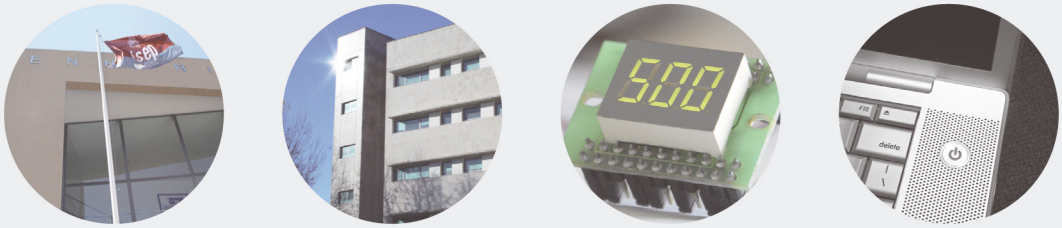
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES



Deteção e Tracking de pessoas e objetos com recurso a LiDAR

JOÃO GABRIEL DE SOUSA MOREIRA
julho de 2023

POLITÉCNICO
DO PORTO

Politécnico do Porto
Instituto Superior de Engenharia do Porto

Deteção e Tracking de pessoas e objetos com recurso a LiDAR

João Gabriel de Sousa Moreira

Mestrado em Engenharia Electrotécnica e de Computadores
Área de Especialização em Sistemas Autónomos

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto

Julho, 2023

Esta dissertação satisfaz, parcialmente, os requisitos que constam da Ficha de Unidade Curricular de Tese/Dissertação, do 2º ano, do Mestrado em Engenharia Electrotécnica e de Computadores, Área de Especialização em Sistemas Autónomos.

Candidato: João Gabriel de Sousa Moreira, Nº 1150672,
1150672@isep.ipp.pt

Orientação Científica: André Miguel Pinheiro Dias, apd@isep.ipp.pt

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
Rua Dr. António Bernardino de Almeida, 431, 4200–072 Porto

Julho, 2023

Agradecimentos

Gostaria de agradecer a todos os que contribuíram para a realização desta dissertação. Em primeiro lugar, quero expressar a minha gratidão ao meu orientador, pela orientação e apoio ao longo de todo o processo. O seu conhecimento e experiência foram fundamentais para o desenvolvimento deste trabalho.

Também gostaria de agradecer aos meus colegas de curso, que estiveram sempre disponíveis para trocar ideias, discutir conceitos e fornecer críticas construtivas. A sua colaboração enriqueceu o trabalho e contribuiu para a sua qualidade.

Não posso deixar de mencionar a minha família e amigos, pelo seu apoio incondicional e encorajamento ao longo desta jornada académica. Agradeço por estarem sempre ao meu lado, acreditando em mim e motivando-me a alcançar os meus objetivos.

Por fim, expresso a minha gratidão a todas as fontes de conhecimento, instituições e autores cujas obras consultei e que foram essenciais para a fundamentação teórica desta dissertação.

A todos os mencionados e a todos os que de alguma forma contribuíram para este trabalho, o meu sincero agradecimento. A realização desta dissertação não teria sido possível sem o apoio e contribuição de cada um de vocês.

Resumo

Esta dissertação explora os avanços no campo da deteção e *tracking* de pessoas utilizando a tecnologia LiDAR. Destacam-se as vantagens do LiDAR em relação às abordagens baseadas em câmaras, devido à sua capacidade de obter informações tridimensionais (3D) do ambiente. Além disso, é apresentado um estudo sobre as metodologias atuais de deteção de objetos utilizando *deep learning*, bem como os métodos de *tracking* mais recentes.

A dissertação também aborda a implementação de ferramentas auxiliares para a preparação de um *dataset* personalizado, destinado ao treino de um modelo de deteção. O processo de treino e suas implicações são explicados em detalhes. Ao combinar as vantagens do LiDAR com as técnicas de deteção e *tracking*, esta dissertação procura contribuir para uma integração desta tecnologia em diversas áreas e aplicações, promovendo avanços na automação e segurança.

Palavras-Chave: LiDAR, inteligência artificial, *machine learning*, redes neuronais, deteção de objetos, *tracking* de objetos, deteção de pessoas, *deep learning*, *dataset* personalizado, treino de modelo *deep learning*.

Abstract

This dissertation explores the advancements in the field of people detection and tracking using LiDAR technology. The advantages of LiDAR over camera-based approaches are highlighted, thanks to its ability to obtain three-dimensional information about the environment. Additionally, a study is presented on current methodologies for object detection using deep learning and the latest tracking methods.

The dissertation also addresses the implementation of auxiliary tools for preparing a customized dataset for training a detection model. The training process and its implications are explained in detail. By combining the advantages of LiDAR with detection and tracking techniques, this dissertation seeks to facilitate the integration of this technology in various areas and applications, promoting advancements in automation and security.

Keywords: LiDAR, artificial intelligence, machine learning, neural networks, object detection, object tracking, people detection, deep learning, customized dataset, deep learning model training.

Índice

Lista de Figuras

Lista de Tabelas

Listagens

Lista de Acrónimos

2D	bidimensional
2DBN	*Two-Dimensional Backbone Network*
3D	tridimensional
3DBN	*Three-Dimensional Backbone Network*
ANN	*Artificial Neural Networks*
AP	*Average Precision*
bbox	*bounding box*
bboxes	*bounding boxes*
BEV	*Bird's Eye View*
BN	*Backbone Network*
CNN	*Convolutional Neural Networks*
COD	*Core Object Detection*
CPU	*Central Processing Unit*
CUDA	*Compute Unified Device Architecture*
DL	*Deep Learning*
DN	*Detector Network*
DNN	*Deep Neural Networks*
DTFT	*Discrete Fourier Transform*
FCN	*Fully Convolutional Network*
FE	*Feature Extraction*
FN	*False Negative*
FOV	*Field of View*

FP	*False Positive*
FPN	*Feature Pyramid Network*
FPS	*Farthest Point Sampling*
FV	*Front-View*
GNN	*Graph Neural Network*
GPS	*Global Positioning System*
GPU	*Graphics Processing Unit*
IA	Inteligência Artificial
ID	identificador único
IMU	*Inertial Measurement Unit*
IoU	*Intersection over Union*
IPO	Instituto Português de Oncologia
ISEP	Instituto Superior de Engenharia do Porto
KITTI	Karlsruhe Institute of Technology and Toyota Technological Institute
LiDAR	*Light Detection and Ranging*
mAP	*mean Average Precision*
ML	*Machine Learning*
MOT	*Multiple Object Tracking*
MRGCN	*Multimodal Relational Graph Convolutional Network*
MT	Metodologias Tracidionais
P-R	*Precision-Recall*
PR	*Prediction Refinement*
ReLU	*Rectified Linear Activation Unit*
RGB	Red, Green and Blue
RL	*Reinforcement Learning*
ROS	*Robot Operating System*

RPN	*Region Proposal Network*
SDR	*Sensor Data Representation*
SECOND	*Sparsely Embedded Convolutional Detection*
SL	*Supervised Learning*
SORT	*Simple Online and Realtime Tracking*
SOT	*Single Object Tracking*
TN	*True Negative*
ToF	*Time of Flight*
TP	*True Positive*
UL	*Unsupervised Learning*
VFE	*Voxel Feature Encoding*

Capítulo 1

Introdução

Nos últimos anos, tem havido avanços significativos na deteção e *tracking* de objetos através da integração de tecnologias como o *Light Detection and Ranging* (LiDAR) [1][2]. O LiDAR permite obter informações 3D do ambiente, proporcionando uma nova perspetiva na deteção de objetos. Esta abordagem introduz níveis adicionais de processamento devido à elevada quantidade de informações capturadas pelo sensor. As técnicas mais clássicas efetuam a deteção de objetos baseando-se principalmente na análise de imagens capturadas por câmaras, onde a identificação da presença e localização dos objetos é realizada através do processamento dos pixeis bidimensional (2D) presentes nas imagens. Os avanços na tecnologia LiDAR permitem melhorias na precisão e alcance, possibilitando um mapeamento mais preciso dos objetos e dos seus movimentos. Os modelos mais recentes apresentam sensores de alta resolução e capacidades de varrimento mais rápidas, facilitando a deteção de objetos de menor dimensão a maiores distâncias [3]. Além disso, os sistemas LiDAR são capazes de operar em diferentes ambientes e condições climáticas, incluindo situações com pouca luminosidade. Com base no processamento das informações 3D obtidas pelo LiDAR, os modelos de deteção podem segmentar e identificar objetos, levando em consideração a sua localização espacial precisa e a sua forma 3D. Após a identificação dos objetos, é possível então o *tracking* da sua trajetória ao longo do tempo. Esta capacidade de deteção e *tracking* é fundamental em aplicações como a condução autónoma, onde é essencial ter uma perceção precisa do ambiente em todas as dimensões.

A tecnologia de deteção e *tracking* de objetos baseada em LiDAR possui diversas aplicações na indústria. Na indústria de transporte, pode ser utilizada em sistemas de condução autónoma [4]. Na agricultura, pode auxiliar no mapeamento e monitorização de culturas [5]. Na construção, pode contribuir para levantamentos topográficos e planeamento de terrenos [6]. A deteção e *tracking* de objetos baseados em LiDAR também podem ser aplicados em vigilância e segurança [7]. Além disso, a tecnologia LiDAR apresenta potenciais aplicações nas áreas da arqueologia [8] e silvicultura [9].

Esta dissertação tem como objetivo explorar os avanços na deteção e *tracking* de objetos utilizando LiDAR. Serão discutidas as técnicas mais relevantes levando em consideração os desafios e soluções propostas. Ao compreendermos o potencial do LiDAR na deteção e *tracking* de objetos em ambientes 3D, podemos impulsionar ainda mais a automação e segurança em várias áreas, estabelecendo as bases para sistemas inteligentes capazes de perceber e interagir com o mundo de forma mais completa e precisa.

1.1 Objetivos

O objetivo principal desta dissertação é a implementação de um sistema de deteção e *tracking* de pessoas recorrendo à tecnologia LiDAR. Embora o sistema poderá detetar qualquer tipo de objeto, com o devido treino, esta dissertação irá ter como foco principal a deteção e *tracking* de pessoas.

Para atingir o objetivo desta dissertação, os seguintes objetivos específicos serão abordados:

- Investigar e selecionar algoritmos e métodos adequados para a deteção de pessoas com recurso ao sensor LiDAR.
- Investigar e selecionar algoritmos e métodos adequados para o *tracking* de objetos 3D.
- Integrar um sistema de deteção de objetos utilizando implementações *open-source*.
- Implementar um sistema de *tracking* de objetos recorrendo ao filtro Kalman.
- Avaliar o desempenho do sistema implementado em termos de precisão de deteção e custo computacional.
- Contribuir para o conhecimento e avanço na área de deteção e *tracking* de pessoas usando tecnologia LiDAR, fornecendo uma solução integrada que poderá ser utilizada através da *framework Robot Operating System* (ROS).

Ao cumprir estes objetivos, espera-se que a dissertação resulte na implementação de um sistema de deteção e *tracking* de pessoas, utilizando a tecnologia LiDAR. O sistema poderá ser aplicado em diversas áreas, contribuindo para o desenvolvimento de soluções avançadas e inovadoras.

Capítulo 2

Estado de Arte

Neste capítulo, serão apresentadas duas metodologias distintas: as Metodologias Tracidionais (MT), que são baseada em princípios geométricos, e o *Deep Learning* (DL). Ambas são consideradas relevantes e constituem a base para o desenvolvimento deste estudo.

Além disso, serão explorados estudos sobre *Machine Learning* (ML) e DL [10], com ênfase nas diferenças fundamentais entre abordagens. Serão discutidos os benefícios e limitações de cada uma, bem como sua aplicabilidade em sistemas de deteção. Será também fornecida uma explicação dos conceitos básicos sobre redes neuronais [11], que são componentes essenciais do DL, destacando sua estrutura e funcionamento.

Ao longo do capítulo, também serão apresentadas diferentes abordagens existentes atualmente para deteção e *tracking* de objetos, discutindo suas vantagens, desafios e aplicações. Dessa forma, será fornecido um panorama abrangente das soluções disponíveis atualmente.

2.1 LiDAR

2.1.1 Sensores complementares

Em áreas como a condução autónoma, verificamos um aumento na quantidade e variedade de sensores utilizados. Particularmente nesta área, é pouco comum encontrar marcas de automóveis que se restrinjam exclusivamente a uma única solução

de sensores.

> *"Each type of car sensor has its advantages. Together they complement each other perfectly."*
>
> (Felix Modes, BMW Group)

O sensor LiDAR não é o único sensor capaz de gerar informação 3D sobre o seu ambiente em redor. Existem outros sensores que possuem características igualmente interessantes, apresentando vantagens e desvantagens relativamente ao LiDAR. Os sensores ultrassónicos 3D disponibilizam informação espacial com um custo unitário e energético inferior ao LiDAR. No entanto, sofrem de atenuação significativa após alguns metros, o que os torna menos adequados como sensor primário para deteção de objetos em distâncias maiores, ao contrário do LiDAR. Esses sensores são amplamente utilizados na indústria automóvel para auxiliar no estacionamento, devido à sua precisão em curtas distâncias [12][13].

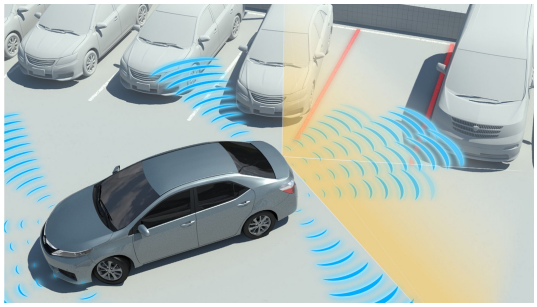


Figura 2.1: Sensor ultrassónico em assistência de estacionamento.

As câmaras apresentam-se como os sensores mais comuns no mercado para projetos de deteção de objetos, sendo a câmara "Microsoft Kinect"um exemplo disso. Elas têm a vantagem de ser o único sensor capaz de capturar e distinguir cores, o que pode ser uma vantagem em objetos específicos, como semáforos e sinais de trânsito. No entanto, a informação obtida por meio das câmaras requer um elevado poder de processamento para obter informações 3D concretas. Outra desvantagem é a sua sensibilidade a fatores ambientais e meteorológicos, como falta de luz ou sujidade na lente. Devido a essas limitações, esse sensor é usado na indústria automóvel para funções de assistência à manutenção de faixa [12][13].

O radar é o sensor mais próximo do LiDAR em termos de características e modo de funcionamento, no entanto, apresenta algumas diferenças importantes. A sua capacidade de precisão em pequenos detalhes é inferior, uma vez que opera em ondas de rádio com um comprimento de onda de 4mm (a 77 GHz), em comparação com as ondas infravermelhas do LiDAR, que possuem comprimentos de onda entre 905-1,550nm. Esta precisão do radar piora à medida que a distância aumenta. Além disso, o seu *Field of View* (FOV) vertical e a resolução angular também são inferiores aos do LiDAR, o que é um aspeto importante para a classificação de objetos [12][13].

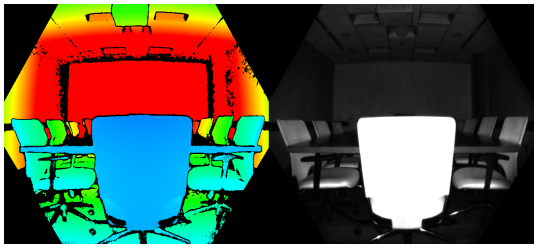


Figura 2.2: Captação de profundidade com a câmara Microsoft Kinect.

Figura 2.3: Deteção de obstáculos com sensor radar.

2.2 Metodologia de Análise

No âmbito da deteção de objetos existem dois ramos principais que diferenciam e categorizam a sua abordagem, designados como análise descritiva e análise preditiva. A análise descritiva concentra-se maioritariamente na compreensão do passado, enquanto a análise preditiva visa prever o futuro.

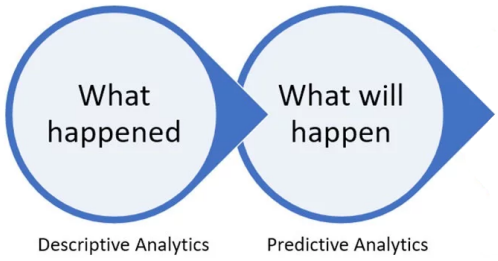


Figura 2.4: Representação da diferença entre análise descritiva e preditiva [14].

2.2.1 Análise Descritiva

A análise descritiva é uma tipologia de análise que exige a definição de um modelo matemático que represente adequadamente o fenómeno a ser observado. Esta abordagem requer um conhecimento aprofundado sobre o fenómeno em questão e está sujeita a erros humanos na elaboração da sua descrição matemática, o que frequentemente a torna considerada como precária, ou com precisão limitada. Além

disso, a análise descritiva depende da disponibilidade de dados históricos precisos e completos, bem como de um entendimento completo dos fatores e variáveis que influenciam o fenómeno em estudo. A interpretação dos resultados obtidos pela análise descritiva também pode ser subjetiva, exigindo uma análise cuidadosa para evitar conclusões enganosas. Portanto, embora a análise descritiva forneça uma valiosa visão retrospetiva, é importante reconhecer as suas limitações e complementá-la com outras abordagens, como a análise preditiva, para obter uma compreensão mais abrangente do fenómeno em questão.

2.2.2 Análise Preditiva

A análise preditiva tem a capacidade de descobrir padrões que auxiliam na identificação de fenómenos, sem a necessidade de fornecer informações descritivas específicas sobre os mesmos. Estes modelos, como o ML, são geralmente mais simples de construir em comparação com os modelos descritivos, embora exijam tempo para treino. Esse processo de treino (padrões de treino) envolve alimentar o modelo com conjuntos de dados onde as saídas (*outputs*) são mapeadas para as respetivas entradas (*inputs*). Durante o treino, o modelo aprende a reconhecer os padrões presentes nos dados e a realizar previsões com base neles. É importante salientar que o desempenho da análise preditiva depende da qualidade e representatividade dos dados de treino utilizados, assim como da escolha adequada do algoritmo de aprendizagem selecionado. Uma vez treinado, o modelo é capaz de fazer previsões em tempo real com base em novos dados, permitindo antecipar eventos futuros e tomar decisões informadas [10].

2.3 Deep Learning aplicado a LiDAR

Neste capítulo iremos abordar a aplicação das técnicas de Inteligência Artificial (IA), ML e DL. Explorar-se-á como a metodologia DL pode melhorar a deteção de objetos recorrendo ao sensor LiDAR. É discutido as vantagens e desvantagens entre a utilização de metodologias DL relativamente a abordagens mais tradicionais, MT.

2.3.1 Inteligência Artificial

Numa perspetiva de comparação, os seres humanos e os animais exibem uma notável capacidade de identificar prontamente uma flor na natureza assim que a avistam, graças à acumulação de experiência registada nos cérebros de seus antecessores ao longo de milhões de anos, transmitida de geração em geração. Além disso, possuem conhecimentos complementares, como o reconhecimento de diferentes tipos de flores, o entendimento dos seus habitats e a distinção entre diversas tipologias florais. Em contraste, os computadores são máquinas desprovidas desse conhecimento ancestral

transmitido, onde as imagens são interpretadas meramente como extensos vetores numéricos ou sequências de bits, desprovidos de contexto significativo. Neste contexto, a IA desempenha um papel fundamental ao se aplicar precisamente no âmbito da contextualização. Ela possibilita que os sistemas interpretem e compreendam esses vetores de informação provenientes do mundo visual, de maneira análoga ao cérebro humano. Para alcançar esse objetivo, a IA recorre ao uso de sensores que se assemelham aos olhos humanos, capacitando as máquinas a agirem adequadamente, com base nas informações adquiridas. Desta forma, a IA emerge como algo crucial para impulsionar a capacidade de análise e perceção do mundo visual por parte das máquinas.

Nos últimos anos, tem havido um notável progresso no desenvolvimento e no conhecimento, especialmente na última década. Este progresso tem impulsionado significativamente a criação e o desenvolvimento de sistemas inteligentes que estão a aproximar-se cada vez mais dos sistemas cognitivos humanos. Estes sistemas inteligentes estão a tornar-se capazes de realizar tarefas complexas de forma autónoma, demonstrando um impacto positivo em diversos aspetos do nosso quotidiano, incluindo desde o setor financeiro e empresarial até o âmbito social. A ML, um ramo da IA, é fundamentada em modelos analíticos que são capazes de produzir uma diversidade de resultados, tais como previsões, regras, recomendações e resultados similares [10]. O aumento no desenvolvimento e uso de métodos de ML e IA na última década deve-se a fatores como a disponibilidade crescente de dados (*big data*), o aumento significativo do poder computacional disponível em computadores comuns e/ou sistemas embebidos (por exemplo, *Raspberry Pi*), e o desenvolvimento de novas *frameworks* que simplificam o processo de desenvolvimento desses sistemas. Uma das principais vantagens desses sistemas é a sua capacidade de evitar a necessidade de traduzir o conhecimento humano sobre um objeto, principalmente no contexto da deteção, para uma linguagem de máquina. Isso reduz processos demorados e propensos a erros, simplificando a integração do conhecimento diretamente no sistema inteligente. [15].

Para uma melhor compreensão das diferenças entre tecnologias da IA, é essencial apresentar os fundamentos integrantes da mesma. Posteriormente, serão explicadas e distinguidas as metodologias ML, *Artificial Neural Networks* (ANN), *Deep Neural Networks* (DNN) e MT. Essa abordagem permitirá aos leitores compreenderem com mais clareza as características e aplicações específicas de cada uma dessas tecnologias.

Machine Learning

A aplicação de ML significa que se observe um melhoramento, ao longo de um período de tempo, numa tarefa que lhe tenha sido designada. O conceito de melhoramento depende do conjunto de métricas usadas para o efeito, podendo ser uma

melhoria na precisão, eficácia ou eficiência. Utilizando métodos de previsão, o objetivo do ML é que se consiga automatizar processos que até agora eram inteiramente analíticos, tal como deteção de objetos ou tradução linguística. Em situações onde o *dataset* a ser estudado apresenta uma dimensão elevada, como acontece nas nuvens de pontos tridimensionais obtidas pelo sensor LiDAR, o ML demonstra uma grande aplicabilidade em executar tarefas como classificação e regressão [10].

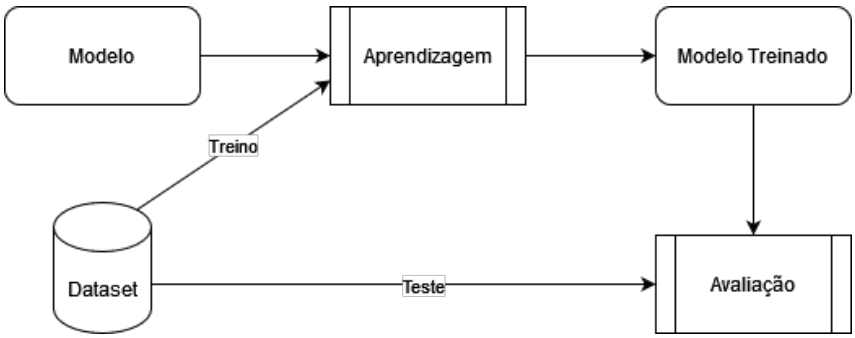


Figura 2.5: Aprendizagem de um algoritmo *machine learning*.

O ML pode ainda ser subdividido em três categorias categorizando o seu tipo de adquirir conhecimento sobre o *dataset*, nomeadamente aprendizagem supervisionada, aprendizagem sem supervisão e por último aprendizagem por reforço.

Aprendizagem supervisionada, do inglês *Supervised Learning* (SL), é regularmente utilizada nos mercados de ações digitais para obter previsões sobre a valorização ou desvalorização da cotação de um dado *asset* [16]. Para treinar um sistema em SL é necessário um *dataset* que contenha exemplos não só de *inputs* mas também com *labeled outputs*.

A segunda categoria designa-se de aprendizagem sem supervisão, do inglês *Unsupervised Learning* (UL), e esta é usada quando é requerido que o sistema detete padrões sem ter existido treino com rótulos (*labels*) ou especificações. O seu treino requer apenas um único dado de entrada, estando apenas interessado em descobrir grupos de elementos que partilhem propriedades ou características comuns. Este modelo é frequentemente usada em negócios digitais de comunicação onde o objetivo é enviar conteúdo digital personalizado consoante o grupo de características onde cada elemento se encontra [17].

E por último aprendizagem por reforço, do inglês *Reinforcement Learning* (RL), que não necessita do par *input*/*output*. Este modelo de ML necessita de uma descrição do estado do sistema, um objetivo final, e uma lista de ações permitidas juntamente com as restrições às quais estas se sujeitam consoante os seus resultados. Após serem fornecidos estes dados, este modelo tenta alcançar o seu objetivo final com uma aprendizagem baseada na tentativa e erro. Este modelo é atrativo para situações onde é possível obter um ambiente fechado, como é o caso dos jogos [18].

Artificial Neural Networks

As ANN ganharam significativa popularidade na última década devido à sua flexibilidade estrutural, permitindo adaptação a uma variedade de contextos. As ANN consistem em representações matemáticas de unidades de processamento interconectadas, chamadas neurónios artificiais, que se baseiam no princípio de processamento de informação em sistemas biológicos, como o cérebro humano. Cada conexão entre neurónios tem a função de transmitir sinais cuja intensidade varia em função de uma variável, sendo que o valor dessa intensidade é ajustado durante o processo de treino para obter os melhores resultados possíveis. Esses sinais são processados pelos neurónios subsequentes quando a intensidade ultrapassa um determinado limite definido pela função de ativação [10].

Uma rede neuronal pode ser composta por várias camadas. Tipicamente, uma camada de entrada recebe dados a serem analisados (por exemplo, uma imagem ou uma nuvem de pontos), enquanto uma camada de saída é responsável por classificar as características presentes nesses dados. Além desse par de camadas, pode haver camadas ocultas, que são responsáveis pela aprendizagem do mapeamento não linear entre a entrada e a saída [19], conforme ilustrado na Figura 2.6. No entanto, uma ANN requer algumas definições manuais, como o número de camadas e neurónios, taxa de aprendizagem, função de ativação, entre outros, devido à limitação de não poderem ser aprendidas pelo algoritmo [10].

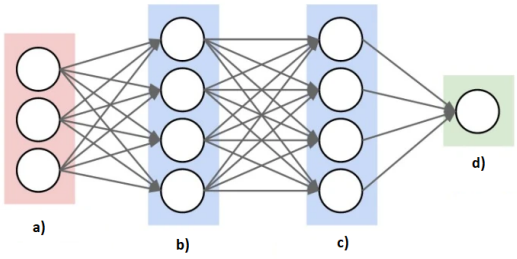


Figura 2.6: a) *Input layer*, b) primeira *layer* oculta, c) segunda *layer* oculta e d) *output layer*.

Deep Neural Networks

As DNN são um subconjunto do ML e têm uma estrutura baseada no modelo ANN. No entanto, as DNN são mais ramificadas, profundas e complexas. Além disso, elas beneficiam do uso de neurónios mais avançados em comparação com as ANN, permitindo a realização de operações avançadas, como a convolução, e o uso de múltiplas funções de ativação. Todas essas características permitem que as DNN descubram automaticamente as representações necessárias para uma tarefa de aprendizado. Por

outro lado, os algoritmos baseados em ML e ANN que não possuem essas características são frequentemente denominados de aprendizagem superficial (*shallow machine learning*), como é o caso do *shallow autoencoder* e das *decision trees*.

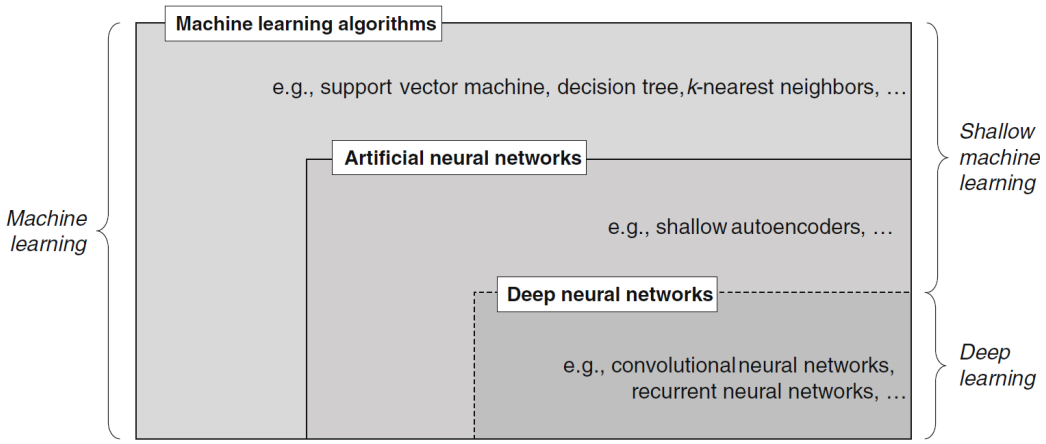


Figura 2.7: Diagrama de conceitos ML e as suas subcategorias [19].

As DNN são frequentemente usadas em circunstâncias onde é necessário o processamento de *datasets* grandes e de alta definição, onde estas superam os algoritmos ML (ex. *point cloud*, vídeo, imagem, áudio). A vantagem dos algoritmos *shallow* baseados em ML surge quando o *dataset* é de baixa dimensão, sendo possível exceder os resultados de uma DNN [20].

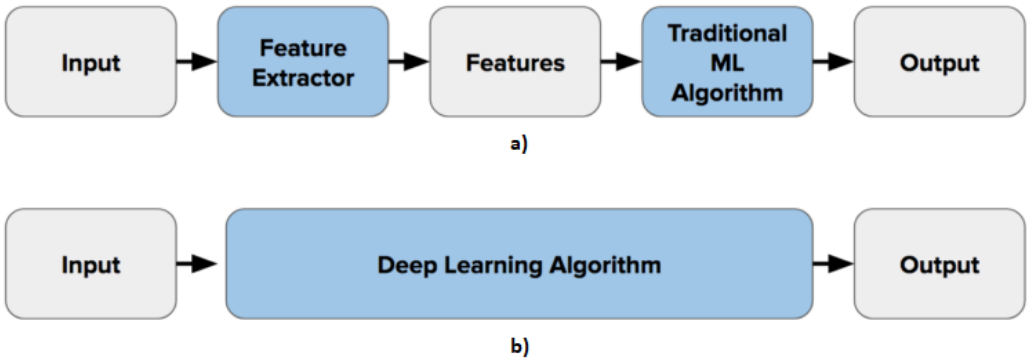


Figura 2.8: Diferença entre a) machine learning e b) deep learning.

2.3.2 Metodologias Tradicionais

Existe uma controvérsia em torno do significado atual de MT. Diversas fontes propõem diferentes interpretações do termo. Por um lado, há aqueles que a consideram como abordagens que se baseiam em análises descritivas, exigindo uma definição matemática precisa do objeto no algoritmo, estabelecida por um ser humano. Por outro

lado, existem aqueles que argumentam que a MT engloba métodos ML que não dependem explicitamente do uso de redes neuronais, sejam elas simples ou complexas [21] [10].

Nesta dissertação, adotaremos a caracterização da MT de acordo com a primeira definição mencionada. Embora muitas técnicas de MT tenham surgido no início deste século, atualmente são consideradas menos relevantes devido ao surgimento das redes neuronais e à rápida adoção da metodologia DL.

Vantagens

Apesar de as abordagens baseadas em redes neuronais e DL serem amplamente adotadas na deteção de objetos atualmente, as MT apresentam vantagens significativas em situações em que o objeto a ser detetado possui características de geometria simples. Nessas circunstâncias, o desenvolvimento de redes neuronais seria excessivo, tornando as MT uma alternativa mais adequada. Além disso, a falta de um conjunto de dados de treino diversificado ou a inviabilidade de obtê-lo pode ser um obstáculo para o uso de redes neuronais, resultando em previsões de deteção com qualidade pouco satisfatória [22]. O treino de redes neuronais requer a disponibilidade de conjuntos de dados ricos em informações sobre o objeto em questão, o que nem sempre é viável. Além disso, a complexidade dos parâmetros internos dos algoritmos DL dificulta a afinação e refinamento desses modelos quando os resultados de deteção não atendem às expectativas, muitas vezes exigindo a repetição do processo de treino com um conjunto de dados mais abrangente.

Por outro lado, as abordagens tradicionais das MT oferecem flexibilidade significativa na afinação dos parâmetros. É possível modificar o código-fonte para incorporar novos modelos matemáticos que complementem ou adicionem objetos a serem detetados. Além disso, as MT são menos exigentes em recursos computacionais e armazenamento, o que permite a integração em sistemas computacionais mais simples, como micro-controladores em sistemas embebidos. Estas características tornam as MT uma opção viável em cenários onde o uso de técnicas DL não é prático ou justificado [21].

Portanto, reconhece-se que as MT possuem méritos indiscutíveis em determinados contextos, proporcionando flexibilidade, eficiência e viabilidade em situações específicas. No entanto, é importante considerar suas limitações em relação à capacidade de lidar com objetos de geometria complexa e à dependência de conjuntos de dados de treino abrangentes para obter resultados de deteção de qualidade.

Aplicações

As dificuldades acrescidas na utilização de *Convolutional Neural Networks* (CNN) 3D, devido à dimensão e densidade dos *datasets* fornecidos pelos sensores 3D, influenciou o reaparecimento das MT. A *Discrete Fourier Transform* (DTFT) é uma das técnicas tradicionais utilizadas para melhorar a rapidez das CNN [23] [24]. Para os métodos DL atingirem uma percentagem de deteção aceitável é necessário o uso de *datasets* de alta dimensão (*big data*). Quando tal não é possível, existem MT dedicadas a aumentar artificialmente a quantidade de exemplos existentes num *dataset* (*data augmentation*). Esta técnica envolve a geração de novos exemplos, com base nos existentes, que contenham pequenas diferenças em rotação e escala [25].

2.4 Dataset e Benchmarks

Um *dataset* é uma coleção de dados, desde imagens a *point clouds*, que podem ser utilizados para treinar e validar algoritmos de ML com o intuito deste encontrar padrões previsíveis no conjunto do *dataset*. Todas as implementações de referência sobre deteção de objetos com recurso a ML adotam uma metodologia de treino supervisionado (abordado neste capítulo, na subsecção 2.3.1) utilizando os *datasets* não só para o seu treino, mas também para avaliação (*benchmark*) sobre a eficiência do algoritmo através das *bounding boxes* (bboxes) e classificações presentes no *dataset*, fornecendo assim um importante *ground truth*. Atualmente, existem *datasets* que disponibilizam dados 3D provenientes de sensores LiDAR, frequentemente referidas como nuvem de pontos (*point cloud*). Estes dados incluem as anotações correspondentes, caracterizando as classes às quais cada objeto pertence. Essa informação adicional constitui uma base fundamental para o desenvolvimento de sistemas de deteção de objetos com recurso a técnicas provenientes de ML.

Tabela 2.1: Comparação entre os três *datasets* mais utilizados.

Nome	**Ano**	**PC Frames**	**Classes**	**Localização**
KITTI	2012	15,400	8	Karlsruhe (Alemanha)
Waymo	2019	200,000	4	SF, Phoenix, Mt. View (EUA)
NuScenes	2019	400,000	23	Boston (EUA), Singapura (SG)

2.4.1 KITTI

Um dos primeiros e mais relevantes *datasets* criados foi o Karlsruhe Institute of Technology and Toyota Technological Institute (KITTI) [26], lançado em 2012. Atualmente, continua a ser um dos *dataset* e *benchmark* mais utilizados, contendo aproximadamente 15.000 *point clouds*, devidamente identificadas e classificadas. Estes dados são obtidos por meio de um sensor LiDAR Velodyne HDL-64E, juntamente

com duas câmaras *stereo* e localização por *Global Positioning System* (GPS). Estes sensores estão montados num veículo (fig. 2.9) e toda a informação (*data*) proveniente deles é sincronizada a uma taxa de 10 *Farthest Point Sampling* (FPS).

Os *datasets* KITTI são obtidos através de uma condução em condições de boa luminosidade em áreas rurais e autoestradas de Karlsruhe, Alemanha. Este *dataset* contém objetos caraterizados consoante oito classes diferentes, porém, apenas três dessas classes (carro, pedestre e ciclista) são utilizadas para as métricas de avaliação. É importante destacar que, nas avaliações na *database* KITTI, os objetos detetados pelo LiDAR são apenas considerados se também forem visíveis pelas câmaras. Caso contrário, o objeto é ignorado.

Contudo, devido ao tamanho reduzido do seu *dataset* tridimensional, a maioria dos métodos DL requerem a utilização de *data augmentation* para atingirem resultados de precisão satisfatórios. Além disso, o sensor LiDAR utilizado produz *point clouds* com baixa densidade a longo alcance, o que torna desafiadora a avaliação da deteção em objetos que estão a longa distância do veículo usado para recolher esses dados, como é o caso da condução em autoestradas.

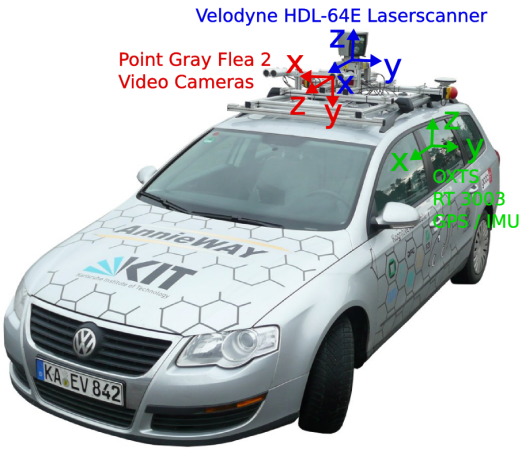


Figura 2.9: Distribuição de sensores no automóvel KITTI.

2.4.2 NuScenes

O mais recente *dataset*, intitulado por NuScenes [27] (2019), foi desenvolvido especialmente para aplicações de condução autónoma. Este *dataset* apresenta uma maior variedade de sensores em comparação com o KITTI, nomeadamente um LiDAR de 32 feixes, cinco radares diferentes, seis câmaras Red, Green and Blue (RGB), uma unidade *Inertial Measurement Unit* (IMU) e GPS, os quais se encontram distribuídos de modo a obter um campo de visão 360º. Além disso, o *dataset* contém aproximadamente 1 milhão de objetos devidamente identificados e seguidos pelo sensor por períodos de 20 segundos. Os dados foram recolhidos e registados pela NuTonomy, a empresa responsável pelo lançamento do NuScenes, durante a condução em

várias condições meteorológicas e de luminosidade em áreas diferentes de Boston e Singapura. Esta recolha de informação proporcionou uma variedade de sentidos de circulação de tráfico, uma vez que Singapura adota o sentido de circulação pela via da esquerda, semelhante ao adotado pelo Reino Unido.

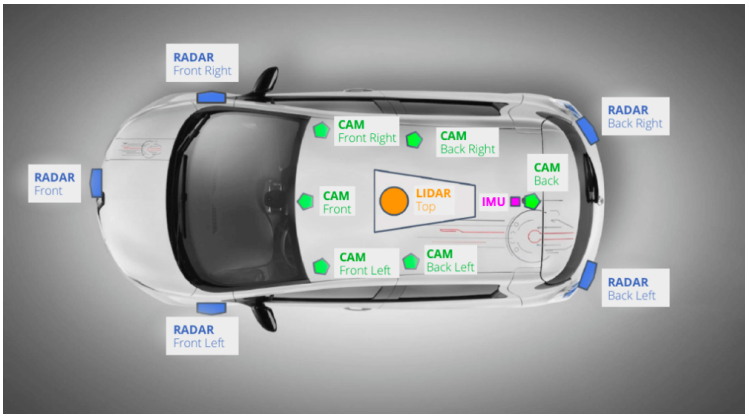


Figura 2.10: Distribuição de sensores no automóvel da NuTonomy para a captura de dados integrados no *dataset* NuScenes.

2.4.3 Waymo

Com lançamento no mesmo ano do NuScenes, o *dataset* Waymo é disponibilizado no mercado contendo aproximadamente 10 milhões de objetos identificados e seguidos em intervalos de 20 segundos, em diversas condições meteorológicas. Os dados integrantes deste *dataset* foram obtidos em áreas urbanas e suburbanas, inicialmente nas cidades de San Francisco, Mountain View e Phoenix e, mais tarde, em Los Angeles, Detroit e Seattle, em diferentes condições de luminosidade. No que diz respeito aos sensores, este *dataset* inclui um LiDAR de médio alcance, quatro LiDAR direcionais de curto alcance e cinco câmaras RGB de alta resolução. A combinação entre os LiDAR de curto e longo alcance permite a obtenção de um nível de detalhe elevado em objetos próximos ao automóvel, bem como uma densidade suficiente a longo alcance para ser possível a identificação de objetos a grandes distâncias.

2.4.4 Métrica de Avaliação do KITTI

A métrica de avaliação utilizada no *dataset* KITTI [26] para a tarefa de avaliação da deteção de objetos é baseada na *mean Average Precision* (mAP) com um limiar de *Intersection over Union* (IoU) para as tarefas de deteção de objetos em 3D e BEV.

Para a deteção de objetos 3D, são estabelecidos limiares de IoU específicos para cada classe. O limiar de IoU é de 0,7 para a classe "carro"e de 0,5 para as classes "peão"e "ciclista". Já para a tarefa de deteção BEV, é utilizado um limiar de IoU 2D entre as bboxes previstas e de referência.

Figura 2.11: Automóvel autónomo da Waymo.

A métrica de Avaliação do KITTI é calculada através das fórmulas de IoU, *precision* e *recall*. A IoU é definida pela seguinte fórmula:

$$IoU = \frac{volume(B_{proposto} \cap B_{verdadeiro})}{volume(B_{proposto} \cup B_{verdadeiro})} \tag{2.1}$$

Onde $B_{proposto}$ representa as bboxes previstas e $B_{verdadeiro}$ representa as bboxes de referência.

A *precision* (p) é definida como a razão entre o número de *True Positive* (TP) (N_{TP}) e o número total de deteções ($N_{TotalDetections}$):

$$p = \frac{N_{TP}}{N_{TotalDetections}} \tag{2.2}$$

A *recall* (r) é definida como a razão entre o número de TP (N_{TP}) e o número total de exemplos positivos reais ($N_{GroundTruth}$):

$$r = \frac{N_{TP}}{N_{GroundTruth}} \tag{2.3}$$

A curva de *Precision-Recall* (P-R) é construída ao representar a precisão no eixo y e a *recall* no eixo x para diferentes valores de limiar de IoU. A área sob a curva P-R é denominada *Average Precision* (AP).

A Métrica de mAP é calculada pela média da AP de todas as classes avaliadas.

Estas métricas proporcionam uma avaliação abrangente e quantitativa do desempenho do modelo de deteção de objetos, com base nas caixas delimitadoras previstas e nas caixas delimitadoras de referência fornecidas pelo *dataset* KITTI.

2.5 Estudo sobre Deteção de Objetos

Atualmente, existem inúmeros métodos diferentes para detetar e identificar objetos em *point clouds*. Contudo, é possível estabelecer uma base de arquitetura comum a todos eles. O seu funcionamento assenta em três pilares essenciais, nomeadamente *Sensor Data Representation* (SDR), *Feature Extraction* (FE) e *Core Object Detection* (COD).

2.5.1 Sensor Data Representation

No SDR, uma *point cloud* é representada com os dados obtidos pelo sensor LiDAR. Como a saída de dados do LiDAR representa uma *point cloud*, podendo conter pontos indesejados e dimensão indefinida (ver 2.12), será necessário proceder ao processamento da mesma antes de ser introduzida como *input* nos processos de extração de *features*.

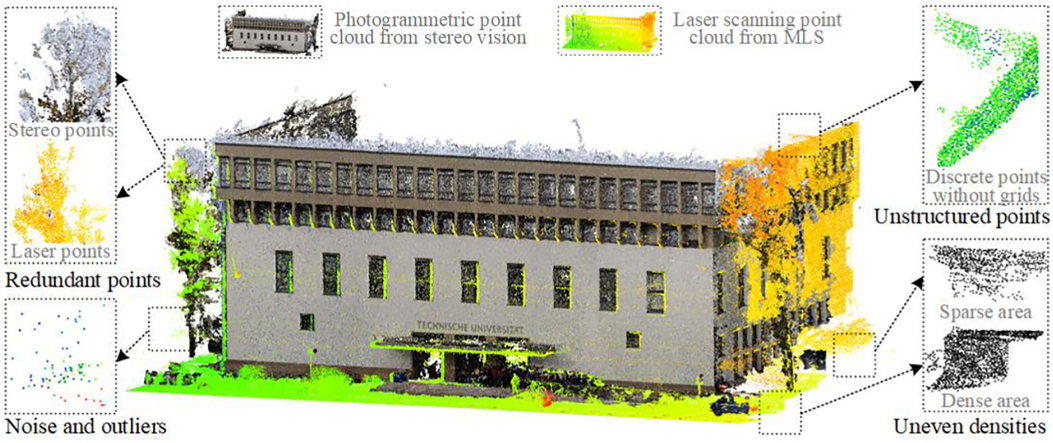


Figura 2.12: Obstáculos de usar uma *point cloud raw* [28].

A seguir, serão apresentadas as abordagens utilizadas no processamento e *downsampling* de uma *point cloud*.

Point-based

A *point cloud* recebida pelo sensor LiDAR passa por um processo de redução no número total de pontos (*downsampling*), utilizando métodos como *Random Sampling* e *Furthest Point Sampling*, a fim de limitar o número total de pontos a um valor N. Isto permite que a *point cloud* seja uma representação mais simples e objetiva do ambiente adquirido (ex. 2.13)[28].

Voxel-based

O *Voxel-based* consiste na associação de pontos a cubos de dimensão e posição pré-definidas, igualmente distribuída pelo espaço tridimensional. Dado um referencial

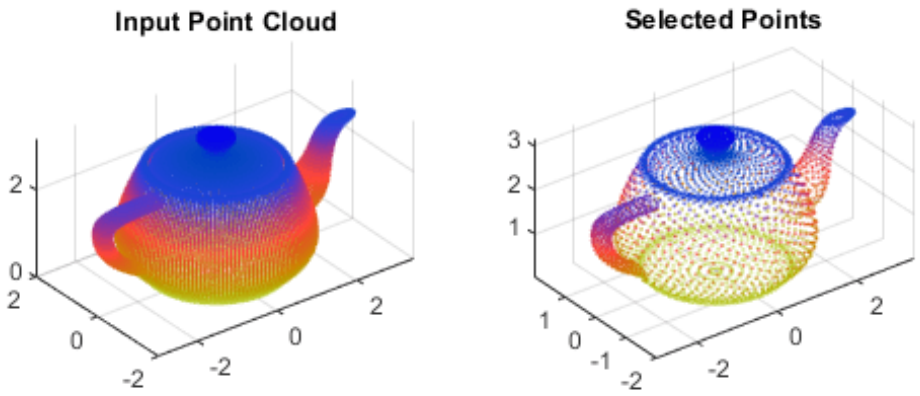


Figura 2.13: Aplicação de *downsampling* numa nuvem de pontos.

cartesiano 3D com uma dimensão $[X,Y,Z]$, repartido igualmente em cubos de dimensão definida $[u_X,u_Y,u_Z]$, um *voxel* será a representação de um cubo [28].

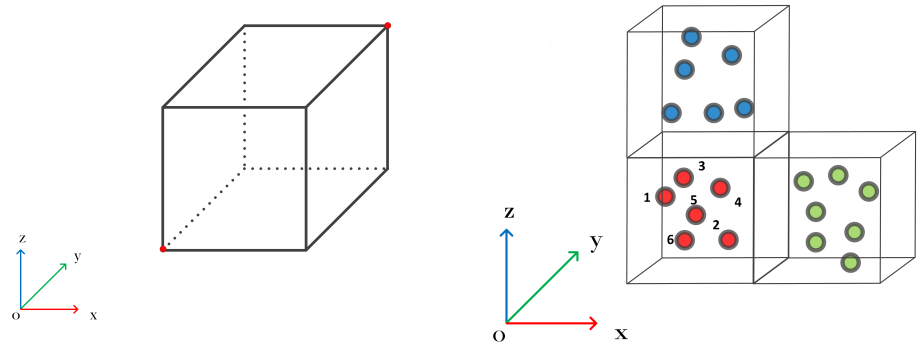


Figura 2.14: Representação de um cubo voxel e sua associação com a *point cloud* [29].

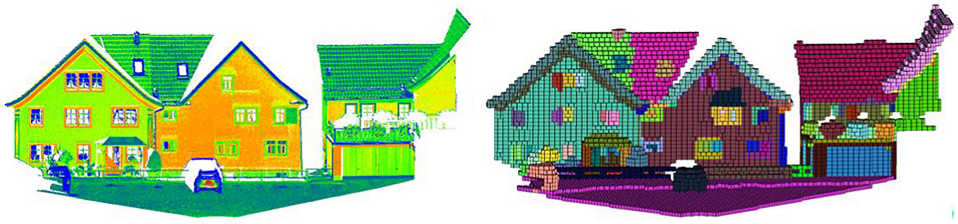
Os pontos pertencentes à *point cloud* são depois associados ao seu *voxel* pertencente com base na sua localização [30][29].

Pillar-based

As técnicas *pillar* repartem um espaço tridimensional $[X,Y,Z]$ igualmente em pilares de dimensão definida $[u_X,u_Y,Z]$, excluindo o eixo Z. A associação de pontos subsequente é semelhante aos métodos utilizados nas abordagens *voxel* [28][31][32].

Projection-based

Os pontos tridimensionais pertencentes à *point cloud* são representados num plano bidimensional por meio de duas técnicas - *Front-View* (FV) e BEV.



(a) (b)

Figura 2.15: Diferença entre a) *point cloud* original e b) renderizada em Voxels [30].

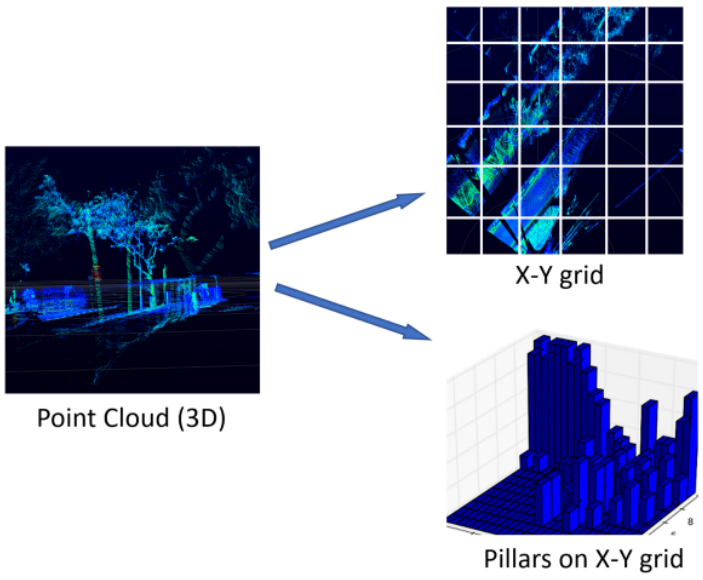


Figura 2.16: Transformação de pontos em pilares.

- **FV**: Esta técnica permite visualizar a *point cloud* num plano vertical 2D idêntico à informação obtida através de uma câmara, conforme observado na figura 2.17, com a penalização de ter menos precisão em detalhe do que a mesma. A redução significativa na quantidade de pontos proporcionada por esta técnica implica um menor poder computacional para a classificação de objetos, seja por meio de redes neuronais ou abordagens heurísticas/geométricas.

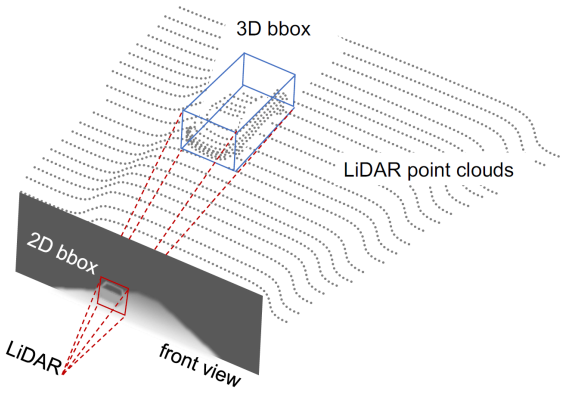


Figura 2.17: Front-View [33].

- **BEV**: Semelhante à abordagem anterior, nesta técnica a *point cloud* será transformada num plano 2D horizontal. Ao utilizar a informação de intensidade fornecida pelo LiDAR, é possível calcular com precisão as distâncias aos objetos, permitindo assim a visualização aérea do ambiente ao redor do sensor.

(a) (b)

Figura 2.18: Imagem a) com plano frontal convertida ao b) *bird's eye view* [34]

Graph-based

Neste método, o objetivo é a visualização da *point cloud* como um gráfico. No entanto, devido ao grande número de pontos presentes na *point cloud*, a aplicação direta desta técnica não é eficiente. Por este motivo, é comum utilizar uma técnica *voxel-based* como etapa inicial. Após a redução de tamanho da *point cloud*, é definido um raio fixo, denotado como r. Nesta abordagem, os pontos são considerados como

nós (*nodes*) e as suas ligações com os pontos na sua vizinhança dentro de um raio r como cantos (*edges*). Dada uma *point cloud* $P = \{p_1, ..., p_n\}$, onde $p_i = (x_i, r_i)$ representa um ponto tridimensional e com *initial feature* r_i, o gráfico será formulado da seguinte forma [35]:

$$G = (P, E), \quad E = (p_i, p_j) \mid \|x_i - x_j\|_2 < d \tag{2.4}$$

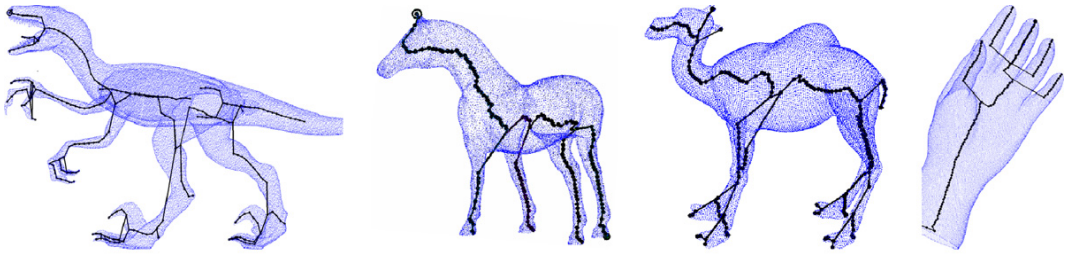


Figura 2.19: Graph-based [35].

2.5.2 Feature Extraction

Neste segundo bloco operacional, são aplicadas técnicas para extrair possíveis *features* da *point cloud* transformada no bloco SDR. No âmbito desta dissertação de mestrado, será abordada em detalhe uma técnica denominada por *Backbone Network* (BN), tanto na sua vertente *Three-Dimensional Backbone Network* (3DBN) como na sua vertente *Two-Dimensional Backbone Network* (2DBN). Estas duas vertentes focam-se em espaços diferentes, e a escolha da técnica a ser utilizada dependerá da abordagem aplicada no SDR. Caso a abordagem utilizada envolva a transformação de planos, como é o caso dos métodos FV e BEV da categoria *Projection-based*, não será possível utilizar a vertente 3DBN, uma vez que o *dataset* perdeu a sua característica tridimensional.

Uma 2DBN é composta por uma *Fully Convolutional Network* (FCN) aplicada numa estrutura de dados 2D limitada, tal como a saída de dados proveniente de algoritmos como o BEV e o FV. Existem diversas arquiteturas de *backbone*, porém, apenas quatro dessas implementações servem de base a muitas outras, nomeadamente a *Feature Pyramid Network* (FPN) [36], U-Net [37], ResNet [38] e VGG [39]. O funcionamento básico de uma 2DBN para deteção de objetos 3D envolve um sequência de etapas. Primeiramente, são aplicadas camadas de convolução 2D (2D Conv), seguidas por deformação 2D (2D DefConv), ou convoluções 2D dilatadas (2D DilConv). Em seguida, é utilizada a normalização *Batch* [40] e, por último, é aplicado um retificador *Rectified Linear Activation Unit* (ReLU) [41].

No contexto das 3DBNs, as abordagens frequentemente utilizadas incluem a CNN como primeira abordagem, seguindo-se as abordagens PointNet/PointNet++ [32][42] e, por último, os modelos *Graph Neural Network* (GNN) [43].

Convolutional Neural Network

As CNN substituíram o processo manual demorado de extrair *features* de *datasets* para identificar objetos. Este processo substituto proporciona uma abordagem escalável por meio de princípios de álgebra linear, em particular a multiplicação de matrizes, para identificar padrões que auxiliem na identificação e classificação de objetos. Embora esta abordagem seja relativamente eficiente e rápida com as combinações de treino adequadas [44], a mesma pode demonstrar-se computacionalmente exigente durante a fase de treino, dependendo da *Graphics Processing Unit* (GPU) disponível no sistema.

As CNN possuem três camadas principais: a primeira é a *convolution layer*, a segunda é a *pooling layer* e a terceira é a *fully-connected layer*. A complexidade da CNN aumenta progressivamente em cada camada. As primeiras camadas são responsáveis por identificarem pequenas *features*, como a cor (exclusivo em *datasets* provenientes de câmaras RBG) e os cantos (vértices e arestas). À medida que a *data* a ser analisada percorre as sucessivas camadas da CNN, o algoritmo começa a reconhecer elementos ou objetos de maior escala até, finalmente, conseguir identificar o objeto completo.

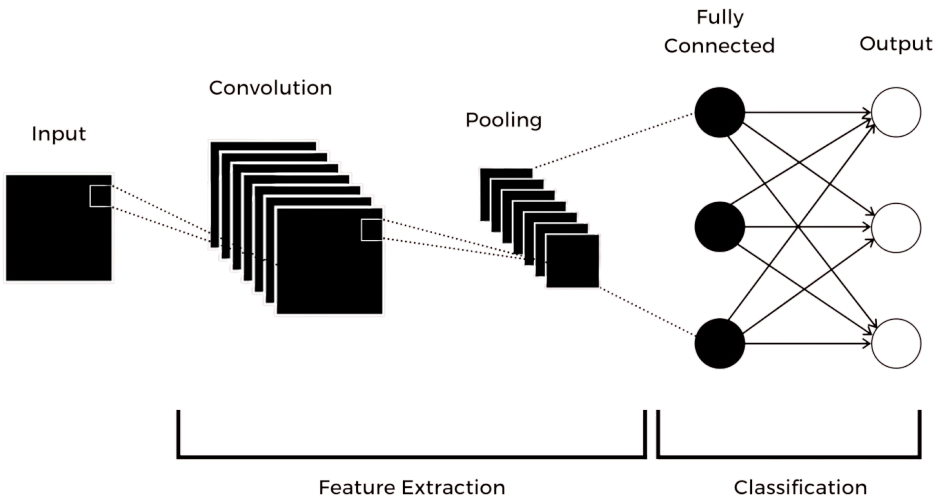


Figura 2.20: Camadas de uma CNN.

Dado que uma *point cloud* (3D) representa um *dataset* com uma estrutura irregular e uma maior quantidade de pontos comparativamente a uma imagem (2D), as

redes CNN, originalmente projetadas para processar imagens 2D, não são passíveis de serem aplicadas diretamente numa *point cloud* devido à elevada capacidade computacional que seria necessário para a processar. Portanto, para utilizar as redes CNN na análise de uma *point cloud*, é necessário realizar um pré-processamento dos dados, o que envolve transformar a *point cloud* num *dataset* mais regular e estruturado, por meio de métodos como *bird's eye view*, *front view* ou *voxels*. Com esta transformação, torna-se possível aplicar uma CNN 2D, seguida de uma CNN 3D, numa amostra mais reduzida de dados [45][46].

PointNet/PointNet++

A PointNet [32] foi o primeiro modelo robusto proposto para processar diretamente dados provenientes do sensor LiDAR, sem a necessidade de utilizar os métodos demonstrados no bloco operacional SDR. O seu funcionamento baseia-se em aprender uma codificação espacial de cada ponto para, posteriormente, agregar todos os pontos individuais e as suas *features* numa *point cloud* global. Este modelo é capaz de executar classificações e segmentações em estruturas de dados que contenham no máximo 100 pontos por *frame*, o que o torna menos adequado para processar *point clouds* provenientes de sensores LiDAR que produzem um grande *output* de dados, como é o caso de alguns sensores utilizados em condução autónoma, que conseguem atingir aproximadamente 100,000 pontos por *frame*.

O modelo, representado na figura 2.21, começa por receber n pontos como entrada para a rede de classificação, que posteriormente são submetidos a transformações. Utilizando *max pooling*, as *features* são agregadas e fornecidas a um bloco *multi-layer perceptron*, obtendo à saída os resultados das classificações para as classes desejadas (k). A rede de segmentação funciona de forma independente do modelo, executando apenas a concatenação das *features* globais e locais [32].

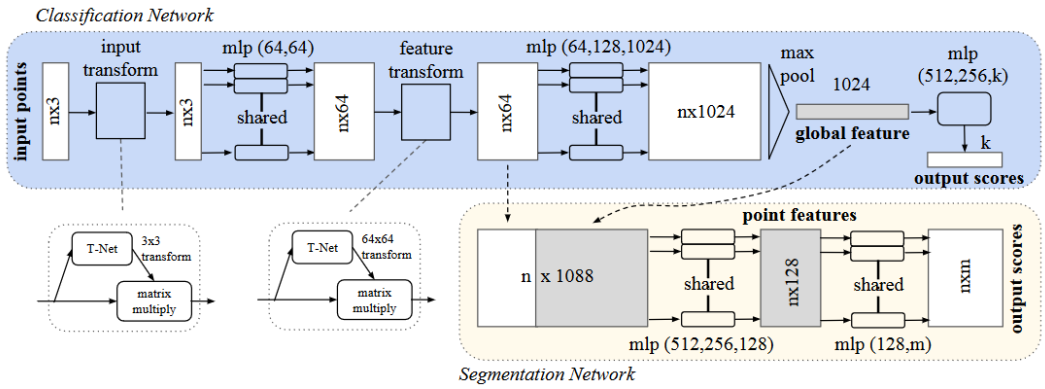


Figura 2.21: Arquitetura da PointNet [32].

Com base no modelo de funcionamento da *PointNet*, a *PointNet++* [42] propõe

camadas de aprendizagem que combinam *features* de múltiplas escalas com densidades diversas. As *features* locais são extraídas de uma pequena área e, em seguida, agrupadas em unidades maiores. Estas unidades são processadas para obter *features* de nível mais alto.. Este processo é repetido recursivamente até serem obtidas todas as *features* da *point cloud*.

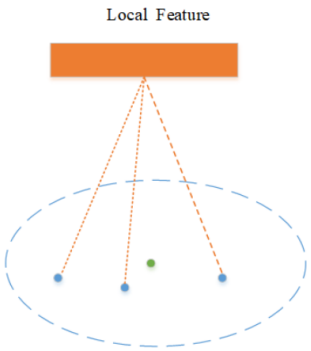


Figura 2.22: Aprendizagem de *features* da PointNet++.

As três camadas principais do modelo são a *sampling layer*, *grouping layer* e a *PointNet* layer. A primeira camada utiliza o método *farthest point sampling* para selecionar um conjunto de pontos da *point cloud* que defina o centróide de uma região local. Segue-se a *grouping layer*, onde a estas regiões locais são atribuídos um conjunto de pontos que se encontrem na vizinhança do seu centróide. Utilizando uma mini-*PointNet*, os conjuntos de pontos locais são abstraídos em representações de alto-nível.

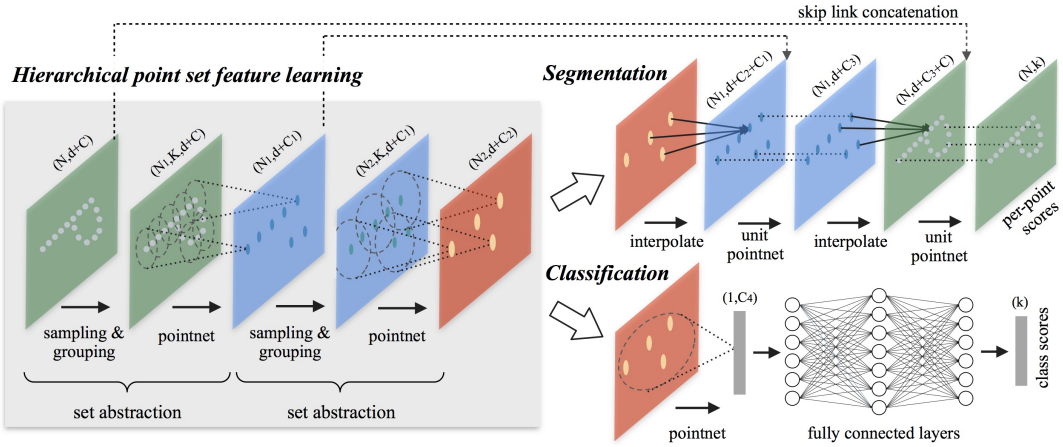


Figura 2.23: Ilustração da arquitetura de aprendizagem hierárquica de *features* e a sua aplicação em segmentação e classificação usando pontos no espaço euclidiano 2D como exemplo [42].

Graph Neural Network

Após a aplicação de uma abordagem *graph based*, a fim de obter uma *point cloud* livre de pontos desnecessários e mais leve (*downscale*), segue-se a utilização de uma 3DBN *Graph*. Esta implementação utiliza uma GNN para codificar *features* com recurso aos nós e cantos [43].

Quando não é possível observar o *dataset* num espaço tridimensional, como é o caso das *point clouds* processadas através de métodos como o BEV e o FV, é necessário a utilização de uma 2DBN. Esta *backbone* consiste numa FCN, onde é possível extrair *features* a partir de um *dataset* bidimensional e estruturado de entrada.

2.5.3 Core Object Detection

Por último, nesta etapa, utilizam-se as *features* extraídas no bloco anterior para realizar uma estimativa da sua classificação, considerando a classe do objeto e o seu tamanho, de modo a gerar uma caixa 3D ao redor do objeto, designada como bbox. Este bloco está dividido em duas categorias, as redes de deteção (*detector networks*) e o aprimoramento de previsão (*prediction refinement*). Em abordagens de deteção de objetos simples, apenas são utilizadas as redes de deteção para identificar objetos. Estas conseguem identificar todos os pontos pertencentes a uma *feature*, extraída previamente pelo bloco *feature extractor*, de modo a calcular um resultado de previsão e gerar uma bbox 3D ao redor do objeto. No entanto, para melhorar a precisão e a eficácia da previsão, algumas abordagens utilizam métodos adicionais de *prediction refinement*. Estas recebem como argumento de entrada a bbox 3D obtida pelas redes de deteção e refinam-na para extrair *features* adicionais, contribuindo para melhorar a classificação e o resultado da previsão, assim como estimar a posição, tamanho e orientação do objeto [28].

Detector Networks

As abordagens *Detector Network* (DN) podem ser categorizadas como *anchor-based* ou *anchor-free*. As abordagens *anchor-based* utilizam múltiplas âncoras de tamanhos predefinidos como referência para obter a classificação e regressão da localização e o tamanho de um objeto. Com uma abordagem diferente, as redes *anchor-free* utilizam as *binary labelelling* obtidas no módulo operacional FE para identificar pontos ou áreas que pertencem a um objeto. Em seguida, para cada ponto, é calculada uma previsão da sua bbox e *confidence score*.

Prediction Refinement

O modelo *Prediction Refinement* (PR) é utilizado exclusivamente na deteção de objetos em duas fases. Este modelo recebe como entrada a bbox proposta pela DN,

processa uma amostragem de alta precisão e extrai *features* adicionais para melhorar a classificação e a bbox calculada anteriormente pela DN.

Quanto à aplicação deste modelo em espaços tridimensionais, existem três categorias principais que distinguem as suas implementações:

- **Point-based:** Esta abordagem permite a utilização de uma representação natural de uma *point cloud*. Neste modelo, é utilizada uma pequena amostra de pontos da bbox proposta pela DN, que são codificados com as suas coordenadas, intensidades e classificações. Esses pontos são, posteriormente, introduzidos em arquiteturas, como a PointNet ou PointNet++, para refinar as suas classificações e BBox [47].

- **Graph-based:** Esta abordagem consiste em dois módulos, um para a extração de *features* locais (R-GCN) e outro para a extração de *features* globais. O módulo R-GCN processa a bbox proposta pela DN e codifica as *features* nos pontos corretos [47]. Após a codificação dos pontos, é construído um gráfico que é processado por camadas *Multimodal Relational Graph Convolutional Network* (MRGCN) [48] para obter um vetor de *features* locais. Em seguida, o gráfico é processado pela camada EdgeConv [49] para aprender *features* globais para todas as propostas. Para cada proposta, as *features* globais do C-GCN [50] são concatenadas com as *features* locais do R-GCN [51] e, em seguida, são introduzidas em duas *fully connected layers* para refinar a sua classificação e bbox.

- **Voxel-based:** Nesta abordagem, a bbox proposta pela DN é repartida em *voxels* e codificada com as *features* obtidas no bloco FE. As *features* de cada *voxel* são extraídas através de arquiteturas *VFE-based*, como a VoxelNet [52]. Em seguida, os *voxels* são processados por uma camada 3D SpConv (*sparce convolution network*), resultando num *downscale* do espaço 3D. Por fim, o uso de *fully connector layers* permite refinar a classificação e estimativa da bbox.

Quanto à sua aplicação em ambientes 2D, estes operam com o BEV para melhorar as bbox 3D. O modelo PR recebe uma bbox 2D fornecido pela DN e com a implementação de uma *fully connected layer* é estimada a cordenada z, sendo assim possível o refinamento da classificação da BBox 3D.

2.6 Detetores de Objetos End-to-End

Nesta secção, serão apresentadas algumas abordagens de deteção, classificação e segmentação de objetos. Além disso, serão discutidas as diferenças entre as várias abordagens e comparadas com o seu tempo de inferência com base no *dataset* KITTI.

Conforme demonstrado por Georgios Zamanakos et al. [28], não existe um método de deteção 3D que, considerando o mesmo *dataset*, seja superior em todos os aspetos relevantes, como precisão nas várias classes ou tempo de inferência. Algumas abordagens obtêm uma maior precisão na classificação de carros, enquanto outras alcançam melhor classificação de pedestres. As melhorias na precisão média das implementações de deteção de objetos com recurso a LiDAR têm aumentado significativamente num curto período de tempo, como ilustrado na figura 2.24. A implementação do VoxelNet [52] em 2018, considerada na altura a implementação estado-de-arte para dados 3D provenientes de LiDAR, conseguiu uma precisão de 65,11% no KITTI (classe carro com dificuldade moderada). Três anos depois, são nos apresentados valores em torno de 81,17% com a proposta do CenterPoint [53].

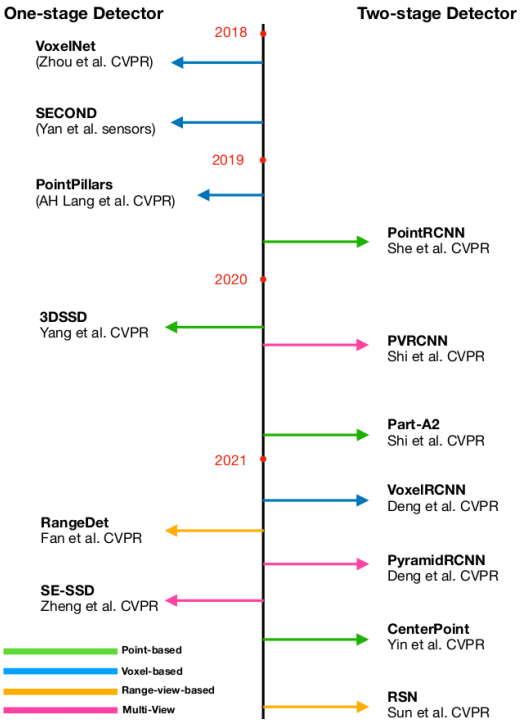


Figura 2.24: Ordem cronológica dos detetores de objetos tridimensionais [1].

No entanto, os métodos de deteção de objetos nos *benchmarks* existentes são avaliados principalmente com base na sua precisão. Por conseguinte, é encorajador a utilização de mais recursos computacionais no desenvolvimento de novos métodos de deteção, como o aproveitamento de altas capacidades de memória e frequências de relógio das novas arquiteturas de *Central Processing Unit* (CPU) e GPU, a fim de alcançar um aumento percentual extra na precisão. Isto leva-nos a uma situação em que é difícil determinar se o aumento de performance trazido pelos novos detetores é resultado da nova abordagem proposta, ou se é uma consequência da utilização de mais recursos computacionais [54].

2.6.1 VoxelNet

O método VoxelNet [52] baseia-se na representação voxelizada dos dados LiDAR. Ele divide o espaço 3D numa grelha tridimensional de voxels e transforma a nuvem de pontos numa representação estruturada de voxels ocupados. Esta abordagem permite que a rede neuronal convolucional processe diretamente dos voxels, aproveitando a estrutura espacial dos dados. A arquitetura do VoxelNet utiliza camadas convolucionais 3D para extrair características discriminativas dos voxels e realizar a deteção de objetos. O VoxelNet demonstrou desempenho promissor em termos de precisão, porém, pode ser computacionalmente exigente devido ao elevado número de voxels.

2.6.2 PointPillars

O método PointPillars [31] apresenta-se como uma abordagem alternativa à representação voxelizada. Em vez de utilizar voxels, o PointPillars utiliza uma representação baseada em pilares (pillars) e pontos contidos nesses pilares. Ele divide o espaço 3D em pilares verticais e projeta os pontos da nuvem em pilares 2D, que são tratados como canais de entrada para a rede neuronal convolucional. Esta representação reduz a dimensionalidade dos dados e melhora a eficiência computacional. O PointPillars utiliza camadas convolucionais 2D para processar os pilares e obter informações sobre a presença de objetos. Esta abordagem apresenta uma combinação de eficiência e desempenho, sendo capaz de atingir alta precisão com menor custo computacional em comparação com o VoxelNet.

2.6.3 SECOND

O método *Sparsely Embedded Convolutional Detection* (SECOND) [55] também é baseado na representação voxelizada dos dados LiDAR, semelhante ao VoxelNet. Ele utiliza um Voxel Feature Encoding (VFE) para extrair características da nuvem de pontos e gerar uma representação voxelizada. No entanto, o SECOND introduz uma camada de convolução esparsa (*sparse convolution*) para processar eficientemente a representação voxel, reduzindo significativamente o custo computacional em comparação com as redes convolucionais tradicionais. A arquitetura do SECOND inclui uma *Region Proposal Network* (RPN) que gera bboxes e realiza a classificação dos objetos. Esta abordagem destaca-se pela sua eficiência e capacidade de lidar com nuvens de pontos esparsas, proporcionando um bom equilíbrio entre precisão e velocidade de inferência.

2.6.4 CenterPoint

O método CenterPoint [53] é uma abordagem recente para a deteção 3D de objetos que se concentra na deteção precisa dos centros dos objetos. Ele utiliza a representação ponto-a-ponto dos dados LiDAR, onde cada ponto é considerado como uma unidade de entrada. O CenterPoint adota uma arquitetura de rede neuronal convolucional que processa diretamente os pontos, aplicando módulos de atenção espacial (*Spatial Attention Modules*) para identificar os centros dos objetos. Esta abordagem evita a necessidade de representações volumétricas, como voxels ou pilares, resultando numa representação mais eficiente em termos de memória e processamento. O CenterPoint demonstrou bons resultados em termos de precisão e velocidade de inferência, sendo particularmente adequado para lidar com objetos de diferentes tamanhos e densidades.

2.6.5 Benchmark

Tabela 2.2: Comparação de performance na deteção BEV: precisão média (%) no KITTI **dataset** com dificuldade moderada. Todas as inferências foram executadas com uma NVIDIA GTX1080Ti, com exceção do VoxelNet, onde foi utilizada uma NVIDIA TITAN X [2]. *Este tempo de inferência pode ser reduzido caso se utilize o TensorRT em vez do PyTorch [1].

	Carro			Peão			
Método	Fácil	Moderado	Difícil	Fácil	Moderado	Difícil	Tempo
VoxelNet [52]	89.35	79.26	77.39	46.13	40.74	38.11	0.223
PointPillars [31]	90.07	86.56	82.81	57.60	48.64	45.78	0.024*
SECOND [55]	88.07	79.37	77.95	55.10	46.27	44.76	0.050
CenterPoint [53]	88.92	86.29	85.22	60.60	56.23	53.67	0.090

Tabela 2.3: Comparação de performance na deteção 3D: precisão média (%) no KITTI **dataset** com dificuldade moderada. Todas as inferências foram executadas com uma NVIDIA GTX1080Ti, com excepção do VoxelNet, onde foi utilizada uma NVIDIA TITAN X [2]. *Este tempo de inferência pode ser reduzido caso se utilize o TensorRT em vez do PyTorch [1].

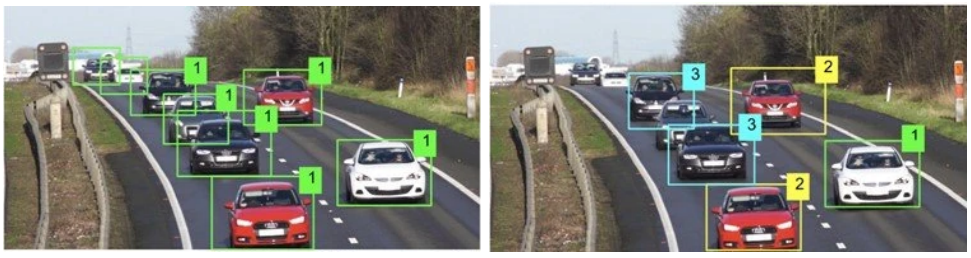
	Carro			Peão			
Método	Fácil	Moderado	Difícil	Fácil	Moderado	Difícil	Tempo
VoxelNet [52]	77.47	65.11	57.73	39.48	33.69	31.51	0.223
PointPillars [31]	82.58	74.31	68.99	51.45	41.92	38.39	0.024*
SECOND [55]	83.13	73.66	66.20	51.07	42.56	37.29	0.050
CenterPoint [53]	85.59	75.56	73.52	56.85	52.33	47.84	0.090

2.6.6 Revisão das metodologias

Cada um dos modelos de deteção apresentados neste capítulo é capaz de cumprir os objetivos previamente definidos para a solução final desta dissertação (capítulo 1.1). No entanto, existem pequenas vantagens e desvantagens entre cada um deles. O CenterPoint não tem concorrência quando comparadas as AP em todo o espectro de classes e dificuldade, no entanto, o seu tempo de inferência só não é superior ao do VoxelNet, tornando-o num candidato não tão apelativo para aplicações em tempo real. São facilmente encontradas implementações do VoxelNet devido à sua popularidade, no entanto os seus resultados de AP em todo o espectro juntamente com o seu elevado tempo de inferência tornam-no num método ultrapassado pelos seus sucessores. Muito semelhantes nas métricas de AP encontram-se o PointPillars e o SECOND, com o primeiro a ganhar vantagem numa redução de tempo de inferência. No entanto, o método SECOND, utilizando um SDR *voxel-based*, evita depender apenas da densidade de pontos na nuvem original, sendo especialmente útil em áreas com densidade de pontos baixa ou irregular, onde o método *pillar-based*, como o PointPillars, pode encontrar desafios na representação adequada dessas regiões.

2.7 Tracking

O desenvolvimento e evolução de tecnologias que permitem que os sistemas computacionais detetem os objetos criaram o ambiente propício para que surgisse a necessidade de os conseguir seguir. O *tracking*, termo inglês proveniente de *object tracking*, consiste no processo de seguir um objeto. Neste caso, a aplicação utiliza as deteções executadas previamente pelos algoritmos de deteção e gera uma identificação única para cada um deles. Além disso, possibilita a previsão da trajetória do objeto, permitindo o seu *tracking* mesmo que o objeto fique temporariamente oculto da visão sensorial do sistema [56].



(a) Object Detection (b) Object Tracking

Figura 2.25: Diferença entre a) classificação obtida por algoritmos de deteção de objetos e b) identificação de cada objeto detetado por métodos de *object tracking*.

Apesar de ser possível aplicar os algoritmos de deteção mencionados anteriormente em cada *frame* para obter sucessivamente as bboxes, essa abordagem é extremamente ineficiente tanto em termos de poder computacional necessário quanto em tempo de execução [57]. Uma alternativa eficiente é utilizar algoritmos de deteção nos primeiros *frames*, seguindo por métodos eficientes de *tracking*. Esta abordagem é a mais adequada e eficiente para dimensionar sistemas inteligentes de deteção e *tracking* de objetos.

Os algoritmos de *object tracking* estão subdivididos em *Single Object Tracking* (SOT) e *Multiple Object Tracking* (MOT), os quais serão apresentados nas próximas subsecções.

2.7.1 SOT

O algoritmo SOT, também conhecido como *visual object tracking*, é amplamente utilizado para o *tracking* de objetos. Ele requer a definição do objeto a ser seguido, processando-o na primeira imagem fornecida como entrada. É importante salientar que apenas um objeto pode ser seguido utilizando este algoritmo, independentemente do número total de objetos presentes.

O SOT é classificado como uma abordagem que não depende necessariamente da deteção prévia de objetos (*detection-free tracking)*. Em vez disso, é necessário fornecer manualmente um número fixo de objetos na primeira imagem, que são subsequentemente localizados nas imagens seguintes. Isso significa que o algoritmo não é capaz de lidar com cenários em que novos objetos aparecem espontaneamente na cena. Por essa razão, o SOT é frequentemente utilizado em sistemas de monitorização e navegação, onde a presença de novos objetos não é um problema comum.

Algoritmos como o SiamMask [58] são exemplos de abordagens eficientes de SOT. Esses algoritmos necessitam apenas de uma imagem de referência para inicialização e conseguem gerar máscaras segmentadas que delimitam o objeto de interesse. Além disso, são capazes de executar essa tarefa em tempo real, com taxas de até 35 *frames* por segundo, o que os torna adequados para aplicações que exigem *tracking* em tempo real.

2.7.2 MOT

Ao contrário dos algoritmos SOT, os métodos MOT utilizam uma abordagem baseada na deteção (*tracking-by-detection*). Isso envolve o uso de um detetor de objetos independente que processa todos os *frames*, gerando assim bboxes. Estas bboxes são, posteriormente, processadas por algoritmos de *tracking* para atribuir um identificador único (ID) a cada objeto.

Estes sistemas são amplamente utilizados em aplicações de condução autónoma devido à sua capacidade de associar rapidamente os resultados dos algoritmos de

deteção de objetos sem cometer erros na troca de identidades entre múltiplos objetos. Abordagens como o *Simple Online and Realtime Tracking* (SORT) [59] beneficiam do uso de filtros Kalman e do algoritmo de associação *Hungarian* [60] para rastrear objetos com alto desempenho. No entanto, existe a possibilidade de um objeto receber um ID diferente se desaparecer e reaparecer posteriormente [56].

Para resolver parcialmente esse problema, foi proposta uma abordagem chamada DeepSORT [61], que combina o SORT com uma rede de re-identificação de pedestres capaz de extrair *features* dos indivíduos detetados, possibilitando a categorização adequada quando eles reaparecem.

Embora os algoritmos mencionados tenham demonstrado sua eficiência, não é possível aplicar diretamente algoritmos baseados no processamento de imagens ao processamento de *point clouds*, devido à sua maior dimensão. Para contornar este desafio, podem ser aplicadas técnicas de projeção para transformar objetos tridimensionais em múltiplas imagens bidimensionais (*frames*). No entanto, surge o dilema de que um grande número de projeções aumentará a carga computacional necessária, enquanto um número reduzido de projeções resultará em falta de informações necessárias para uma precisão adequada. Assim como ocorreu com a adaptação de métodos de deteção de objetos para trabalhar com dados tridimensionais, novos algoritmos de *tracking* têm sido propostos. Um exemplo é o AB3DMOT [62], que estende o algoritmo SORT para a terceira dimensão.

2.7.3 Revisão das metodologias

Uma das principais diferenças entre SOT e MOT é a capacidade de lidar com múltiplos objetos em movimento. Enquanto o SOT se concentra no *tracking* de um objeto específico, o MOT lida com a associação e *tracking* de vários objetos simultaneamente. Tendo em conta os objetivos definidos previamente para esta dissertação, é de extrema importância a identificação correta de não apenas um, mas vários objetos ou pessoas, de modo a conseguir desenhar o trajeto de cada objeto único. O SOT é adequado quando o número de objetos é limitado e não há uma mudança significativa na composição da cena, enquanto que o MOT é mais apropriado para cenários com múltiplos objetos em movimento, onde a capacidade de *tracking* e associação de objetos é fundamental.

Capítulo 3

Fundamentos

Este capítulo apresenta os fundamentos relacionados ao desenvolvimento de sistemas autónomos e à deteção e *tracking* de objetos. Começa com uma introdução ao ROS e destaca a importância do LiDAR na geração de *point clouds* detalhadas do ambiente. O modelo SECOND é explorado como uma arquitetura eficiente para a deteção de objetos em nuvens de pontos 3D, utilizando DL. A função de custo do detetor SECOND, que combina vários componentes para garantir uma deteção precisa, é discutida.

3.1 ROS

A escolha da plataforma base para esta dissertação recaiu sobre o ROS. Além de ser uma ferramenta muito familiar aos alunos que frequentam o mestrado de MEECSA, o ROS é uma estrutura gratuita e *open-source* que define previamente os componentes, interfaces e ferramentas necessárias para o desenvolvimento e construção de robôs autónomos.

O ROS oferece uma ampla gama de recursos e funcionalidades que são especialmente úteis para o desenvolvimento de sistemas de deteção e *tracking* utilizando sensores LiDAR. A capacidade do ROS em utilizar mensagens e tópicos torna-o uma ferramenta poderosa para coordenar os dados provenientes de vários sensores, como um sensor LiDAR e uma câmara, simultaneamente. Os dados de diferentes sensores podem ser publicados em tópicos específicos e, em seguida, consumidos por diferentes *nodes* do ROS para processamento, análise e tomada de decisões.

Uma das vantagens do ROS é a capacidade de gravar as comunicações entre os diferentes *nodes* num arquivo chamado *bag*. Esse arquivo regista todas as mensagens dos tópicos ativos durante a gravação, permitindo reconstruir posteriormente o ambiente onde os dados foram recolhidos. Esta funcionalidade é especialmente útil no desenvolvimento de algoritmos de deteção e *tracking*, pois possibilita a análise e validação dos resultados num ambiente controlado e reproduzível, sem a necessidade de submeter o robô ou os sensores a inúmeros testes em campo.

Além disso, o ROS beneficia de uma comunidade ativa de contribuidores e de uma vasta biblioteca de pacotes e ferramentas disponíveis. Esses pacotes abrangem desde drivers para diferentes sensores, incluindo LiDAR, até algoritmos de processamento de dados e deteção de objetos. Esta diversidade de recursos facilita o desenvolvimento e a integração de soluções completas de deteção e *tracking* utilizando sensores LiDAR.

Assim, ao utilizar o ROS como base para esta dissertação, há benefícios decorrentes não apenas da familiaridade com a plataforma, mas também das funcionalidades poderosas do ROS para o processamento e integração de dados de sensores, bem como do suporte da comunidade e da ampla variedade de pacotes disponíveis.

3.2 LiDAR

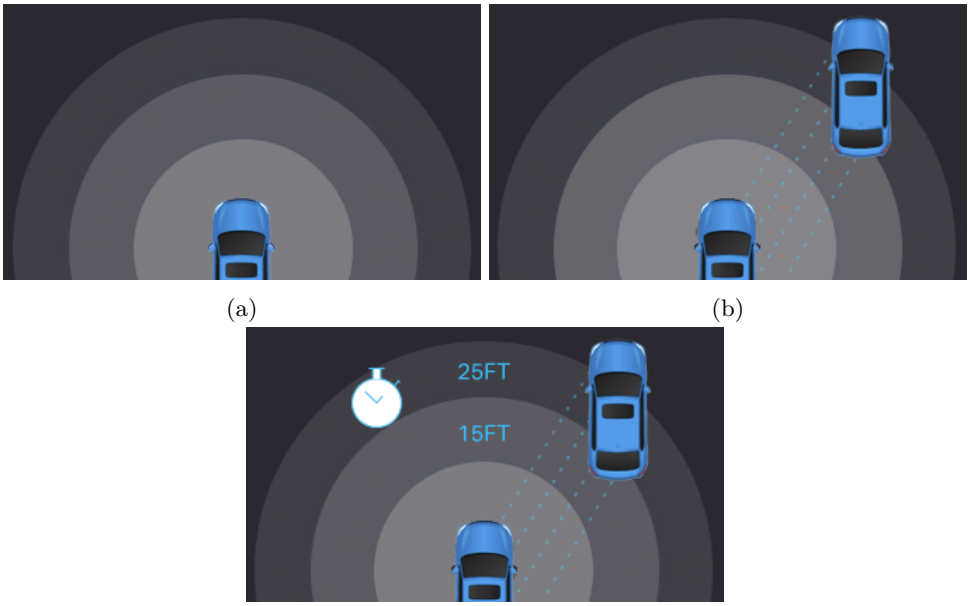
O LiDAR é uma tecnologia avançada que utiliza feixes de laser para extrair informações tridimensionais do ambiente circundante. O sensor emite pulsos de luz em direção aos objetos ao seu redor, conforme ilustrado na Figura 3.1a. Os pulsos de luz refletem nos objetos e retornam ao sensor, conforme demonstrado na Figura 3.1b. Através da medição da diferença de tempo entre o envio e a receção dos pulsos (*Time of Flight* (ToF)) e da multiplicação dessa diferença pela velocidade da luz, é possível estimar a distância até aos objetos que refletiram os pulsos, tal como exemplificado na Figura 3.1c. Este processo ocorre em alta frequência, permitindo a construção de um mapa tridimensional detalhado do ambiente [63].

O LiDAR fornece as coordenadas 3D relativas ao sensor, as quais podem ser calculadas da seguinte forma:

$$\begin{aligned} x &= d \cdot \cos(\omega) \cdot \cos(\phi) \\ y &= d \cdot \cos(\omega) \cdot \sin(\phi) \\ z &= d \cdot \sin(\omega) \end{aligned} \tag{3.1}$$

Aqui, d representa a distância medida por meio do ToF, ϕ é o ângulo de rotação (yaw) em relação ao eixo Z, e ω é o ângulo de inclinação ($pitch$) para cada feixe de laser emitido [28].

O sensor LiDAR produz uma nuvem de pontos, também conhecida como *point cloud*, em que cada ponto contém coordenadas 3D (xyz) que representam a sua



(a) (b)

(c)

Figura 3.1: Funcionamento de um sensor LiDAR.

posição espacial, juntamente com informações adicionais, como a intensidade de retorno (r). Alguns sensores LiDAR também fornecem uma dimensão adicional denominada por elongação (e), que indica a extensão do pulso de laser além de sua largura nominal [64].

A saída de dados do sensor LiDAR, composta pelas coordenadas 3D, intensidade de retorno e, quando aplicável, a elongação, constitui uma nuvem de pontos que representa uma visão detalhada e tridimensional do ambiente circundante. Esta informação é essencial para o desenvolvimento de algoritmos de deteção e *tracking* de pessoas e objetos, permitindo a identificação, localização e análise dos elementos presentes no ambiente.

3.3 Deteção

O modelo SECOND destacou-se como uma arquitetura eficiente e eficaz para a deteção de objetos em nuvens de pontos 3D. No seu ano de lançamento, introduziu no mercado uma abordagem inovadora ao utilizar convoluções esparsas para processar diretamente os dados provenientes do LiDAR, o que permite uma representação mais rica das informações espaciais e estruturais presentes numa nuvem de pontos [55].

Além disso, o modelo SECOND possui uma arquitetura modular e flexível. A sua escalabilidade possibilita obter resultados de precisão comparáveis aos detetores complexos mais recentes, mantendo tempos de inferência consideravelmente inferiores, conforme demonstrado em [54].

Outro fator relevante é a disponibilidade de implementações e recursos existentes em repositórios remotos, que contêm código bem documentado e atualizado, o que simplifica significativamente o processo de integração do modelo em outros projetos [2].

Com bases nas considerações mencionadas anteriormente, a decisão de implementar o modelo SECOND foi tomada atendendo à sua capacidade de lidar eficientemente com dados esparsos provenientes do LiDAR, à sua arquitetura modular e flexível, e à disponibilidade de recursos e implementações já existentes. Estes fatores contribuem para uma integração mais suave e eficiente do modelo em projetos relacionados.

3.3.1 SECOND

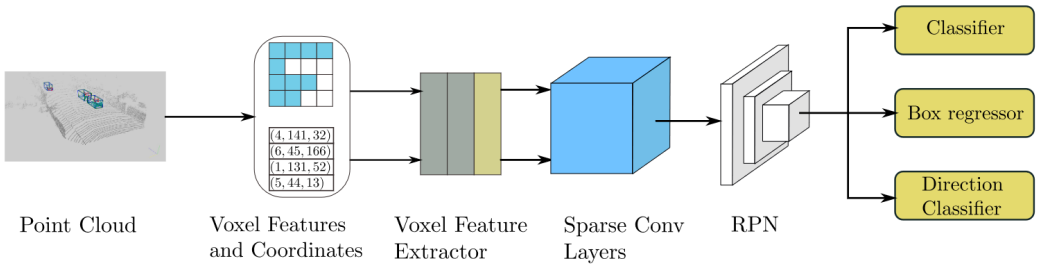


Figura 3.2: Arquitetura proposta pelo SECOND [55].

O SECOND [55] é um detetor baseado em DL composto por três componentes principais: (1) um *Voxel Feature Encoding* (VFE) semelhante ao utilizado no VoxelNet [52], (2) uma camada de convolução esparsa (*sparse convolution*) como camada intermédia, e (3) uma RPN. A arquitetura proposta pelo SECOND é ilustrada na Figura 3.2.

A camada VFE extrai *features* da nuvem de pontos e gera uma representação em forma de voxels. Essa representação em voxels permite uma manipulação mais eficiente dos dados tridimensionais. No VFE do SECOND, cada voxel contém informações como intensidade, coordenadas espaciais e características de reflectância. Além disso, uma rede de convolução 3D é aplicada aos voxels para extrair recursos semânticos. Esta combinação de informações espaciais e semânticas permite que o VFE capture com precisão os detalhes dos objetos na nuvem de pontos.

Em seguida, a camada de convolução esparsa processa a representação voxel através de convoluções esparsas. As convoluções esparsas concentram-se apenas nos voxels relevantes, evitando cálculos desnecessários em voxels vazios ou com pouca informação. Isso resulta numa redução significativa na quantidade de cálculos necessários, tornando o treino e a inferência mais eficientes em termos de tempo e recursos computacionais.

A RPN recebe como entrada o resultado da camada de convolução esparsa, gerando um conjunto de bboxes candidatas, juntamente com suas classificações e pontuações de confiança. Através da aplicação de filtros de convolução 3D, a RPN analisa as *features* esparsas obtidas na etapa anterior para localizar e classificar os objetos presentes na cena. O conjunto resultante de bboxes contém as deteções propostas pela rede e é posteriormente utilizado para realizar a deteção e o *tracking* de pessoas e objetos na nuvem de pontos.

Função de Custo

A função de custo utilizada no detetor SECOND [55] é composta por várias componentes, cada uma desempenhando um papel específico no treino do modelo. A combinação dessas componentes resulta na função de custo final.

A primeira componente é a *focal loss*, que é utilizada para lidar com o desequilíbrio de classes na deteção de objetos. A *focal loss* é definida da seguinte forma:

$$\mathrm{FL}(p_t) = -\alpha_t(1-p_t)^{\gamma}\log(p_t) \tag{3.2}$$

Nesta equação, p_t representa a probabilidade prevista para a classe do objeto, α_t é um fator de ponderação que controla o equilíbrio entre as classes e γ é um parâmetro que ajusta o grau de foco na correção dos exemplos difíceis. A *focal loss* penaliza de forma mais intensa as previsões incorretas para exemplos difíceis, enquanto reduz a penalização para exemplos fáceis.

A segunda componente é a *SmoothL1 loss*, que é utilizada para a regressão das bboxes. Esta função de custo lida com a presença de *outliers*, reduzindo o seu impacto na estimativa precisa das coordenadas da caixa delimitadora. A *SmoothL1 loss* é definida da seguinte forma:

$$\mathrm{SmoothL1}(x) = \begin{cases} 0.5x^2, & \text{se } |x| < 1 \\ |x| - 0.5, & \text{caso contrário} \end{cases} \tag{3.3}$$

A terceira componente é uma forma aprimorada de regressão de ângulo (*angle loss regression*) para estimar a orientação dos objetos. Esta função de custo utiliza a diferença de ângulos entre a orientação prevista e a orientação verdadeira para medir a precisão da estimativa. A fórmula da regressão de ângulo é definida da seguinte forma:

$$L_\theta = \mathrm{SmoothL1}(\sin(\theta_p - \theta_t)) \tag{3.4}$$

Nesta equação, θ_p representa o ângulo previsto e θ_t é o ângulo verdadeiro.

Por fim, a função de custo total (L_{total}) é calculada como uma combinação ponderada das componentes mencionadas anteriormente, utilizando hiperparâmetros (β_1,

β_2, β_3) para ajustar a importância relativa de cada componente na função de custo final. A fórmula da função de custo total é definida da seguinte forma:

$$L_{\text{total}} = \beta_1 L_{\text{cls}} + \beta_2(L_{\text{reg}} - \theta + L_{\text{reg-outros}}) + \beta_3 L_{\text{dir}} \tag{3.5}$$

É importante destacar que os valores dos hiperparâmetros devem ser ajustados durante o treino do modelo, a fim de obter o melhor desempenho possível.

A função de custo descrita acima desempenha um papel crucial no processo de treino do detetor SECOND, permitindo a aprendizagem e o aprimoramento das previsões de classe, regressão das bboxes e estimativa de orientação dos objetos.

3.4 Tracking

A abordagem adotada neste trabalho consiste em realizar o *tracking* de objetos com recurso à metodologia MOT. A implementação desta metodologia começa por um algoritmo designado de *Hungarian Algorithm*, com o intuito de associar os dados de deteção às trajetórias dos objetos. Esta é uma técnica eficiente, que visa encontrar a atribuição ótima entre as deteções e as trajetórias existentes, levando em consideração as distâncias ou similaridades entre elas. Após a associação dos dados, é aplicado um filtro de Kalman, um método estatístico de filtragem, para estimar e prever a posição e a velocidade dos objetos seguidos ao longo do tempo. O filtro de Kalman utiliza modelos dinâmicos e informações de medição para realizar uma fusão inteligente dos dados, permitindo estimativas mais precisas, mesmo em cenários com presença de ruído ou incerteza.

Esta combinação de técnicas de correspondência e filtragem possibilita a obtenção de resultados robustos e confiáveis no *tracking* de objetos em tempo real. Esta abordagem é amplamente aplicada em diversas áreas, como condução autónoma, vigilância e reconhecimento de atividades [62] [61].

3.4.1 Hungarian Algorithm

A deteção e o *tracking* são tarefas fundamentais na visão computacional moderna, com inúmeras aplicações que vão desde sistemas de vigilância [7] até sistemas autónomos industriais [65]. Um dos principais desafios dessas tarefas é o estabelecimento de correspondências entre objetos em diferentes *frames* para garantir uma associação e continuidade precisas. O *hungarian algorithm* [60], também conhecido como algoritmo de Munkres, oferece uma solução eficiente para o problema de associação de dados, minimizando o custo total das atribuições entre objetos.

Desenvolvido por Harold Kuhn em 1955 e redescoberto por James Munkres em 1957, o *hungarian algorithm* é uma técnica de otimização combinatória utilizada para resolver problemas de atribuição. No contexto do *tracking* de objetos, o problema de

atribuição visa encontrar a melhor atribuição de objetos a recursos, dada uma matriz de custo que quantifica o custo de associação entre objetos e recursos. O objetivo é encontrar a atribuição ótima que minimize o custo total, levando em consideração fatores como a forma dos objetos, o movimento e as relações espaciais [60].

Dada uma matriz (Tabela 3.1) com o mesmo número de linhas e colunas que representa as deteções e *tracks* existentes, o objetivo é associar as deteções aos *tracks* com base na maior área de sobreposição IoU ou na menor distância euclidiana.

	Track 1	Track 2	Track 3	Track 4
Deteção 1	76	32	85	48
Deteção 2	11	94	27	72
Deteção 3	63	19	53	89
Deteção 4	36	41	68	97

Tabela 3.1: Matriz de associação

Ao utilizar o *hungarian algorithm* no *tracking* com o objetivo de associar deteções e *tracks* com a maior área de sobreposição IoU possível, estamos perante uma associação designada de *maximization problem*. No entanto, para uma associação em *maximization problem* ser bem sucedida, é necessário converter a nossa matriz inicial para um estado de *minimization problem* - isto significa subtrair o valor mais elevado em cada célula da matriz original, que no caso da matriz 3.1 é 97.

	Track 1	Track 2	Track 3	Track 4
Deteção 1	21	65	12	49
Deteção 2	86	3	70	25
Deteção 3	34	78	44	8
Deteção 4	61	56	29	0

Tabela 3.2: Matriz de associação após a conversão para um estado de *minimization problem*.

Após a subtração do valor mais elevado em cada célula da matriz original, procede-se à subtração do menor valor em cada linha da matriz resultante.

	Track 1	Track 2	Track 3	Track 4
Deteção 1	9	53	0	37
Deteção 2	83	0	67	22
Deteção 3	26	70	36	0
Deteção 4	61	56	29	0

Tabela 3.3: Matriz de associação após a subtração do menor valor em cada linha da matriz 3.2.

Além dos passos mencionados anteriormente, é necessário subtrair o menor valor de cada coluna, na própria coluna.

	Track 1	Track 2	Track 3	Track 4
Deteção 1	0	53	0	37
Deteção 2	74	0	67	22
Deteção 3	17	70	36	0
Deteção 4	52	56	29	0

Tabela 3.4: Matriz de associação após a subtração do menor valor de cada coluna da matriz 3.3.

Com este processo, o objetivo é determinar o número mínimo de linhas necessárias traçar para abranger todas as células que possuem um valor de zero. O algoritmo estabelece que, para uma matriz NxN, deveremos de ter um valor mínimo de linhas igual a N.

Conforme observado na tabela 3.4, o atual número mínimo de linhas é três. Isto significa que será necessário efetuar um prolongamento do *algoritmo*, no qual serão criados zeros adicionais. Esta etapa implica a subtração das as células não traçadas com linha pelo menor valor que não se encontra numa linha ou coluna traçada, adicionando o valor subtraído aos elementos que já estão traçados duas vezes.

	Track 1	Track 2	Track 3	Track 4
Deteção 1	0	53	0	54
Deteção 2	74	0	67	39
Deteção 3	0	53	19	0
Deteção 4	35	39	12	0

Tabela 3.5: Matriz de associação após a subtração das células não traçadas da matriz 3.4.

Com a necessidade de traçar quatro linhas para cobrir todos os zeros, consegue-se alcançar a associação ótima.

	Track 1	Track 2	Track 3	Track 4
Deteção 1	0	53	0	54
Deteção 2	74	0	67	39
Deteção 3	0	53	19	0
Deteção 4	35	39	12	0

Tabela 3.6: Matriz de associação final.

3.4.2 Filtro de Kalman

O filtro de Kalman é um algoritmo de estimativa amplamente utilizado que desempenha um papel fundamental em várias aplicações, como o *tracking* de objetos, sistemas de navegação e fusão de sensores. Este filtro foi desenvolvido por Rudolf Kalman na década de 1960 e, desde então, é amplamente adotado devido à sua eficácia e versatilidade [66]. É especialmente adequado para situações em que é necessário

estimar o estado de um sistema com base em informações incompletas e ruidosas. O filtro de Kalman é capaz de lidar com incertezas nos dados de entrada e fornece estimativas ótimas do estado real do sistema, minimizando o erro de estimativa.

O funcionamento do filtro de Kalman baseia-se num modelo matemático que descreve o comportamento dinâmico do sistema. Esse modelo inclui equações que descrevem a evolução do estado do sistema ao longo do tempo e as relações entre as medidas dos sensores e o estado verdadeiro do sistema. O filtro de Kalman utiliza essas equações para atualizar iterativamente as estimativas do estado com base nas novas medições recebidas.

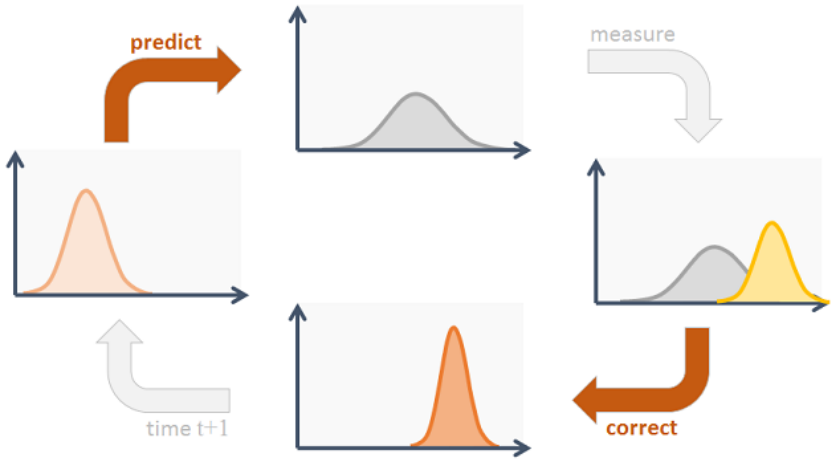


Figura 3.3: Exemplificação do ciclo de um filtro Kalman.

O filtro de Kalman opera por meio da propagação e atualização de distribuições gaussianas e suas covariâncias, conforme ilustrado na Figura 3.3. Esse processo é dividido em duas etapas principais, a previsão (*prediction*) e a atualização (*update*). Na etapa de previsão, o filtro utiliza o modelo dinâmico do sistema para estimar o estado futuro com base no estado atual e nas matrizes de controlo disponíveis. Em contrapartida, na etapa de atualização, o filtro combina a estimativa prévia com as medidas dos sensores, a fim de obter uma estimativa mais precisa do estado atual do sistema.

As equações do filtro de Kalman são expressas, matematicamente, da seguinte forma:

Etapa de Previsão:

$$\text{Estado da Previsão} : \hat{x}_{k|k-1} = F_k \hat{x}_{k-1|k-1} + B_k u_k$$
$$\text{Covariância da Previsão} : P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k$$

Etapa de Atualização:

$$\begin{aligned}
\text{Inovação} &: y_k = z_k - H_k \hat{x}_{k|k-1} \\
\text{Matriz de Inovação} &: S_k = H_k P_{k|k-1} H_k^T + R_k \\
\text{Ganho de Kalman} &: K_k = P_{k|k-1} H_k^T (S_k + R_k)^{-1} \\
\text{Estado da Atualização} &: \hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k y_k \\
\text{Covariância da Atualização} &: P_{k|k} = (I - K_k H_k) P_{k|k-1}
\end{aligned}$$

Onde $\hat{x}_{k|k-1}$ representa a estimativa prévia do estado no tempo k com base nas informações até o tempo $k-1$, $P_{k|k-1}$ é a covariância da estimativa prévia, F_k é a matriz de transição de estado, B_k é a matriz de controlo, u_k é o vetor de controlo, Q_k é a covariância do ruído do processo, z_k é a medida do sensor no tempo k, H_k é a matriz de observação, R_k é a covariância do ruído da medição, y_k é a inovação (diferença entre a medida real e a estimativa prévia), S_k é a matriz de inovação, K_k é o ganho de Kalman e I é a matriz identidade [67].

Estas equações descrevem o processo de estimativa do estado do sistema utilizando o filtro de Kalman. O filtro é aplicado de forma iterativa para atualizar as estimativas do estado à medida que novas medições são obtidas. Este processo de previsão e atualização permite obter estimativas mais precisas e confiáveis do estado real do sistema, tendo em consideração as incertezas presentes nos dados de entrada.

Capítulo 4

Implementação

O presente capítulo descreve a implementação detalhada do sistema de deteção e *tracking* de objetos utilizando a arquitetura proposta. Serão abordados os principais componentes da implementação, incluindo o módulo de deteção baseado no modelo SECOND, o módulo de *tracking* utilizando o abordagens como o Hungarian Algorithm e o Kalman Filter, bem como o processo completo para criação de anotações da nuvem de pontos de modo a obtermos bons resultados no processo de treino da rede neuronal.

4.1 Arquitetura

Este capítulo apresenta a arquitetura geral do sistema proposto, a qual desempenha um papel fundamental na compreensão do funcionamento e da integração dos diferentes componentes. A Figura 4.1 ilustra a visão geral desta arquitetura, destacando os principais módulos e as interações entre eles.

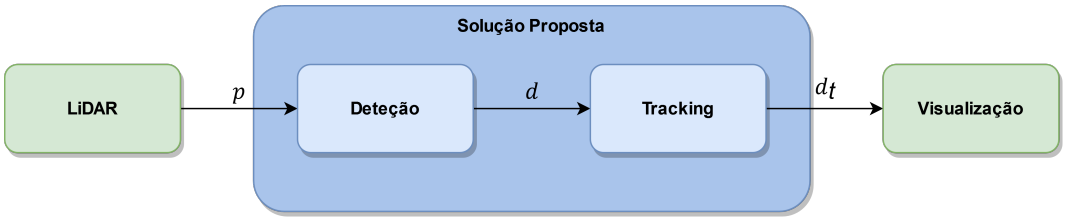


Figura 4.1: Arquitetura da solução proposta para o sistema de deteção e *tracking*.

A solução proposta é composta por dois nodes principais: o *node* "Deteção" e o *node* "Tracking". O primeiro recebe a nuvem de pontos proveniente do sensor LiDAR, representada por p:

$$p = \text{sensor\_msgs::PointCloud2}$$

Esta nuvem de pontos é recebida através de um tópico ROS. Em seguida, são geradas propostas de deteção pelo modelo e posteriormente publicadas, sendo representadas por d:

$$d = \text{jsk\_recognition\_msgs::BoundingBoxArray}$$

A estrutura de dados d possui dois campos principais: (1) uma mensagem *header*, comum em tópicos ROS, representada por std_msgs::Header; e (2) um *array* de bboxes, do tipo jsk_recognition_msgs::BoundingBox e contendo as seguintes características:

- `std_msgs/Header header`: Informações de cabeçalho, como o *timestamp* e o sistema de coordenadas de referência;
- `geometry_msgs/Pose pose`: Posição e orientação da bbox;
- `geometry_msgs/Vector3 dimensions`: Dimensões da bbox, correspondente à largura, altura e profundidade;
- `float32 value`: Valor associado à bbox, utilizado para atribuir uma medida de confiança à deteção (*score*);
- `uint32 label`: Valor indicativo à classe representada pela bbox.

O *node* "Tracking", desempenhando um papel crucial no processo de identificação e previsão das deteções, subscreve o tópico d e associa as novas propostas de deteções com as propostas obtidas na iteração anterior, obtendo uma previsão da próxima posição. As deteções associadas e previstas são então publicadas para visualização através do tópico d_t:

$$d_t = \text{jsk\_recognition\_msgs::BoundingBoxArray}$$

Nos subcapítulos seguintes, serão abordados com maior detalhe a implementação e os algoritmos utilizados em cada um destes nodes. Compreender esta arquitetura é de extrema importância para obter uma visão geral do sistema, bem como para compreender as etapas de processamento e integração de todos os componentes.

4.1.1 Deteção

O *node* responsável pela deteção de objetos, denominado de "Deteção" e ilustrado na Figura 4.2, é composto por dois blocos essenciais. O primeiro bloco, designado de

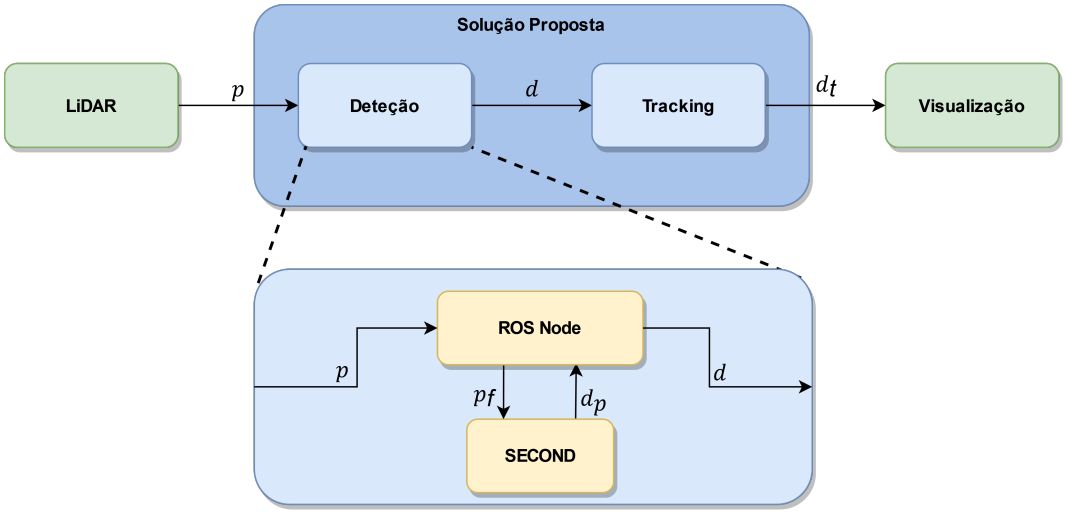


Figura 4.2: Arquitetura proposta para o sistema de deteção.

"ROS Node", desempenha um papel de suporte à rede de deteção SECOND, sendo responsável por gerir o fluxo de dados no sistema. Subscreve as mensagens enviadas pelo sensor LiDAR (p) e realiza um processamento dos dados 3D, filtrando apenas as informações relevantes que serão utilizadas pela rede neuronal, p_f:

$$p_f = \begin{bmatrix} x_1 & y_1 & z_1 & i_1 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

Estas informações incluem a posição tridimensional de cada ponto e a sua intensidade. Em seguida, esses dados são fornecidos ao modelo de deteção, que realiza a inferência e retorna um conjunto de deteções propostas, representado por d_p:

$$d_p = \begin{bmatrix} x_1 & y_1 & z_1 & w_1 & l_1 & h_1 & score_1 & label_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

Esse conjunto é composto pelas posições tridimensionais das deteções propostas, juntamente com a largura, comprimento e altura da bbox, respetivamente. Além disso, cada deteção possui uma pontuação de confiança (*score*) e uma classe detetada (*label*). Por fim, o "ROS Node" agrega as informações obtidas pelo modelo, representadas por d_p, e constrói um objeto bbox, que é publicado no tópico d.

4.1.2 Tracking

O primeiro processo do componente *Tracking* é associar as novas deteções, recebidas através da subscrição ao tópico d, com as deteções existentes em iterações anteriores, e para isso recorre ao bloco operacional "Identificação". Depois de ter as associações corretamente identificadas, sabemos com maior pormenor quais das deteções são novas, quais já se encontravam presentes anteriormente, e quais desapareceram. Esta informação vai ser bastante útil pois servirá de entrada para o segundo bloco funcional, Previsão.

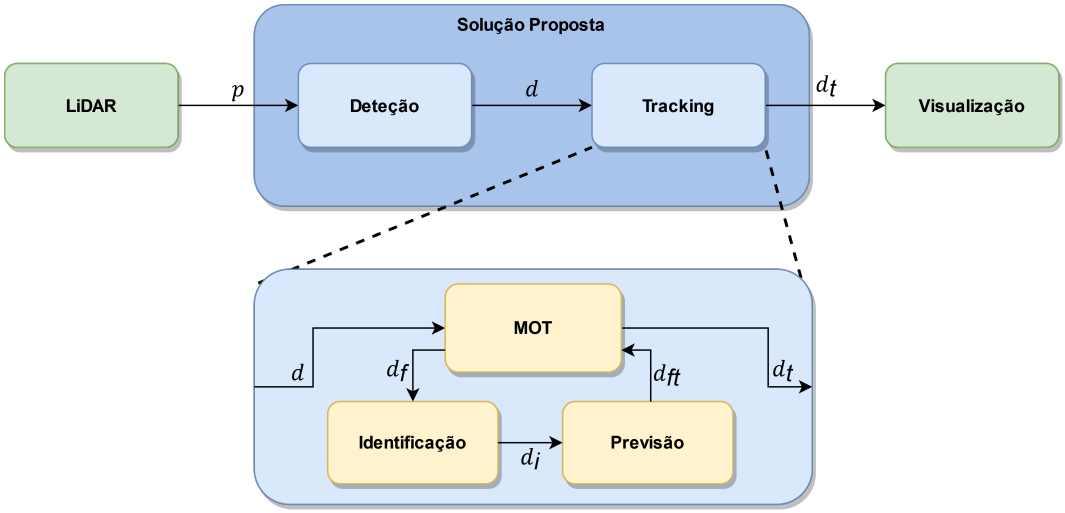


Figura 4.3: Arquitetura da solução proposta para o *node* "Tracking".

MOT

O algoritmo de MOT desempenha um papel fundamental na integração dos algoritmos de Identificação e Previsão, bem como na interação com o ambiente externo, atuando como uma interface que conecta esses algoritmos e facilita a troca de informações.

Uma das principais responsabilidades do algoritmo MOT é a subscrição do tópico de deteções d publicado pelo algoritmo de Deteção. Isso permite que o MOT receba as informações mais recentes sobre as deteções de objetos propostas pelo detetor. Estas deteções são então processadas, filtrando apenas a informação essencial, representada por d_f:

$$d_f = \begin{bmatrix} x_1 & y_1 & z_1 & w_1 & l_1 & h_1 & id_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

Esta informação é composta pelas posições tridimensionais das deteções propostas, juntamente com a largura, comprimento, altura e ID da bbox. Posteriormente, é

enviada para o algoritmo de identificação de modo a calcular as associações entre iterações.

Além disso, o algoritmo MOT também desempenha um papel importante na criação das bboxes, estimadas com a informação obtida pelo algoritmo de Previsão, representado por d_ft:

$$d_ft = \begin{bmatrix} x_p1 & y_p1 & z_p1 & id_1 \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

Estas bboxes são disponibilizadas para visualização através da publicação do tópico d_t.

Identificação

Neste bloco funcional, o nosso objetivo é estabelecer correspondências entre deteções em iterações consecutivas para melhor acompanhar a trajetória dos objetos no espaço. Para isso, foi implementado um algoritmo responsável por realizar esta associação, como pode ser observado na figura 4.4.

O algoritmo recebe como entrada uma matriz de deteções da iteração atual, d_f. Utilizando o método Hungarian, são determinadas as correspondências mais prováveis entre as deteções da iteração atual e as deteções de iterações anteriores, tendo em conta o parâmetro MAX_COST que define o custo máximo para uma associação ser considerada válida.

Para associar as deteções, o algoritmo percorre a matriz de deteções e compara as suas características, como posição e tamanho. Se uma deteção na iteração atual possuir características semelhantes a uma deteção na iteração anterior anterior, elas são consideradas correspondentes e o ID da deteção anterior é atribuído à deteção atual.

No caso de deteções não correspondentes, verificamos se estas apareceram em iterações anteriores, de acordo com outro parâmetro, HIST, que define o número de iterações disponíveis em histórico. Se uma deteção atual não tiver uma correspondência direta, procuramos por deteções semelhantes nas iterações anteriores. Se encontrarmos uma correspondência suficientemente próxima, respeitando o parâmetro de custo máximo de uma associação e o parâmetro de histórico, o ID da deteção anterior é atribuído à deteção atual. As novas deteções, ou seja, deteções que não possuem correspondência nas iterações anteriores, recebem um novo ID.

No final, o algoritmo disponibiliza uma matriz de deteções contendo os identificadores únicos atribuídos corretamente, representado por d_i:

$$d_i = \begin{bmatrix} x_1 & y_1 & z_1 & id_i \\ \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

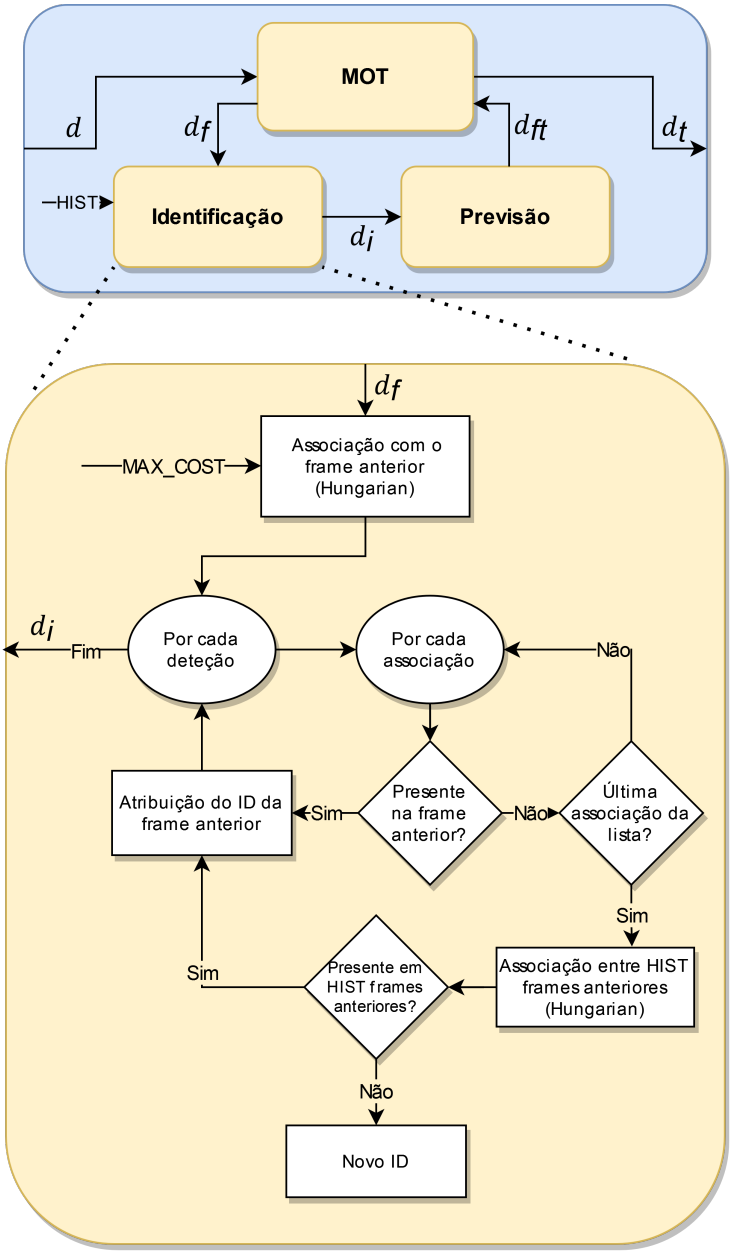


Figura 4.4: Arquitetura proposta para o sistema de associação e identificação.

Previsão

Após obtermos uma matriz de deteção pelo algoritmo de Identificação (secção 4.1.2), o objetivo deste algoritmo é utilizar de forma robusta a informação limitada disponível para acompanhar o objeto de forma efetiva ao longo do tempo. É nesse contexto que o filtro de Kalman demonstra-se eficaz, ao utilizar um modelo de movimento do objeto detetado, bem como os ruídos de medição e de processo previamente definidos. Dessa forma, o filtro de Kalman permite estimar a próxima posição do objeto com base nas informações disponíveis, oferecendo uma abordagem para lidar com as imperfeições do detetor, como ruído de medição e oclusão do objeto. A arquitetura do algoritmo Previsão está ilustrada na figura 4.5.

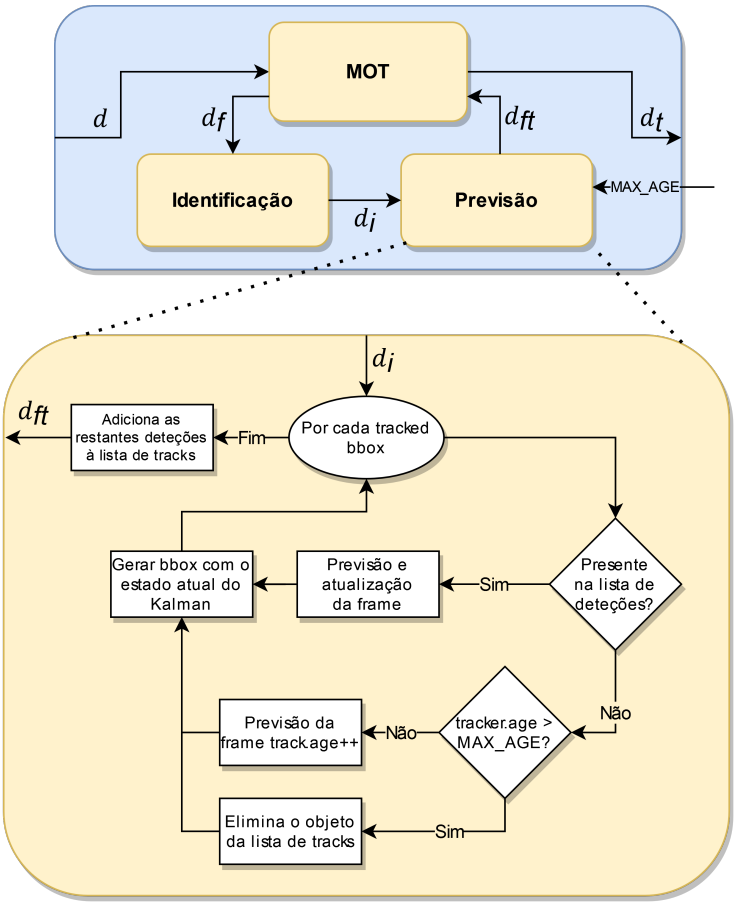


Figura 4.5: Arquitetura proposta para o sistema de Previsão.

A implementação do algoritmo "Previsão" começa por receber uma matriz de deteções propostas em contexto da iteração atual, representada por d_i. De modo a clarificar, uma *track* refere-se a uma deteção que possua uma instância da implementação do filtro Kalman.

Prossegue-se para um processo de verificação cruzada entre os IDs das *tracks* existentes e os IDs das deteções obtidas por meio de d_i. Esta verificação é realizada

para determinar quais *tracks* correspondem às deteções atuais e quais necessitam de ser atualizados.

Para os *tracks* que têm um ID correspondente nas deteções d_i, o filtro de Kalman é utilizado para prever a posição atual do objeto a ser seguido. É executada uma previsão para propagar o estado interno do filtro de Kalman no tempo, levando em consideração o modelo de movimento do objeto. Em seguida é executado uma atualização, para corrigir a previsão gerada pelo filtro de Kalman, com as posições obtidas pelo detetor (neste caso, o único sensor disponível). Além disso, a idade do *track* é atualizado para zero, indicando que foi atualizado na iteração atual.

No caso de um *track* não ter um ID correspondente nas deteções atuais, significa que não foi detetado na iteração atual. Nesse caso, é verificado o *delay* do *track*, ou seja, o número de iterações consecutivas em que não foi detetado desde a iteração atual. Se o *track* estiver oculto por um período que ultrapasse a idade máxima definida para o algoritmo de Previsão (MAX_AGE), o *track* é removido da lista de *tracks*. Caso contrário, a idade do *track* é incrementada em um, sendo executada uma previsão do filtro de Kalman para estimar a posição atual do objeto.

Além disso, são criados novos *tracks* para as deteções da iteração atual que não possuem um ID correspondente na lista de *tracks* existentes.

4.2 Ambiente de desenvolvimento

Um dos grandes desafios de quem pretende utilizar uma implementação existente de um modelo de deteção e/ou *tracking* é a preparação do ambiente de desenvolvimento local onde o modelo vai ser executado. Ao longo do processo de pesquisa e implementação desta dissertação, foi notório a quantidade de repositórios remotos onde quase ou nada especificavam as versões das bibliotecas utilizadas para o desenvolvimento do modelo. Maioritariamente das implementações são em Python, beneficiando de bibliotecas existentes como o PyTorch e TensorFlow que fornecem ferramentas para a criação e treino de modelos *DL*. No entanto, os ambientes em Python são conhecidos por não lidarem muito bem com retro-compatibilidade entre versões, causando um enorme transtorno em projetos onde podem existir inúmeras bibliotecas, como o caso dos modelos de deteção.

Após a necessidade de gerir eficientemente as bibliotecas Python utilizadas no projeto, optou-se por utilizar a plataforma Anaconda. A Anaconda é uma ferramenta de gestão de ambientes e pacotes Python amplamente reconhecida e utilizada na comunidade de ciência de dados e desenvolvimento de software. É possível criar ambientes virtuais isolados que contêm versões específicas das bibliotecas necessárias para executar um projeto com sucesso, ajudando a evitar conflitos entre bibliotecas e garante a reprodutibilidade do ambiente de desenvolvimento em diferentes sistemas.

Além disso, o Anaconda oferece um vasto repositório de bibliotecas prontas para uso.

Com isto, foi possível chegar ao estado onde o ambiente de desenvolvimento estava capaz de executar inferências e treinar o modelo. Ao utilizar o Anaconda, foi possível registar este ambiente consistente e confiável para o projeto, garantindo a compatibilidade e a estabilidade das bibliotecas utilizadas entre diferentes computadores.

```
name: second_ros
channels:
  - pytorch-lts
  - nvidia
  - defaults
dependencies:
  - _libgcc_mutex=0.1
  - _openmp_mutex=5.1
  - blas=1.0
  - bzip2=1.0.8
  - ca-certificates=2022.10.11
```

Listagem 4.1: Excerto do ambiente de desenvolvimento.

4.3 Treino do modelo

O repositório "second.pytorch", onde está implementado o modelo de deteção SECOND, inclui vários modelos pré-treinados. É possível realizar testes de inferência desses modelos em qualquer nuvem de pontos publicada num tópico ROS. Os modelos pré-treinados estão organizados por classes de deteção específicas. Há um modelo treinado apenas com dados de carros, outro com dados exclusivamente de pedestres e, por fim, um modelo que abrange as classes mais comuns no conjunto de dados KITTI: carro, ciclista, pedestre e carrinha.

Realizou-se um teste para avaliar a precisão da inferência utilizando uma sessão gravada com o sensor LiDAR fornecido pelo INESC TEC, juntamente com os modelos pré-treinados disponíveis no repositório. Rapidamente se verificou que as deteções propostas pelo modelo continham um elevado número de falsos positivos, ou seja, eram detetados carros e pessoas onde, na realidade, não existiam. Devido à dificuldade em encontrar modelos pré-treinados adicionais desta arquitetura de deteção e a possibilidade do problema estar relacionado com as diferenças nas características entre o sensor utilizado no treino (2.4.1) e o sensor utilizado nesta dissertação, optou-se por treinar o modelo com um *dataset* obtido durante a implementação desta dissertação.

Nesta secção, serão apresentadas as ferramentas e os métodos utilizados para o treino do modelo.

4.3.1 Pré-processamento da nuvem de pontos

Com o objetivo de evitar contratempos nas secções futuras de processamento da nuvem de pontos, optamos por realizar a remoção de pontos inválidos presentes no *log* adquirido através do sensor LiDAR. O valor inválido NaN (Not a Number) é frequentemente utilizado para representar pontos ausentes, inválidos ou não detetados. Tal situação pode ocorrer quando não são identificados objetos ou superfícies detetáveis na nuvem de pontos capturada pelo sensor LiDAR, resultando numa nuvem vazia caso todos os pontos obtidos sejam inválidos.

Dessa forma, foi implementado um mecanismo de subscrição do tópico de saída de dados do sensor LiDAR, que possibilita a verificação individual de cada nuvem de pontos em relação à presença de pontos inválidos. Quando identificados, esses pontos são eliminados e a nuvem resultante, contendo exclusivamente pontos válidos, é publicada num tópico distinto. Esta abordagem visa garantir a consistência dos dados utilizados nas etapas subsequentes do processamento da nuvem de pontos, evitando potenciais interferências decorrentes de pontos inválidos e mantendo a qualidade geral do fluxo de informação.

4.3.2 Criação do ground-truth

O fornecimento de *ground-truth* em modelos de DL é essencial para o treino eficaz. Este consiste em anotações (*labels*) precisas associadas à nuvem de pontos fornecida ao modelo para o treino, permitindo que o modelo aprenda a mapear corretamente os dados de entrada para as saídas desejadas. Ele atua como referência, possibilitando a avaliação do desempenho do modelo e a *backpropagation* do erro para ajuste dos parâmetros. Sem o *ground-truth*, o modelo não teria uma base sólida para aprender e melhorar seu desempenho durante o treino.

Anotação do Dataset

Para gerar as anotações das classes, foi utilizado uma aplicação na *toolbox* do MATLAB [68] chamada Lidar Labeler [69]. Esta ferramenta facilita a tarefa de anotar objetos em dados 3D oferecendo uma interface gráfica amigável que permite aos utilizadores visualizar e interagir com as nuvens de pontos de forma rápida e eficiente. A sua capacidade de oferecer múltiplas perspetivas de visualização das nuvens de pontos (4.6), permitindo uma análise abrangente e precisa dos dados, torna-a numa ferramenta versátil.

O primeiro passo consiste em importar o *dataset* que iremos utilizar no treino para a ferramenta Lidar Labeler, que possui a conveniência de suportar um ficheiro

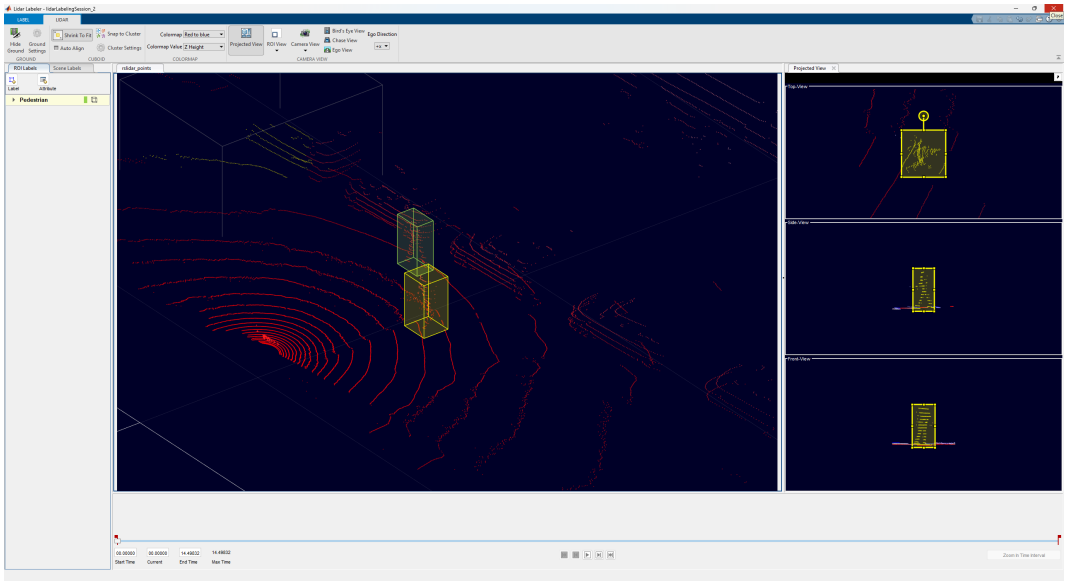


Figura 4.6: Anotação de uma pessoa utilizando o Lidar Labeler.

bag obtido através da *framework* ROS. A tarefa de anotar objetos, seja em 2D ou 3D, é conhecida por ser tediosa e repetitiva, exigindo um trabalho manual extenso. Considerando que uma sessão gravada com o sensor VLP16, configurado com uma taxa de publicação de 10Hz, pode conter cerca de 600 *frames* em apenas um minuto de sessão, é necessário gerar manualmente o mesmo número de anotações. Para reduzir esse tempo e aumentar a eficiência, utilizamos o algoritmo embutido na ferramenta chamado "Point Cloud Temporal Interpolator". Esta funcionalidade permite preencher as lacunas nas *frames* onde não existem anotações, estimando-as com base na interpolação das *frames* que contem anotações. Após a interpolação, é necessário apenas pequenas correções nas dimensões da bbox estimada pela ferramenta para garantir uma maior consistência no objeto em que queremos detetar.

Após anotarmos todas as *frames* do *dataset* com as classes desejadas, que neste caso é apenas "Pedestrian", procedemos à extração das anotações no único formato suportado pela ferramenta: uma instância do objeto "groundTruthLidar" no ambiente MATLAB. O objeto "groundTruthLidar" é composto pelos seguintes componentes:

- **dataSource**: Esta propriedade especifica a fonte dos dados da nuvem de pontos utilizada para as anotações.
- **labelDefs**: Esta propriedade especifica a hierarquia das anotações obtidas, contendo informações como nome, classe e grupo.
- **labelData**: Esta propriedade contém os dados anotados numa matriz. Inclui informações de identificação, posição e *timestamps* de data/hora para as anotações.

Nesta instância do objeto "groundTruthLidar", o foco principal reside nos dados contidos no objeto "labelData". Este objeto é composto por uma matriz numérica de tamanho M por 9, em que cada linha adopta o formato [xctr, yctr, zctr, xlen, ylen, zlen, xrot, yrot, zrot], onde:

- M corresponde ao número de anotações na *frame*.
- xctr, yctr e zctr especificam o centro da bbox anotada.
- xlen, ylen e zlen especificam o comprimento da bbox ao longo dos eixos x, y e z, respetivamente.
- xrot, yrot e zrot especificam os ângulos de rotação da bbox ao longo dos eixos x, y e z, respetivamente.

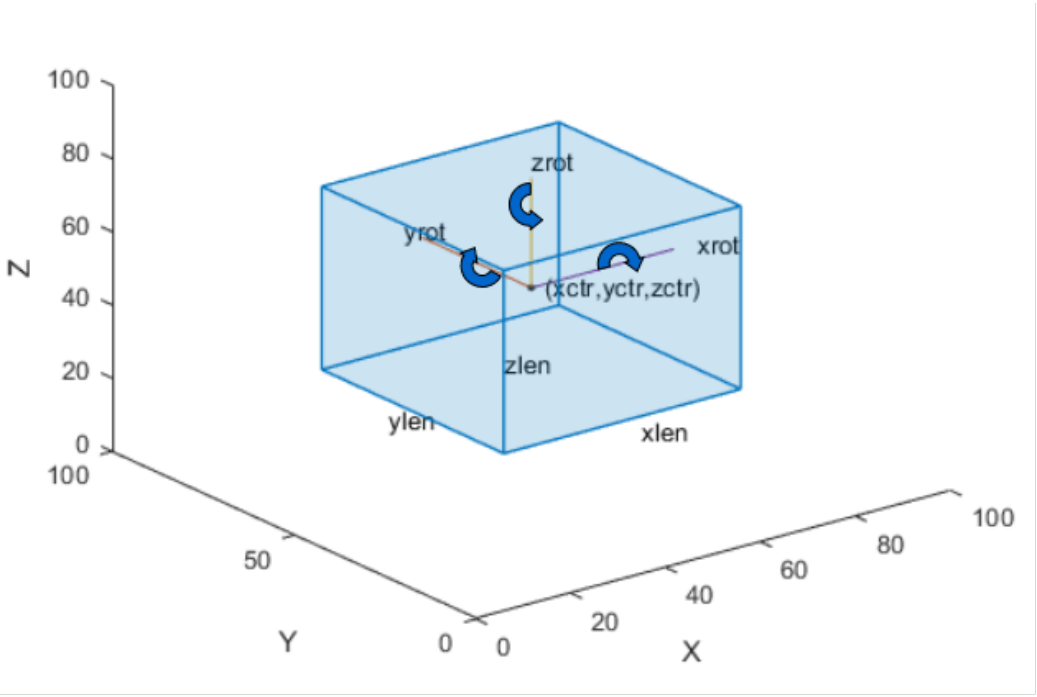


Figura 4.7: Determinação da posição da bbox anotada consoante os valores do objeto labelData [68].

Conversão para formato KITTI

Apesar de termos obtido todos os dados necessários para representar uma anotação como uma bbox, ainda não é suficiente para treinar a implementação do modelo SECOND utilizada nesta dissertação. Isso ocorre porque a implementação do modelo baseia-se no formato de dados utilizado pelo KITTI, que possui um formato de anotação diferente do formato obtido com a ferramenta Lidar Labeler. O formato de anotação do KITTI pode ser visto na Tabela 4.1. Portanto, é necessário converter

as anotações da ferramenta Lidar Labeler para o formato equivalente KITTI de modo a treinar adequadamente o modelo.

Tabela 4.1: Formato de anotação utilizado pelo KITTI [26].

Values	Name	Description
1	type	Describes the type of object: 'Car', 'Van', 'Truck', 'Pedestrian', 'Person_sitting', 'Cyclist', 'Tram', 'Misc' or 'DontCare'
1	truncated	Float from 0 (non-truncated) to 1 (truncated), where truncated refers to the object leaving image boundaries
1	occluded	Integer (0,1,2,3) indicating occlusion state: 0 = fully visible, 1 = partly occluded, 2 = largely occluded, 3 = unknown
1	alpha	Observation angle of object, ranging [-pi..pi]
4	bbox	2D bounding box of object in the image (0-based index): contains left, top, right, bottom pixel coordinates
3	dimensions	3D object dimensions: height, width, length (in meters)
3	location	3D object location x, y, z in camera coordinates (in meters)
1	rotation_y	Rotation ry around Y-axis in camera coordinates [-pi..pi]
1	score	Only for results: Float, indicating confidence in detection, needed for p/r curves, higher is better.

Nesta dissertação, desenvolveu-se a ferramenta "matlab2kitti" em Python, com o propósito de converter objetos do tipo "groundTruthLidar" obtidos através da exportação de anotações feitas pela ferramenta Lidar Labeler. A ferramenta "matlab2kitti" segmenta as informações do objeto "groundTruthLidar" exportado e reorganiza-as utilizando uma *dataframe*, adicionando também os campos ausentes da Tabela 4.1. Esta reorganização é necessária para adaptar as anotações geradas pela ferramenta Lidar Labeler ao formato de anotação utilizado pelo conjunto de dados KITTI, garantindo assim um formato coerente ao utilizado na implementação do modelo SECOND.

Para além dos campos adicionais considerados, há também uma diferença no sistema de coordenadas utilizado pela ferramenta Lidar Labeler em comparação com o sistema de coordenadas utilizado no *dataset* KITTI. Conforme descrito no

artigo publicado pelos autores do KITTI [26], as posições espaciais das anotações são definidas em relação ao sistema de coordenadas da câmara de referência, e não ao sistema de coordenadas do LiDAR. Isso apresenta dois problemas: (1) os sensores estão separados por uma distância significativa e (2) o sistema de coordenadas da câmara (utilizado nas anotações KITTI) não coincide com o sistema de coordenadas do sensor LiDAR (utilizado na ferramenta Lidar Labeler). Estas diferenças podem ser observadas na figura 2.9.

Para abordar essa particularidade, a ferramenta "matlab2kitti" utiliza as matrizes de calibração da câmara fornecidas pelo KITTI para converter os pontos do referencial do LiDAR para o referencial da câmara. Na Figura 4.8, estão representados alguns dos sensores utilizados no conjunto de dados KITTI, juntamente com as matrizes de transformação de referenciais.

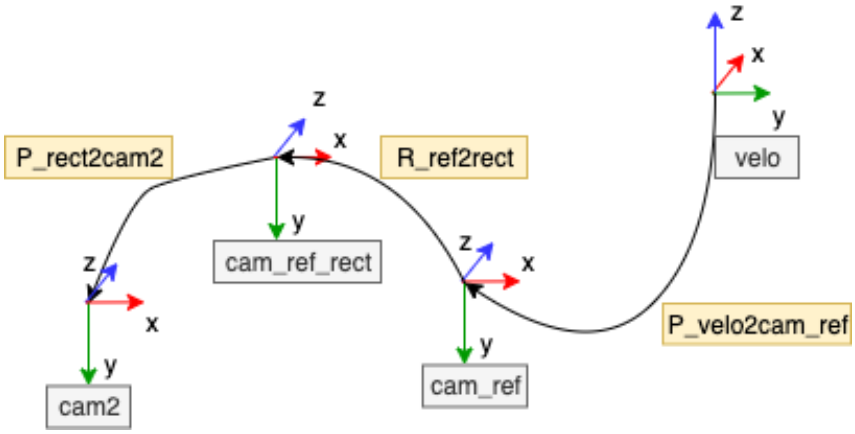


Figura 4.8: Esquema de conversão entre referenciais dos sensores utilizados no *dataset* KITTI, representando os retângulos amarelados as matrizes de conversão.

As anotações efetuadas estão no referencial do sensor LiDAR, "velo". Será necessário realizar uma transformação euclidiana, denominada P_velo2cam_ref, para converter do referencial LiDAR para o referencial da câmara de referência, "cam_ref". Após essa transformação, será necessário aplicar uma matriz de rotação, R_ref2rect, para retificar os pontos no referencial da câmara de referência. Após as transformações referidas, a ferramenta "matlab2kitti" gera uma visualização da *dataframe* final através de um Jupyter Notebook desenvolvido para essa finalidade. Em simultâneo, as anotações são escritas em ficheiros de texto separados, fazendo correspondência direta entre número total de ficheiros e número total de *frames*. Na figura 4.9 está representado um exemplo de uma *dataframe* final com o formato utilizado pelo *dataset* KITTI, como demonstrado na tabela 4.1.

```
from IPython.display import display
import matlab2kitti as m2k
lc = m2k.LabelConverter(m2k.CALIB_FILE_PATH, m2k.MAT_FILE_PATH, m2k.LABEL_OUT_PATH)
display(lc.labels)
```

[1] ✓ 0.4s

File	Label	Type	Truncated	Occluded	Alpha	BBox	Dimensions	Location	Rotation
0	0	Pedestrian	0.0	0	0	[0, 0, 50, 50]	[1.692, 0.609, 0.959]	[-1.082, -0.114, 6.739]	0
	1	Pedestrian	0.0	0	0	[0, 0, 50, 50]	[1.805, 0.899, 0.957]	[1.846, -0.082, 4.899]	0
1	0	Pedestrian	0.0	0	0	[0, 0, 50, 50]	[1.695, 0.61, 0.957]	[-1.221, -0.114, 6.734]	0
	1	Pedestrian	0.0	0	0	[0, 0, 50, 50]	[1.814, 0.877, 0.94]	[1.856, -0.085, 4.864]	0
2	0	Pedestrian	0.0	0	0	[0, 0, 50, 50]	[1.698, 0.611, 0.955]	[-1.359, -0.114, 6.73]	0
...	...	...	...	...	...	...	...	...	...
142	1	Pedestrian	0.0	0	0	[0, 0, 50, 50]	[1.686, 0.738, 0.888]	[-9.65, 0.006, 6.497]	0
143	0	Pedestrian	0.0	0	0	[0, 0, 50, 50]	[1.65, 0.834, 0.75]	[-20.985, 0.308, 5.27]	0
	1	Pedestrian	0.0	0	0	[0, 0, 50, 50]	[1.686, 0.738, 0.885]	[-9.818, 0.007, 6.507]	0
144	0	Pedestrian	0.0	0	0	[0, 0, 50, 50]	[1.646, 0.837, 0.749]	[-21.127, 0.314, 5.254]	0
	1	Pedestrian	0.0	0	0	[0, 0, 50, 50]	[1.685, 0.737, 0.882]	[-9.987, 0.009, 6.518]	0

290 rows × 8 columns

Figura 4.9: Visualização do Jupyter Notebook contendo as anotações corrigidas visíveis através de uma *dataframe*.

4.3.3 Configuração de treino

A organização da configuração de treino é dividida em várias secções contendo informações e parâmetros essenciais para o modelo de deteção utilizado, estando dividido em diferentes secções, onde cada secção representa um componente específico do modelo, incluindo a rede utilizada, o gerador de voxels, o extrator de características, o RPN e as configurações da *loss*. Cada componente possui seus próprios parâmetros ajustáveis que podem ser modificados conforme necessário.

A implementação do modelo de deteção utilizado possuí vários ficheiros de configuração disponíveis para uso. Os ficheiros de configuração utilizados possuem diversos parâmetros modificáveis, no entanto poucos são os que não requerem um conhecimento profundo e modificações adicionais no código *core* do modelo de deteção. Isto deve-se ao facto dos ficheiros de configurações existentes terem sido parametrizados de modo a gerarem a melhor classificação possível em todas as classes de deteção existentes no *dataset* KITTI.

Como tal, para esta dissertação foi utilizado a configuração de treino disponível no repositório da implementação do modelo.

Capítulo 5

Resultados

Este capítulo apresenta os resultados obtidos no âmbito da dissertação, com foco na avaliação do desempenho do sistema de deteção desenvolvido. O capítulo inicia com uma descrição do *hardware* utilizado, incluindo informações sobre o sensor LiDAR e o computador. Em seguida, são apresentados os detalhes da aquisição do *dataset* utilizado para o treino e validação do modelo de deteção, bem como uma descrição das métricas de avaliação utilizadas.

5.1 Hardware

5.1.1 Sensor LiDAR

O sensor LiDAR utilizado nesta dissertação foi o RS-Helios 5515, da RoboSense, representado na figura 5.1. A RoboSense é uma empresa de tecnologia especializada em sensores LiDAR. Estes sensores são projetados como uma ferramenta para auxiliar a condução autónoma, fornecendo uma visão tridimensional em tempo real do ambiente. Estas características possibilitam aos veículos a identificação de obstáculos, pedestres, ciclistas e outros elementos no cenário de condução. As características deste sensor estão representadas na tabela 5.1.

Figura 5.1: Gama RS-Helios.

Tabela 5.1: Tabela de especificações do sensor RS-Helios 5515 fornecida pelo fabricante.

Característica	**RS-Helios 5515**
Nº de Feixes	32
Comp. de onda (λ)	905nm
Segurança	Classe 1 Eye safe
Alcance	150m (110m @ 10% NIST)
Ponto cego	$\leq$0.2m
Precisão do Alcance (Típico)	$\pm$3cm (3σ)
Campo de Visão Horizontal	360°
Campo de Visão Vertical	70° (-55° +15°)
Resolução Horizontal	0.2°/0.4°
Resolução Vertical	Até 1.33°
Taxa de Frames	10Hz/20 Hz
Velocidade de Rotação	600/1200rpm (10/20Hz)
Pontos Por Segundo	$\sim$576,000pts/s (Single Return)
	$\sim$1,152,000pts/s (Dual Return)
Pacote UDP Inclui	Coordenadas Espaciais, Intensidade, Timestamp, etc.
Automotive Ethernet	100M Base T1
Saída	Pacotes UDP via Ethernet
Tensão de Operação	9V - 32V
Consumo de Energia	12W
Temperatura de Operação	-30°C +60°C
Temperatura de Armazenamento	-40°C +85°C
Proteção de Entradas	IP67, IP6K9K
Sincronização de Tempo	$GPRMC com 1PPS, PTP&gPTP
Dimensão	ϕ97.5mm $\times$ H100 mm
Peso (sem cabos)	$\sim$1.0 kg

5.1.2 Sistema Computacional

No âmbito desta dissertação, foi utilizado um computador *desktop* tanto para a execução das inferências como para o treino do modelo de deteção. Um requisito fundamental para o desenvolvimento deste projeto, bem como de qualquer projeto de DL, é a utilização de uma GPU Nvidia com uma quantidade significativa e rápida de memória interna. Esta necessidade surge devido às intensas operações matemáticas requeridas pelas redes neuronais e ao uso da linguagem de programação *Compute Unified Device Architecture* (CUDA) como interface com a GPU. Na tabela 5.2 estão representadas as características desse computador.

Tabela 5.2: Características do computador utilizado nesta dissertação.

Componente	**Característica**
Processador	AMD Ryzen 7 3700X Octa-Core 3.6GHz c/ Turbo 4.4GHz
Memória RAM	32Gb DDR4 3200MHZ
Armazenamento	SSD M.2 500GB NVMe
Placa Gráfica	EVGA GeForce RTX 3060 Ti 8GB GDDR6
Sistema Operativo	Ubuntu 20.04 LTS

5.2 Deteção

5.2.1 Aquisição do Dataset

O *dataset* utilizado para o treino e validação do modelo de deteção foi adquirido no Instituto Superior de Engenharia do Porto (ISEP), junto à entrada do laboratório do INESC. A escolha do local para a aquisição do *dataset* deu-se maioritariamente por ser uma zona de elevada concentração de alunos que percorrem o percurso desde o ISEP até à estação de metro do Pólo Universitário ou Instituto Português de Oncologia (IPO).

Foram utilizadas para o *dataset* sete sessões de captura de dados, nas quais foram observadas diferentes situações, incluindo pessoas em locais com visão desobstruída pelo sensor, pessoas parcialmente visíveis devido à presença de carros estacionados entre elas e o sensor, pessoas em pares com mochilas nas costas e casos em que as pessoas mudam de direção e orientação de forma repentina.

5.2.2 Métricas de deteção

Na avaliação de um sistema de deteção, são utilizados quatro termos para descrever os resultados dos testes: *True Positive* (TP), *True Negative* (TN), FP e *False Negative* (FN). O TP refere-se aos casos em que o modelo identifica corretamente

Figura 5.2: Pátio do INESC, ISEP.

um objeto positivo. O TN ocorre quando o modelo corretamente descarta um objeto negativo. Já o FP ocorre quando o modelo identifica, erradamente, um objeto negativo como positivo. Por fim, o FN ocorre quando o modelo não deteta um objeto positivo. Encontra-se na figura 5.3 uma representação destas nomenclaturas, faltando apenas o TN devido a não ser representável visualmente. Estes termos são fundamentais para entender as métricas que irão ser discutidas nesta secção.

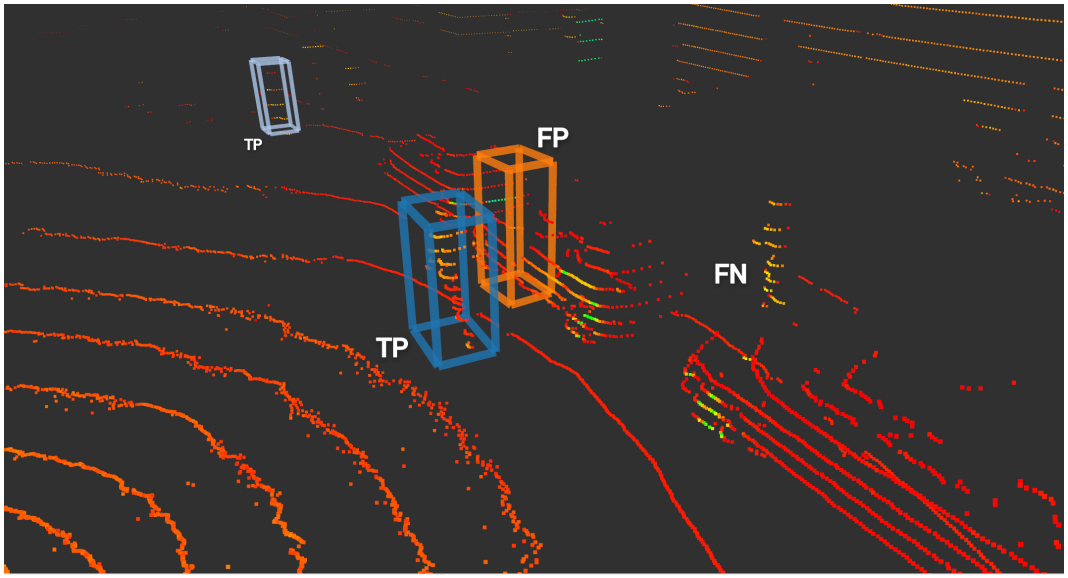


Figura 5.3: Representação de TP, FP e FN.

Accuracy

A Accuracy (exatidão) é uma métrica que indica a taxa geral de classificações corretas realizadas pelo modelo, medindo a proporção de todas as classificações corretas

em relação ao número total de amostras. A fórmula para calcular a Accuracy é a seguinte:

$$A = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.1}$$

Um valor elevado de Accuracy significa uma classificação correta na maioria das deteções, tanto nos TP como TN.

Precision

A Precision (precisão) é obtida através do rácio entre o número de TP e a soma dos verdadeiros positivos com os FP. A fórmula para o cálculo da Precision é:

A Precision é uma métrica representativa da capacidade do modelo em realizar deteções corretas e precisas. A fórmula para calcular a Precision é a seguinte:

$$P = \frac{TP}{TP + FP} \tag{5.2}$$

Um valor de Precision elevado implica que poucas deteções incorretas são feitas, minimizando assim os falsos positivos.

Recall

O Recall é uma métrica representativa da capacidade do modelo em identificar corretamente todos os objetos relevantes. A sua fórmula é dada por:

$$R = \frac{TP}{TP + FN} \tag{5.3}$$

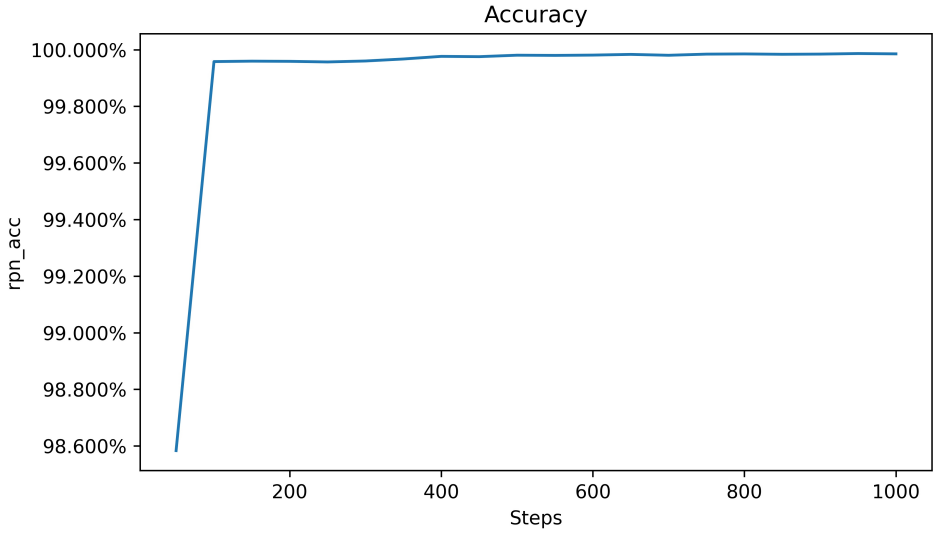
Quanto maior o seu valor, menor a probabilidade de haver objetos positivos não detetados.

F1 Score

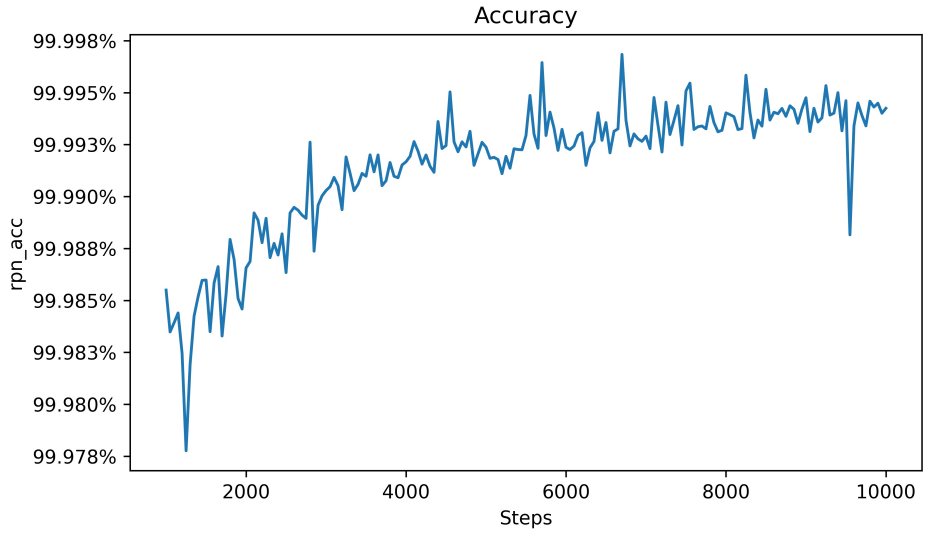
O F1 Score é uma métrica que combina a Precision e o Recall numa única medida para avaliar o desempenho geral de um modelo de deteção, fornecendo um equilíbrio entre a capacidade de realizar deteções corretas (Precision) e a capacidade de identificar corretamente todos os casos relevantes (Recall). O F1 Score é calculado a partir da média harmónica da Precision e do Recall, levando em consideração tanto os TP quanto os FP e FN. A fórmula para calcular o F1 Score é a seguinte:

$$F1Score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{5.4}$$

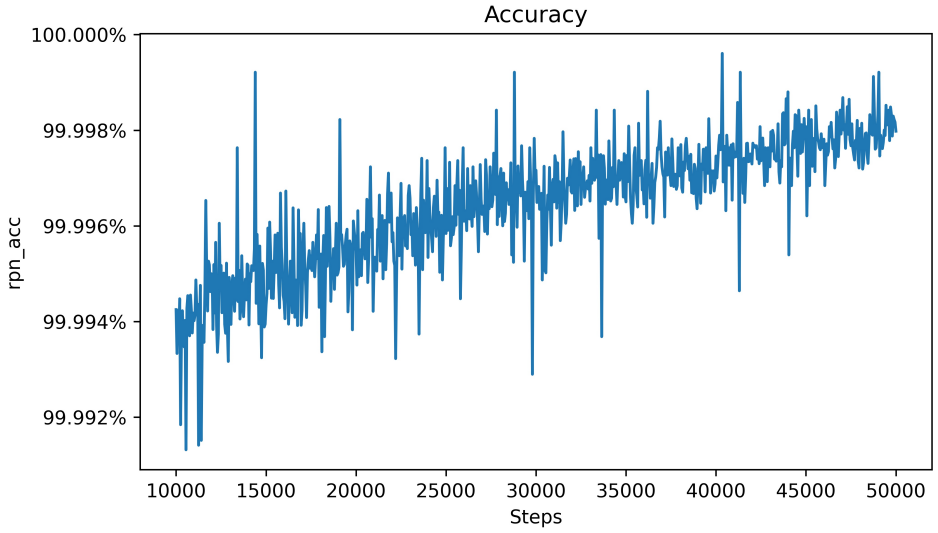
O valor do F1 Score varia de 0 a 1, onde um valor mais próximo de 1 indica um melhor desempenho geral do modelo.



(a) Accuracy entre os *steps* de treino 0 e 1,000.

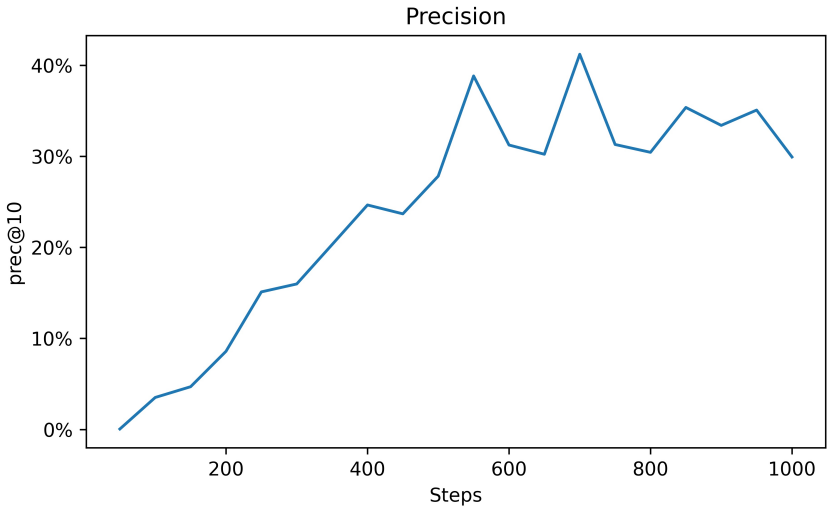


(b) Accuracy entre os *steps* de treino 1,000 e 10,000.

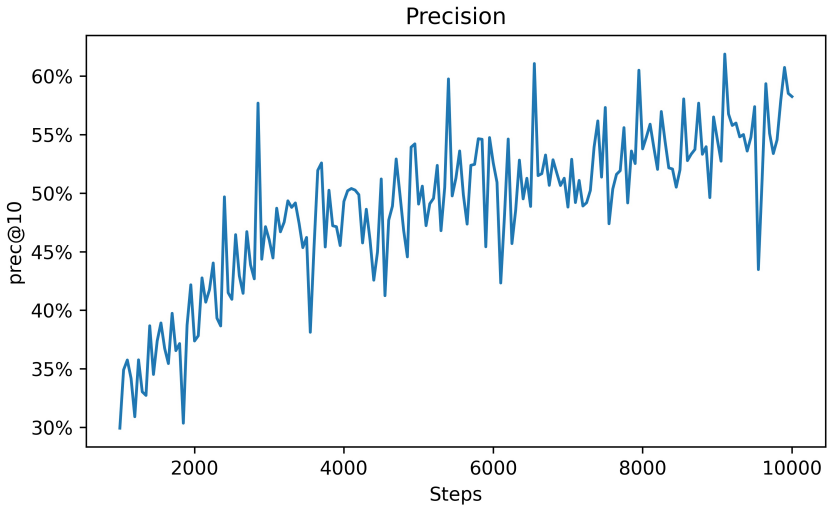


(c) Accuracy entre os *steps* treino 10,000 e 50,000.

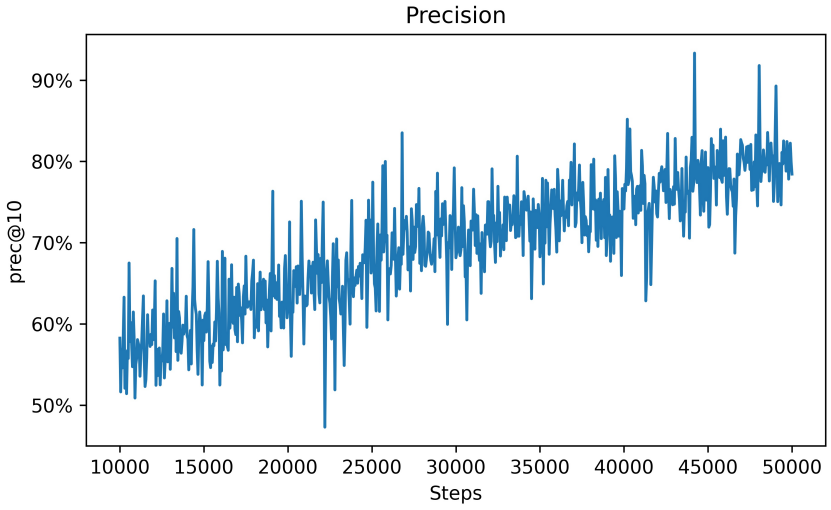
Figura 5.4: Resultados da Accuracy.



(a) Precision entre os *steps* de treino 0 e 1,000.

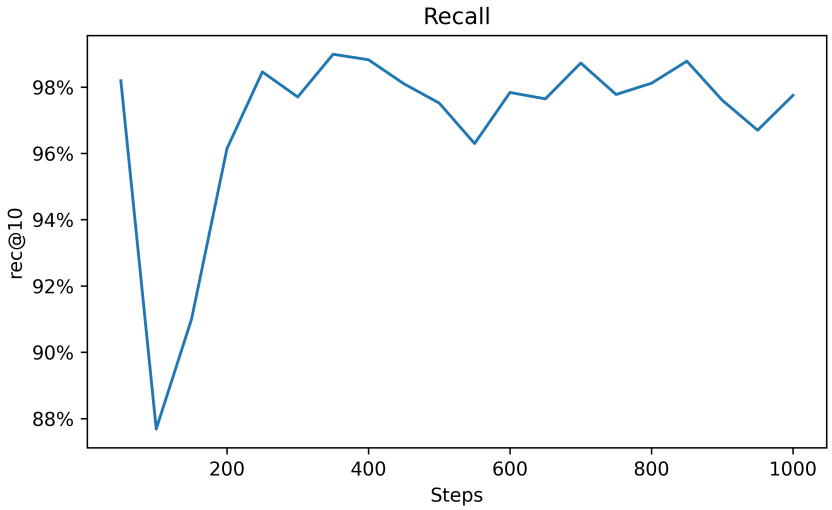


(b) Precision entre os *steps* de treino 1,000 e 10,000.

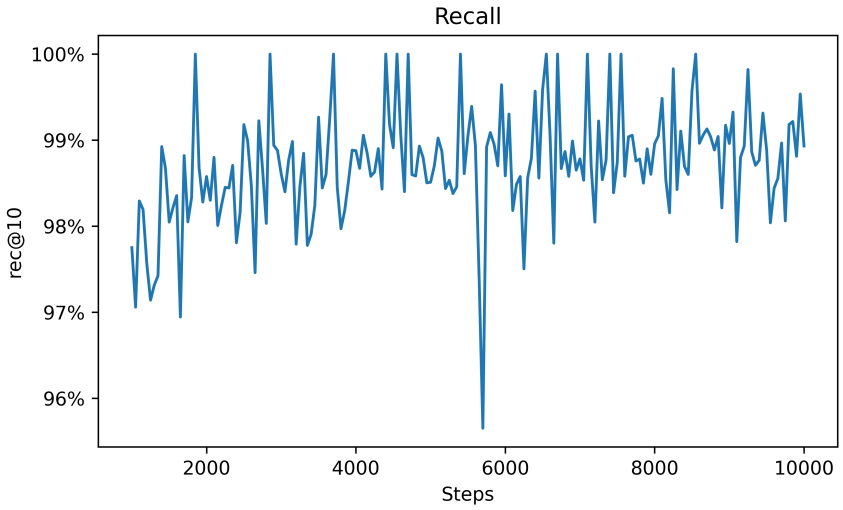


(c) Precision entre os *steps* de treino 10,000 e 50,000.

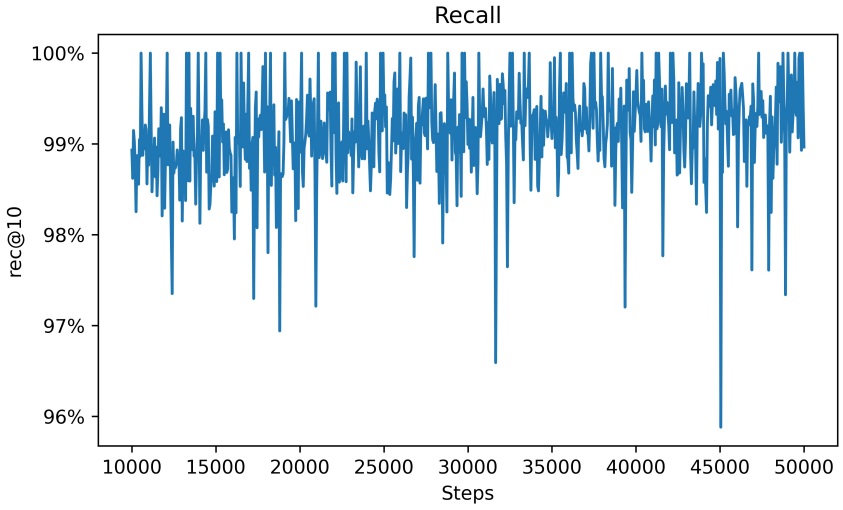
Figura 5.5: Resultados da Precision para um *threshold* IoU de 10%.



(a) Recall entre os *steps* de treino 0 e 1,000.

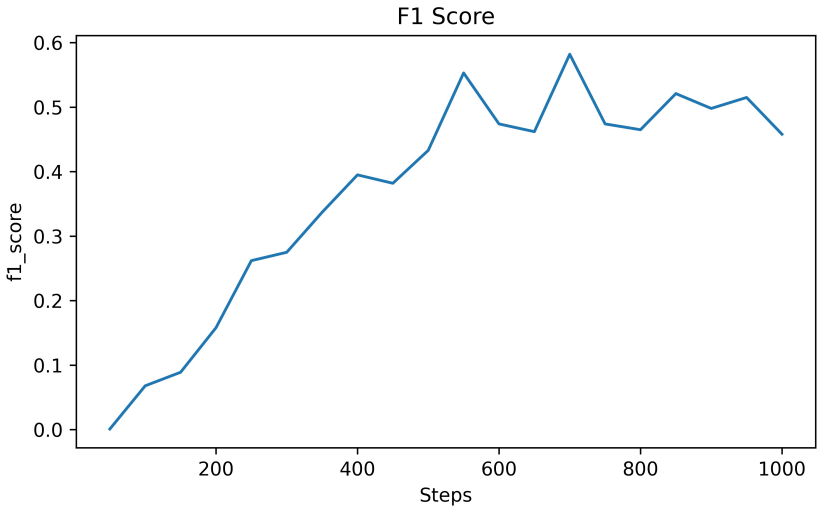


(b) Recall entre os *steps* de treino 1,000 e 10,000.

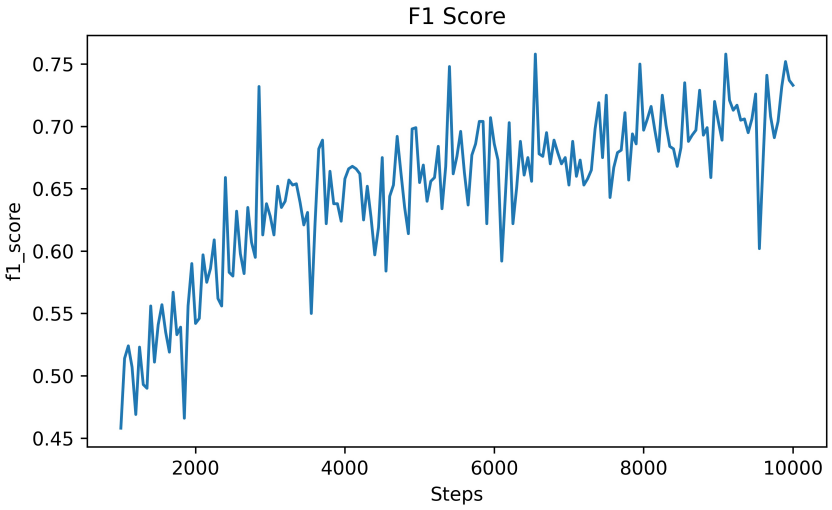


(c) Recall entre os *steps* de treino 10,000 e 50,000.

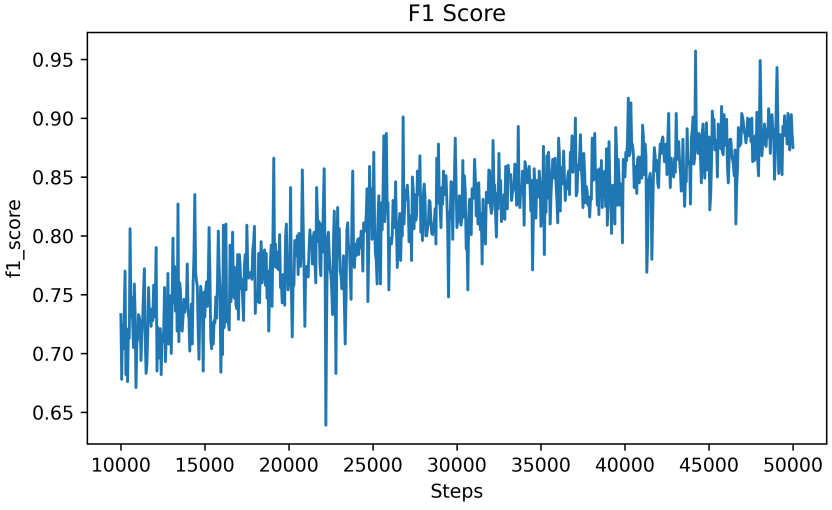
Figura 5.6: Resultados da Recall para um *threshold* IoU de 10%.



(a) F1 Score entre os *steps* de treino 0 e 1,000.



(b) F1 Score entre os *steps* de treino 1,000 e 10,000.



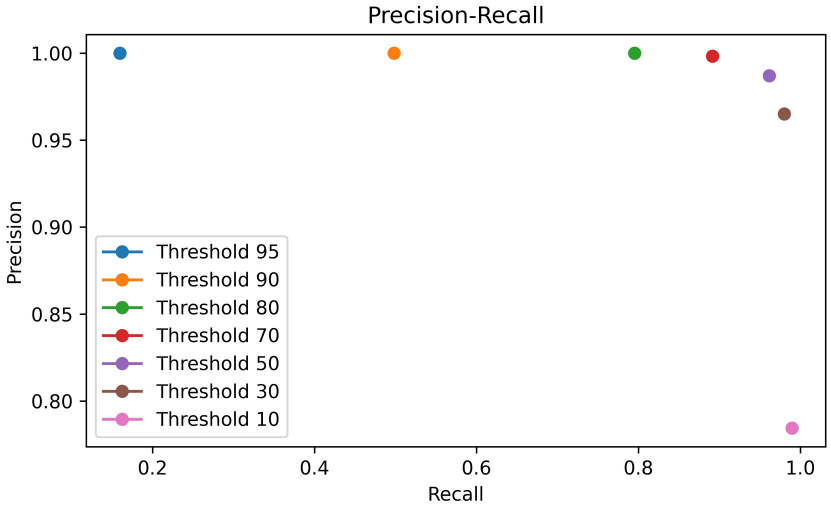
(c) F1 Score entre os *steps* de treino 10,000 e 50,000.

Figura 5.7: Resultados do F1 Score para um *threshold* IoU de 10%.

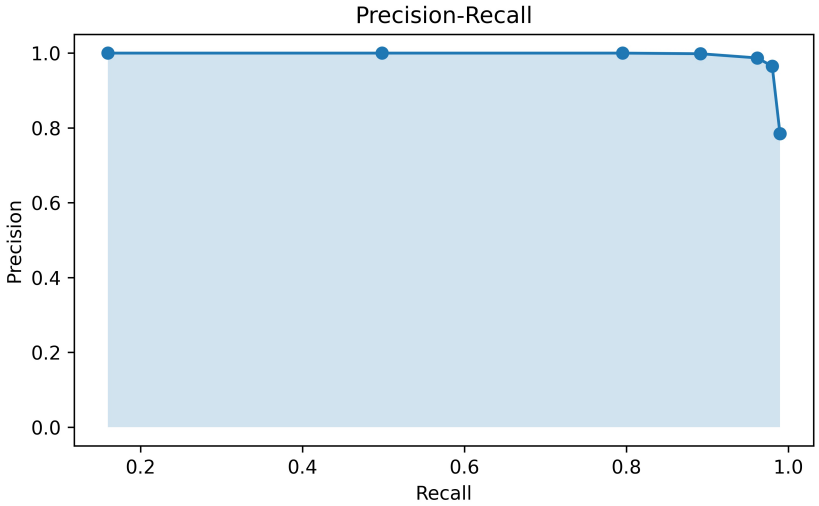
Average Precision

O AP é uma métrica que mede o desempenho de um modelo de classificação ao avaliar a Precision e o Recall em diferentes *thresholds* de IoU, resumindo a curva P-R num único valor, fornecendo uma medida consolidada da qualidade geral das deteções do modelo. Quanto maior o seu valor, melhor é o desempenho do modelo em identificar corretamente os objetos de interesse.

Após o cálculo da área da curva P-R, observada na figura 5.8, obtivemos uma AP de 98.5%.



(a) Representação da relação entre a Precision e o Recall em diferentes *thresholds* IoU.



(b) Representação da curva P-R. A área desta curva, sombreada a azul, representa o AP do nosso modelo.

Figura 5.8: Resultados do cálculo da curva Precision-Recall.

5.2.3 Custo computacional

O custo computacional da inferência do modelo de deteção foi estimado com base na diferença de tempo entre o fornecimento da nuvem de pontos e a receção das propostas de deteção resultantes da inferência. Os resultados, representados na figura 5.9, são apresentados em segundos, representando o tempo total de execução da inferência. Verificou-se que o tempo médio de inferência no nosso *dataset* foi de 0,021 segundos (46 FPS), revelando-se significativamente mais rápido em comparação com o tempo de inferência divulgado pelos autores do modelo no conjunto de dados KITTI (representado na tabela 2.3). Essa discrepância pode ser atribuída às melhorias contínuas na implementação do modelo em linguagem Python, aliadas à eficiência das instruções executadas pelas GPU modernas.

```
1    tic = time.time()
2    boxes_lidar, scores, label = self.
         inference(cloud)
3    toc = time.time()
4    inference_time = toc-tic
5    fps = 1/(inf_time)
```

Listagem 5.1: Cálculo do custo computacional do modelo.

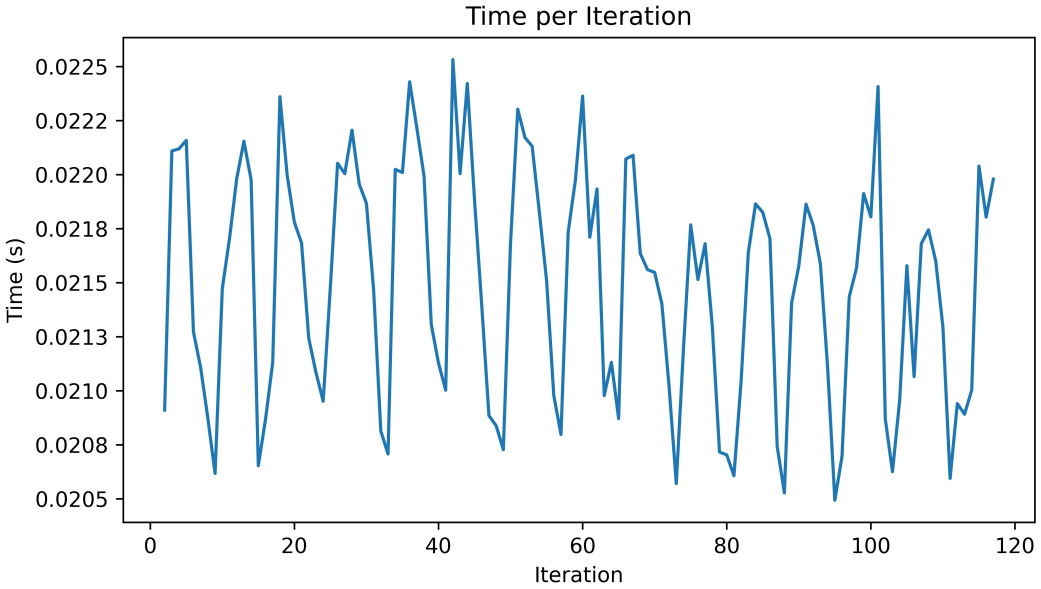


Figura 5.9: Tempo de execução de cada inferência do modelo de deteção.

5.2.4 Erro de Estimação

Para uma melhor avaliação da eficiência do detetor, foi desenvolvido um avaliador que compara, em cada iteração, as bboxes propostas pelo detetor com as bboxes presentes nas anotações geradas pela ferramenta Lidar Labeler. Os resultados dessa avaliação estão apresentados na Figura 5.10 para BEV e na Figura 5.11 para 3D, onde foi medido o IoU entre as bboxes e calculado o erro estimado das deteções em todo o *dataset*. Os resultados revelaram uma média do erro IoU estimado de 0.22 em BEV e 0.30 em 3D, com um desvio padrão de 0.11 e 0.10 respetivamente. As diferenças de posição e tamanho das bboxes propostas pelo modelo e as anotadas podem ser visualizadas na Figura 5.12.

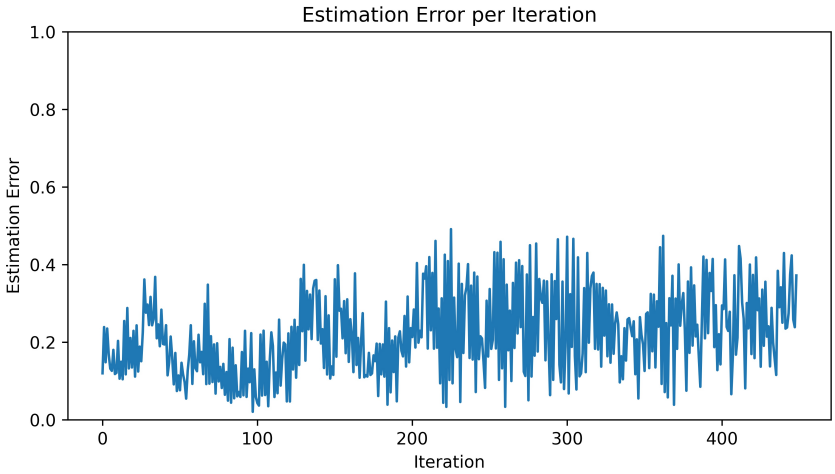


Figura 5.10: Representação dos resultados do erro estimado entre as deteções propostas pelo detetor e as anotações (*ground-truth*) em BEV.

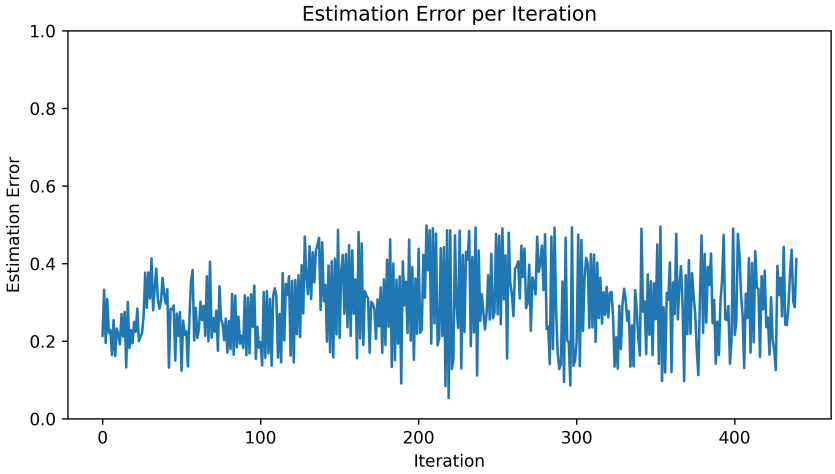


Figura 5.11: Representação dos resultados do erro estimado entre as deteções propostas pelo detetor e as anotações (*ground-truth*) em 3D.

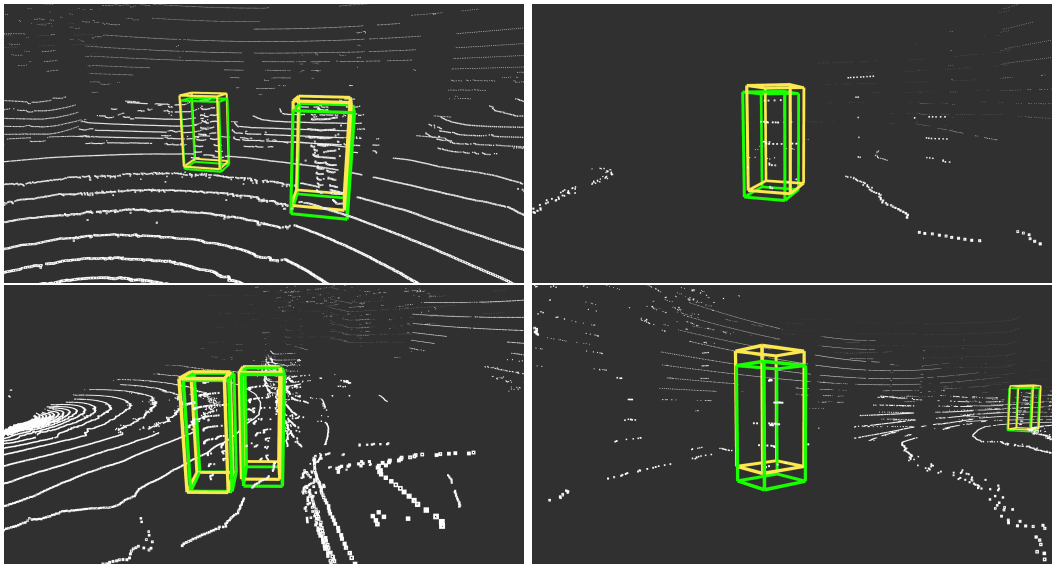


Figura 5.12: Representação da bbox proposta pelo modelo de deteção a amarelo e a bbox anotada (*ground-truth*) a verde.

5.3 Tracking

5.3.1 Parâmetros

Durante a implementação dos algoritmos de associação de IDs e previsão de posições das deteções fornecidas pelo modelo SECOND, foram estabelecidos parâmetros com o objetivo de otimizar o desempenho e garantir o funcionamento adequado do algoritmo. Nesta secção, iremos demonstrar as configurações utilizadas para os resultados obtidos.

Identificação

Os parâmetros utilizados no processo de identificação das deteções foram *inputs* no algoritmo Hungarian. Os parâmetros que melhor se adaptaram ao uso do nosso *dataset* foram os seguintes:

- MAX_COST = 1
- HIST = 10

Previsão

Os parâmetros utilizados no algoritmo de previsão foram os seguintes:

- MAX_AGE = 10
- DETECTOR_STD_MEAS = 0.1

- DETECTOR_FREQUENCY = 0.1
- PROCESS_NOISE = 0.01

Os parâmetros DETECTOR_STD_MEAS, DETECTOR_FREQUENCY e PROCESS_NOISE são utilizados para implementar um filtro de Kalman para cada um dos *tracks* identificados. O primeiro representa o desvio padrão do ruído de medição do sensor em metros, geralmente denotado como σ, o segundo representa o intervalo de tempo entre as medições em segundos, normalmente denotado como dt, enquanto que o terceiro representa o ruído do processo do filtro Kalman, normalmente denotado como w.

O filtro de Kalman utilizado é linear, tendo o estado do objeto sido modelado com as variáveis x, y, z, v_x, v_y, v_z, representando as coordenadas espaciais e as velocidades nas direções x, y e z, respetivamente. A sua configuração prosseguiu-se da seguinte forma:

- Função de medição definida para mapear o estado do objeto para as medições observadas, utilizando a matriz de medição H:

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

- Ruído de medição definido como uma matriz diagonal R, onde o desvio padrão do ruído de medição foi definido como σ:

$$R = \begin{bmatrix} \sigma_x^2 & 0 & 0 \\ 0 & \sigma_y^2 & 0 \\ 0 & 0 & \sigma_z^2 \end{bmatrix}$$

- Ruído do processo definido como uma matriz diagonal Q, definido como w:

$$Q = \begin{bmatrix} w & 0 & 0 & 0 & 0 & 0 \\ 0 & w & 0 & 0 & 0 & 0 \\ 0 & 0 & w & 0 & 0 & 0 \\ 0 & 0 & 0 & w & 0 & 0 \\ 0 & 0 & 0 & 0 & w & 0 \\ 0 & 0 & 0 & 0 & 0 & w \end{bmatrix}$$

- Função de transição de estado definida pela matriz de transição F, que relaciona o estado atual com o próximo estado, considerando também o intervalo

de tempo entre as medições dt:

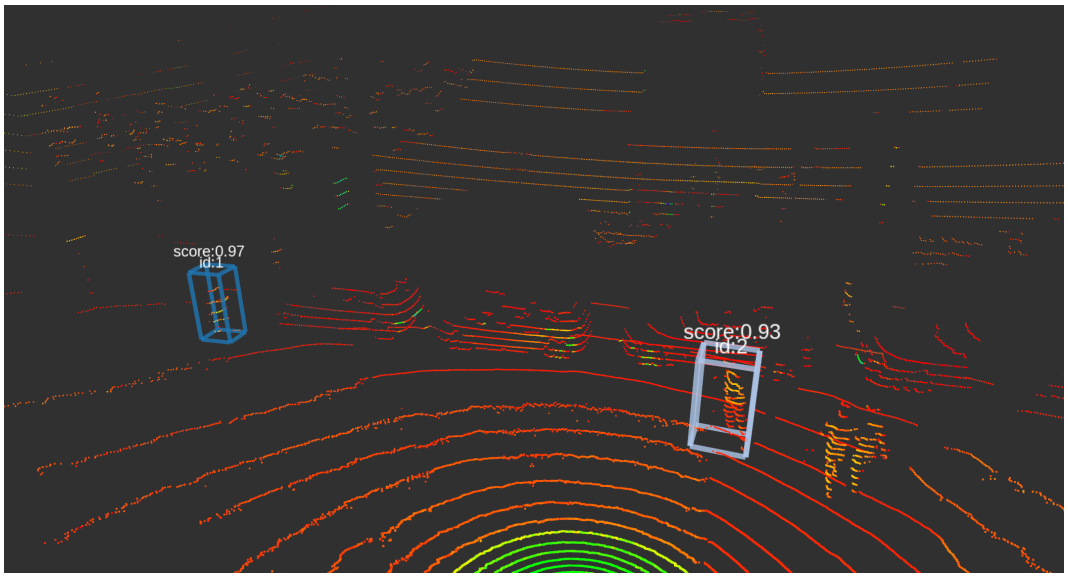
$$F = \begin{bmatrix} 1 & 0 & 0 & dt & 0 & 0 \\ 0 & 1 & 0 & 0 & dt & 0 \\ 0 & 0 & 1 & 0 & 0 & dt \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

5.3.2 Visualização

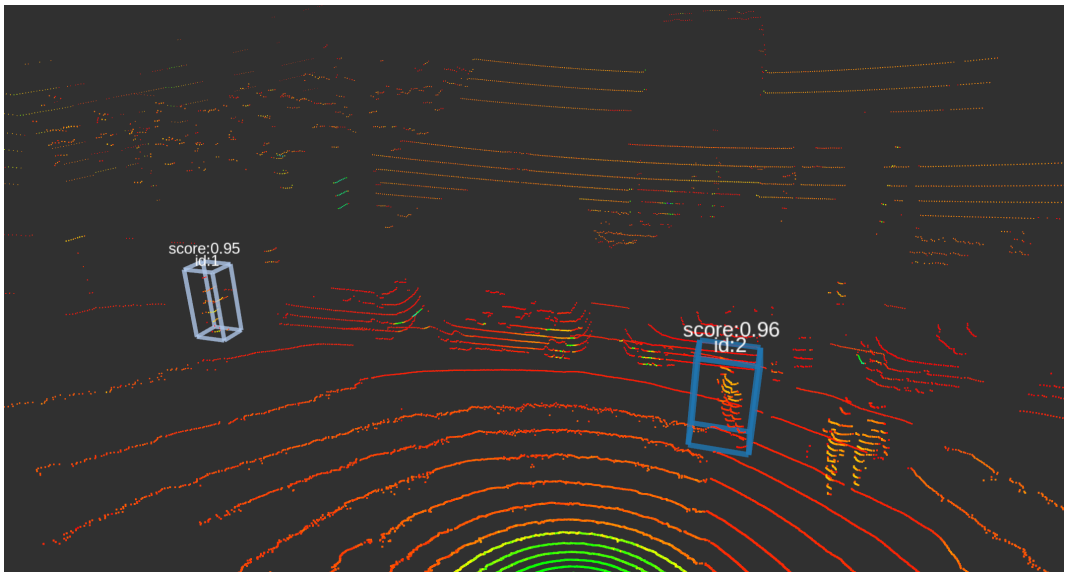
Na figura 5.13 podemos observar o correto funcionamento do algoritmo de identificação das deteções. As *bbox* observadas são do tópico *d*, significando isto que os seus IDs são gerados aleatoriamente pelo modelo de deteção, enquanto o texto representado por "id:x" é gerado pelo algoritmo de identificação. A cor das *bbox* são representada pelo seu ID, mudando de cor se o ID também se alterar. Observando a figura b), verificamos que apesar do ID da deteção, representado pela cor da *bbox*, ter mudado em relação à figura a), o algoritmo de identificação continuou a representar cada pessoa com o seu ID original.

Na figura 5.14 podemos observar o funcionamento do algoritmo de previsão das deteções. Utilizando o filtro de Kalman, conseguimos estimar a posição da *bbox* após o desaparecimento da deteção.

Na Figura 5.15 é possível observar o trajeto percorrido por cada pessoa desde o momento em que o algoritmo de *tracking* a começou a seguir. Este comportamento é verificado em todos os *logs* obtidos, e estão representados pelas figuras 5.16, 5.17, 5.18, 5.19, 5.20 e 5.21.

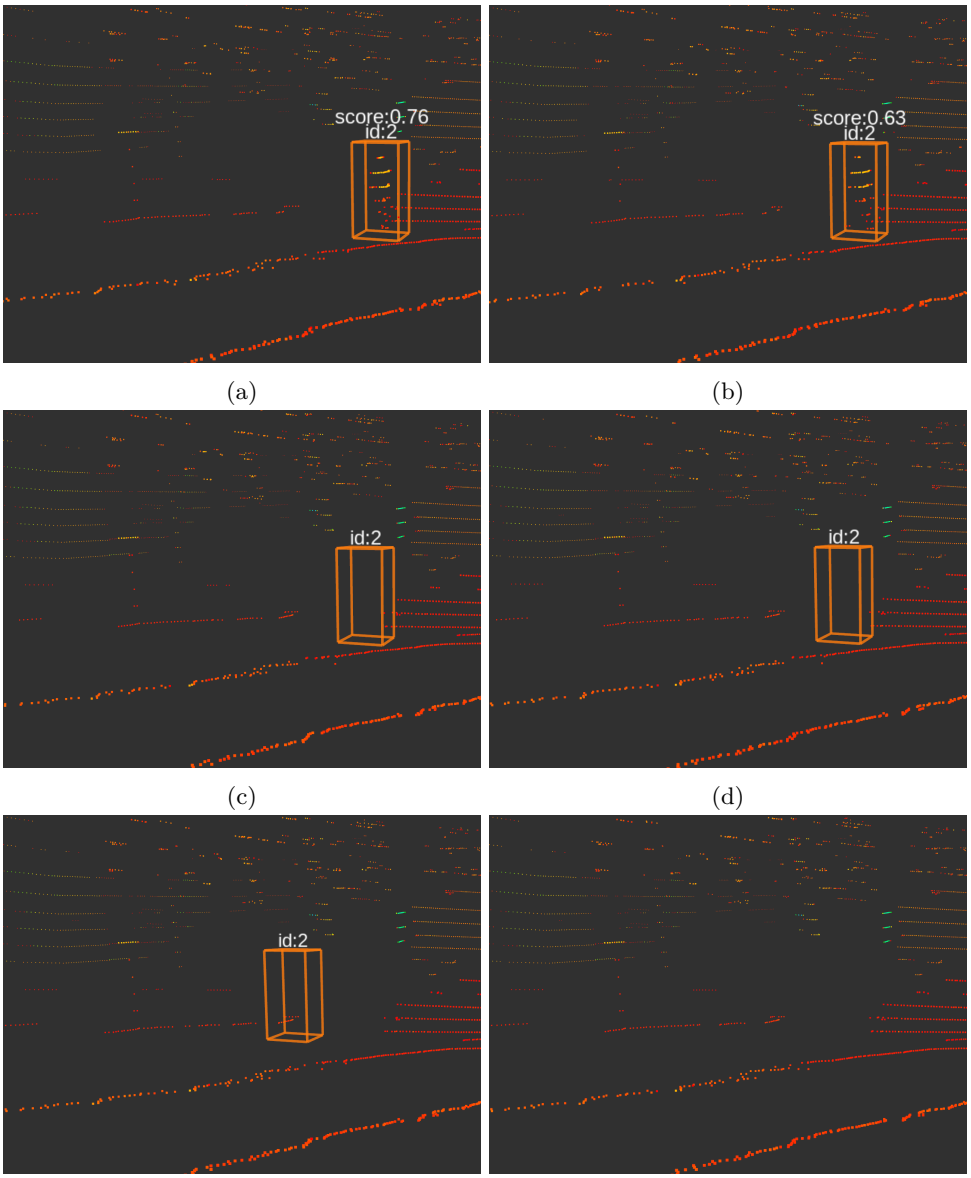


(a)



(b)

Figura 5.13: Funcionamento correto do algoritmo de identificação.

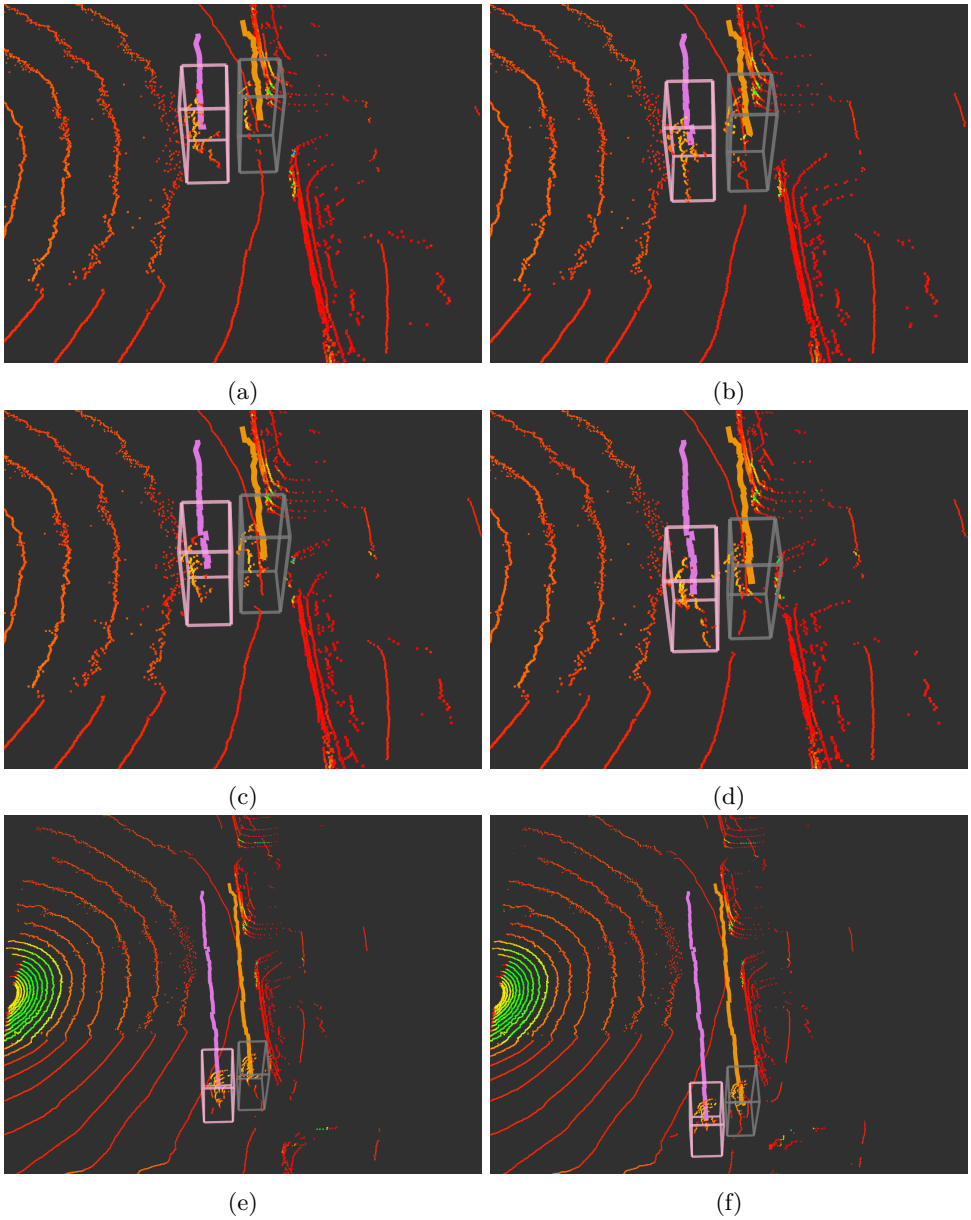


(a) (b)

(c) (d)

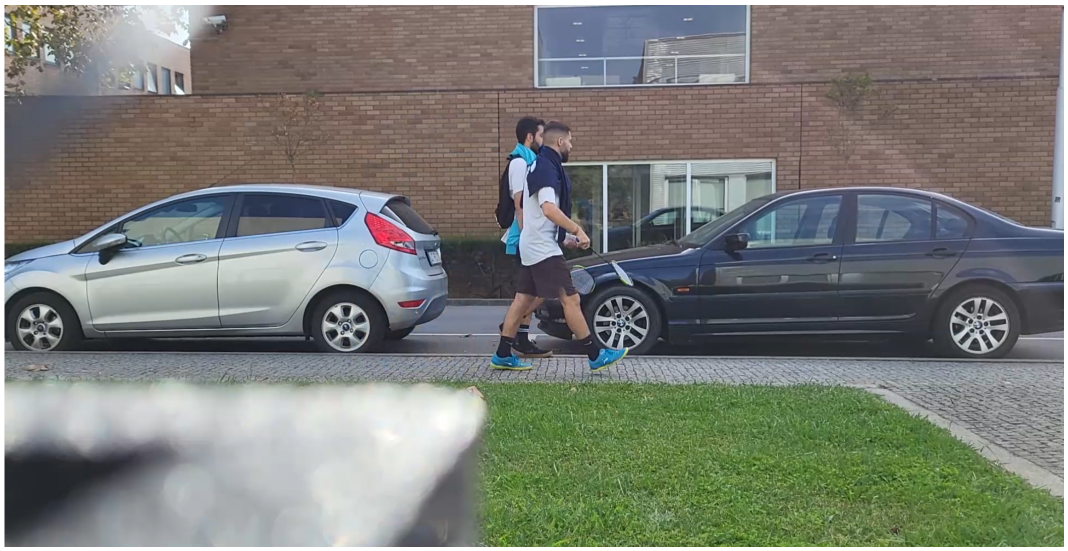
(e) (f)

Figura 5.14: Visualização do *tracking* de uma pessoa em (a) e (b). O algoritmo de previsão mantém a representação da *bbox* da pessoa em (c) e (d), utilizando o filtro de Kalman para obter previsões com base no modelo de movimento anterior. Após 10 iterações, podemos observar o algoritmo de previsão a descartar a previsão em (e) e (f).

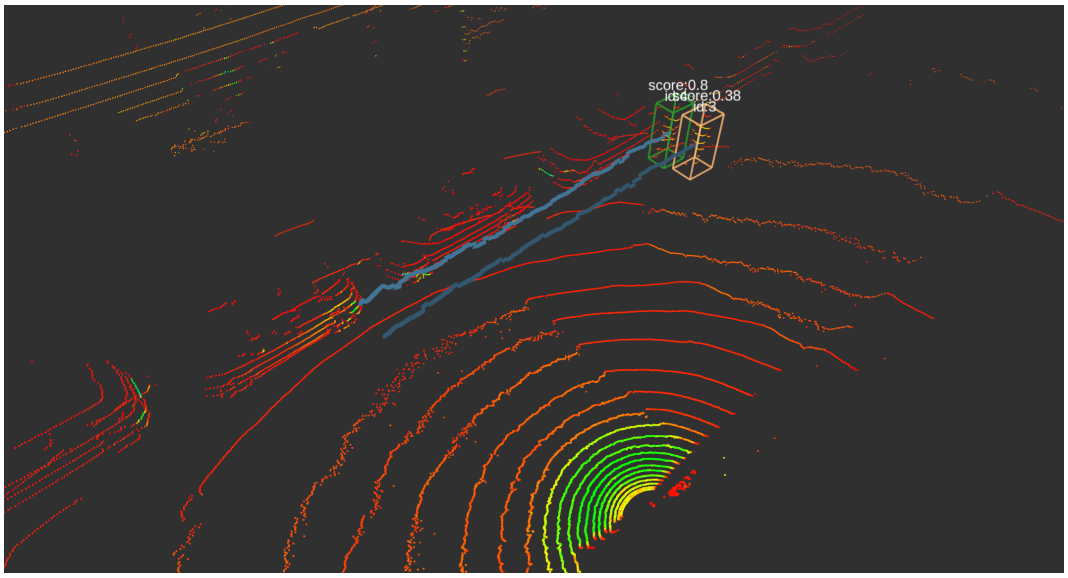


(a) (b) (c) (d) (e) (f)

Figura 5.15: Visualização do caminho percorrido por cada pessoa em iterações sequenciais.

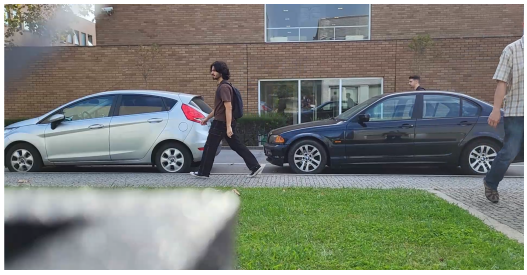


(a)



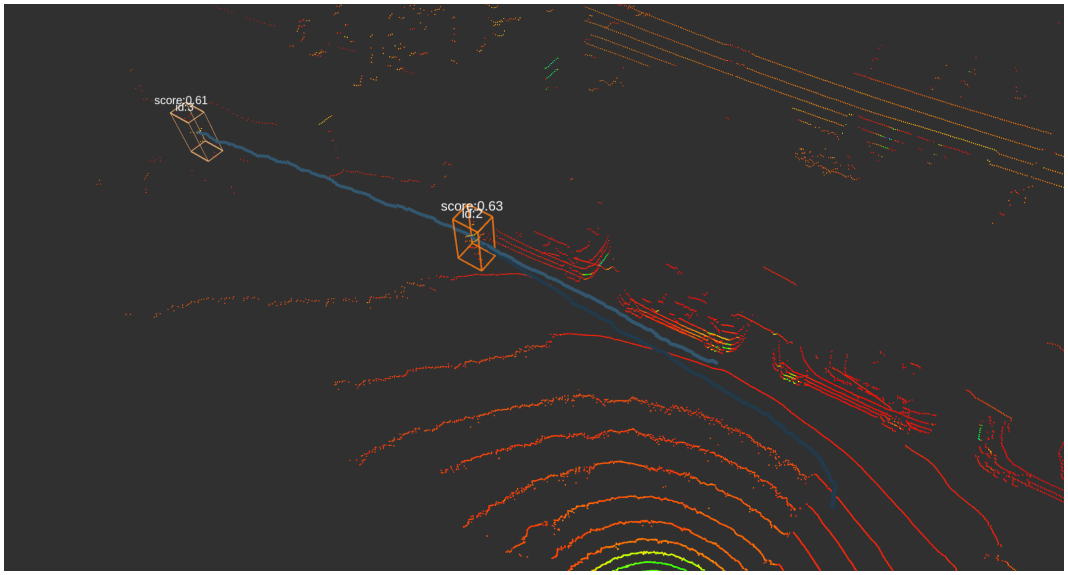
(b)

Figura 5.16: Trajetória de dois pedestres a caminhar paralelamente entre si.



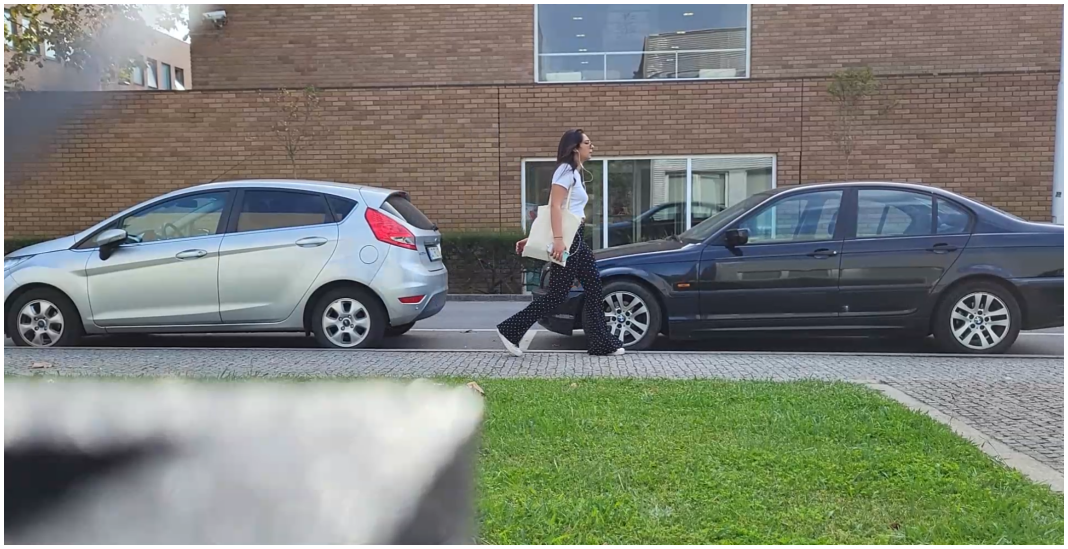
(a)

(b)

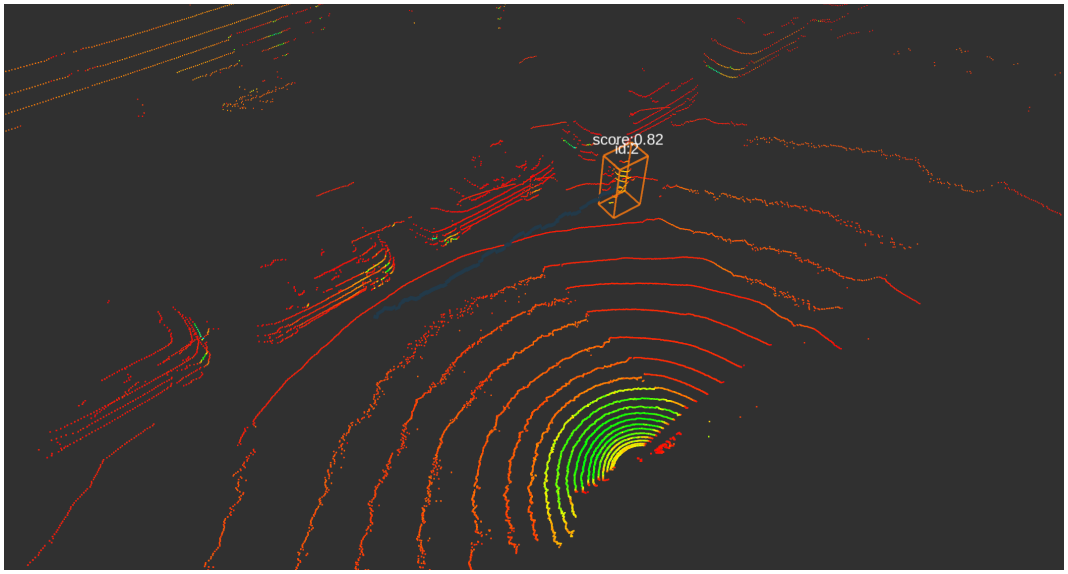


(c)

Figura 5.17: Trajetória equivalente de dois pedestres com um desfasamento.

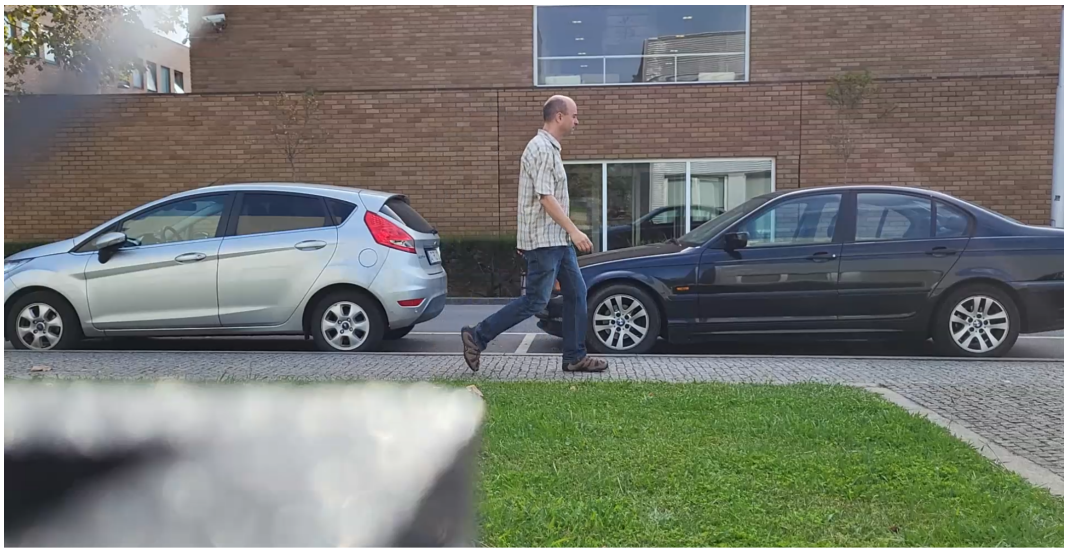


(a)

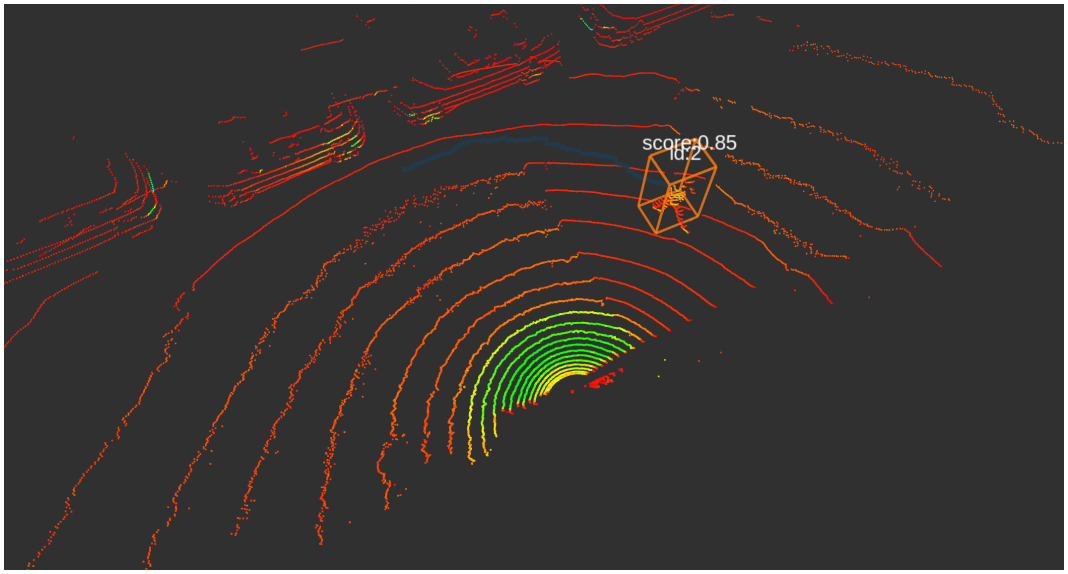


(b)

Figura 5.18: Trajetória linear de um pedestre sozinho.



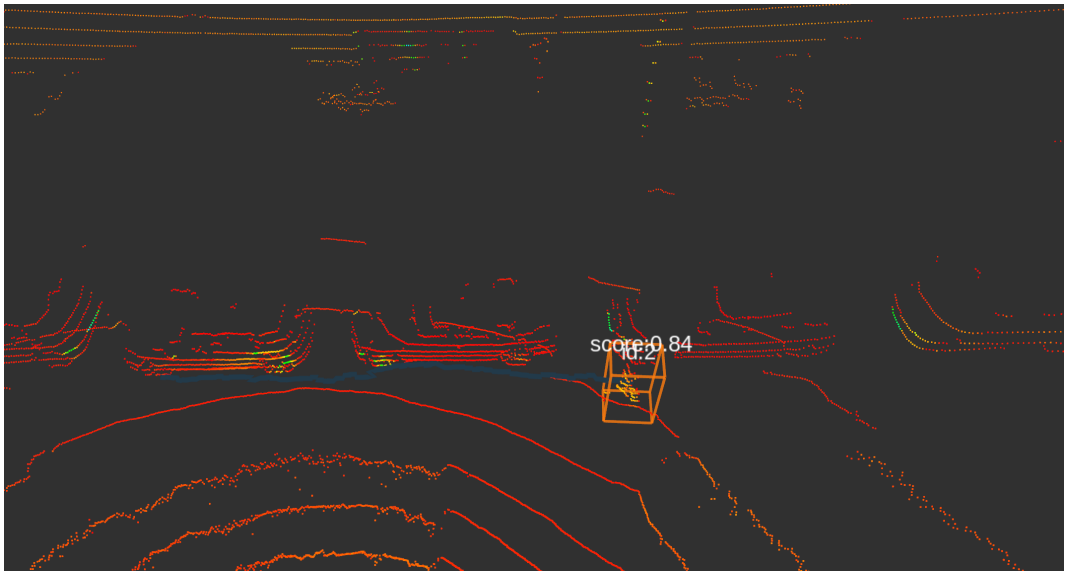
(a)



(b)

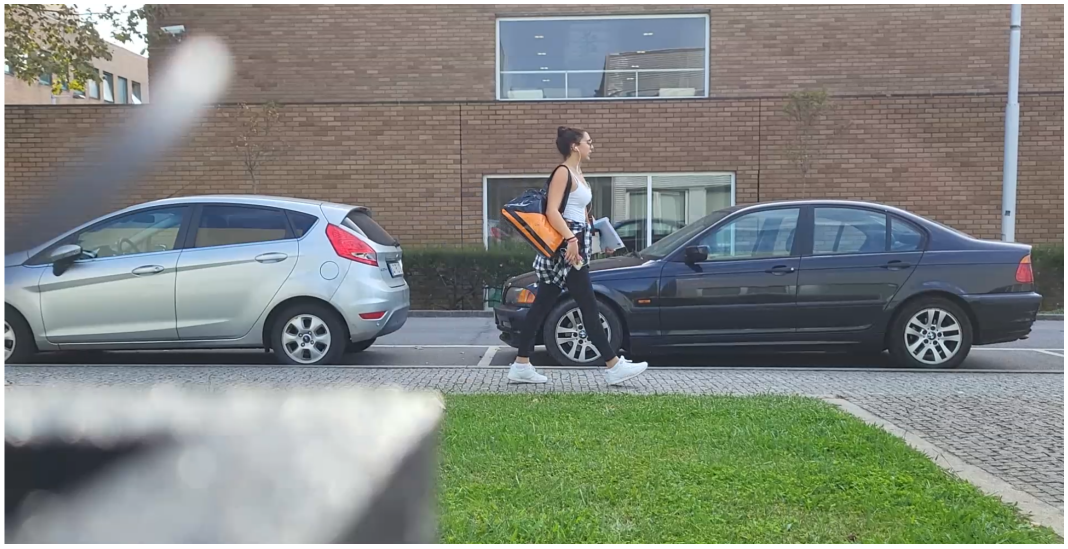
Figura 5.19: Trajetória angular de um pedestre sozinho.

(a)

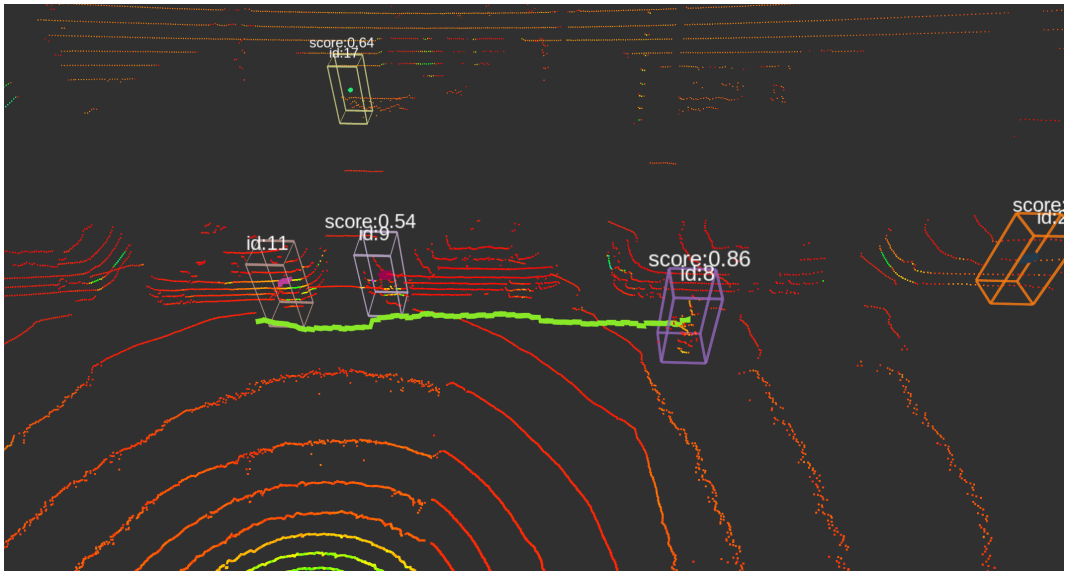


(b)

Figura 5.20: Trajetória de um pedestre que carrega um saco do lixo.



(a)



(b)

Figura 5.21: Trajetória de um pedestre com um saco de desporto, onde é possível identificar alguns FP na sua vizinhança.

Capítulo 6

Conclusão e Trabalho Futuro

Ao longo desta dissertação, foram abordados e cumpridos os objetivos inicialmente propostos. Foi realizada uma investigação sobre algoritmos e métodos adequados para a deteção e *tracking* de pessoas utilizando sensores LiDAR, permitindo a seleção das abordagens mais adequadas para cada etapa do processo. A solução proposta nesta dissertação demonstrou um desempenho satisfatório de deteção de pessoas, conseguindo alcançar um valor F1 Score a rondar os 90% para o nosso *dataset*. Os algoritmos selecionados apresentaram boa consistência na classificação e identificação de pessoas. Além disso, a utilização do filtro Kalman no processo de *tracking* permitiu acompanhar com sucesso o movimento das pessoas identificadas, proporcionando uma estimativa precisa das suas trajetórias. A integração do sistema de deteção de objetos, recorrendo a implementações *open-source*, com um sistema de *tracking* e a utilização da *framework* ROS (Robot Operating System) possibilitaram uma abordagem modular e flexível, facilitando o reaproveitamento do sistema em futuros projetos. A avaliação do desempenho do sistema demonstrou resultados satisfatórios em termos de precisão de deteção e custo computacional. As métricas utilizadas mostraram que o sistema foi capaz de detetar com precisão a maioria das pessoas presentes no ambiente de teste, com uma taxa de falsos positivos relativamente baixa. Além disso, o sistema apresentou uma boa eficiência computacional, garantindo tempos de resposta adequados para aplicações em tempo real.

Com base nas conclusões obtidas nesta dissertação, sugere-se a extensão do uso da solução proposta de deteção e *tracking* para outros cenários, podendo ser utilizada para detetar a presença de aves próximas às turbinas eólicas, *geofencing*, permitindo

a implementação de medidas preventivas para evitar acidentes e garantir a segurança dessas aves. Poderá também ser utilizado para auxiliar na localização de objetos não desejáveis à superfície da água, aplicando-se em lagos, rios e oceanos.

Referências

[1] W. Zimmer, E. Ercelik, X. Zhou, X. J. D. Ortiz, and A. Knoll, "A survey of robust 3d object detection methods in point clouds," 2022. [Citado nas páginas ix, xiii, 1, 28 e 30]

[2] K. Vedder, "Current approaches and future directions for point cloud object detection in intelligent agents," 2021. [Citado nas páginas xiii, 1, 30 e 38]

[3] R. Roriz, J. Cabral, and T. Gomes, "Automotive lidar technology: A survey," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 7, pp. 6282–6297, 2022. [Citado na página 1]

[4] Y. Li, L. Ma, Z. Zhong, F. Liu, M. A. Chapman, D. Cao, and J. Li, "Deep learning for lidar point clouds in autonomous driving: A review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 8, pp. 3412–3432, 2021. [Citado na página 2]

[5] M. Kragh, R. N. Jørgensen, and H. Pedersen, "Object detection and terrain classification in agricultural fields using 3d lidar data," in *Computer Vision Systems* (L. Nalpantidis, V. Krüger, J.-O. Eklundh, and A. Gasteratos, eds.), (Cham), pp. 188–197, Springer International Publishing, 2015. [Citado na página 2]

[6] J. Wang, W. Sun, W. Shou, X. Wang, C. Wu, H.-Y. Chong, Y. Liu, and C. Sun, "Integrating bim and lidar for real-time construction quality control," *Journal of Intelligent & Robotic Systems*, vol. 79, pp. 417–432, Aug 2015. [Citado na página 2]

[7] T. D'orazio, M. Leo, and A. Distante, "Eye detection in face images for a driver vigilance system," in *IEEE Intelligent Vehicles Symposium, 2004*, pp. 95–98, IEEE, 2004. [Citado nas páginas 2 e 40]

[8] C. M. Albrecht, C. Fisher, M. Freitag, H. F. Hamann, S. Pankanti, F. Pezzutti, and F. Rossi, "Learning and recognizing archeological features from lidar data," in *2019 IEEE International Conference on Big Data (Big Data)*, pp. 5630–5636, 2019. [Citado na página 2]

[9] H. Liu, X. Shen, L. Cao, T. Yun, Z. Zhang, X. Fu, X. Chen, and F. Liu, "Deep learning in forest structural parameter estimation using airborne lidar data,"

IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, vol. 14, pp. 1603–1618, 2021. [Citado na página 2]

[10] C. Janiesch, P. Zschech, and K. Heinrich, "Machine learning and deep learning," *Electronic Markets*, vol. 31, pp. 685–695, Sep 2021. [Citado nas páginas 5, 8, 9, 10, 11 e 13]

[11] A. Abraham, "Artificial neural networks," *Handbook of measuring system design*, 2005. [Citado na página 5]

[12] "An introduction to automotive lidar." `https://www.ti.com/`. [Citado na página 6]

[13] "Automotive sensores." `https://www.bmw.com/en/innovation/automotive-sensors.html`. [Citado na página 6]

[14] "What is prescriptive analytics and how can it help you?." `https://www.vovia.com/blog/analytics/what-is-prescriptive-analytics/`. [Citado nas páginas ix e 7]

[15] P. R. Norvig and S. A. Intelligence, "A modern approach," *Prentice Hall Upper Saddle River, NJ, USA: Rani, M., Nayak, R., & Vyas, OP (2015). An ontology-based adaptive personalized e-learning system, assisted by software agents on cloud storage. Knowledge-Based Systems*, vol. 90, pp. 33–48, 2002. [Citado na página 9]

[16] A. Jayanth Balaji, D. Harish Ram, and B. B. Nair, "Applicability of deep learning models for stock price forecasting an empirical study on bankex data," *Procedia Computer Science*, vol. 143, pp. 947–953, 2018. 8th International Conference on Advances in Computing & Communications (ICACC-2018). [Citado na página 10]

[17] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. [Citado na página 10]

[18] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018. [Citado na página 10]

[19] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`. [Citado nas páginas ix, 11 e 12]

[20] C. Rudin, "Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead," *Nature Machine Intelligence*, vol. 1, pp. 206–215, May 2019. [Citado na página 12]

[21] K. Arai and S. Kapoor, eds., *Advances in Computer Vision*. Springer International Publishing, 2020. [Citado na página 13]

[22] I. Jovančević, H.-H. Pham, J.-J. Orteu, R. Gilblas, J. Harvent, X. Maurice, and L. Brèthes, "3d point cloud analysis for detection and characterization of defects on airplane exterior surface," *Journal of Nondestructive Evaluation*, vol. 36, p. 74, Oct 2017. [Citado na página 13]

[23] T. Highlander and A. Rodriguez, "Very efficient training of convolutional neural networks using fast fourier transform and overlap-and-add," 2016. [Citado na página 14]

[24] T. Highlander and A. Rodriguez, "Very efficient training of convolutional neural networks using fast fourier transform and overlap-and-add," in *BMVC*, 2015. [Citado na página 14]

[25] A. Ioannidou, E. Chatzilari, S. Nikolopoulos, and I. Kompatsiaris, "Deep learning advances in computer vision with 3d data: A survey," *ACM Comput. Surv.*, vol. 50, apr 2017. [Citado na página 14]

[26] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3354–3361, 2012. [Citado nas páginas xiii, 14, 16, 57 e 58]

[27] H. Caesar, V. Bankiti, A. H. Lang, S. Vora, V. E. Liong, Q. Xu, A. Krishnan, Y. Pan, G. Baldan, and O. Beijbom, "nuscenes: A multimodal dataset for autonomous driving," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11621–11631, 2020. [Citado na página 15]

[28] G. Zamanakos, L. Tsochatzidis, A. Amanatiadis, and I. Pratikakis, "A comprehensive survey of lidar-based 3d object detection methods with deep learning for autonomous driving," *Computers & Graphics*, vol. 99, pp. 153–181, 2021. [Citado nas páginas ix, 18, 19, 26, 28 e 36]

[29] J. Wang, R. Lindenbergh, and M. Menenti, "Evaluating voxel enabled scalable intersection of large point clouds," *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. II-3/W5, pp. 25–31, 08 2015. [Citado nas páginas ix e 19]

[30] Y. Xu, X. Tong, and U. Stilla, "Voxel-based representation of 3d point clouds: Methods, applications, and its potential use in the construction industry," *Automation in Construction*, vol. 126, p. 103675, 2021. [Citado nas páginas ix, 19 e 20]

[31] A. H. Lang, S. Vora, H. Caesar, L. Zhou, J. Yang, and O. Beijbom, "Pointpillars: Fast encoders for object detection from point clouds," 2018. [Citado nas páginas 19, 29 e 30]

[32] R. Charles, H. Su, K. Mo, and L. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," 07 2017. [Citado nas páginas ix, 19, 23 e 24]

[33] S. Zhou, W. Wang, X. Li, and Z. Jin, "A spike learning system for event-driven object recognition," 01 2021. [Citado nas páginas ix e 21]

[34] M. Simon, S. Milz, K. Amende, and H.-M. Gross, "Complex-yolo: Real-time 3d object detection on point clouds," 2018. [Citado nas páginas ix e 21]

[35] M. Natali, S. Biasotti, G. Patanè, and B. Falcidieno, "Graph-based representations of point clouds," *Graphical Models*, vol. 73, no. 5, pp. 151–164, 2011. [Citado nas páginas ix e 22]

[36] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017. [Citado na página 22]

[37] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015* (N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, eds.), (Cham), pp. 234–241, Springer International Publishing, 2015. [Citado na página 22]

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016. [Citado na página 22]

[39] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014. [Citado na página 22]

[40] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning* (F. Bach and D. Blei, eds.), vol. 37 of

Proceedings of Machine Learning Research, (Lille, France), pp. 448–456, PMLR, 07–09 Jul 2015. [Citado na página 22]

[41] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *ICML*, pp. 807–814, 2010. [Citado na página 22]

[42] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," 2017. [Citado nas páginas ix, 23, 24 e 25]

[43] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009. [Citado nas páginas 23 e 26]

[44] R. Meng, S. G. Rice, J. Wang, and X. Sun, "A fusion steganographic algorithm based on faster r-cnn," *Computers, Materials & Continua*, vol. 55, no. 1, pp. 1–16, 2018. [Citado na página 23]

[45] J. Huang and S. You, "Point cloud labeling using 3d convolutional neural network," 2016. [Citado na página 24]

[46] X. Li, J. Guivant, N. Kwok, Y. Xu, R. Li, and H. Wu, "Three-dimensional backbone network for 3d object detection in traffic scenes," 2019. [Citado na página 24]

[47] S. Shi, X. Wang, and H. Li, "Pointrcnn: 3d object proposal generation and detection from point cloud," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–779, 2019. [Citado na página 27]

[48] G. Li, M. Müller, G. Qian, I. C. D. Perez, A. Abualshour, A. K. Thabet, and B. Ghanem, "Deepgcns: Making gcns go as deep as cnns," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021. [Citado na página 27]

[49] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon, "Dynamic graph cnn for learning on point clouds," 2018. [Citado na página 27]

[50] W. Liu, Q. Cheng, Z. Deng, and M. Jia, "C-gcn: A flexible csi phase feature extraction network for error suppression in indoor positioning," *Entropy*, vol. 23, no. 8, 2021. [Citado na página 27]

[51] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. v. d. Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," 2017. [Citado na página 27]

[52] Y. Zhou and O. Tuzel, "Voxelnet: End-to-end learning for point cloud based 3d object detection," 2017. [Citado nas páginas 27, 28, 29, 30 e 38]

[53] T. Yin, X. Zhou, and P. Krahenbuhl, "Center-based 3d object detection and tracking," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11784–11793, 2021. [Citado nas páginas 28 e 30]

[54] X. Wang and K. M. Kitani, "Cost-aware evaluation and model scaling for lidar-based 3d object detection," 2022. [Citado nas páginas 28 e 37]

[55] Y. Yan, Y. Mao, and B. Li, "Second: Sparsely embedded convolutional detection," *Sensors*, vol. 18, no. 10, 2018. [Citado nas páginas x, 29, 30, 37, 38 e 39]

[56] D. Wang, C. Huang, Y. Wang, Y. Deng, and H. Li, "A 3d multiobject tracking algorithm of point cloud based on deep learning," *Mathematical Problems in Engineering*, vol. 2020, pp. 1–10, 12 2020. [Citado nas páginas 31 e 33]

[57] "Object tracking with deep learning." `https://viso.ai/deep-learning/object-tracking/`. [Citado na página 32]

[58] Q. Wang, L. Zhang, L. Bertinetto, W. Hu, and P. H. Torr, "Fast online object tracking and segmentation: A unifying approach," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2019. [Citado na página 32]

[59] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple online and realtime tracking," in *2016 IEEE international conference on image processing (ICIP)*, pp. 3464–3468, IEEE, 2016. [Citado na página 33]

[60] H. W. Kuhn, "The hungarian method for the assignment problem," *Naval Research Logistics (NRL)*, vol. 52, 1955. [Citado nas páginas 33, 40 e 41]

[61] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," 2017. [Citado nas páginas 33 e 40]

[62] X. Weng, J. Wang, D. Held, and K. Kitani, "Ab3dmot: A baseline for 3d multi-object tracking and new evaluation metrics," 2020. [Citado nas páginas 33 e 40]

[63] "Velodyne lidar." `https://velodynelidar.com/`. [Citado na página 36]

[64] P. Sun, H. Kretzschmar, X. Dotiwalla, A. Chouard, V. Patnaik, P. Tsui, J. Guo, Y. Zhou, Y. Chai, B. Caine, V. Vasudevan, W. Han, J. Ngiam, H. Zhao, A. Timofeev, S. Ettinger, M. Krivokon, A. Gao, A. Joshi, Y. Zhang, J. Shlens, Z. Chen, and D. Anguelov, "Scalability in perception for autonomous driving: Waymo open dataset," in *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2443–2451, 2020. [Citado na página 37]

[65] S. Benhimane, H. Najafi, M. Grundmann, Y. Genc, N. Navab, and E. Malis, "Real-time object detection and tracking for industrial applications.," in *VISAPP (2)*, pp. 337–345, 2008. [Citado na página 40]

[66] G. Welch, G. Bishop, *et al.*, "An introduction to the kalman filter," 1995. [Citado na página 42]

[67] R. J. Meinhold and N. D. Singpurwalla, "Understanding the kalman filter," *The American Statistician*, vol. 37, no. 2, pp. 123–127, 1983. [Citado na página 44]

[68] T. M. Inc., "Matlab version: 9.10.0 (r2021a)," 2021. [Citado nas páginas x, 54 e 56]

[69] T. M. Inc., "Lidar toolbox version: 1.1 (r2021a)," 2022. [Citado na página 54]