na Seção 3.3 será abordado como os dados provenientes da pesquisa serão coletados.

3.1 Classificação da Pesquisa

Prodanov e Freitas (2013) definem pesquisa como "um conjunto de ações propostas para encontrar a solução para um problema, que tem por base procedimentos racionais e sistemáticos". Gil et al. (2002) também definem pesquisa como um procedimento racional e sistemático, cujo objetivo é encontrar soluções para o problema proposto.

Os autores propõem diversas formas de classificação de pesquisa, tendo em vista diferentes pontos de vista como da abordagem, da natureza, dos objetivos ou de seus procedimentos. A classificação desta pesquisa é resumida na Tabela 3.1.

3.1.1 Abordagem da Pesquisa

Para Moresi et al. (2003), a abordagem da pesquisa pode ser quantitativa ou qualitativa. Uma pesquisa quantitativa busca analisar o que for possível sobre um assunto para buscar uma quantificação sobre aspectos do assunto abordado. Já uma pesquisa qualitativa deseja entender um assunto por meio da interpretação de dados e informações relacionadas a ele.

Este trabalho propõe uma abordagem mista de pesquisa quantitativa e qualitativa, pois busca medir o nível de precisão das técnicas aplicadas ao problema proposto, e também uma análise sobre as razões deste resultado.

3.1.2 Natureza da Pesquisa

Definir a natureza da pesquisa tem relação com o escopo de seus resultados e conhecimentos obtidos. O foco de uma pesquisa básica é universal, e o de uma pesquisa aplicada é local (MORESI et al., 2003). Isso ocorre porque uma pesquisa aplicada é voltada para um problema específico, e uma pesquisa básica não. Ambas produzem conhecimentos válidos, mas com escopos diferentes.

A pesquisa deste trabalho pode ser considerada aplicada, pois busca aplicar uma coleção de conhecimentos em um conjunto de objetivos práticos.

3.1.3 Objetivos da Pesquisa

Prodanov e Freitas (2013) definem três grandes classificações para a natureza de pesquisa a partir do ponto de vista de seus objetivos: exploratória, descritiva e explicativa. A pesquisa exploratória ocorre quando se quer obter mais informações sobre o assunto abordado e possibilita, segundo Gil et al. (2002), uma investigação de intuições sobre o tema. Este trabalho é mais bem caracterizado como uma pesquisa exploratória, abordando o assunto por meio de elementos mais notáveis no campo de pesquisa, elaborando conexões entre assuntos relacionados.

3.1.4 Procedimentos

Diversos procedimentos podem ser realizados durante a elaboração da pesquisa. Em um primeiro momento, foi realizada uma análise bibliográfica (MORESI et al., 2003), para se encontrar informações relevantes aos temas de pesquisa, e fundamentar alguns conhecimentos iniciais sobre os temas. Em vários momentos, utilizou-se de algo que pode ser considerado, de acordo com Gil et al. (2002), como pesquisa de coorte para consultar dados de documentações das bibliotecas de código e também informações em comunidades voltadas para o desenvolvimento de produtos de software.

Além disso, pela própria natureza do assunto abordado, sendo este um simulador de veículos autônomos, são realizados procedimentos que podem ser considerados, por Moresi et al. (2003), como pesquisas de laboratório.

Tabela 3.1 – Classificação da Pesquisa

Abordagem da Pesquisa	**Natureza da Pesquisa**	**Objetivos da Pesquisa**	**Procedimentos**
Quantitativa e Qualitativa	Aplicada	Exploratória	Pesquisa Bibliográfica, Pesquisa de Coorte, Pesquisa de Laboratório

Fonte: Elaborada pelo autor.

3.2 Processo Metodológico

As atividades deste projeto foram desenvolvidas utilizando a metodologia Kanban de forma adaptada. O Kanban é uma metodologia de desenvolvimento altamente customizável que, de forma geral, consiste em distribuir tarefas em listas, onde cada lista representa um estado no processo de desenvolvimento (BOEG, 2010). Esta metodologia

foi adaptada para incluir também as partes não relacionadas a desenvolvimento do projeto, como a pesquisa bibliográfica ou elaboração de textos. Sendo assim, aborda todas as atividades relacionadas ao trabalho.

Foi possível separar a execução do trabalho em algumas macro-atividades, e com base nelas, foi elaborado um cronograma para a sua execução, que pode ser visto em Tabela 3.2 e Tabela 3.3. Conforme estabelecido pelo processo de execução de TCC na UnB, o trabalho é dividido em duas partes: TCC1 e TCC2, cada tabela representa uma etapa de execução do trabalho.

Tabela 3.2 – Cronograma para desenvolvimento do TCC1

Cronograma TCC1	**Julho**	**Agosto**	**Setembro**	**Outubro**	**Novembro**
Definir o tema	X				
Definir escopo da pesquisa	X				
Pesquisar referencial teórico	X	X			
Realizar a pesquisa		X	X	X	
Documentar o processo de desenvolvimento da pesquisa			X	X	
Apresentar TCC1					X

Fonte: Elaborada pelo autor.

Tabela 3.3 – Cronograma para desenvolvimento do TCC2

Cronograma TCC2	**Dezembro**	**Janeiro**	**Fevereiro**	**Março**	**Abril**	**Maio**
Modificações no escopo	X					
Pesquisar referencial teórico	X	X				
Realizar a pesquisa	X	X	X	X		
Documentar o processo de desenvolvimento da pesquisa	X			X	X	
Apresentar TCC2						X

Fonte: Elaborada pelo autor.

Os detalhes para as atividades identificadas para a realização do TCC são:

- **Definir o tema**: esta atividade tem por objetivo definir o tema do trabalho;
- **Definir escopo da pesquisa**: a atividade busca delimitar qual será o escopo da pesquisa para o trabalho de forma a observar quais são os limites da pesquisa;

- **Pesquisar referencial teórico**: a atividade busca encontrar referências para o tema da pesquisa, para entender o que é o assunto, e o que é necessário para a pesquisa;

- **Realizar a pesquisa**: consiste em realizar atividades de desenvolvimento para alcançar os objetivos propostos, mais bem detalhadas na Subseção 3.2.2;

- **Documentar o processo de desenvolvimento da pesquisa**: a atividade tem por objetivo realizar a escrita dos resultados do processo de pesquisa e informações obtidas durante a pesquisa;

- **Apresentar TCC1**: nesta atividade, são apresentados o trabalho e os resultados ao professor orientador e à banca examinadora;

- **Modificações no escopo**: após a apresentação para a banca examinadora e coleta de seu *feedback*, é um momento oportuno para se refletir no escopo e organização do trabalho e aplicar estas mudanças, o que é feito nesta atividade, e

- **Apresentar TCC2**: é feita a apresentação final do trabalho para a banca examinadora.

3.2.1 Levantamento Bibliográfico

O levantamento bibliográfico foi utilizado para construir uma base inicial de conhecimentos sobre o assunto. Foi executado pelo acesso a múltiplas bases de dados, como os portais indexados pelos Periódicos Capes e também pelo *Google Scholar*. Pesquisas foram feitas com os termos: "*autonomous vehicles*" (Tabela 3.4), "*carla simulator*" (Tabela 3.5), "*yolo carla simulator*" (Tabela 3.6) e "*inverse perspective mapping*" (Tabela 3.7). Além disso, em alguns casos, foi utilizada a exploração das referências.

Tabela 3.4 – Artigo selecionados na busca por *autonomous vehicles*

Título	**Autor**	**Ano**
Autonomous Vehicles and Public Health	David Rojas-Rueda, Mark J. Nieuwenhuijsen, Haneen Khreis e Howard Frumkin	2020
Understanding autonomous vehicles: A systematic literature review on capability, impact, planning and policy	Asif Faisal, Md Kamruzzaman, Tan Yigitcanlar e Graham Currie	2019

Fonte: Elaborada pelo autor.

Tabela 3.5 – Artigo selecionados na busca por *carla simulator*

Título	**Autor**	**Ano**
CARLA: An open urban driving simulator	lexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez e Vladlen Koltun	2017
KITTI-CARLA: a KITTI-like dataset generated by CARLA Simulator	Jean-Emmanuel Deschaud	2021
Physical Adversarial Attack on Vehicle Detector in the Carla Simulator	Tong Wu, Xuefei Ning, Wenshuo Li, Ranran Huang, Huazhong Yang e Yu Wang	2020

Fonte: Elaborada pelo autor.

Tabela 3.6 – Artigo selecionados na busca por *yolo carla simulator*

Título	**Autor**	**Ano**
Traffic Sign Detection using Carla and Yolo in Python	Yogesh Valeja, Shubham Pathare, Dipen Patel e Mohandas Pawar	2021
Speed traffic sign detection on the CARLA simulator using YOLO	Martí Sánchez Juanola	2019

Fonte: Elaborada pelo autor.

Tabela 3.7 – Artigo selecionados na busca por *inverse perspective mapping*

Título	**Autor**	**Ano**
Stereo inverse perspective mapping: theory and applications	Massimo Bertozz, Alberto Broggi e Alessandra Fascioli	1997
Multimodal inverse perspective mapping	Miguel Oliveira, Vitor Santos e Angel D. Sappa	2014
Implementation of inverse perspective mapping algorithm for the development of an automatic lane tracking system	A.M. Muad, A. Hussain, S.A. Samad, M.M. Mustaffa e B.Y. Majlis	2004
Stabilization of Inverse Perspective Mapping Images based on Robust Vanishing Point Estimation	Marcos Nieto, Luis Salgado, Fernando Jaureguizar e Julian Cabrera	2007

Fonte: Elaborada pelo autor.

Também foram incluídos artigos indicados pelos autores da ferramenta YOLO (Tabela 3.8), e dados coletados de relatórios de organizações como a *World Health Organization* (WHO, 2018) e a Sociedade de Engenheiros Automotivos (SAE, 2021).

Tabela 3.8 – Artigos da ferramenta YOLO

Título	Autor	Ano
You Only Look Once: Unified, Real-Time Object Detection	Joseph Redmon, Santosh Divvala, Ross Girshick e Ali Farhadi	2016
YOLO9000: Better, Faster, Stronger	Joseph Redmon e Ali Farhadi	2016
YOLOv3: An Incremental Improvement	Joseph Redmon e Ali Farhadi	2018

Fonte: Elaborada pelo autor.

3.2.2 Realizar Pesquisa

Algumas atividades foram definidas para a realização da pesquisa. Elas são:

- **Instalação do software CARLA**: fazer a transferência e a configuração dos arquivos necessários para executar o software do simulador;
- **Instalar o pacote de desenvolvimento gráfico da placa de vídeo**: realizar a instalação de todas as dependências para poder executar processamento com apoio da placa de vídeo;
- **Instalar o *OpenCV* (BRADSKI, 2000) com suporte a processamento na placa de vídeo**: realizar a instalação da biblioteca, compilada localmente para incluir suporte a processamento na placa de vídeo;
- **Instalar o software para execução independente do YOLO**: esta etapa é feita para instalar o YOLO, suas dependências e alguns modelos já treinados, para realizar testes com as configurações normais e garantir seu funcionamento correto;
- **Refatorar o exemplo do CARLA**: a refatoração deve ser feita para transformar o exemplo de um único *script* para um conjunto organizado em um módulo *Python*;
- **Integrar YOLO no exemplo CARLA para prova de conceito**: adicionar suporte à utilização do YOLO como um sensor no simulador CARLA, utilizando a placa de vídeo para realizar a detecção de objetos, integração feita por meio do *OpenCV* (BRADSKI, 2000);
- **Melhorar desempenho de detecção do YOLO**: refatorar o código da integração para obter melhorias de desempenho na detecção de objetos, inclusive com alterações das bibliotecas e implementações utilizadas;
- **Implementar sensor de câmera estereoscópica no simulador CARLA**: criar sensor que possibilite a coleta de duas imagens, de forma simultânea, a partir de câmeras posicionadas a uma distância conhecida;

- **Implementar algoritmo de cálculo IPM**: implementar o algoritmo para permitir a conversão de uma coordenada da imagem para uma coordenada no mundo real e permitir o cálculo da distância;
- **Integrar IPM e Yolo**: utilizar o YOLO para descobrir a posição de um objeto na imagem e converter essa posição em coordenadas com o IPM;
- **Implementar algoritmo de estereoscopia**: implementar um algoritmo que faça o cálculo da distância a partir da posição de um mesmo objeto em duas imagens;
- **Integrar estereoscopia e YOLO**: utilizar o YOLO para descobrir a posição de um objeto na imagem e realizar o cálculo da distância por meio da estereoscopia, e
- **Elaborar cenário para testes**: criar um cenário no simulador CARLA com objetos em posições fixas e conhecidas, que permita a realização de testes dos sensores e algoritmos implementados para garantir seu funcionamento correto.

3.3 Medição de resultados

O aspecto quantitativo deste trabalho faz necessário encontrar formas de comparar numericamente os resultados obtidos com um valor real, a *ground truth*. Este termo quer dizer verificar se os resultados obtidos pela visão computacional estão medindo corretamente o mundo real (KONDERMANN, 2013). Em conformidade com os objetivos deste trabalho, é necessário medir a distância entre dois veículos. Para isso é preciso estabelecer um método para mensurar esta distância, de forma que permita comparar com os resultados obtidos pelos algoritmos.

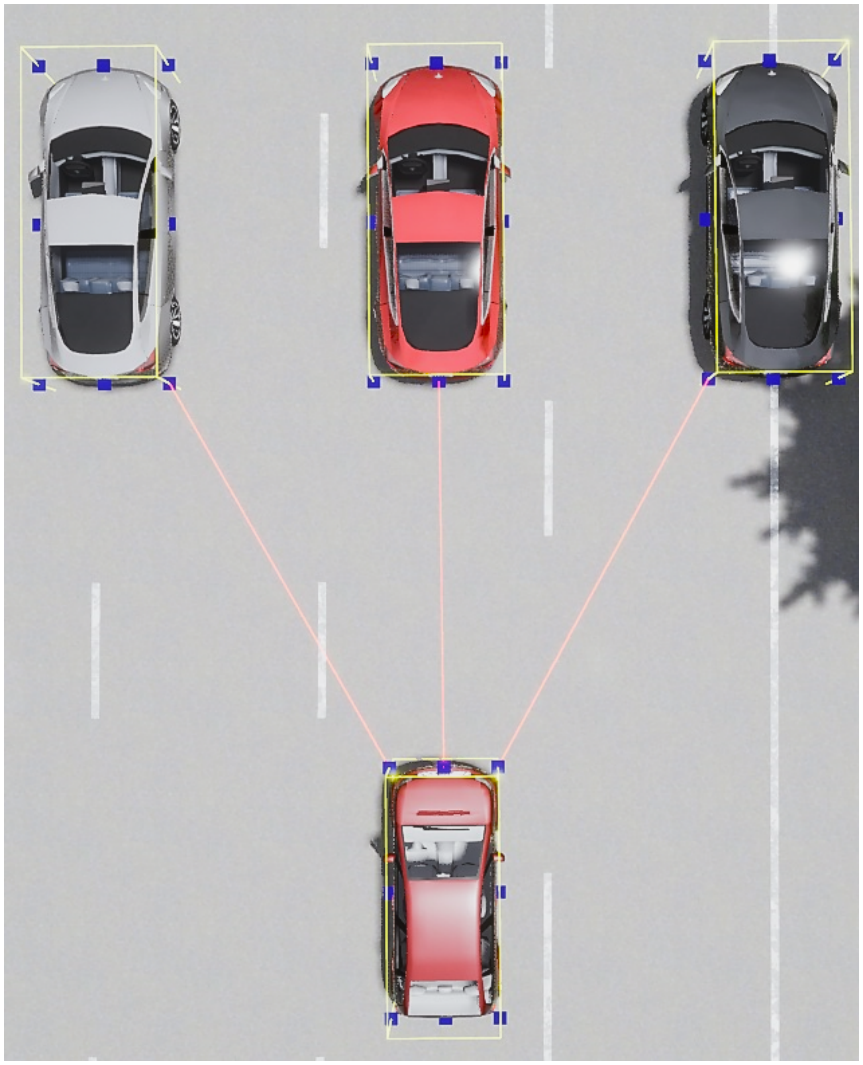
Como este trabalho utiliza um simulador, obter dados precisos da simulação é uma possibilidade. O simulador CARLA cerca cada veículo com uma caixa delimitadora, e fornece qual é a posição do centro desta caixa, e o quanto ela se estende em cada eixo. A Figura 3.1 faz uma representação visual deste cálculo. Com as informações da simulação, é possível definir pontos de medição nestas caixas de forma que, para cada veículo, existam 8 pontos de medição. Eles podem ser vistos como os quadrados de cor azul na Figura 3.1. Na figura, os pontos estão sendo exibidos no centro de altura do carro, mas isso é apenas para facilitar a visualização. O valor real é medido na altura do solo. A distância entre dois pontos de medição é dada pela Equação 3.1, e a distância entre veículos é dada pela menor distância entre seus pontos de medição, sendo vista na linha vermelha, na Figura 3.1.

$$d(x_1, y_1, z_1, x_2, y_2, z_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} \tag{3.1}$$

Para obter os resultados quantitativos de cada algoritmo testado, são feitas medições em distâncias diferentes, utilizando cenários de teste no simulador. O foco deste

trabalho está na medição de distâncias até carros, outras classes de objeto foram desconsideradas. Os cenários elaborados consistem em posicionar um veículo a uma distância fixa do veículo de medição, e então é capturada a imagem das câmeras no veículo de medição, inseridas em um conjunto de dados, que contem as imagens de todos os cenários. Com este conjunto de dados será feito o cálculo das distâncias e comparado com o valor real da distância. Ao todo são 44 cenários, 22 ocorrem com o simulador configurado com o mapa no modo diurno, ao meio-dia, e 22 no modo noturno, à meia-noite, ambos com o clima limpo e sem nuvens. O veículo de medição sempre é mantido na mesma posição, a diferença entre os cenários se dá pelo outro veículo, que muda sua posição em 1 metro a cada cenário, variando de uma posição mínima de 3 metros de distância do veículo de medição para uma posição máxima de 25 metros a partir do veículo de medição.

Figura 3.1 – Caixas delimitadoras, pontos de medição e medição de distância. São utilizados múltiplos carros para demonstrar que a escolha é feita com a menor entre todas as distâncias calculadas. De forma que o resultado é alterada tanto pela distância longitudinal quanto latitudinal.



Fonte: Elaborada pelo autor.

4 Resultados Obtidos

Foi realizada a integração entre o simulador CARLA com o algoritmo de detecção de imagens YOLO, permitindo sua visualização, próxima do tempo real, na aplicação cliente. Também foi criado um modo de câmera estereoscópica e algoritmos de cálculo de distância por estereoscopia e IPM, que funcionam em conjunto com a detecção de objetos do YOLO.

Utilizaram-se as versões YOLOv3 e YOLOv4, de forma que será possível comparar os resultados entre as versões, e verificar se as melhorias na detecção de objetos do YOLOv4 influenciam nos cálculos de distância. Com o objetivo de alcançar desempenho próximo de tempo real, as alterações citadas na Seção 4.2 foram feitas no código do exemplo do simulador CARLA, que serviu de base para o projeto.

Este capítulo está dividido em seis partes. A Seção 4.1 aborda preparações necessárias para o desenvolvimento tecnológico do trabalho. A Seção 4.2 aborda o código fonte utilizado como base para o projeto, e explica sobre algumas modificações realizadas. A Seção 4.3 discorre sobre os tipos de sensores implementados neste trabalho e alguns de seus detalhes. A Seção 4.4 explica como os dados de sensores diferentes foram combinados para realizar o cálculo da distância. A Seção 4.5 aborda alguns detalhes da implementação para se obter mais desempenho. A Seção 4.6 mostra os resultados coletados no projeto.

4.1 Configuração de ambiente

Alguns pré-requisitos para a execução do projeto foram necessários. Utilizou-se a biblioteca *OpenCV* (BRADSKI, 2000) para realizar alguns procedimentos de manipulação de imagem e implementações[1,2] do YOLOv3 e YOLOv4, em PyTorch (PASZKE et al., 2019). Elas foram escolhidas pela facilidade de instalação e integração com o projeto, já com suporte a processamento na placa de vídeo do computador.

Além disso, foi necessário instalar o simulador CARLA e a biblioteca para a utilização no cliente *Python*. O simulador CARLA funciona de maneira independente. No momento de sua inicialização, o simulador fica aguardando a conexão de clientes. Para a coleta de dados, foi utilizado o projeto CARLA *Scenario Runner*[3], que funciona como um cliente CARLA, mas que deve ser o primeiro a se conectar ao simulador para configurar o cenário.

[1] https://github.com/eriklindernoren/PyTorch-YOLOv3
[2] https://github.com/WildflowerSchools/pytorch-YOLOv4
[3] https://github.com/carla-simulator/scenario_runner

4.2 Partindo do exemplo de controle manual do CARLA

O desenvolvimento começou a partir do exemplo de código *manual_control.py*[4], porque este exemplo possui a interface gráfica já implementada, utilizando a *PyGame*, e um módulo que lida com os comandos do usuário. Além disso, traz exemplos de como lidar com diversos sensores e funcionalidades do CARLA. Como o exemplo é um único *script Python*, foi necessário adaptá-lo para que estivesse em múltiplos arquivos. Isso foi feito para facilitar a modificação e implementação de funcionalidades.

O código foi dividido nos módulos *game*, *gui*, *sensors* e *yolo*. O módulo *game* cuida da execução principal do cliente, gerencia a conexão com o simulador, captura os comandos do usuário, e executa ações baseadas neles. Além de instanciar os objetos essenciais para a execução do cliente, instancia os sensores no simulador e os gerencia. Já o módulo *gui*, possui as classes responsáveis por exibir informações na tela do cliente, sobrepondo a imagem dos sensores. O módulo *sensors* traz o código que implementa os sensores integrados ao cliente. Por fim, o módulo *yolo* traz a implementação da integração com o YOLO e algoritmos de medição de distância, no submódulo *distance_measure*.

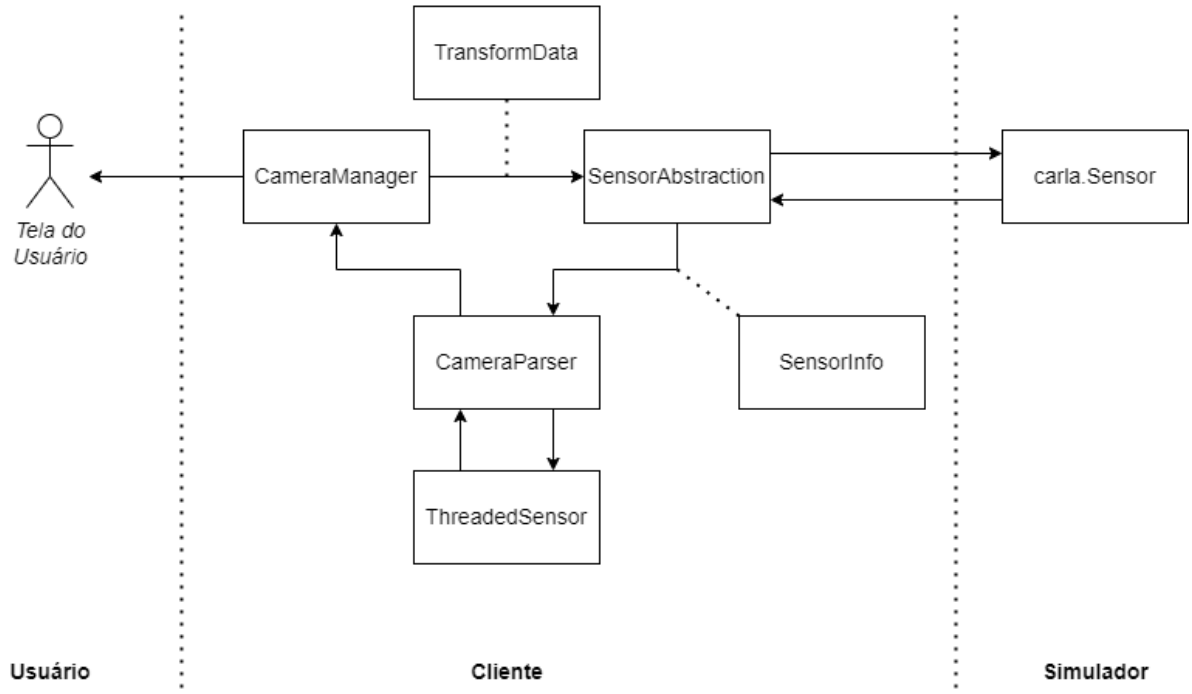
4.3 Sensores Implementados

No módulo *game*, existe a classe *CameraManager*, que era a responsável por criar câmeras virtuais, ou interpretar dados de outros tipos de sensores, e exibir seus resultados na tela para o usuário. Esta classe foi modificada e uma abstração de sensor, *SensorAbstraction*, passou a ser utilizada para fornecer um controle maior de acesso aos sensores, e permitir a implementação de uma câmera estéreo. O código que interpreta os resultados dos sensores também foi separado para uma classe, *CameraParser*. A abstração dos sensores é responsável por criar o objeto virtual do sensor na simulação, e conectar os dados recebidos por ele com o *CameraParser*, que dá o destino adequado para estes dados. Essas classes se comunicam com o auxílio das classes *SensorInfo* e *TransformData*, que enviam os dados do sensor e de localizações no mundo virtual, respectivamente. Uma visualização deste processo pode ser vista na Figura 4.1 e no diagrama de sequência da Figura 4.2.

Os sensores são implementados herdando da classe *ThreadedSensor*. Ela é responsável por gerenciar uma *thread* de trabalho que executa o processamento de dados do sensor, lendo trabalhos, *ThreadedSensorJob*, de uma fila, e executando o método desejado para cada sensor que a implementa.

[4] https://github.com/carla-simulator/carla/blob/master/PythonAPI/examples/manual_control.py

Figura 4.1 – Diagrama das classes envolvidas no controle de sensores. As áreas "*Usuário*"e "*Simulador*"são simplificações, não exibindo o processo feito internamente

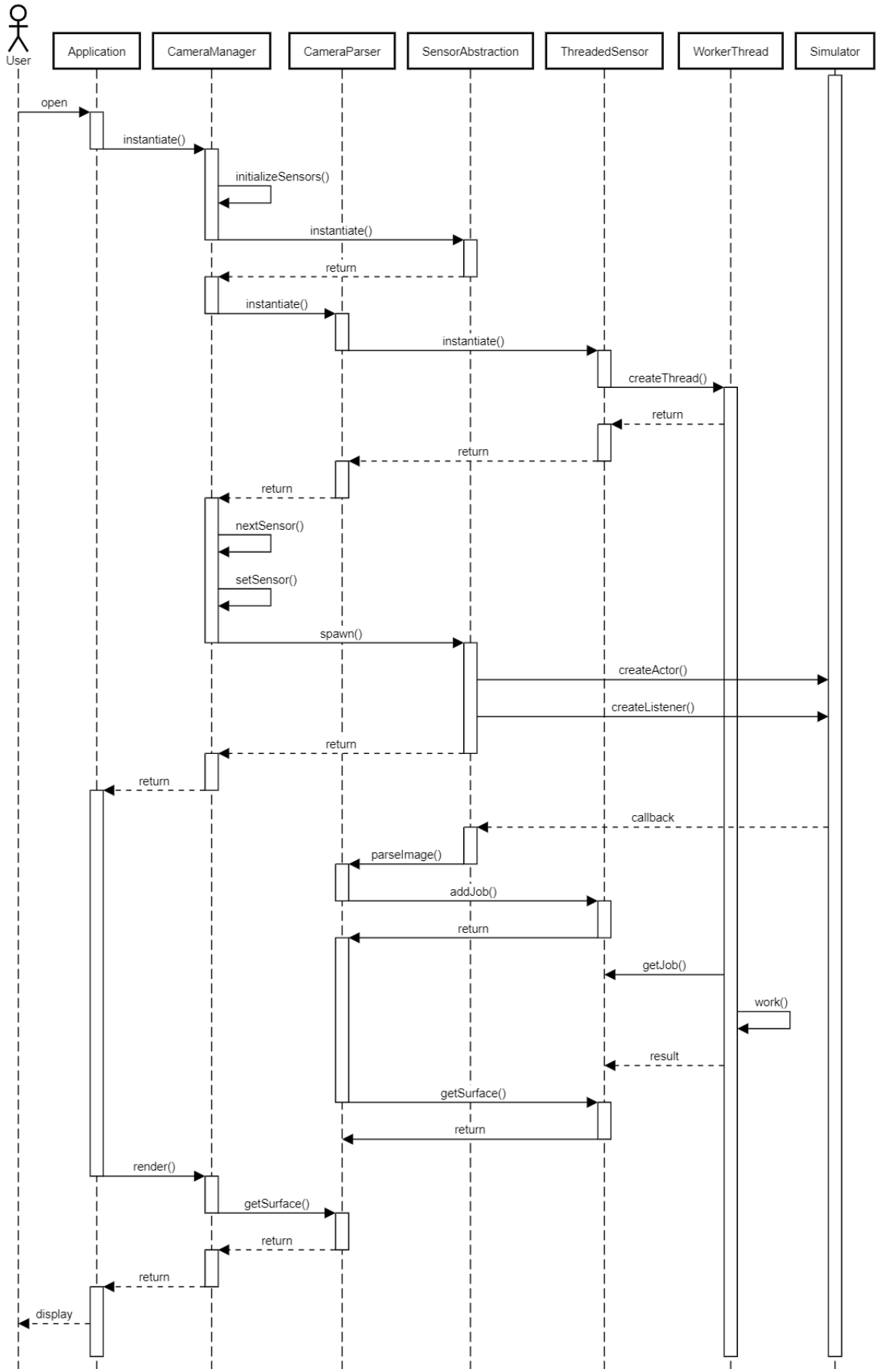


Fonte: Elaborada pelo autor.

Quando uma classe herdeira da *ThreadedSensor* recebe a nova imagem, ela é inserida em uma fila de processamento. Essa fila é processada de forma assíncrona em uma *thread* específico. Esta *thread* é executado de forma contínua, aguardando novas imagens. Ao receber uma nova imagem, ela verifica se essa não é uma imagem muito antiga, com base na diferença entre o horário atual e o horário da criação da imagem. Caso seja, ela será descartada em favor de uma imagem mais nova. Isso foi feito por questões de desempenho, e será mais bem abordado na Seção 4.5. Caso a imagem não tenha sido descartada, ela é preparada para ser processada pelo algoritmo do sensor.

O sensor YOLO é definido como uma câmera virtual, que captura imagens em cores, com resolução de 1280 por 720 pixels. Seu campo de visão é de 72 graus na horizontal. Caso esteja funcionando no modo estéreo, são criadas duas câmeras com distância de 60 centímetros entre elas. Essa distância entre câmeras foi escolhida com base nos resultados obtidos por Zaarane et al. (2020), por Sánchez-Ferreira et al. (2016) e pelos parâmetros definidos por Geiger et al. (2013). O posicionamento da câmera no veículo é, preferencialmente, virado para o mesmo sentido da frente do carro. O funcionamento do sensor pode ser observado, de forma generalizada, no fluxograma da Figura 4.3.

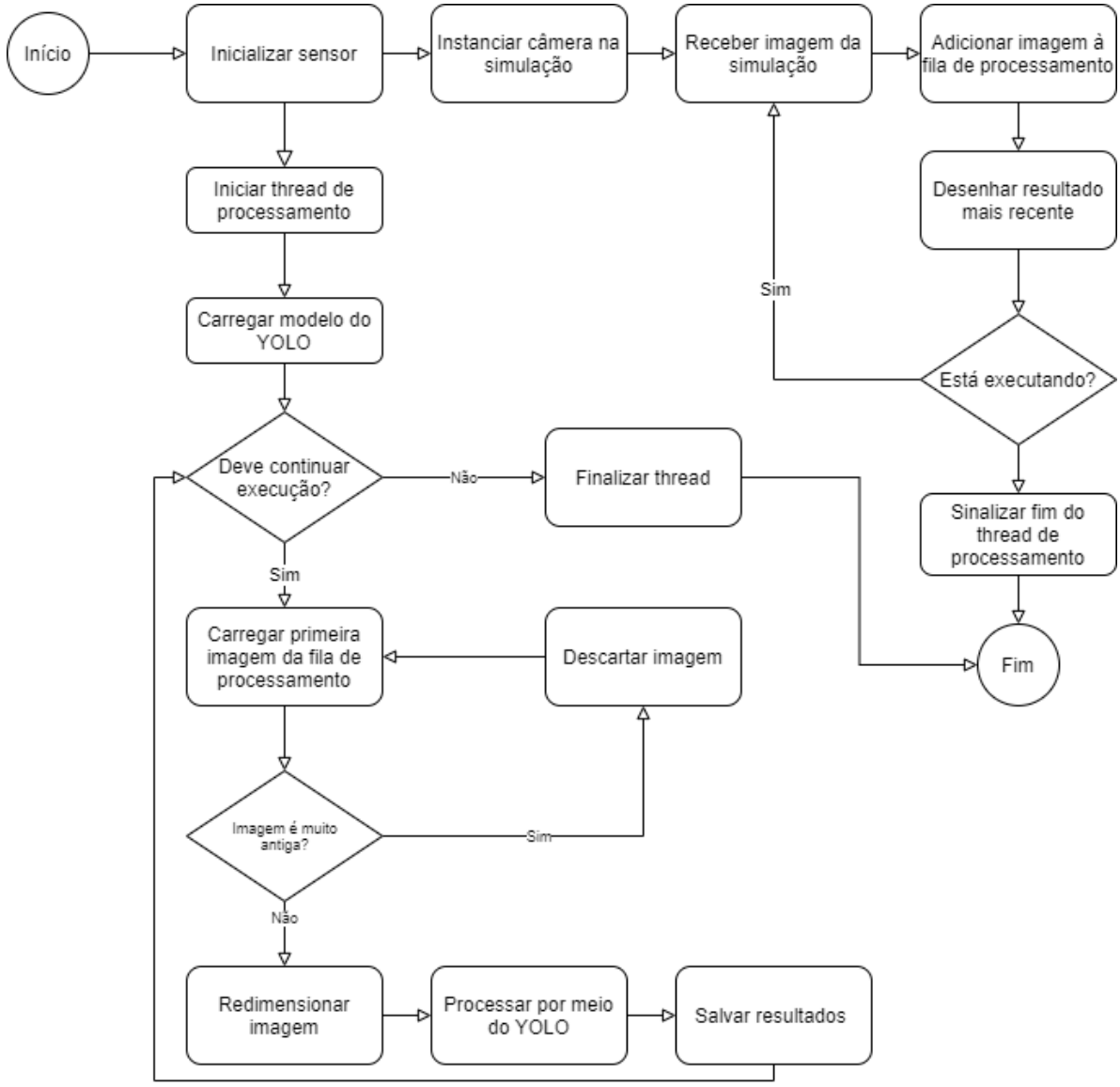
Figura 4.2 – Diagrama de sequência para a execução do CameraManager



Fonte: Elaborada pelo autor.

Duas coisas ocorrem de forma síncrona ao iniciar o cliente: é instanciado o sensor no programa cliente, e a representação virtual do sensor é inicializada no servidor da simulação. No cliente, instanciar o sensor é algo que depende do seu tipo, por exemplo, para o *YoloSensor* é carregado o modelo da rede neural YOLOv3 ou YOLOv4, a partir de alguns arquivos. Já para o *IPMSensor*, é calculada a matriz P, definida na Seção 2.1. Para isso é utilizada uma implementação baseada em Oliveira, Santos e Sappa (2015). Após instanciar a câmera e o sensor, é feita a integração entre eles, onde o sensor fica preparado para receber as imagens enviadas pela câmera a cada passo executado no servidor da simulação.

Figura 4.3 – Fluxograma da execução do sensor YOLO



Fonte: Elaborada pelo autor.

Caso a imagem seja inserida no YOLO, o resultado é uma lista das possíveis clas-

sificações feitas pela rede neural. Essas classificações são lidas e, para cada uma, são extraídas as posições dos pontos que definem o quadro da área de interesse da classificação; a confiança da rede neural na classificação, e a categoria em que aquele objeto foi classificado. Caso a confiança esteja abaixo de um valor mínimo, ela é descartada. Para as previsões restantes, elas são enviadas como resultado e a aplicação cliente desenha as predições na tela do usuário, sendo possível ver como o resultado é exibido na tela da aplicação cliente na Figura 4.4. Aqui, o resultado já contém os dados da integração dos sensores, que será abordada na Seção 4.4.

Ao todo, são exibidas cinco linhas com informações. A primeira linha mostra a posição, em pixels, do centro da marcação na imagem. Já a segunda linha mostra a porcentagem da confiança da classificação do YOLO e a classificação obtida. Na terceira linha é exibido o valor da conversão das coordenadas do ponto de medição na imagem para as coordenadas do mundo, por meio do IPM. Na quarta linha é exibido o resultado de um método de medição não abordado neste trabalho. Por fim, na quinta linha, é exibido o resultado da distância medida pelo IPM.

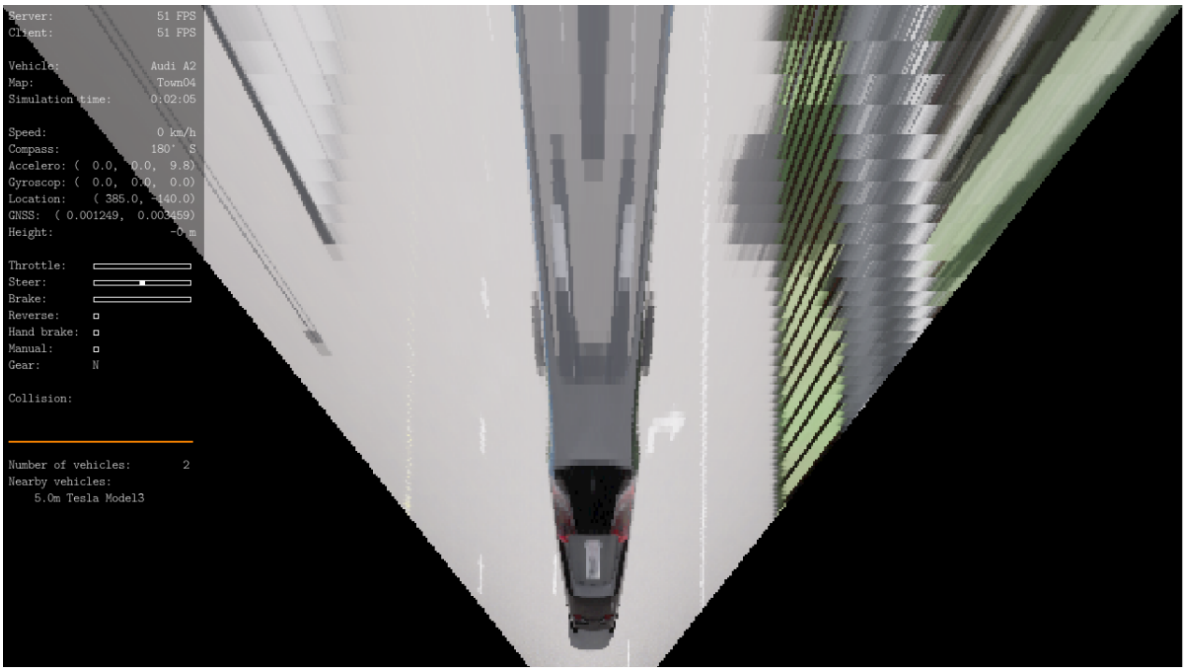
Figura 4.4 – Dados exibidos na tela a partir do sensor YOLO

Fonte: Elaborada pelo autor.

Caso a imagem seja inserida no IPM, o resultado é uma imagem que aparenta dar a vista de cima da cena, o que pode ser visto na Figura 4.5. O modo estereoscópico pode ser visto na Figura 4.6. Aqui, a demarcação possui todas as informações do sensor não estereoscópico adicionadas das linhas 6 e 7, que trazem o resultado do cálculo da distância a partir de implementações baseadas em dois métodos de estereoscopia: Strbac

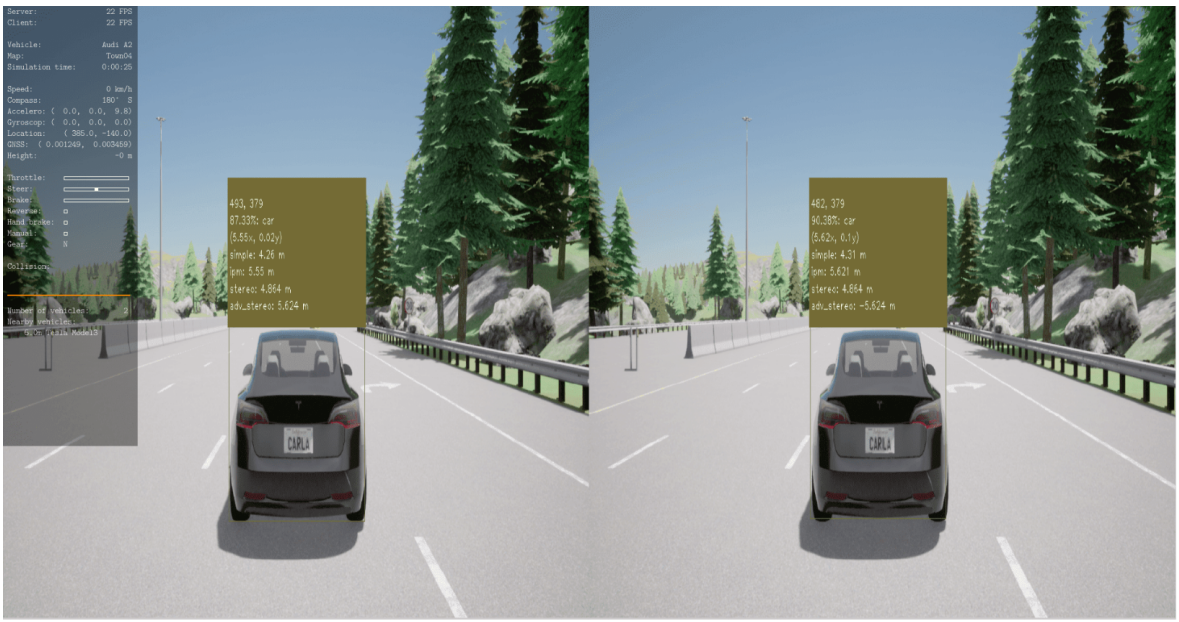
et al. (2020) e Zaarane et al. (2020), respectivamente.

Figura 4.5 – Dados exibidos na tela a partir do sensor IPM



Fonte: Elaborada pelo autor.

Figura 4.6 – Dados exibidos na tela a partir do modo estéreo do sensor YOLO

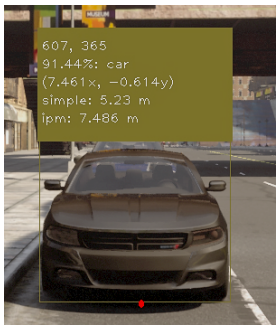


Fonte: Elaborada pelo autor.

4.4 Integrando Sensores

A integração dos sensores tem por objetivo obter o valor da distância, em metros, do carro em que os sensores estão instalados até outro objeto. O valor de entrada é a imagem capturada pelas câmeras no carro. Nessa imagem, utiliza-se o YOLO para encontrar e demarcar veículos, pedestres, e outros objetos. A partir das posições demarcadas, é elencado um ponto de medição. A marcação é em formato retangular, e o ponto de medição é a metade da aresta inferior do retângulo, pois é o ponto que, geralmente, está mais próximo do chão. Na Figura 4.7, o ponto vermelho representa o ponto de medição.

Figura 4.7 – Demarcação feita pelo YOLO, o ponto vermelho é o ponto de medição



Fonte: Elaborada pelo autor.

O ponto de medição facilita a integração entre os sensores. Para calcular a distância com o IPM, basta converter este ponto do plano de coordenadas da imagem para o plano de coordenadas do carro. A maior dificuldade desta integração é encontrar os valores para calibrar as matrizes do IPM, que mesmo sendo coletados do simulador, precisam ser convertidos adequadamente.

Um ponto importante ao lidar com estereoscopia é identificar o mesmo objeto nas duas imagens recebidas. Para isso, foi utilizado um método baseado no proposto por Strbac et al. (2020), que utiliza as detecções do YOLO para realizar esta identificação. Esse método possui duas partes, a primeira é comparar todas as detecções e montar listas de candidatos para cada detecção, com base em critérios de eliminação: a posição da detecção na imagem da câmera esquerda deve estar mais à direita que a detecção da câmera direita, e os objetos detectados devem ser do mesmo tipo. Na segunda etapa, entre as detecções candidatas, deve ser selecionada a detecção com maior similaridade de tamanho. Isso é feito encontrando o mínimo erro quadrático médio. O cálculo do erro entre duas detecções é descrito na Equação 4.1, onde w é a largura da detecção, e h é a altura da detecção, ambas em pixeis. P é um valor que determina se haverá mais peso para o tamanho horizontal ou vertical. Depois de calcular o erro entre os candidatos, é selecionada a detecção com o menor erro. A partir desta detecção, é possível utilizar uma

das equações de estereoscopia, abordadas na Seção 2.1, para encontrar a distância até o objeto.

$$erro = P * (w_L - w_R)^2 + (1 - P) * (h_L - h_R)^2 \tag{4.1}$$

4.5 Considerações Sobre o Desempenho

O desempenho dos algoritmos YOLO e de IPM é um fator importante para a aplicação que se busca desenvolver. Para obter um desempenho próximo de tempo real, foram realizadas algumas iterações na forma como as classificações são feitas.

Primeiramente, buscava-se gerar uma classificação para cada uma das imagens geradas pela simulação. Entretanto, isso causava um problema, porque o tempo de gerar uma predição era maior que o tempo que levava para uma nova imagem ser gerada. Isso causava uma falta de sincronia entre as imagens e as predições, e quando se exibia o resultado da predição no cliente, ele estava cerca de 20 segundos atrasado em relação ao que estava ocorrendo na simulação.

Depois desta tentativa, o código dos sensores foi modificado, inicialmente, apenas para o YOLO, mas, posteriormente, generalizado para o IPM, com o intuito de executar as predições de forma paralela. Por isso, a implementação foi modificada para seguir o padrão de design de software *Thread Pool*. Este padrão consiste em criar uma fila de tarefas a serem executadas, e possuir *threads* de trabalho que executarão as tarefas de forma paralela. Com base nisso, foram testados diversos tamanhos para a quantidade de *threads* de trabalho utilizadas. Entretanto, utilizar mais de uma *thread* de trabalho diminui o desempenho em relação à diferença entre o momento em que os resultados são exibidos e o momento em que a imagem é processada, adicionando latência às predições. Isso aconteceu devido à concorrência do acesso à placa gráfica ao executar as predições em paralelo. Por isso, foi utilizada apenas uma *thread* de trabalho, o que ainda assim trouxe melhora no desempenho em relação à versão anterior. Agora, as predições não bloqueiam a *thread* de execução principal da aplicação cliente.

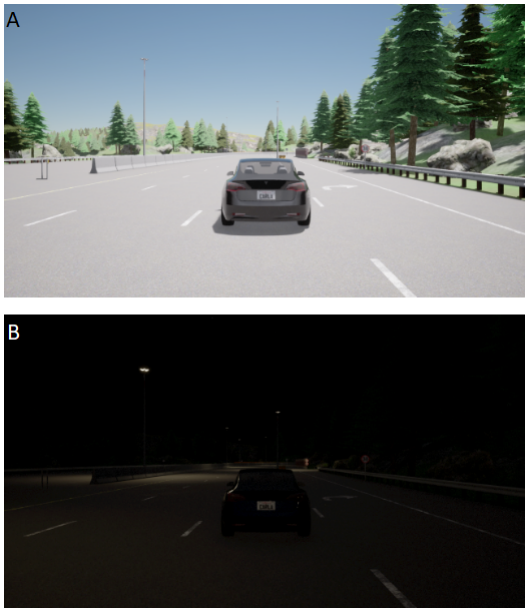
A modificação que trouxe melhor desempenho em relação à diferença entre o momento em que os resultados são exibidos e o momento em que as imagens são geradas foi definir um critério para descarte de imagens da fila. Isso quer dizer que nem todas as imagens serão processadas, mas isso é compensado pelo fato da predição ser muito mais próxima do momento atual da simulação. As imagens são descartadas, caso sejam mais atrasadas que 0,1 segundo. Outras variáveis que afetam o desempenho são o tamanho da imagem processada, o modelo carregado na rede neural, e em quantas classes ele tenta classificar os objetos.

Já para o IPM, converter a imagem completa é um processo computacionalmente caro. Entretanto, quando é feita a integração com o sensor YOLO, o custo computacional é reduzido, já que o IPM converterá apenas um ponto, para obter a distância até determinado objeto. Outro fator que aumenta o desempenho do IPM é a utilização de caches para guardar as matrizes utilizadas e não precisar de um novo cálculo. Isso acontece porque a transformação está ligada aos parâmetros da câmera e, caso eles não mudem, não é necessário recalcular.

4.6 Coleta de Dados

As coletas foram realizadas conforme o descrito na Seção 3.3. Foram considerados dois cenários, um configurado para as condições noturnas e outro para condições diurnas. A diferença pode ser vista na Figura 4.8. Além disso, discriminou-se o resultado com o YOLOv3 e com YOLOv4. Os resultados numéricos podem ser vistos nas tabelas do Apéndice B. Quando um dos métodos falhou em detectar a distância, ela foi atribuída como zero.

Figura 4.8 – Comparação entre os cenários testados. A imagem A exibe a cena diurna e a imagem B a cena noturna, ambas no mesmo local e com o mesmo modelo de veículo.



Fonte: Elaborada pelo autor.

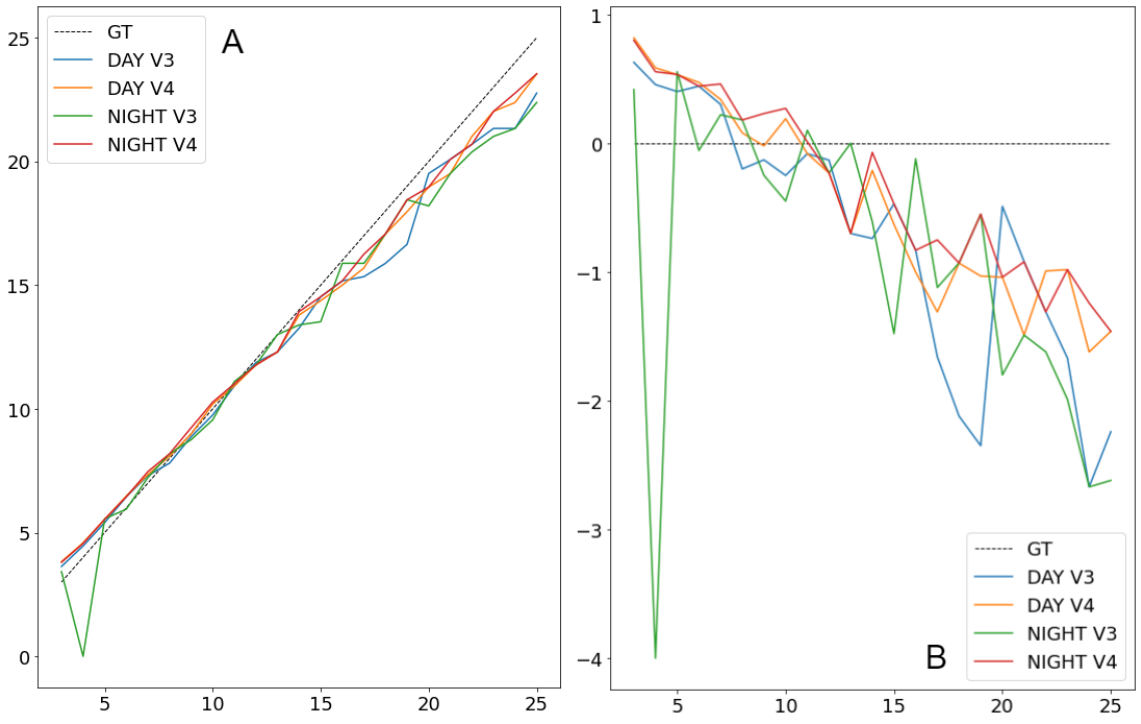
Os resultados podem ser observados em Figura 4.9, Figura 4.10 e Figura 4.11. Cada figura possui dois gráficos, sendo: o da esquerda representando o valor coletado em cada medição, e o da direita o erro entre o valor coletado e o valor real. Neste caso, um erro positivo equivale à predição ser maior que o valor real, e um negativo a um valor

inferior ao real. As linhas nos gráficos são marcadas de acordo com o conjunto de dados e versão do YOLO que representam. "*GT*" representa o valor real. "*DAY V3*" é o conjunto de dados diurnos com o YOLOv3. "*NIGHT V3*" é o conjunto de dados noturnos com o YOLOv3. "*DAY V4*" é o conjunto de dados diurnos com o YOLOv4. "*NIGHT V4*" é o conjunto de dados noturnos com o YOLOv4. Todas as medições estão em metros.

Para comparar os resultados entre os métodos, foi utilizado o cálculo do erro médio quadrático (RMSE), que é definido na Equação 4.2, e os erros podem ser vistos na Tabela 4.1. Além disso foi calculado a média do erro relativo para cada método, que pode ser visto na Tabela 4.2.

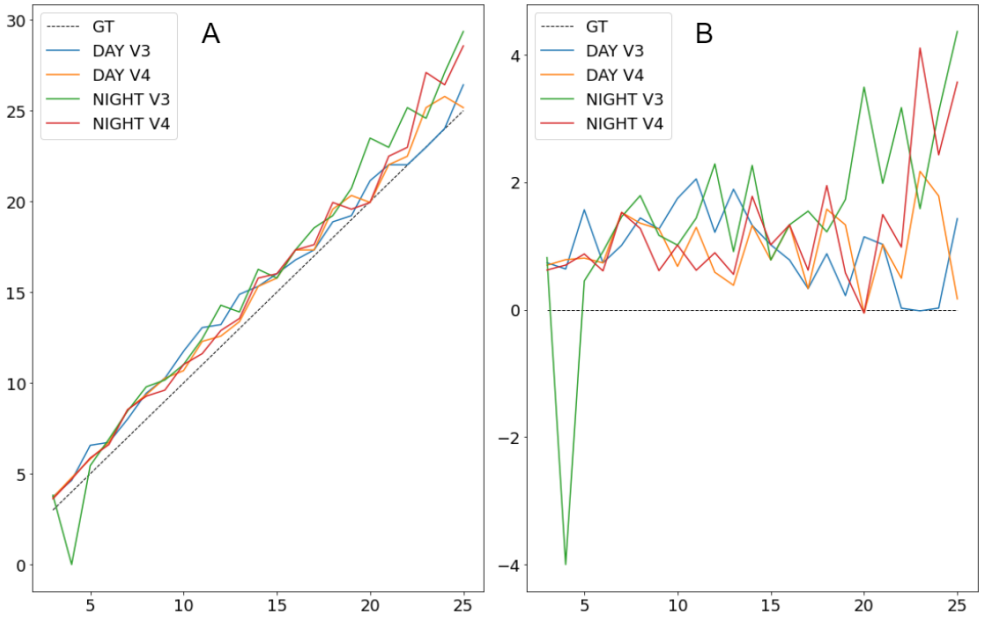
$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{4.2}$$

Figura 4.9 – Resultados da utilização do método baseado em IPM. Para ambos os gráficos as medidas estão em metros e o eixo X representa a distância medida. No gráfico A o eixo Y representa os valores obtidos pela medição da distância. Já no gráfico B, o eixo Y representa o erro entre o valor obtido e o valor real.



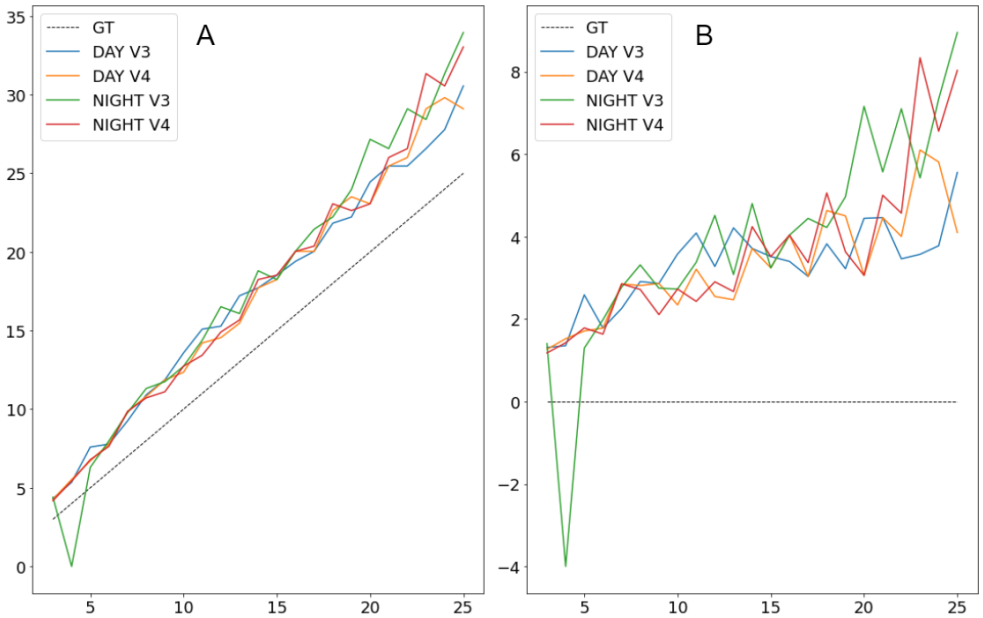
Fonte: Elaborada pelo autor.

Figura 4.10 – Resultados da utilização da implementação baseada no método estéreo de Strbac et al. (2020). Para ambos os gráficos as medidas estão em metros e o eixo X representa a distância medida. No gráfico A o eixo Y representa os valores obtidos pela medição da distância. Já no gráfico B, o eixo Y representa o erro entre o valor obtido e o valor real



Fonte: Elaborada pelo autor.

Figura 4.11 – Resultados da utilização da implementação baseada no método estéreo de Zaarane et al. (2020). Para ambos os gráficos as medidas estão em metros e o eixo X representa a distância medida. No gráfico A o eixo Y representa os valores obtidos pela medição da distância. Já no gráfico B, o eixo Y representa o erro entre o valor obtido e o valor real



Fonte: Elaborada pelo autor.

Tabela 4.1 – Erros absolutos, em metros, calculados com RMSE

Método	**YOLOv3 Dia (m)**	**YOLOv4 Dia (m)**	**YOLOv3 Noite (m)**	**YOLOv4 Noite (m)**
IPM + YOLO	1,208	0,867	1,443	**0,759**
Método baseado em Strbac et al. (2020)	1,134	**1,11**	2,138	1,587
Método baseado em Zaarane et al. (2020)	**3,456**	3,535	4,695	4,104

Tabela 4.2 – Média do erro relativo para cada método

Método	**YOLOv3 Dia (%)**	**YOLOv4 Dia (%)**	**YOLOv3 Noite (%)**	**YOLOv4 Noite (%)**
IPM + YOLO	6,67	6,12	9,78	**5,77**
Método baseado em Strbac et al. (2020)	10,29	**9,38**	16,95	10,3
Método baseado em Zaarane et al. (2020)	27,49	**26,41**	33,84	27,48

5 Conclusão

Considerando o que foi apresentado no Capítulo 4, pode ser observado que, para os objetivos específicos definidos em Seção 1.1, este trabalho conseguiu sucesso. Sendo digno de nota:

- **Explorar o funcionamento e as aplicações do simulador CARLA**: foi possível desenvolver um conhecimento do funcionamento do simulador CARLA e de seus componentes, e também como utilizar a API disponibilizada para controlar a simulação, criando agentes no simulador, alterando configurações e controlando veículos. O exemplo de código construído passa por diversas das funcionalidades básicas do CARLA, e demonstra como utilizar sensores; como instanciar veículos e interagir com o simulador, além de ser construído de forma integrada a um cliente, que permite visualizar o resultado em tempo real das alterações feitas;
- **Integrar o simulador CARLA com o algoritmo YOLO para detecção de objetos no formato de um sensor no veículo virtual**: foi possível analisar as imagens geradas no simulador com o algoritmo YOLO, e exibir os resultados de sua execução na tela, além de incluir no código opções para a customização de qual modelo será carregado e alterações de parâmetros;
- **Implementar algoritmos de detecção de distância baseados em câmeras**: foram implementados algoritmos baseados em estereoscopia e na técnica IPM, que executam em tempo real, de forma que foi possível comparar seus resultados, e
- **Implementar prova de conceito que demonstre o funcionamento da detecção de objetos em conjunto com estimação de distância**: foi realizada a integração entre as técnicas de detecção de objetos e detecção de distância, utilizando uma câmera normal ou uma câmera estéreo, ambas no simulador. Com isto, foi possível identificar a distância entre um objeto e o veículo. A validação deste trabalho foca somente em carros, mas o código possibilita expansões futuras e detecções de outros tipos de objetos.

Assim como observado por Strbac et al. (2020), quando se utiliza o YOLO como detector de objetos em métodos de estereoscopia, depende-se bastante das áreas de interesse retornadas pelo algoritmo. Nos resultados deste trabalho, quando relacionado à estereoscopia, o YOLOv4 é, em geral, superior ao YOLOv3, porque as áreas de interesse retornadas pelo algoritmo são mais precisas. Como o foco dos testes é em relação ao cálculo da distância, o nível de certeza mínimo do algoritmo YOLO foi utilizado em um

valor baixo. Mesmo assim, o YOLOv3 não conseguiu detectar o veículo nas imagens de distância de 4 metros no conjunto de dados noturnos. Isso pode ser observado nas tabelas do Apéndice B, e nos gráficos da Seção 4.6, o que teve um impacto na avaliação final do método por meio do RMSE.

Nos resultados do cálculo do erro com RMSE, o método de IPM saiu-se melhor, seguido pela implementação baseada em Strbac et al. (2020) e, por fim, a implementação baseada no método de Zaarane et al. (2020). O mesmo acontece nos erros relativos. Vale observar que, para este trabalho, a maior adaptação foi feita ao método de Zaarane et al. (2020), já que neste trabalho é utilizado um procedimento diferente para a detecção dos objetos e para cruzar os dados entre as imagens. Esta é uma possibilidade para explicar o erro elevado quando se aplica este método.

A utilização de um simulador de veículos autônomos permitiu entrar em contato com diversos conteúdos, métodos e tecnologias relacionadas a veículos autônomos. Observar seu funcionamento é algo que trouxe maior entendimento sobre os conceitos. Sem o simulador, não seria possível realizar alguns dos testes e desenvolvimentos necessários para chegar neste ponto.

Ainda assim, há alguns elementos que não entraram no escopo deste trabalho, e podem servir de referência para futuras pesquisas. A metodologia de teste pode ser aprimorada. Atualmente dados são coletados a partir de um conjunto de dados limitados e de posições de fixas para os veículos. Também se tem a limitação de serem considerados apenas carros. É possível desenvolver um método para coleta contínua de dados, que ocorra durante a execução de um cenário mais longo, onde é avaliada a distância do veículo ao longo de uma rota definida. Além disso, este trabalho teve como foco o cálculo da distância para um veículo em linha reta. Trabalhos futuros podem considerar posições extras para o veículo medido, ou utilizar mais de um veículo, além de aumentar o conjunto de dados, variando em condições climáticas, de iluminação e localização.

Referências

BERTOZZ, M.; BROGGI, A.; FASCIOLI, A. Stereo inverse perspective mapping: Theory and applications. *Image and Vision Computing*, v. 16, n. 8, p. 585–590, jun. 1998. ISSN 0262-8856. Citado na página 34.

BLOOMENTHAL, J.; ROKNE, J. Homogeneous coordinates. *The Visual Computer*, v. 11, n. 1, p. 15–26, jan. 1994. ISSN 0178-2789, 1432-8726. Citado na página 30.

BOCHKOVSKIY, A.; WANG, C.-Y.; LIAO, H.-Y. M. YOLOv4: Optimal Speed and Accuracy of Object Detection. *arXiv:2004.10934 [cs, eess]*, abr. 2020. Disponível em: <http://arxiv.org/abs/2004.10934>. Citado na página 29.

BOEG, J. Kanban em 10 passos. *Tradução de Leonardo Campos, Marcelo Costa, Lúcio Camilo, Rafael Buzon, Paulo Rebelo, Eric Fer, Ivo La Puma, Leonardo Galvão, Thiago Vespa, Manoel Pimentel e Daniel Wildt. C4Media*, 2010. Citado na página 38.

BRADSKI, G. The OpenCV Library. *Dr. Dobb's Journal: Software Tools for the Professional Programmer*, v. 25, n. 11, 2000. ISSN 1044-789X. Disponível em: <https://www.elibrary.ru/item.asp?id=4934581>. Citado 2 vezes nas páginas 42 e 45.

DOSOVITSKIY, A. et al. CARLA: An Open Urban Driving Simulator. In: *Proceedings of the 1st Annual Conference on Robot Learning*. PMLR, 2017. p. 1–16. ISSN 2640-3498;. Disponível em: <https://proceedings.mlr.press/v78/dosovitskiy17a.html>. Citado 2 vezes nas páginas 19 e 25.

DWORAK, D. et al. Performance of LiDAR object detection deep learning architectures based on artificially generated point cloud data from CARLA simulator. In: *2019 24th International Conference on Methods and Models in Automation and Robotics (MMAR)*. [S.l.: s.n.], 2019. p. 600–605. Citado na página 26.

ELMQUIST, A.; NEGRUT, D. Methods and Models for Simulating Autonomous Vehicle Sensors. *IEEE Transactions on Intelligent Vehicles*, v. 5, n. 4, p. 684–692, dez. 2020. ISSN 2379-8904. Citado na página 25.

Epic Games. *Unreal Engine*. 2019. Disponível em: <https://www.unrealengine.com>. Citado na página 25.

FAISAL, A. et al. Understanding autonomous vehicles: A systematic literature review on capability, impact, planning and policy. *Journal of Transport and Land Use*, Journal of Transport and Land Use, v. 12, n. 1, p. 45–72, 2019. ISSN 1938-7849. Disponível em: <https://www.jstor.org/stable/26911258>. Citado na página 19.

GEIGER, A. et al. Vision meets robotics: The KITTI dataset. *The International Journal of Robotics Research*, v. 32, n. 11, p. 1231–1237, set. 2013. ISSN 0278-3649, 1741-3176. Citado 2 vezes nas páginas 33 e 47.

GIL, A. C. et al. *Como Elaborar Projetos de Pesquisa*. [S.l.: s.n.], 2002. v. 4. Citado 2 vezes nas páginas 37 e 38.

HOPKINS, D.; SCHWANEN, T. Talking about automated vehicles: What do levels of automation do? *Technology in Society*, v. 64, p. 101488, fev. 2021. ISSN 0160791X. Citado na página 19.

JANAI, J. et al. Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art. *Foundations and Trends® in Computer Graphics and Vision*, v. 12, n. 1–3, p. 1–308, 2020. ISSN 1572-2740, 1572-2759. Citado 3 vezes nas páginas 20, 23 e 24.

JEONG, J.; KIM, A. Adaptive Inverse Perspective Mapping for lane map generation with SLAM. In: *2016 13th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. Xian, China: IEEE, 2016. p. 38–41. ISBN 978-1-5090-0821-6. Citado na página 33.

JUANOLA, M. S. Speed traffic sign detection on the CARLA simulator using YOLO. 2019. Disponível em: <http://repositori.upf.edu/handle/10230/42548>. Citado na página 29.

KOCIĆ, J.; JOVIČIĆ, N.; DRNDAREVIĆ, V. Sensors and Sensor Fusion in Autonomous Vehicles. In: *2018 26th Telecommunications Forum (TELFOR)*. [S.l.: s.n.], 2018. p. 420–425. Citado 2 vezes nas páginas 23 e 24.

KONDERMANN, D. Ground truth design principles: An overview. In: *Proceedings of the International Workshop on Video and Image Ground Truth in Computer Vision Applications*. New York, NY, USA: Association for Computing Machinery, 2013. (VIGTA '13), p. 1–4. ISBN 978-1-4503-2169-3. Citado na página 43.

LANCTOT, R. *Accelerating the Future: The Economic Impact of the Emerging Passenger Economy*. 2017. Disponível em: <https://newsroom.intel.com/newsroom/wp-content/uploads/sites/11/2017/05/passenger-economy.pdf>. Citado na página 19.

MENG, X.; WANG, H.; LIU, B. A robust vehicle localization approach based on gnss/imu/dmi/lidar sensor fusion for autonomous vehicles. *Sensors*, Multidisciplinary Digital Publishing Institute, v. 17, n. 9, p. 2140, 2017. Citado na página 23.

MORESI, E. et al. Metodologia da pesquisa. *Brasília: Universidade Católica de Brasília*, v. 108, n. 24, p. 5, 2003. Citado 2 vezes nas páginas 37 e 38.

MUAD, A. et al. Implementation of inverse perspective mapping algorithm for the development of an automatic lane tracking system. In: *2004 IEEE Region 10 Conference TENCON 2004*. Chiang Mai, Thailand: IEEE, 2004. A, p. 207–210. ISBN 978-0-7803-8560-3. Citado na página 30.

NHTSA. *Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey*. National Highway Traffic Safety Administration, 2015. Disponível em: <https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812115>. Citado na página 19.

NIETO, M. et al. Stabilization of Inverse Perspective Mapping Images based on Robust Vanishing Point Estimation. In: *2007 IEEE Intelligent Vehicles Symposium*. Istanbul, Turkey: IEEE, 2007. p. 315–320. ISBN 978-1-4244-1067-5 978-1-4244-1068-2. ISSN 1931-0587. Citado na página 33.

OLIVEIRA, M.; SANTOS, V.; SAPPA, A. D. Multimodal inverse perspective mapping. *Information Fusion*, v. 24, p. 108–121, jul. 2015. ISSN 15662535. Citado na página 49.

OSIŃSKI, B. et al. CARLA Real Traffic Scenarios – novel training ground and benchmark for autonomous driving. *arXiv:2012.11329 [cs]*, set. 2021. Disponível em: <http://arxiv.org/abs/2012.11329>. Citado na página 26.

PASZKE, A. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. arXiv, 2019. Citado na página 45.

PRACIANO, B. J. G. Multi-Camera Framework for Object Detection and Distance Estimation. p. 98, 2020. Citado na página 33.

PRODANOV, C. C.; FREITAS, E. C. D. *Metodologia Do Trabalho Científico: Métodos e Técnicas Da Pesquisa e Do Trabalho Acadêmico-2ª Edição*. [S.l.]: Editora Feevale, 2013. Citado 2 vezes nas páginas 37 e 38.

REDMON, J. et al. You Only Look Once: Unified, Real-Time Object Detection. *arXiv:1506.02640 [cs]*, maio 2016. Disponível em: <http://arxiv.org/abs/1506.02640>. Citado 2 vezes nas páginas 27 e 28.

REDMON, J.; FARHADI, A. YOLO9000: Better, Faster, Stronger. *arXiv:1612.08242 [cs]*, dez. 2016. Disponível em: <http://arxiv.org/abs/1612.08242>. Citado na página 29.

REDMON, J.; FARHADI, A. YOLOv3: An Incremental Improvement. *arXiv:1804.02767 [cs]*, abr. 2018. Disponível em: <http://arxiv.org/abs/1804.02767>. Citado na página 29.

Rojas-Rueda, D. et al. Autonomous Vehicles and Public Health. *Annual Review of Public Health*, v. 41, n. 1, p. 329–345, 2020. Citado na página 19.

SAE. *J3016C: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles - International*. [S.l.], 2021. Disponível em: <https://www.sae.org/standards/content/j3016_202104/>. Citado 3 vezes nas páginas 23, 24 e 41.

Sánchez-Ferreira, C. et al. A real-time stereo vision system for distance measurement and underwater image restoration. *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, v. 38, n. 7, p. 2039–2049, out. 2016. ISSN 1678-5878, 1806-3691. Citado na página 47.

SHINNERS, P. *PyGame*. 2011. Disponível em: <http://pygame.org/>. Citado na página 27.

STRBAC, B. et al. YOLO Multi-Camera Object Detection and Distance Estimation. In: *2020 Zooming Innovation in Consumer Technologies Conference (ZINC)*. Novi Sad, Serbia: IEEE, 2020. p. 26–30. ISBN 978-1-72818-259-9. Citado 11 vezes nas páginas 11, 13, 34, 35, 51, 52, 56, 57, 59, 60 e 71.

TANVEER, M. H.; SGORBISSA, A. An Inverse Perspective Mapping Approach using Monocular Camera of Pepper Humanoid Robot to Determine the Position of Other Moving Robot in Plane:. In: *Proceedings of the 15th International Conference on*

Informatics in Control, Automation and Robotics. Porto, Portugal: SCITEPRESS - Science and Technology Publications, 2018. p. 219–225. ISBN 978-989-758-321-6. Citado na página 30.

TUOHY, S. et al. Distance determination for an automobile environment using inverse perspective mapping in OpenCV. In: *IET Irish Signals and Systems Conference (ISSC 2010)*. Cork, Ireland: IET, 2010. p. 100–105. ISBN 978-1-84919-252-1. Citado 2 vezes nas páginas 31 e 32.

VAJGL, M.; HURTIK, P.; NEJEZCHLEBA, T. Dist-YOLO: Fast Object Detection with Distance Estimation. *Applied Sciences*, v. 12, n. 3, p. 1354, jan. 2022. ISSN 2076-3417. Citado na página 33.

VALEJA, Y. et al. Traffic Sign Detection using Clara and Yolo in Python. In: *2021 7th International Conference on Advanced Computing and Communication Systems (ICACCS)*. [S.l.: s.n.], 2021. v. 1, p. 367–371. ISSN 2575-7288. Citado na página 29.

VARGHESE, J. Z.; BOONE, R. G. Overview of autonomous vehicle sensors and systems. In: *International Conference on Operations Excellence and Service Engineering*. [S.l.]: sn, 2015. p. 178–191. Citado na página 23.

WACHENFELD, W.; WINNER, H. The Release of Autonomous Vehicles. In: MAURER, M. et al. (Ed.). *Autonomous Driving: Technical, Legal and Social Aspects*. Berlin, Heidelberg: Springer, 2016. p. 425–449. ISBN 978-3-662-48847-8. Disponível em: <https://doi.org/10.1007/978-3-662-48847-8_21>. Citado 3 vezes nas páginas 19, 20 e 25.

WHO. *Global Status Report on Road Safety*. [S.l.], 2018. Disponível em: <https://www.who.int/violence_injury_prevention/road_safety_status/2018/en/>. Citado 2 vezes nas páginas 19 e 41.

WU, T. et al. Physical Adversarial Attack on Vehicle Detector in the Carla Simulator. *arXiv:2007.16118 [cs]*, ago. 2020. Disponível em: <http://arxiv.org/abs/2007.16118>. Citado na página 30.

ZAARANE, A. et al. Real-Time Vehicle Detection Using Cross-Correlation and 2D-DWT for Feature Extraction. *Journal of Electrical and Computer Engineering*, v. 2019, p. 1–9, jan. 2019. ISSN 2090-0147, 2090-0155. Citado na página 34.

ZAARANE, A. et al. Distance measurement system for autonomous vehicles using stereo camera. *Array*, v. 5, p. 100016, mar. 2020. ISSN 25900056. Citado 9 vezes nas páginas 12, 13, 34, 47, 51, 56, 57, 60 e 72.

ZHANG, Z. Determining the Epipolar Geometry and its Uncertainty: A Review. *International Journal of Computer Vision*, p. 35, 1998. Citado na página 34.

Apêndices

APÊNDICE A – Código-fonte

O código da aplicação pode ser encontrado acessando a página do repositório no *Github*: https://github.com/guilherme1guy/carla_darknet_integration.

APÊNDICE B – Tabelas de resultados

Tabela B.1 – Resultado das medições de IPM

Distância Real (m)	**YOLOv3 Dia (m)**	**YOLOv4 Dia (m)**	**YOLOv3 Noite (m)**	**YOLOv4 Noite (m)**
3	3,632	3,823	**3,421**	3,803
4	**4,459**	4,589	0,0	4,559
5	**5,405**	5,537	5,56	5,538
6	6,447	6,476	**5,947**	6,447
7	7,305	7,344	**7,225**	7,464
8	7,804	**8,084**	8,184	8,184
9	8,874	**8,983**	8,754	9,234
10	9,753	**10,194**	9,554	10,274
11	10,922	10,923	11,104	**11,014**
12	**11,872**	11,773	11,773	11,773
13	12,302	12,303	**13,003**	12,303
14	13,263	13,792	13,393	**13,932**
15	**14,532**	14,372	13,523	**14,532**
16	15,172	15,002	**15,883**	15,172
17	15,343	15,692	15,882	**16,252**
18	15,882	17,072	**17,073**	17,072
19	16,652	17,971	**18,452**	**18,452**
20	**19,513**	18,962	18,202	18,962
21	**20,083**	19,511	19,512	20,082
22	20,693	**21,011**	20,382	20,692
23	21,333	**22,021**	21,012	**22,021**
24	21,332	22,381	21,332	**22,761**
25	22,761	**23,541**	22,382	**23,541**

Tabela B.2 – Resultado das medições da implementação baseada no método de Strbac et al. (2020)

Distância Real (m)	YOLOv3 Dia (m)	YOLOv4 Dia (m)	YOLOv3 Noite (m)	YOLOv4 Noite (m)
3	3,735	3,696	3,816	**3,62**
4	**4,636**	4,783	0,0	4,698
5	6,566	5,808	**5,449**	5,873
6	6,733	6,733	6,909	**6,607**
7	**8,008**	8,525	8,456	8,525
8	9,438	9,355	9,788	**9,272**
9	10,263	10,263	10,164	**9,61**
10	11,745	**10,677**	11,011	11,011
11	13,05	12,291	12,436	**11,616**
12	13,213	**12,584**	14,285	12,891
13	14,888	**13,381**	13,909	13,552
14	**15,32**	**15,32**	16,262	15,777
15	16,016	**15,777**	**15,777**	16,016
16	**16,779**	17,329	17,329	17,329
17	**17,329**	**17,329**	18,545	17,618
18	**18,876**	19,575	19,219	19,945
19	**19,219**	20,328	20,727	19,575
20	21,141	**19,945**	23,49	**19,945**
21	**22,022**	**22,022**	22,98	22,491
22	**22,022**	22,491	25,168	22,98
23	**22,98**	25,168	24,583	27,104
24	**24,024**	25,782	27,104	26,427
25	26,427	**25,168**	29,363	28,569

Tabela B.3 – Resultado das medições da implementação baseada no método de Zaarane et al. (2020)

Distância Real (m)	YOLOv3 Dia (m)	YOLOv4 Dia (m)	YOLOv3 Noite (m)	YOLOv4 Noite (m)
3	4,312	4,267	4,406	**4,179**
4	**5,355**	5,525	0,0	5,427
5	7,588	6,712	**6,296**	6,786
6	7,782	7,782	7,985	**7,636**
7	**9,257**	9,854	9,775	9,854
8	10,911	10,814	11,315	**10,719**
9	11,865	11,865	11,75	**11,109**
10	13,579	**12,344**	12,73	12,73
11	15,088	14,211	14,378	**13,43**
12	15,277	**14,549**	16,516	14,904
13	17,214	**15,47**	16,081	15,669
14	**17,713**	**17,713**	18,803	18,242
15	18,518	**18,242**	**18,242**	18,518
16	**19,4**	20,036	20,036	20,036
17	**20,036**	**20,036**	21,443	20,37
18	**21,826**	22,634	22,222	23,061
19	**22,222**	23,505	23,966	22,634
20	24,445	**23,061**	27,161	**23,061**
21	**25,463**	25,464	26,571	26,005
22	**25,463**	26,005	29,102	26,571
23	**26,571**	29,102	28,425	31,34
24	**27,779**	29,811	31,34	30,557
25	30,557	**29,101**	33,952	33,034