UNIVERSIdade FEEVALE

MAURICIO BONETTI TARDETTI

Navegação autônoma utilizando simulador CARLA

Novo Hamburgo

2020

Mauricio Bonetti Tardetti

Navegação autônoma utilizando simulador CARLA

Trabalho de Conclusão de Curso

apresentado como requisito parcial

à obtenção do grau de Bacharel em

Ciência da Computação pela

Universidade Feevale

Orientador: Ricardo Ferreira de Oliveira

Novo Hamburgo

2020

Agradecimentos

Gostaria de agradecer a todos os que ajudaram nesses momentos difíceis. Em especial a minha mãe Elaine Fátima Bonetti e minha namorada Karine Bender por todo o apoio durante todo o percurso da graduação e TC.

Aos amigos, minha gratidão, pelo apoio emocional - nos períodos mais difíceis do trabalho. Também estendo minha gratidão ao mestre Ricardo Ferreira de Oliveira, orientador deste trabalho.

Agradeço também a oportunidade de estudar em uma instituição como a FEEVALE, que só foi possível graças uma bolsa do PROUNI.

Em memória de Tiago de Souza um dos meus melhores amigos que deu a vida ao defender uma amiga de um feminicídio no dia 5 de outubro de 2020. *“Diante da vastidão do tempo e da imensidão do universo, é um imenso prazer para mim dividir um planeta e uma época com você.” (Carl Sagan).*

Resumo

Seguindo a evolução dos sistemas de aprendizagem de máquina nos anos recentes, permitidos pelos avanços possibilitados pelo uso de placas gráficas para processamento altamente paralelizado, grandes benefícios foram proporcionados para as diversas áreas relacionadas como a visão computacional e o desenvolvimento de redes neurais cada vez mais complexas. A conjunção destas e diversas outras áreas de estudo beneficiaram e permitem hoje em dia o desenvolvimento de modelos eficientes de navegação autônoma utilizando variadas técnicas ou combinando-as para obter melhores resultados. Este trabalho tem como objetivo combinar técnicas de aprendizagem de máquina, *pathfinding* e visão computacional para desenvolvimento de um modelo de direção autônoma. Para esse desenvolvimento, será utilizado o simulador CARLA que permite a simulação de diversos cenários em ambientes urbanos, incluindo sistemas dinâmicos que permitem avaliação em condições adversas. Como resultados preliminares da integração de um modelo de detecção de objetos ao agente de navegação uma taxa de sucesso de 72%, mas com melhorias a serem feitas esses resultados podem ser aprimorados.

Palavras-chave: Inteligência artificial, Visão computacional, Direção autônoma.

Abstract

Following the evolution of machine learning systems in recent years, allowed by the advances made possible using graphics cards for highly parallelized processing, which bring great benefits in several related areas, such as computer vision and the development of increasingly complex neural networks. The combination of these and several other areas of study benefits and can be used today in the development of efficient models of automatic navigation, using specific techniques or combining many to obtain better results. This work aims to combine machine learning techniques, pathfinding, and computer vision for the development of an autonomous driving model. For this development, the CARLA simulator will be used. The CARLA simulator allows the simulation of different scenarios in urban environments, including dynamic systems that allow assessments of adverse conditions. Based on studies already conducted in the area, such as Bojarski's model and others for the evaluation and validation of results obtained.

Keywords: Artificial intelligence, Autonomous Driving, Computer Vision.

Lista de Figuras

[Figura 1 - Representação gráfica para um problema de minimização 17](#_Toc55560310)

[Figura 2 – Exemplo de árvore de decisão 18](#_Toc55560311)

[Figura 3 - Exemplo de arvore de decisão com resultados lineares 19](#_Toc55560312)

[Figura 4 - Modelo matemático de um neurônio 20](#_Toc55560313)

[Figura 5 - Representação visual de uma rede neural artificial 21](#_Toc55560314)

[Figura 6 - Gráfico de comparação entre algoritmos de *pathfinding* 22](#_Toc55560315)

[Figura 7 - Exemplo de detecção de rostos, roupas e cabelos 23](#_Toc55560316)

[Figura 8 - Vetor de parâmetros para definição de um veículo 25](#_Toc55560317)

[Figura 9 - Representação dos parâmetros para definição de um veículo 25](#_Toc55560318)

[Figura 10 - Modelos parametrizados por Koller 26](#_Toc55560319)

[Figura 11 - Funções que compõem a DDT 28](#_Toc55560320)

[Figura 12 - Demonstração dos recursos de clima dinâmico do simulador CARLA 29](#_Toc55560321)

[Figura 13 - Exemplo dos sensores em ação no simulador 30](#_Toc55560322)

[Figura 14 - Visualização do sistema utilizado por Bojarski et al. 32](#_Toc55560323)

[Figura 15 - Diagrama da simulação 33](#_Toc55560324)

[Figura 16 - Fórmula para cálculo da métrica de autonomia 33](#_Toc55560325)

[Figura 17 - Resultados obtidos por Dosovitskiy et al. 34](#_Toc55560326)

[Figura 18 - Mapa utilizado na simulação 36](#_Toc55560327)

[Figura 19 - Convergência de um controlador PID 38](#_Toc55560328)

[Figura 20 - Variação da velocidade do agente com o tempo 39](#_Toc55560329)

[Figura 21 - Fórmula para calcular o erro do controlador PID lateral 40](#_Toc55560330)

[Figura 22 - Gráfico do valor de *steer* a cada frame 40](#_Toc55560331)

[Figura 23 - Avanços na precisão através do uso de *Deep Learning* 41](#_Toc55560332)

[Figura 24 - Frameworks e abstrações dos mesmos que fazem uso de CUDA 42](#_Toc55560333)

[Figura 25 - Comparação de desempenho Tensorflow GPU x CPU 43](#_Toc55560334)

[Figura 26 - Detecção de objetos com o *Object Detection API* 44](#_Toc55560335)

[Figura 27 - TensorBoard para acompanhamento do modelo 46](#_Toc55560336)

[Figura 28 - Coleta de imagens com o CARLA-2DBBox 47](#_Toc55560337)

[Figura 29 - Modelo de anotação PascalVOC 48](#_Toc55560338)

[Figura 30 - Interface da ferramenta labelimg 49](#_Toc55560339)

[Figura 31 - mAP para objetos grandes 50](#_Toc55560340)

[Figura 32 - mAP para objetos médios 50](#_Toc55560341)

[Figura 33 - mAP para objetos pequenos 51](#_Toc55560342)

[Figura 34 - Modelo treinado aplicado na simulação 51](#_Toc55560343)

[Figura 35 - Sucessos x Falhas na simulação do agente 53](#_Toc55560344)

[Figura 36 - Velocidade média do agente 54](#_Toc55560345)

[Figura 37 - Distância percorrida pelo agente 54](#_Toc55560346)

Lista de tabelas

[Tabela 1 – Exemplo de regras geradas por um algoritmo de classificação 18](#_Toc55560347)

[Tabela 2 - Classificações dos sistemas de direção 28](#_Toc55560348)

[Tabela 3 - Sensores do tipo câmera 30](#_Toc55560349)

[Tabela 4 – Tipos de sensores para detecção de eventos 30](#_Toc55560350)

[Tabela 5 - Outros tipos de sensores do simulador 31](#_Toc55560351)

[Tabela 6 - Modelos pré-treinados disponíveis 44](#_Toc55560352)

[Tabela 7 - Quadros por segundo máximo esperado para cada modelo 47](#_Toc55560353)

Lista de Abreviaturas e Siglas

|  |  |
| --- | --- |
| CNN | Convolucional Neural Networks |
| GPU | Graphics Computing unit (Processador gráfico) |
| API | *Application Programming Interface* (Interface de programação de aplicações) |
| CARLA  GPS | Car Learning to Act  Global Positioning System (Sistema de posicionamento global) |
| RGB | *Red, Green, Blue* |
| GNSS | *Global Navigation Satellite System* |
| NPC  ODD  mAP  FPS  QPS  SAE  DDT  PID  ML  LIDAR | *Non-playable Character* (Ator não controlável)  *Operational Design Domain* (domínio de operação)  mean Average Precision  Quadros por segundo  Quadros por segundo  Society of Automotive Engineers  Dynamic Driving Task  Proportional integral derivative (Controlador proporcional integral derivativo)  Machine Leaning (Aprendizado de máquina)  *LIght Detecting And Ranging* (Captura em 3D do ambiente através de pontos de reflexão) |

Sumário

[1 Introdução 12](#_Toc55560735)

[1.1 OBJETIVOS 14](#_Toc55560736)

[1.1.1 Objetivo geral 14](#_Toc55560737)

[1.1.2 Objetivos específicos 14](#_Toc55560738)

[1.2 Estrutura do trabalho 14](#_Toc55560739)

[2 metodologia 15](#_Toc55560740)

[3 Aprendizagem de máquina 16](#_Toc55560741)

[3.1 Definição 16](#_Toc55560742)

[3.1.1 Arvores de decisão 17](#_Toc55560743)

[3.1.2 Modelos lineares 18](#_Toc55560744)

[3.2 Aplicações 19](#_Toc55560745)

[3.3 Redes neurais artificiais 20](#_Toc55560746)

[3.4 Pathfinding 21](#_Toc55560747)

[4 Visão computacional 23](#_Toc55560748)

[4.1 aplicações 23](#_Toc55560749)

[4.2 Reconhecimento de objetos 24](#_Toc55560750)

[4.2.1 Objetos em movimento 24](#_Toc55560751)

[4.3 Redes neurais convolucionais aplicadas a visão computacional 26](#_Toc55560752)

[5 Navegação autonoma 27](#_Toc55560753)

[5.1 Estado atual 27](#_Toc55560754)

[5.2 classificação sistemas de direcão 27](#_Toc55560755)

[5.3 Simulador CARLA 29](#_Toc55560756)

[5.3.1 API ABERTA 31](#_Toc55560757)

[5.4 Estudos na área 32](#_Toc55560758)

[6 Desenvolvimento do agente de navegação 35](#_Toc55560759)

[6.1 Planejamento de rotas 35](#_Toc55560760)

[6.2 Controle do estado do agente 36](#_Toc55560761)

[6.3 Controle longitudinal 37](#_Toc55560762)

[6.4 Controle lateral 39](#_Toc55560763)

[7 detecção de objetos e eventos 41](#_Toc55560764)

[7.1 cuda para aceleração de deep learning 42](#_Toc55560765)

[7.2 detecão de objetos com tensorflow 43](#_Toc55560766)

[7.2.1 Treinando um modelo 44](#_Toc55560767)

[7.3 Reconhecimento de objetos aplicado ao carla 46](#_Toc55560768)

[7.3.1 Obtendo imagens para o *dataset* 47](#_Toc55560769)

[7.3.2 Pré-processamento e classificação manual 48](#_Toc55560770)

[7.3.3 Resultados do modelo 49](#_Toc55560771)

[7.3.4 Considerações referentes ao modelo e a simulação 52](#_Toc55560772)

[8 Resultados do agente 53](#_Toc55560773)

[9 CONCLUSÃO 56](#_Toc55560774)

[Referências Bibliográficas 58](#_Toc55560775)

# Introdução

No horizonte da evolução dos meios de transporte, chegou-se a um ponto onde um dos maiores objetivos de grande parte dos fabricantes é trazer modelos que possuam alguma forma de direção autônoma. Até o presente momento não existem veículos completamente autônomos circulando nas ruas das cidades, seja por limitações técnicas, imprevisibilidade do tráfego e outros problemas recorrentes no trânsito. Na vanguarda, com um sistema de piloto automático, está a fabricante Tesla com seu sistema “Autopilot”, porém, como referido anteriormente, assim como outros, ele ainda não possui capacidade total de direção autônoma.

Outro ponto a se levar em consideração a respeito dos veículos autônomos é a forma como sua adoção se daria. Conforme Howard (2013) em uma pesquisa realizada com 107 residentes de Berkeley na California as maiores preocupações dos entrevistados dizem respeito ao preço, confiança e ao controle que terão sobre esses sistemas de direção autônoma, porém, dentre as características que mais atraem para esse tipo de veículo é sua segurança e conveniência.

Há um grande potencial a ser explorado na área de veículos autônomos, formas de torná-los mais seguros, eficientes e confiáveis. Mas todos esses pontos dependem fortemente da adoção destes veículos no dia a dia, de acordo com o “National Center for Statistics and Analysis” (2018) no ano de 2016 houve mais de 37 mil vítimas fatais em acidentes causados por veículos. A estatística para direção sob efeito de bebidas alcóolicas para o mesmo ano é de uma morte a cada 51 minutos.

Muitas das estatísticas anteriormente apresentadas poderiam ser reduzidas com a adoção de uma frota de veículos autônomos. Além disso, problemas de tráfego comuns como congestionamentos poderiam ser tratados de forma muito mais eficiente, uma vez que, com uma comunicação integrada entre os agentes (veículos), poderia se determinar o uso de rotas alternativas assim que o problema começar. Evitando o acúmulo de veículos em áreas com problemas.

Como uma forma de validar e avaliar conceitos relacionados a direção autônoma foi desenvolvido um simulador nomeado *CARLA: An Open Urban Driving Simulator*. “Um simulador de código aberto para pesquisa de direção autônoma. CARLA foi desenvolvido desde sua base para suportar o desenvolvimento, treinamento e validação de sistemas urbanos de direção autônoma, em adição a seu código aberto, o simulador também provê de recurso digitais abertos como cidades, construções e veículos que foram criados para esse propósito e podem ser usados livremente, a plataforma suporta um conjunto flexível de sensores e condições de ambiente.” (DOSOVITSKIY et al., 2017, tradução nossa).

Para fazer uso dos dados gerados a partir do simulador se faz necessário o uso de recursos de visão computacional e de aprendizagem de máquina através de redes neurais. “A área de visão computacional foca em replicar partes da complexidade da visão humana, permitindo computadores processar e identificar imagens e vídeos de forma similar a humana” (MIHAJLOVIC, 2019, tradução nossa). Graças aos avanços na área de inteligência artificial, *deep learning* e redes neurais houve um grande salto para a área, permitindo exceder a capacidade humana em determinadas tarefas conforme Mihajlovic (2019).

Com o uso de redes neurais, tendo como entrada os dados obtidos com uso da visão computacional. Estas redes trabalham com o conceito de neurônios, cada um desses possui um peso associado a ele:

As redes neurais artificiais se diferenciam pela sua arquitetura e pela forma como os pesos associados às conexões são ajustados durante o processo de aprendizado. A arquitetura de uma rede neural restringe o tipo de problema no qual a rede poderá ser utilizada, e é definida pelo número de camadas (camada única ou múltiplas camadas), pelo número de nós em cada camada, pelo tipo de conexão entre os nós (*feedforward* ou *feedback*) e por sua topologia. (FERNEDA, E. apud HAYKIN, 2001, p. 46-49).

Conforme o trabalho de Bojarki et al. (2016) onde é apresentado um modelo *end-to-end* para navegação autônoma utilizando apenas uma câmera frontal no veículo foi possível o desenvolvimento de um modelo através do uso de redes neurais convolucionais, onde este, foi capaz de navegar por ruas e rodovias com ou sem marcações de faixas. Isso nos dá uma expectativa potencial de um modelo combinado com mais sensores como por exemplo: LIDAR, sensores de profundidade, detector de obstáculos e outros recursos disponíveis.

Combinando a gama de sensores disponíveis, com a praticidade para realização dos testes no simulador, é possível a obtenção de uma grande quantidade de dados a serem combinados, para o desenvolvimento de um modelo robusto. De forma a permitir a navegação pode-se utilizar um sistema de *machine learning* baseado em regras, conforme Zhou explica:

Navegação de terreno requer o uso de um mapa e coordenadas absolutas, ou pontos de referência. Em um caso em que coordenadas não estão disponíveis ou não são confiáveis, a escolha óbvia é utilizar o ambiente ao redor para extrair conhecimento e auto localizar-se através da observação de arvores, pedras, contornos de terreno e outros objetos como pontos de referência. (2007, tradução nossa)

Dentre as motivações para o desenvolvimento deste trabalho há o interesse do autor pelo assunto, pela área de aprendizagem de máquina, visão computacional e da linguagem Python que é utilizada para fazer interface com o simulador a ser utilizado no desenvolvimento deste. Além disso, trata-se de uma área promissora e de interesse para a área da ciência da computação.

## OBJETIVOS

### Objetivo geral

Desenvolver um modelo de navegação autônoma no simulador CARLA fazendo uso de recursos de inteligência artificial e visão computacional. Permitindo a navegação entre dois pontos do mapa e avaliar sua aplicabilidade em casos reais.

### Objetivos específicos

* Realizar pesquisa bibliográfica sobre o simulador CARLA e aplicações;
* Realizar pesquisa bibliográfica sobre conceitos de usabilidade;
* Desenvolver e treinar a detecção de objetos (placas, semáforos, faixas, veículos e pedestres) no simulador;
* Desenvolver modelo para controle do veículo;
* Avaliar o desempenho do modelo comparando a trabalhos correlatos;
* Validar parcialmente a aplicabilidade do modelo;

## Estrutura do trabalho

O desenvolvimento do trabalho se dará em 8 capítulos. No capítulo 2 será apresentada metodologia do trabalho e nos capítulos posteriores serão abordados conceitos de aprendizagem de máquina, visão computacional, e direção autônoma, assim como o desenvolvimento do agente de navegação e do modelo de detecção de objetos, assim como os resultados do agente.

# metodologia

Baseado em Prodanov e Freitas (2013), o trabalho pode ser classificado como pesquisa aplicada, pois tem como objetivo gerar conhecimento para resolver um problema específico qual seja o desenvolvimento de um módulo em Python para aplicação no simulador CARLA.

Os objetivos deste trabalho permitem enquadrá-lo no conceito de pesquisa exploratória. Será realizada pesquisa bibliográfica em artigos sobre modelos de direção autônoma visando obter informações que permitam melhorias ao modelo a ser desenvolvido. Esta pesquisa consiste em observar as pesquisas já existentes sobre o assunto, para usar como uma linha de base para início da construção do modelo.

A Forma de abordagem será qualitativa. O modelo será validado com a comparação entre resultados obtidos por outros trabalhos correlatos comparando características relacionadas a sua usabilidade, eficiência e eficácia.

O trabalho iniciará com a pesquisa de técnicas de visão computacional e aplicação de algoritmos de aprendizagem de máquina para se determinar e analisar o cenário onde o veículo da simulação se encontra. Serão utilizados algoritmos de *pathfinding* em cima do dos dados que compõe o mapa do sistema da simulação.

A validação será feita com base no artigo de BOJARSKI, et. al (2016) no qual é desenvolvido um modelo *end-to-end* similar ao proposto por este trabalho.

O Projeto também é classificado como experimental, serão realizadas experimentações incrementais para validar e avaliar melhorias e comparação dos resultados entre as iterações do modelo com trabalhos correlatos.

A metodologia de avaliação do agente final a ser considerada utiliza como base a capacidade do mesmo de completar cenários no simulador. O agente será criado em uma posição aleatória entre as disponíveis no simulador e então será sorteado outro ponto aleatório como destino.

O agente deve através da combinação dos dados do mapa do simulador e recursos de visão computacional navegar sem colidir pela rota traçada entre os pontos. Qualquer colisão que ocorra, o agente será resetado para um novo local com um novo destino.

# Aprendizagem de máquina

Aprendizagem de máquina ou *machine learning* é atualmente uma das áreas em maior evidência no estudo de ciência da computação, através de inovações e facilitações trazidas por diversos frameworks como Tensorflow, Torch, Weka, Scikit e outras ferramentas populares levando a um aumento no número de aplicações dessas tecnologias no dia a dia.

Sendo uma área ampla, compreendendo desde algoritmos de busca de caminho, predições, análise de dados, *data mining*, reconhecimento de padrões, redes neurais e beneficiando diversas outras áreas como a da visão computacional.

## Definição

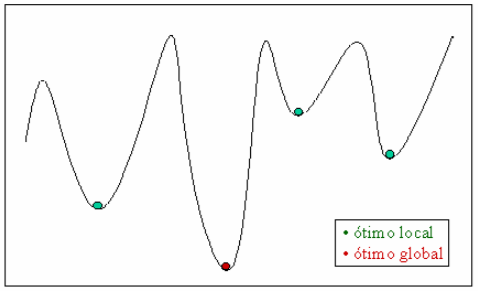
O termo *machine learning* é derivado do verbo aprender, que tem por definição: adquirir conhecimento através de estudo, experiencia ou ensino; tomar consciência de algo através de observação; recorrer a memória; receber instrução. De acordo com Witten et. al. (2011, p. 7) essa definição é insuficiente para definir a aprendizagem computacional. Como saber uma máquina tem conhecimento de algum assunto? A forma de avaliar o conhecimento adquirido por um computador se dá através das respostas.

Além disso, questões como memória são triviais para sistemas de aprendizagem de máquina. A definição mais correta de acordo com Witten refere-se ao desempenho: “Entidade aprendem conforme mudanças de comportamento de forma que as façam ter melhor desempenho no futuro” (Witten et al., 2011, p. 7, tradução nossa), ou seja, sistemas de aprendizagem de máquina através de mudanças de comportamento tentam alcançar melhores resultados.

Tendo como objetivo um melhor resultado, sistemas de aprendizagem de máquinas incorrem em problemas, sua evolução acontece incrementalmente em direção a um melhor resultado e alcançando um ponto onde não há evolução possível, isso pode ser chamado de ótimo local esse fenômeno é demonstrado conforme a Figura 1.

Num problema típico de minimização, encontra-se um ótimo local quando qualquer movimento a ser feito piore o valor atual da função objetivo. Um ótimo global corresponde ao menor valor da função objetivo, entre todos os ótimos locais existentes no espaço de busca. (CHAVES, A., 2003)

Figura 1 - Representação gráfica para um problema de minimização



Fonte: CHAVES, A. (2003)

### Arvores de decisão

Arvores decisão são parte importante de algoritmos de aprendizagem de máquina, principalmente quando envolvem classificação de dados, são representações básicas do “conhecimento” de um algoritmo de ML. Witten et. al. (2011, p. 61) apresenta esse conceito de “conhecimento” apenas como uma forma de chamar a estrutura gerada por esse tipo de algoritmo, uma vez que não é representativo de conhecimento propriamente dito.

Estas arvores são representações de conjuntos de regras gerados por alguns algoritmos de aprendizagem como os algoritmos de classificação, estes normalmente possuem como entrada dados de aprendizagem com seus resultados reais de classificação e a partir deles infere regras que façam uma representação do conjunto, conforme exemplo dado por Witten et. al (20, p. 12) a tabela 1 apresenta regras geradas com dados relativos a condições climáticas e a possibilidade de sair para brincar.

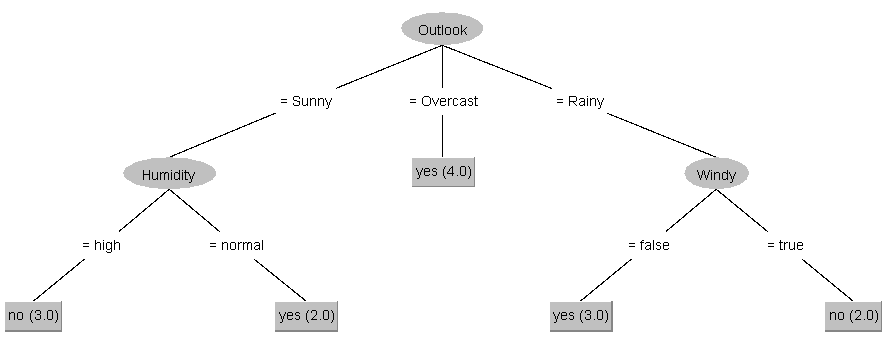
Tabela 1 – Exemplo de regras geradas por um algoritmo de classificação

|  |  |  |
| --- | --- | --- |
| **Regra** | **Saída** | |
| Se tempo = ensolarado e umidade = alta | | brincar = não |
| Se tempo = chuvoso e vento=verdadeiro | | brincar = não |
| Se tempo = nublado | | brincar = sim |
| Se umidade = normal | | brincar = sim |
| Se nenhuma regra anterior | | brincar = sim |

Fonte: WITTEN E. et al (2011)

A partir da tabela 1 pode ser feita a geração da arvore de decisão exposta na Figura 2.

Figura 2 – Exemplo de árvore de decisão



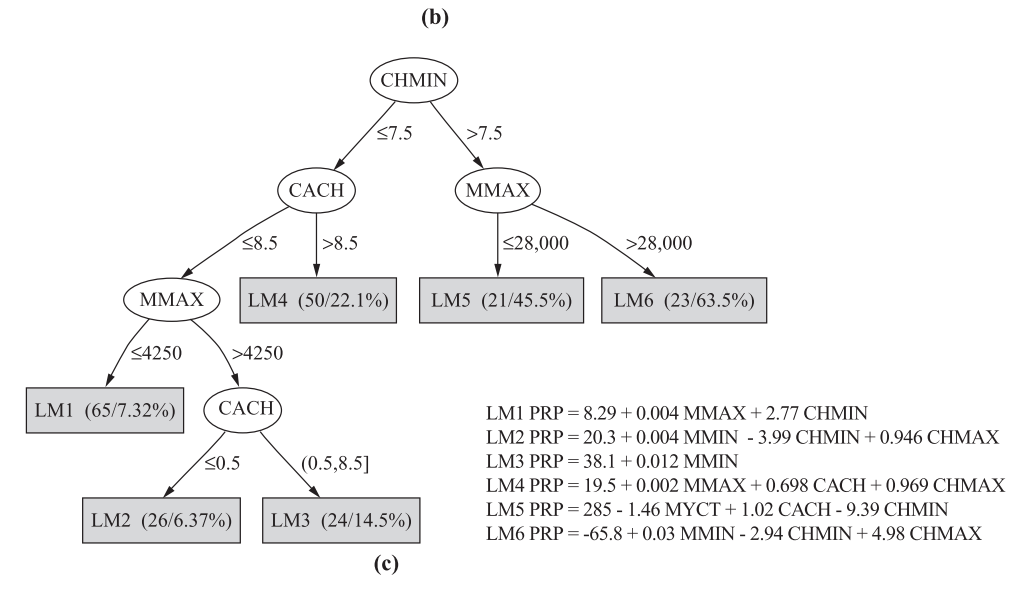
Fonte: elaborada pelo autor

### Modelos lineares

Diferentemente de problemas de classificação, onde o resultado do aprendizado é determinado por um conjunto de respostas o resultado de um algoritmo de predição linear tem como função a soma dos valores e seus pesos conforme Witten et al (p. 15, 2011).

Outra modelo para representação de modelos lineares é através de arvores de decisão semelhante a Figura 2, porém, para estes modelos lineares, os resultados são derivados de uma ou mais funções de regressão, por exemplo, conforme Figura 3, a arvore é expandida a partir das variáveis do modelo e os resultados são funções lineares.

Figura 3 - Exemplo de arvore de decisão com resultados lineares



Fonte: Witten et al (2011)

## Aplicações

Dentre os usos aplicados os algoritmos de *machine learning* Witten et. al (2011, p. 21) descreve a capacidade dos sistemas de aprender a partir das estruturas de tomada de decisões inferidas de um conjunto de dados como uma das características mais importantes desses agentes. O uso desses sistemas já é comum no nosso dia a dia, através da coleta e análise automatizada toda informação que é gerada com ações em um site, por exemplo, pode-se tornar relevante para oferta de novos produtos, propagandas que possuam maior compatibilidade com o perfil do usuário (WITTEN, et. al, 2011, p. 22).

Áreas como a visão computacional se beneficiaram grandemente a exemplo de Witten et al 2011, cientistas da área de meio ambiente buscavam formas de detectar através de imagens de satélites vazamentos de petróleo, esse processo para ser realizado manualmente era dependente de pessoal altamente especializado, a partir do desenvolvimento de um sistema altamente customizado e com validação manual controlando falsos positivos e negativos fazendo uma retroalimentação desse sistema melhorando assim sua capacidade. “Diferente de outras aplicações de aprendizagem de máquina, onde há a implantação de um classificador, nesse tipo de aplicação, a implantação é de sistema de aprendizado” (WITTEN, et. al, 2011, p. 23).

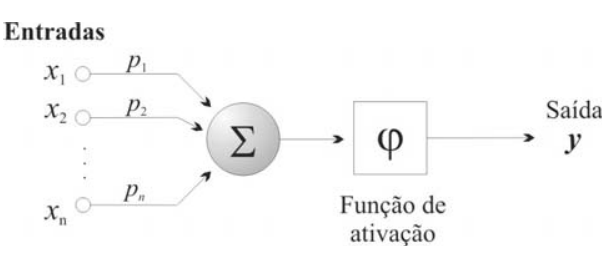
## Redes neurais artificiais

O cérebro humano é formado por bilhões de neurônios. Um neurônio é formado por três seções com funções específicas: corpo, dendritos e axônio de acordo com Ferneda E. (2006).

Como resultado da ação conjunta dos neurônios as informações de entrada são processadas e levam o cérebro a comandar reações físicas. “Um novo conhecimento ou uma nova experiência pode levar a alterações estruturais no cérebro. Tais alterações são efetivadas por meio de um rearranjo das redes de neurônios, reforçando ou inibindo algumas sinapses” (FERNEDA, E. apud HAYKIN, 2001, p. 32-36)

Seguindo as definições uma rede neural, assim como o cérebro humano consiste em diversos neurônios que processam um conjunto de informações a apresentam uma saída. Uma definição matemática de um neurônio é apresentada na Figura 4.

Figura 4 - Modelo matemático de um neurônio

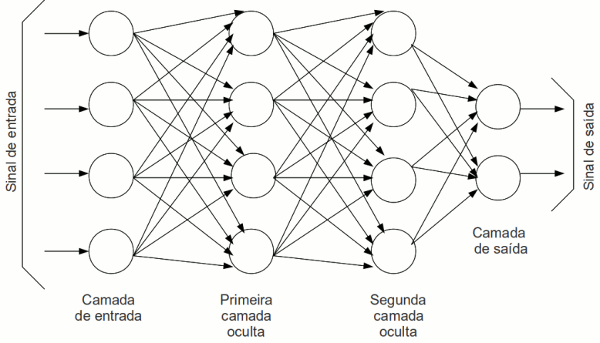


Fonte: (Ferneda, E. 2006)

O Modelo apresentado na Figura 4 trata-se de uma simplificação apresentada por Ferneda, E. (2006) de um modelo definido por Haykin (2001). O modelo consiste em um conjunto de entradas caracterizado por pesos (P1, P2, P3) que são processados por um “somador” que agrupa esses sinais, estes sinais então são processados por uma função que apresenta uma saida *y.*

A combinação desses neurônios matemáticos é o que define uma rede neural artifical. Conforme Ferneda, E. (2006) estas são modelos que buscam simular o processamento de informações do cérebro humano, são agrupamento de neurônios através de conexões sinápticas, essa composição gera uma representação similar ao que pode ser visualizado na Figura 5.

Figura 5 - Representação visual de uma rede neural artificial



Fonte: Monolito Nimbus (2017)

## Pathfinding

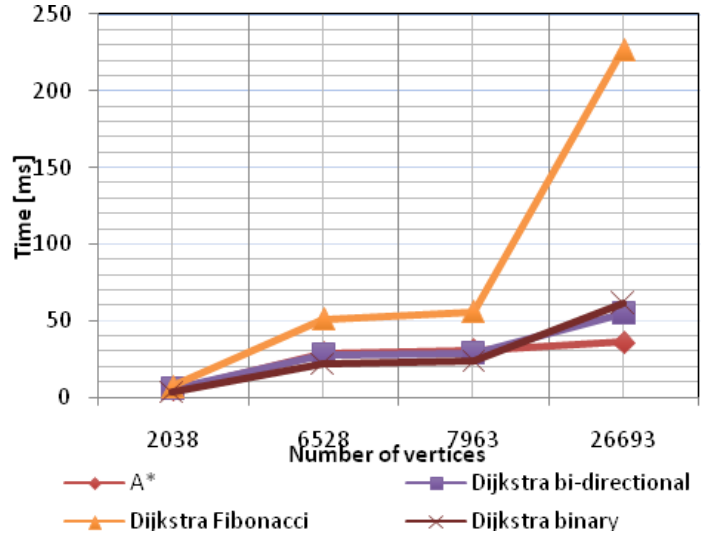
Algoritmos de busca de caminho ou *pathfinding* são essenciais no nosso dia-a-dia por exemplo, através de sistemas de GPS (OSTROWSKI et al., 2015). De acordo com (SCHRIJVER, 2012) é difícil de traçar origem desses algoritmos, mesmo sociedades primitivas e mesmo animais esse tipo de otimização era essencial.

Problemas de busca de caminho foram estudados nos anos de 1950 com o objetivo de busca de caminhos alternativos, buscar uma segunda menor rota caso a primeira estivesse ocupada, isso se aplicaria principalmente para rotas de transporte e até roteamento de chamadas telefônicas (SCHRIJVER, 2012).

Dentre os algoritmos existentes de busca de caminho existem alguns destaques, como o algoritmo de Dijkstra, Bellman-Form, Dijkstra bidirecional e o algoritmo A\*. Conforme (OSTROWSKI et al., 2015) em uma comparação entre estes algoritmos para a determinar o caminho de um sistema de GPS.

A função para definição da menor rota é definida pela soma das intersecções sucessivas na rota. Para definição da rota mais rápida a função é definida pela soma das proporções das distancias entre as interseções e os limites de velocidade nesse caminho entre as intersecções (OSTROWSKI et al., 2015).

Figura 6 - Gráfico de comparação entre algoritmos de *pathfinding*



Fonte: (OSTROWSKI et al., 2015)

Conforme pode ser observado na Figura 7, os algoritmos de Dijkstra em suas variações binárias e bidirecionais, aproximam-se do resultado obtido pelo algoritmo A\*.

# Visão computacional

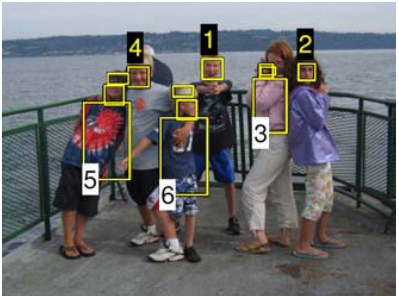
Visão computacional é a área de estudo que trabalha com a identificação, processamento e análise de imagens através de técnicas matemáticas. Conforme Mihahlovic “A área de visão computacional foca em replicar partes da complexidade da visão humana, permitindo computadores processar e identificar imagens e vídeos de forma similar a humana” (2019, tradução nossa).

Graças aos avanços na área de inteligência artificial, *deep learning* e redes neurais trouxeram um grande salto para a área, permitindo exceder a capacidade humana em determinadas tarefas conforme Mihajlovic (2019).

## aplicações

Humanos percebem estruturas tridimensionais do mundo que nos cerca com facilidade, algoritmos por sua vez com os modelos atuais desenvolvidos por pesquisadores da área técnicas para recuperar formas tridimensionais e aparência das imagens de acordo com Szeliski (2011). Na Figura 8 pode-se observar um exemplo da detecção a partir de padrões de cabelos, roupas e rostos.

Figura 7 - Exemplo de detecção de rostos, roupas e cabelos



Fonte: (Sivic, Zitnick, and Szeliski 2006)

A visão computacional hoje tem aplicabilidade em diversas aplicações do mundo real tais como:

* Reconhecimento ótico de caracteres (OCR)
* Inspeção automática
* Varejo
* Construção de modelos 3D
* Imagens médicas
* Segurança automotiva
* Motion Capture
* Vigilância
* Reconhecimento de digitais e biometria

## Reconhecimento de objetos

Entender a forma os sistemas biológicos entendem e reconhecem objetos é um dos maiores objetivos da neurociência computacional. De uma perspectiva computacional reconhecer uma classe de elementos e um elemento em específico tem seu funcionamento de forma similar, até mesmo com um mesmo modelo, porém, variando questões como especificidade e variância de acordo com (RIESENHUBER; POGGIO, 2000).

A maior dificuldade computacional tem a ver com a variabilidade, um sistema de visão precisa generalizar através de muitas variações na aparência, como por exemplo, a perspectiva, iluminação, oclusão e ao mesmo tempo manter uma especificidade. É importante observar que um objeto pode ser reconhecido por uma variedade de níveis de especificidade, por exemplo: um gato pode ser reconhecido como “meu gato” em um nível de individuo ou de forma mais abrangente como um “gato”, “mamífero”, “animal” e outros (RIESENHUBER; POGGIO, 2000, Tradução nossa)

Para mitigar problemas como variações na perspectiva diversas abordagens fazem um alinhamento através de transformações tridimensionais para buscar um alinhamento do modelo com a imagem (KOLLER; LANDSHUT, 1996).

### Objetos em movimento

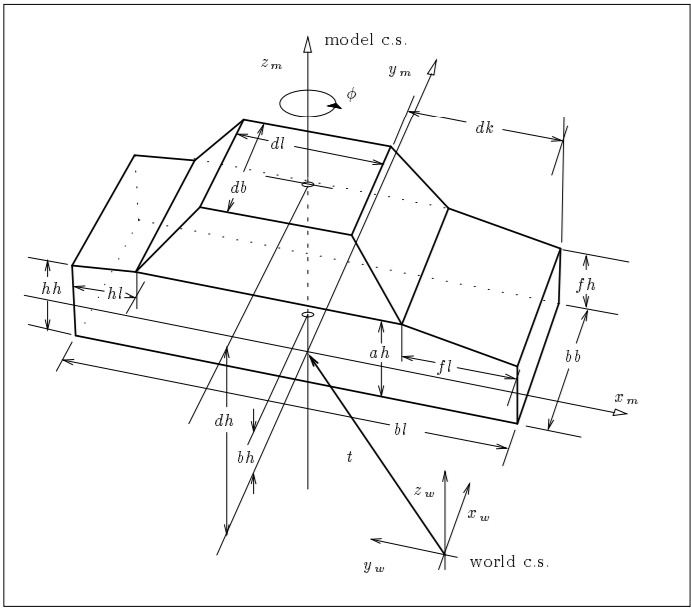
Em seu artigo Koller (1996) define um modelo de movimento no qual é descrita a dinâmica de um veículo em uma estrada. Primeiramente, um modelo paramétrico para definição de um veículo é apresentado na Figura 9. Além disso, outros parâmetros como a posição e ângulo são necessários para essa avaliação. Os parâmetros são representados conforme Figura 10.

Figura 8 - Vetor de parâmetros para definição de um veículo



Fonte: (KOLLER; LANDSHUT, 1996)

Figura 9 - Representação dos parâmetros para definição de um veículo



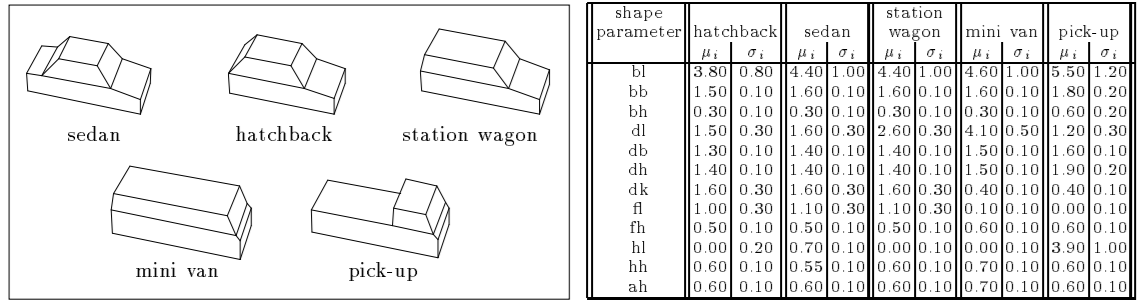
Fonte: (KOLLER; LANDSHUT, 1996)

Para comparação entre imagens e o modelo proposto por Koller, é utilizada uma avaliação das bordas, ou seja, as bordas do modelo 3D são projetadas na imagem 2D, usando algoritmos para eliminação de linhas ocultas. As bordas são induzidas no plano 2D através do uso de uma correlação das distancias do modelo com o uso da distância de Mahalanobis. A fórmula de Mahalanobis trata da correlação entre as distancias com valores de referência, conforme a figura 8 onde há um vetor de distancias de um veículo e através da aplicação é possível comparar os valores lidos com as distâncias de referência para cada tipo de veículo.

#### Processo de classificação

Para a classificação (KOLLER; LANDSHUT, 1996) definiram modelos parametrizados para 5 tipos de veículos, dentre eles *sedan, hatchback, station wagon, minivan, pick-up* confome Figura 11. Esses modelos são selecionados a partir da variação dos parametros de forma observados pela distância de Mahalanobis.

Figura 10 - Modelos parametrizados por Koller



Fonte: (KOLLER; LANDSHUT, 1996)

## Redes neurais convolucionais aplicadas a visão computacional

Com as evoluções trazidas pelo uso de redes neurais convulsionais ao reconhecimento de padrões, através do aprendizado automático de características determinantes através de um *dataset* de treinamento, sendo extremamente eficientes no reconhecimento de imagens (BOJARSKI et al., 2016).

Técnicas do tipo são utilizadas comercialmente há mais de 20 anos, mas apenas nos anos recentes com a evolução dos conjuntos de dados rotulados para treinamento e validação desses algoritmos em conjunto com a implementação desses algoritmos de forma paralela em *GPUs* tornaram essas tarefas mais eficientes e amplamente adotadas (BOJARSKI et al., 2016).

# Navegação autonoma

Os temas abordados nos capítulos 3 e 4 são essenciais para o desenvolvimento de um modelo para navegação autônoma. No contexto deste trabalho, o desenvolvimento se dará através do uso de redes neurais aplicadas para o reconhecimento de objetos e para o controle da direção do veículo. Além disso, há a necessidade de uma busca de caminho através de algoritmos de *pathfinding*.

## Estado atual

Veículos autônomos representam uma grande evolução tecnológica aos problemas de transportes, além da possibilidade de revolucionar totalmente a forma como as pessoas se movimentam pelas cidades (HOWARD; DAI, 2013). Outros motivos para a aplicação desses tipos de veículo envolvem questões como a segurança e seu potencial para redução do número de acidentes (HÄNE et al., 2017).

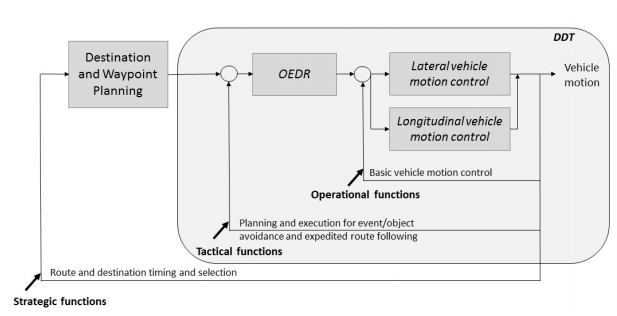
De acordo com o “National Center for Statistics and Analysis” (2018) no ano de 2016 houve mais de 37 mil vítimas fatais em acidentes causados por veículos. A estatística para direção sob efeito de bebidas alcóolicas para o mesmo ano é de uma morte a cada 51 minutos. A percepção popular em uma pesquisa realizada por Howard (2013) com 107 residentes de Berkeley na California as maiores preocupações dos entrevistados dizem respeito ao preço, confiança e ao controle que terão sobre esses sistemas de direção autônoma, porém, dentre as características que mais atraem para esse tipo de veículo é sua segurança e conveniência.

## classificação sistemas de direcão

De acordo com o SAE *(Society of Automotive Engineers)* um veículo pode ser classificado em até 6 níveis de automação, variando do nível 0 onde não há nenhuma forma de automação até o nível 5, no qual o veículo de forma autônoma é capaz de responder a toda gama de situações e cenários. (SAE, 2018). Sendo considerados sistemas com autonomia de direção veículos onde há a implementação entre nível 3 e 5.

A tarefa de direção ou DDT *(dynamic driving task)* de acordo com o SAE (2018), pode ser dividida basicamente em 3 funções, são elas, operacional (controle do veículo), Táticas (planejamento e tomada de decisões e resposta de curto prazo) e estratégica (planejamento de longo prazo, por exemplo: rotas), conforme descrito pela figura X.

Figura 11 - Funções que compõem a DDT



Fonte: SAE (2018).

As classificações de acordo com definição do SAE podem ser encontradas na tabela 2.

Tabela 2 - Classificações dos sistemas de direção

|  |  |
| --- | --- |
| **Nível** | **Descrição** |
| 0 | Nenhuma automação |
| 1 | Assistência de direção (auxílio no controle longitudinal ou lateral) |
| 2 | Sistema básico de controle automático (manutenção em faixa e controle longitudinal) |
| 3 | Funções de direção e detecção de objetos e eventos totalmente automáticas, motorista é necessário como *fallback.* |
| 4 | Evolução do nível 3, onde o sistema de direção consegue lidar com situações adversas sem necessidade do motorista. |
| 5 | Sistema totalmente automatizado, evoluindo do nível 4 com a capacidade de operar em **qualquer** domínio de operação (ODD). |

Fonte: SAE, 2018

Conforme a tabela 2 o mínimo necessário para que a tarefa de direção possa ser considerada autônoma é a partir do momento em que passa a haver a detecção e resposta a objetos e eventos (OEDR). Em níveis superiores, as diferenças ficam no domínio operacional do sistema e não mais necessidade de uma intervenção humana.

Segundo Paden a hierarquia de controle e planejamento de um veículo pode ser dividida em 4 funções, similar ao apresentado por SAE (2018), São elas, o planejamento de rotas, a tomada de decisões baseada em comportamento, planejamento do movimento e finalmente o controle do veículo.

## Simulador CARLA

Publicado inicialmente em 2017 o simulador CARLA tem como objetivo facilitar testes em veículos autônomos.

CARLA é um simulado de código aberto para pesquisa de direção autônoma. O simulador foi desenvolvido desde sua base para permitir o desenvolvimento, treinamento e validação de modelos de direção autônoma urbanos. Em adição a seu Código aberto, o CARLA possui recursos abertos (layouts urbanos, construções, veículos) que foram criados para esse propósito e podem ser utilizados livremente. A plataforma de simulação ainda possui uma gama flexível de sensores e condições de ambiente (DOSOVITSKIY et al., 2017, tradução nossa).

Figura 12 - Demonstração dos recursos de clima dinâmico do simulador CARLA



Fonte: (DOSOVITSKIY et al., 2017b)

Dentre seus recursos, há a inclusão de *layouts* urbanos, múltiplos modelos de veículos, construções, tráfego de pedestres, placas de trânsito, diferentes condições de ambiente conforme Figura 12 e mais, permitindo uso de um grande conjunto de sensores. Essa flexibilidade permite seu uso para treinamento de diversas estratégias de direção, com recursos como coordenadas de GPS, velocidade, aceleração, além de dados detalhados de colisões e demais infrações, os tipos de sensores disponíveis podem ser vistos na tabela 2, 3 e 4.

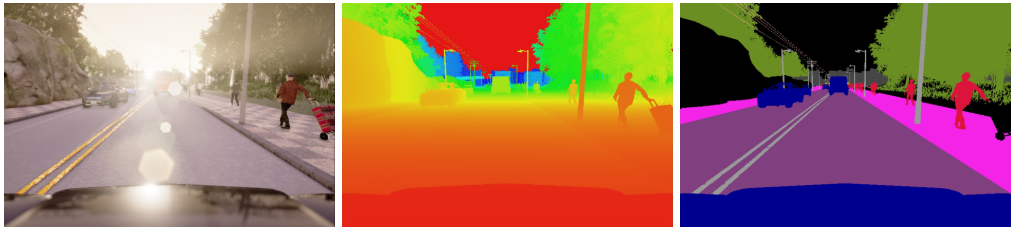
Tabela 3 - Sensores do tipo câmera

|  |  |
| --- | --- |
| *Tipo* | *Descrição* |
| *Depth* | Sensor de profundidade no campo de visão. um mapa de escala de cinza |
| *RGB* | Câmera RGB regular |
| *Semantic segmentation* | Sensor para segmentação de elementos conforme suas “etiquetas” |

Fonte: Documentação CARLA

Os sensores do tipo câmera, funcionam de forma a retornar novos dados a cada novo passo da simulação de acordo com a taxa de quadros do cliente. A partir destes sensores se dará a maior parte da detecção do ambiente. Conforme Figura 13, pode-se observar exemplos de cada um dos sensores descritos na tabela 2.

Figura 13 - Exemplo dos sensores em ação no simulador



Fonte: (DOSOVITSKIY et al., 2017b)

Os sensores descritos na tabela 3 são relacionados a detecção de eventos, como por exemplo, colisão. Seu comportamento de gatilho trata-se justamente do acontecimento da ação específica que eles são projetados para detectar.

Tabela 4 – Tipos de sensores para detecção de eventos

|  |  |
| --- | --- |
| *Tipo* | *Descrição* |
| *Collision* | Notifica eventos de colisão entre o elemento onde está acoplado e outros atores. |
| *Lane invasion* | Notificação de invasão de faixas |
| *Obstacle* | Detecção de obstáculos a frente do ator onde está acoplado |

Fonte: Documentação CARLA

Outros sensores disponíveis conforma a tabela 4 envolvem recursos de geolocalização e orientação através dos sensores de GNSS e IMU, além de sensores para mapeamentos 3D e 2D dos arredores do ator da simulação.

Tabela 5 - Outros tipos de sensores do simulador

|  |  |
| --- | --- |
| *Tipo* | *Descrição* |
| *GNSS* | Sensor para geolocalização |
| *IMU* | Acelerômetro, Giroscópio e Bussola. |
| *LIDAR raycast* | Sensor rotatório que permite a geração de uma visão 3D dos seus arredores |
| *Radar* | Mapeamento 2D da visão e seus elementos em movimento |

Fonte: Documentação CARLA

Conforme Dosovitskiy et al. (2017), um desafio durante o desenvolvimento foi a criação comportamental dos NPC, ao final, o comportamento destes é baseado pelo modelo padrão utilizado pela UE4, com seus parâmetros de física ajustados para maior realismo, adicionando controle de comportamentos conforme regras de trânsito e tomada de decisões em intersecções.

### API ABERTA

Para comunicação entre o mundo da simulação e um modulo de controle há a especificação de uma API trabalhando no modelo cliente-servidor. No qual o servidor é o responsável pela renderização do mundo. O cliente se conecta a este servidor através do uso de sockets para receber os dados dos sensores e enviar comandos para um agente (DOSOVITSKIY et al., 2017b).

A partir da API é possível o controle de aspectos de tráfego, comportamento de pedestre, condições climáticas, sensores a serem utilizados, além, do controle do veículo. Seja a inicialização de veículos para compor o trânsito do mapa onde a simulação ocorrerá, a associação e posicionamento de sensores no ator principal da simulação.

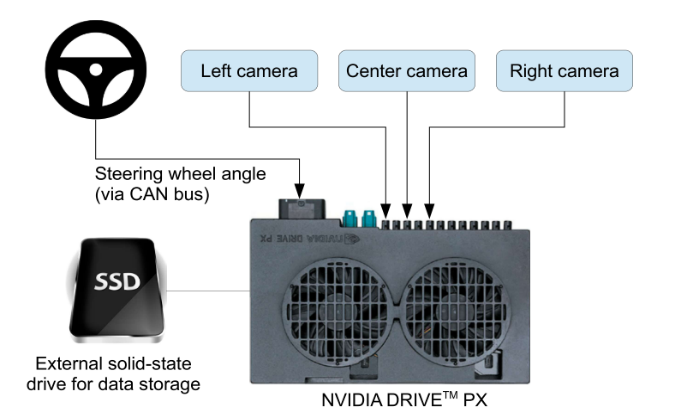
## Estudos na área

Por tratar-se de uma área em grande evolução e relevância nos últimos anos, diversos estudos são conduzidos com objetivo de avaliar e desenvolver modelos de direção autônoma. Por exemplo, Bojarski et al. (2016), Dosovitskiy et al. (2017).

De acordo com Bojarski et al. (2016) o estudo foi baseado no uso de CNNs e apenas 3 câmeras frontais, conforme Figura 14. Diferente de outros modelos de direção autônoma, a proposta é o aprendizado completo de todo o processo, tendo como base no projeto DAVE desenvolvido pela *Defense Advanced Research Projects Agency (DARPA).*

A motivação primária para este trabalho é evitar a necessidade do reconhecimento de características especificamente pensadas para humanos, como por exemplo: marcações de pistas, *guard rails* ou outros veículos*,* evitando criar uma coleção de regras de “*if, then, else”* baseados na observação dessas características. (BOJARSKI et al., 2016, tradução nossa)

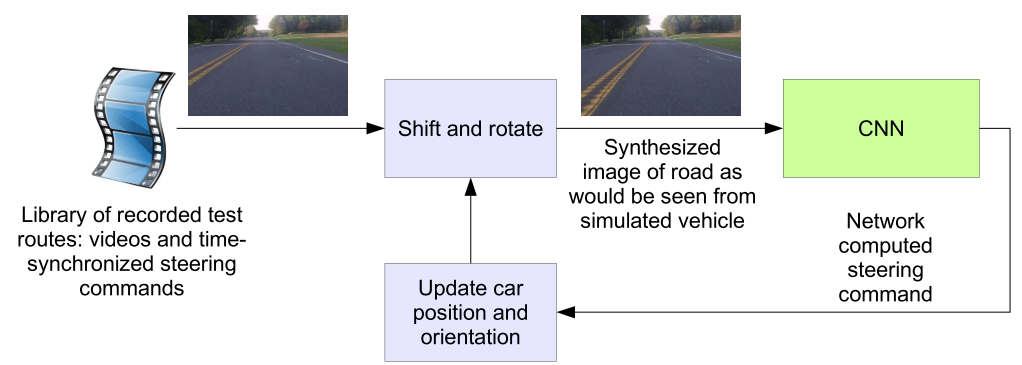
Figura 14 - Visualização do sistema utilizado por Bojarski et al.



Fonte: (BOJARSKI et al., 2016)

Para treinamento e avaliação antes da realização de testes em estradas reais, foram realizados testes utlizando um simulador que a partir de gravações de uma câmera frontal gera imagens de forma a simular a direção. O simulador adapta o próximo quadro para corresponder ao esperado conforme um movimento realizado pela rede neural, esse novo quadro alimenta a nova avaliação dessa rede (BOJARSKI et al., 2016), conforme Figura 15.

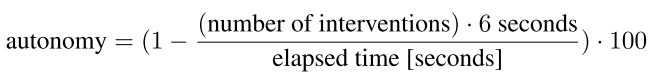
Figura 15 - Diagrama da simulação



Fonte: (BOJARSKI et al., 2016)

A avaliação do sistema proposta por (BOJARSKI et al., 2016) se dá a partir de uma fórmula onde é avaliado o percentual de autonomia do modelo proposto. Baseando-se na métrica do número de intervenções humanas necessárias, estes ajustes manuais ocorrem quando o veículo sai da faixa, assumindo que esse tipo de intervenção leva aproximadamente 6 segundos, conforme Figura 16. “Para uma viagem típica no Condado de Monmouth, NJ, do nosso escritório em Holmdel até Atlantic Highlands, a autonomia alcançada foi de 98%” (BOJARSKI et al., 2016).

Figura 16 - Fórmula para cálculo da métrica de autonomia



Fonte: (BOJARSKI et al., 2016)

Para validação do simulador CARLA no trabalho publicado por Dosovitskiy et al. (2017), foram avaliados três modelos de direção autonoma, dentre eles: um *pipeline* baseado nos subsistemas de percepção visual, planejamento e controle, uma rede neural treinada a partir de imitação, e o terceiro, trata-se de uma rede neural treinada através de aprendizado reforçado (DOSOVITSKIY et al., 2017b).

No modelo utilizando o *pipeline* modular, conforme Dosovitskiy et al. (2017) é utilizada uma segmentação semantica para estimar as faixas, limites da estrada, objetos dinâmicos e outros perigos e um sistema de classificação determina a proximidade das intersecções. Para o planejamento é utilizada uma máquina de estado baseada em regras, implementando politicas voltadas a ambientes urbanos, e o sistema de controle é executado por um controlador PID que atua como a direção, acelelador e freio .

Já o modelo de direção imitada, faz uso de dados obtidos por “motoristas” humanos na cidade de teste do simulador, ao total foram utilizadas 14 horas de gravações e a rede gerada fez uso do otimizador Adam.

O sistema utilizando aprendizado reforçado foram realizados 10 milhões de passos na simulação utilizando nos agentes uma algoritmo chamado *asynchronous advantage actor-critic (A3C)*. Os resultados obtidos por Dosovitskiy et al. (2017) são apresentados conforme a Figura 17 os modelos estão representados pelas seguintes siglas: MP: *Pipeline* modular, IL: Aprendizado imitado e RL: aprendizado reforçado. Os dados da Figura 17 representam a distância percorrida em kilometros por cada modelo até a ocorrência de duas infrações.

Figura 17 - Resultados obtidos por Dosovitskiy et al.



Fonte: (DOSOVITSKIY et al., 2017)

# Desenvolvimento do agente de navegação

Utilizando-se do conceito cliente-servidor no qual o simulador CARLA é construído, é possível a partir de scripts na linguagem Python realizar a conexão a um servidor de simulação e enviar comando a serem executados em um ou mais agentes da simulação.

Para o desenvolvimento do agente é utilizado como base para posicionamento e localização dos sensores necessários para a tarefa de percepção um veículo Model 3 da Tesla disponibilizado junto ao simulador.

Os estágios desse agente são baseados nas características estabelecidas pela SAE, 2018 no qual um sistema autônomo é composto por um planejamento de longo termo (escolha de rotas), um planejamento de curto termo (OEDR) onde é realizada a percepção do ambiente e arredores do veículo e por fim, o controle longitudinal (controle de aceleração e freio) e controle lateral, definindo trocas de faixa e outras ações laterais a serem tomadas.

## Planejamento de rotas

O sistema de planejamento de rotas utilizado, faz uso de uma base de scripts já desenvolvidos e disponibilizados juntamente com o simulador em questão. O sistema de mapas utilizado é gerado a partir do sistema OpenDRIVE e todas as faixas e estradas são tratadas em um conceito de ponto de referência ou *waypoint*.

Esse processo de seleção do caminho se dá através da construção de um dígrafo que onde cada vértice contém a posição do *waypoint* no ambiente da simulação.

A construção do dígrafo que irá determinar a rota é realizada com a biblioteca *networkx* disponível para Python no qual é possível reproduzir estruturas de grafos complexas, além de incluir funções para navegação e algoritmos determinísticos de caminho dentro destes.

O sistema de faixas ou *waypoints* no simulador é composto por vias dos seguintes tipos: Direção, Parada, Ciclovia, Calçada, Bidirecional, Estacionamento entre outros. Baseado nestes dados disponíveis para também incluem facilitadores para verificar as marcações das mesmas e até mesmo obter informações das faixas a direita ou esquerda e verificar a possibilidade de uma troca de faixas.

Para compreender melhor o funcionamento desse sistema é necessário a apresentação do padrão OpenDRIVE definido pela ASAM (*Association for Standardization of Automation and Measuring Systems).* Atualmente em sua versão 1.6, ele define um padrão de descrição de uma rede de estradas estáticas para aplicações de simulação de direção. A proposta inclui também a definição de outros objetos ao longo da estrada, como placas de trânsito e semáforos.

Figura 18 - Mapa utilizado na simulação



Fonte: CARLA

A figura 18 trata de uma representação do mapa 3 do simulador, sendo um dos mais complexos disponíveis, incluindo rotatória e ruas sem saída.

## Controle do estado do agente

Durante a execução do agente em um ambiente com múltiplos veículos e pedestres, o agente de navegação autônoma deve possuir um controle de estado para determinar se em dado momento está bloqueado por outro agente no trânsito, bloqueado por um semáforo no vermelho ou realizando seu percurso regularmente.

A determinação desse estado ocorre através da análise dos dados coletados pelos sensores acoplados ao agente. Essa função é realizada por um modulo de percepção, apresentado no capítulo 7.

## Controle longitudinal

O controle longitudinal é responsável pelo controle da velocidade do agente, parte dos veículos hoje em dia, já possuem alguma forma de controle longitudinal, normalmente chamado de *cruise control*, esse sistema funciona de forma a estabilizar uma velocidade pré-definida.

No contexto do desenvolvimento do agente, essa velocidade será a velocidade máxima da via, obtida através dos dados do mapa do simulador. conforme (Bojarski et al., 2016) o sistema de controle será baseado em um controlador PID.

O sistema PID ou Proporcional Integral Derivativo trata-se de uma técnica baseada nos valores de erro para aplicação de uma ação proporcional, esse controle se dá conforme fórmula.



Com os seguintes termos:

**Kp = Termo proporcional:** *Define um ganho proporcional em cima da observação do erro para produção da resposta.*

**Kd = Termo derivativo:** *Determina proporcionalmente a taxa de variação da variável de controle com relação ao tempo, valores maiores levam a mudanças mais bruscas.*

**Ki = Termo integral:** *Soma de n obersvações do erro para impedir um estado estacionário e corrigir erros de pequenas variações*

**t = tempo**

O controle dos erros, é feito atraves da estrutura *deque* no Python, que consiste de uma estrutura de dados similar a uma fila, porem, com duas pontas, operando de forma similar a uma lista, um limite de itens e funcionamento mais eficiente com custo O(n) para inserções em qualquer ponto.

Separando a formula para a aplicação no código ela é composta por três partes principais:

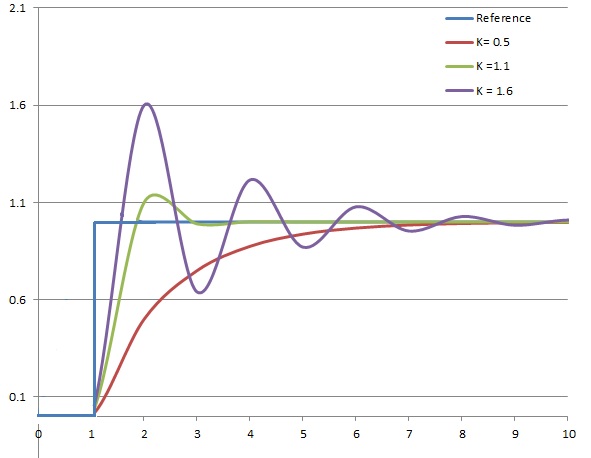
Primeiramente, o cálculo do erro no momento, para o controle longitudinal. Trata-se da diferença entre a velocidade objetivo e a velocidade atual, então é aplicada a multiplicação deste erro ao termo proporcional.

Então, é calculado a diferença entre os últimos dois erros que ocorreram e esse delta é dividido pelo valor de tempo, resultando no valor da derivada que será multiplicada pelo termo derivativo.

Para o termo Integral, é realizada a soma de todos os deltas de erro encontrados na fila, que contem as ultimas 10 observações de erro, sendo multiplicado pelo *t* e então, é mutiplicado pelo termo integral.

Realizando a soma de cada um dos termos, o resultado é uma ação proporcional para a compensação dos erros, esse comportamento pode ser observado conforme a figura 19.

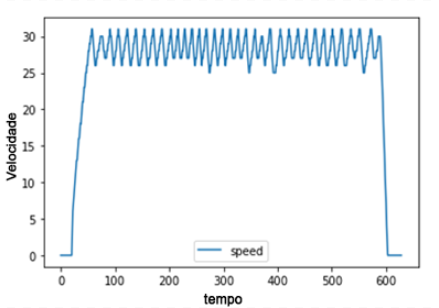
Figura 19 - Convergência de um controlador PID



Como pode ser observado na figura 19, conforme a variação do tempo, há uma compensação dos termos para alcançar o valor de referência, variando os valores e balanceando através do erro para encontrar um ponto de equilíbrio.

Antes de aplicar diretamente o valor obtido ao controle do veículo, é preciso adaptar esse valor a realidade do simulador, no caso, os valores de referência para aceleração e freio vão de -1 a +1, para evitar qualquer valor inferior ou superior aos esperados pelo simulador é realizado a operação de *clip* onde os valores acima ou abaixo do esperado pelo simulador são truncados pelos limites para manter as saídas sob controle.

Figura 20 - Variação da velocidade do agente com o tempo



Como pode ser observado na figura 20, o controlador PID trata de manter a velocidade conforme o parâmetro da velocidade máxima, a variação observada deve-se ao fato da compensação de erro do controlador.

## Controle lateral

Similar ao controle longitudinal também foi empregado o uso de um controlador PID, porém, no caso do controle lateral, há um número maior de variáveis a serem observadas para a determinação do comando a ser realizado. É necessário a observação da rota traçada na etapa de planejamento de rotas.

É necessário saber qual o próximo *waypoint* para encontrar a angulação necessária para alcançar o objetivo. Esse controle é realizado por um planejador local, que está diretamente ligado a tarefa de percepção e controle de curto prazo e será detalhado em capítulo posterior.

Essa tarefa de controle lateral é realizada através da diferença entre o estado de posição e rotação atual e o necessário para a via de destino. Para esse processo, a posição do agente é acrescida dos cossenos da sua angulação para obter assim uma posição final viável. Considerando que tratam-se de coordenadas, a estrutura dos dados é composta por uma tupla com 3 variáveis (x,y,z).

Sendo o objetivo encontrar o ângulo para a transição entre faixas ou acessos, não é necessário o uso do eixo Z, já que este é relacionado à altura. Para avaliarmos o valor de erro com o qual o controlador PID será alimentado são calculados duas tuplas com os deltas para cada variável em relação a posição inicial.

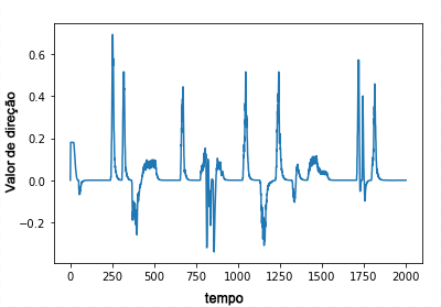
O valor do erro é encontrado através do cosseno da divisão entre o produto dos deltas do veículo e do *waypoint* pela distância entre os pontos multiplicada.

Figura 21 - Fórmula para calcular o erro do controlador PID lateral



Conforma a fórmula apresentada na figura 21, a função *d* refere-se a distância euclidiana entre os pontos. Ainda é necessário encontrar qual o sinal do erro, para que o controlador consiga compensar o movimento. Essa inversão de sinal ocorre, caso o segundo termo da multiplicação das tuplas seja negativo.

Figura 22 - Gráfico do valor de *steer* a cada frame



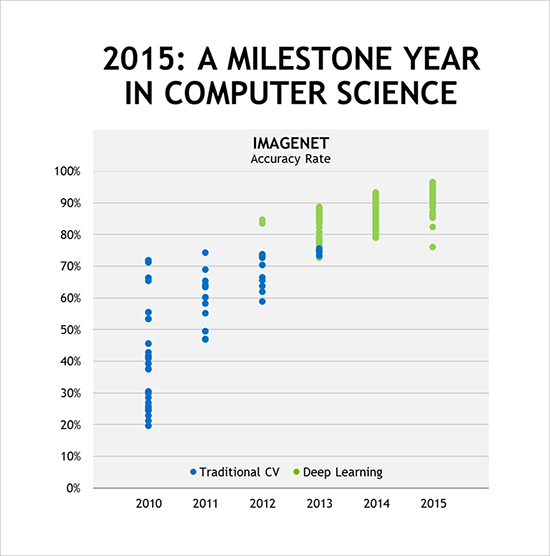
Fonte: Autor

# detecção de objetos e eventos

De acordo com o SAE (2018) uma das partes mais importantes da autonomia de um sistema autônomo trata-se da sua capacidade de detectar objetos e eventos acontecendo em tempo real e responder a estes. Partindo desse princípio há a necessidade de desenvolvimento de algum modelo capaz de reconhecer estes padrões a partir de um fluxo de imagens recebidos pelos sensores disponíveis no veículo.

Dentre as variáveis que tornaram essa tarefa possível estão os avanços que o uso do aprendizado de máquina e o *deep learning* trouxeram para a área de processamento digital de imagens. Com o uso de placas de vídeo para aceleração das operações que compõem esses cenários.

Figura 23 - Avanços na precisão através do uso de *Deep Learning*

  
Fonte: Nvidia, 2015

De acordo com a Nvidia (2015) os avanços com o uso de GPUs para aceleração de tarefas de treinamento em *deep learning* permitindo uma redução de 10 a 20 vezes no tempo de treinamento considerando sua grande capacidade de processamento paralelo, posteriormente, com otimizações na arquitetura das placas e algoritmos o ganho chega a 50 vezes e com expectativa de ganhos maiores. Também conforme figura 23 o uso dessas tecnologias permitiu um grande avanço na precisão no processamento de imagens.

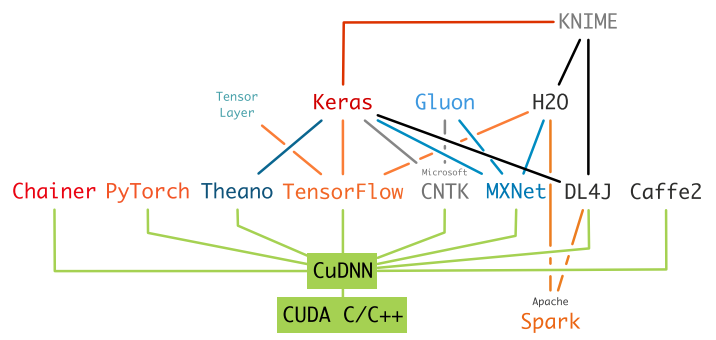
## cuda para aceleração de deep learning

Lançado em 2007 pela Nvidia o sistema CUDA trata-se de uma modelo de programação paralelo fazendo o uso dos núcleos disponíveis nas placas de vídeo da marca, e considerando que elas possuem até milhares de núcleos, por exemplo: Um dos modelos lançados nesse ano possui mais de 10 mil núcleos CUDA trabalhando em paralelo.

Esse modelo, porém, não está limitado a aplicações de aprendizagem de máquina. Qualquer aplicação que possa fazer proveito do uso de tal capacidade de paralelismo pode ser implementada em CUDA C.

Para o uso em aprendizado de máquina e *deep learning* diversos frameworks com esse propósito tem em sua implementação o uso dessa aceleração por hardware. Conforme figura 24, pode-se observar que os principais frameworks de ML fazem o uso desse recurso.

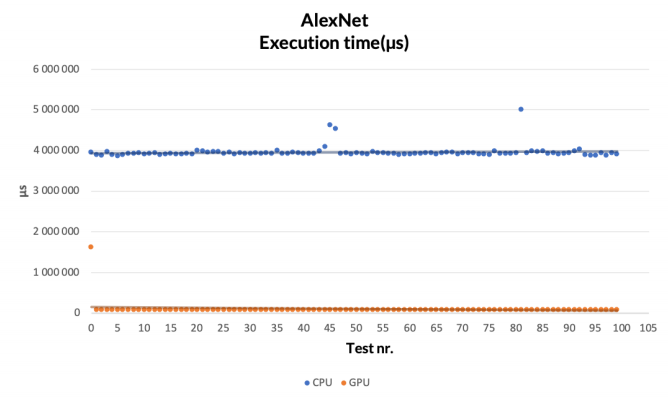
Figura 24 - Frameworks e abstrações dos mesmos que fazem uso de CUDA



Fonte: NGUYEN, G. et al., 2019

Em uma comparação direta entre o desempenho entre GPU e CPU com o uso do framework Tensorflow conforme (LIND, E. et al., 2018) em testes comparativos utilizando *AlexNet –* uma rede neural convolucional projetada por Alex Krizhevsky que competiu no desafio ImageNet para reconhecimento de imagen.Os resultados favorecem muito o paralelismo disponível nas GPUs. Conforme figura 25, avaliando o tempo de execução pode-se observar ganho próximo ao estimado pela Nvidia, aproximando-se das 50 vezes, estes resultados

Figura 25 - Comparação de desempenho Tensorflow GPU x CPU



Fonte: LIND, E. et al., 2018

## detecão de objetos com tensorflow

Conforme proposta de (HUANG, J. et al, 2017) visando alcançar um equilíbrio entre velocidade e acurácia na tarefa de detecção de objetos seria necessária uma forma de comparação equivalente entre múltiplos sistemas, dentre estes: *Faster R-CNN, R-FCN e SSD*.

Para reproduzir e comparar esses diversos sistemas foi proposta uma meta-arquitetura implementando de forma unificada os três modelos apresentados anteriormente. Sendo possível testar de forma fácil diferentes extratores de características e compará-los entre cada sistema.

O sistema de meta-arquitetura proposta por (HUANG, J. et al, 2017) alcança performance de estado da arte na tarefa de detecção de objetos, avaliada sobre o *dataset* COCO, este, trata-se de um conjunto de larga escala composto por milhares de imagens com objetos classificados.

Essa meta-arquitetura que permite a combinação de extratores de características e redes neurais para detecção é disponibilizado como “*TensorFlow Object Detection API”.* Sendo totalmente configurável através de um arquivo “.config” descrevendo parâmetros da rede neural, número de classes do resultado e caminhos dos arquivos de treinamento e de avaliação.

Figura 26 - Detecção de objetos com o *Object Detection API*



Fonte: Tensorflow Object Detection API

Como retorno de um modelo após sua execução o retorno é composto das classificações de cada item do qual a rede neural foi capaz de identificar. A figura 26 trata-se de um exemplo de detecção treinado com o dataset COCO.

### Treinando um modelo

A API disponibilizada é composta por scripts para conversão de um *dataset* para formato esperado *tfrecord*. contendo as imagens e etiquetas para os objetos que compõem a cena.

Primariamente, antes de iniciar um treinamento é necessária a construção ou utilização de um *dataset* pronto. Após a criação dos arquivos *tfrecord* é necessário a definição de um arquivo de configuração. Diversos exemplos são disponibilizados juntamente com a API, estes arquivos são referentes a cada um dos modelos pré-treinados disponibilizados conforme tabela 6. Cada um dos modelos da tabela possui uma precisão média em função da área ou *mean Average Precision* e qual sua velocidade em milissegundos.

Tabela 6 - Modelos pré-treinados disponíveis

|  |  |  |
| --- | --- | --- |
| ***Nome do modelo*** | ***Velocidade (ms)*** | ***COCO mAP[^1]*** |
| [*ssd\_mobilenet\_v1\_coco*](http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v1_coco_2018_01_28.tar.gz) | 30 | 21 |
| [*ssd\_mobilenet\_v1\_0.75\_depth\_coco ☆*](http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v1_0.75_depth_300x300_coco14_sync_2018_07_03.tar.gz) | 26 | 18 |
| [*ssd\_mobilenet\_v1\_quantized\_coco ☆*](http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v1_quantized_300x300_coco14_sync_2018_07_18.tar.gz) | 29 | 18 |
| [*ssd\_mobilenet\_v1\_0.75\_depth\_quantized\_coco ☆*](http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v1_0.75_depth_quantized_300x300_coco14_sync_2018_07_18.tar.gz) | 29 | 16 |
| [*ssd\_mobilenet\_v1\_ppn\_coco ☆*](http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v1_ppn_shared_box_predictor_300x300_coco14_sync_2018_07_03.tar.gz) | 26 | 20 |
| [*ssd\_mobilenet\_v1\_fpn\_coco ☆*](http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v1_fpn_shared_box_predictor_640x640_coco14_sync_2018_07_03.tar.gz) | 56 | 32 |
| [*ssd\_resnet\_50\_fpn\_coco ☆*](http://download.tensorflow.org/models/object_detection/ssd_resnet50_v1_fpn_shared_box_predictor_640x640_coco14_sync_2018_07_03.tar.gz) | 76 | 35 |
| [*ssd\_mobilenet\_v2\_coco*](http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v2_coco_2018_03_29.tar.gz) | 31 | 22 |
| [*ssd\_mobilenet\_v2\_quantized\_coco*](http://download.tensorflow.org/models/object_detection/ssd_mobilenet_v2_quantized_300x300_coco_2019_01_03.tar.gz) | 29 | 22 |
| [*ssdlite\_mobilenet\_v2\_coco*](http://download.tensorflow.org/models/object_detection/ssdlite_mobilenet_v2_coco_2018_05_09.tar.gz) | 27 | 22 |
| [*ssd\_inception\_v2\_coco*](http://download.tensorflow.org/models/object_detection/ssd_inception_v2_coco_2018_01_28.tar.gz) | 42 | 24 |
| [*faster\_rcnn\_inception\_v2\_coco*](http://download.tensorflow.org/models/object_detection/faster_rcnn_inception_v2_coco_2018_01_28.tar.gz) | 58 | 28 |
| [*faster\_rcnn\_resnet50\_coco*](http://download.tensorflow.org/models/object_detection/faster_rcnn_resnet50_coco_2018_01_28.tar.gz) | 89 | 30 |
| [*faster\_rcnn\_resnet50\_lowproposals\_coco*](http://download.tensorflow.org/models/object_detection/faster_rcnn_resnet50_lowproposals_coco_2018_01_28.tar.gz) | 64 |  |
| [*rfcn\_resnet101\_coco*](http://download.tensorflow.org/models/object_detection/rfcn_resnet101_coco_2018_01_28.tar.gz) | 92 | 30 |
| [*faster\_rcnn\_resnet101\_coco*](http://download.tensorflow.org/models/object_detection/faster_rcnn_resnet101_coco_2018_01_28.tar.gz) | 106 | 32 |
| [*faster\_rcnn\_resnet101\_lowproposals\_coco*](http://download.tensorflow.org/models/object_detection/faster_rcnn_resnet101_lowproposals_coco_2018_01_28.tar.gz) | 82 |  |
| [*faster\_rcnn\_inception\_resnet\_v2\_atrous\_coco*](http://download.tensorflow.org/models/object_detection/faster_rcnn_inception_resnet_v2_atrous_coco_2018_01_28.tar.gz) | 620 | 37 |
| [*faster\_rcnn\_inception\_resnet\_v2\_atrous\_lowproposals\_coco*](http://download.tensorflow.org/models/object_detection/faster_rcnn_inception_resnet_v2_atrous_lowproposals_coco_2018_01_28.tar.gz) | 241 |  |
| [*faster\_rcnn\_nas*](http://download.tensorflow.org/models/object_detection/faster_rcnn_nas_coco_2018_01_28.tar.gz) | 1833 | 43 |
| [*faster\_rcnn\_nas\_lowproposals\_coco*](http://download.tensorflow.org/models/object_detection/faster_rcnn_nas_lowproposals_coco_2018_01_28.tar.gz) | 540 |  |

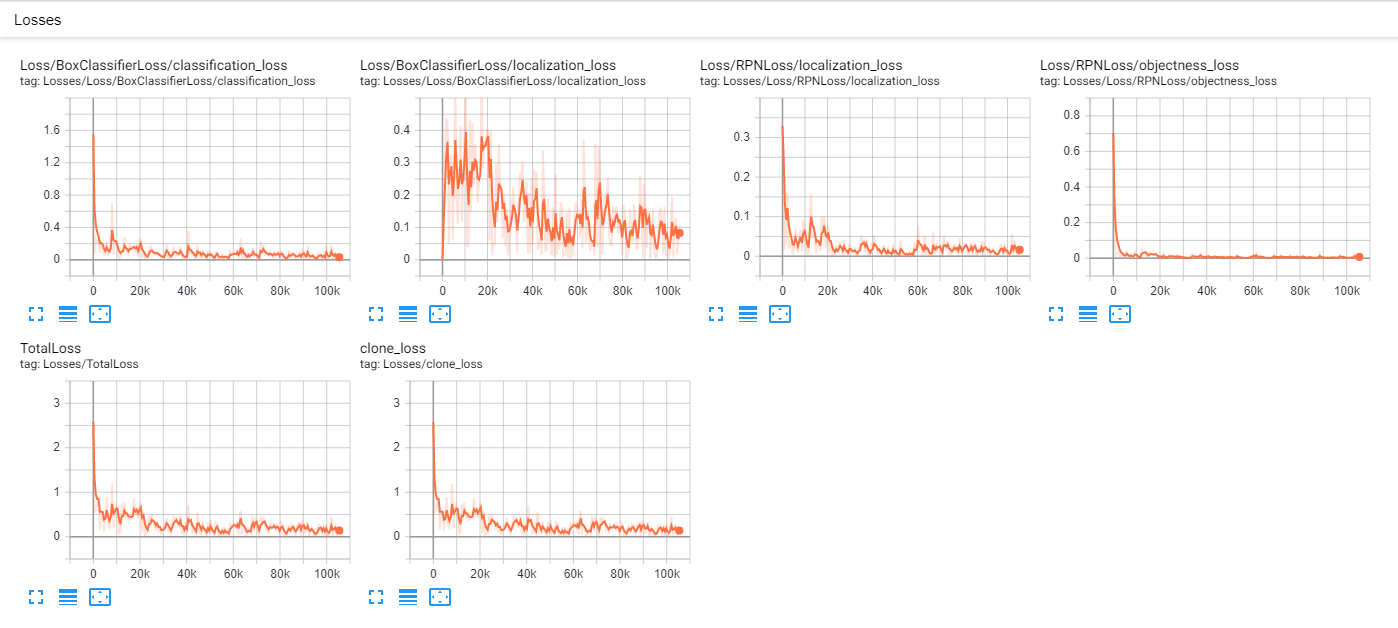
Fonte: TensorFlow Object Detection API

O treinamento a partir da API de detecção de objetos utiliza um destes modelos pré-treinados como base. Com o arquivo de configuração pronto, basta iniciar o treinamento utilizando o script *train.py* com os parâmetros apontando para a configuração criada.

> train.py --logtostderr --train\_dir=training/ --pipeline\_config\_path=training/faster\_rcnn\_inception\_v2\_coco.config

Para acompanhar o processo do treinamento é possível o uso de uma ferramenta chamada ***TensorBoard*** que demonstra de forma gráfica diversos indicadores do estado atual do modelo sendo treinado. Conforme figura 27.

Figura 27 - TensorBoard para acompanhamento do modelo



## Reconhecimento de objetos aplicado ao carla

Conforme descrito no capítulo 7.2 para realizar o treinamento de um modelo para detecção de objetos é necessária a construção de um *dataset* contendo imagens e a categorização dos objetos presentes. Sendo então necessária a coleta de um número considerável de exemplos que serão ensinados a uma rede neural.

Considerando a necessidade de maior desempenho no reconhecimento de objetos no cenário da direção autônoma é necessário o uso de um modelo que alcance um equilíbrio entre performance e acurácia. Dentre as opções de modelos pré-treinados algumas características principais a serem consideradas:

1. Tempo de execução em ms
2. Precisão em mAP

Analisando a tabela 6 dentre os modelos apresentados os modelos que apresentam um melhor equilíbrio entre velocidade e precisão trata-se dos modelos utilizando a *Faster RCNN.* Os modelos do tipo “ssd” apesar da grande velocidade, possuem valores de precisão abaixo dos esperados para a aplicação.

Considerando apenas os modelos do FasterRCNN pode-se aproximar qual a taxa de quadros na simulação irá ser executada. A partir da tabela 17 o modelo que apresenta o melhor desempenho no quesito velocidade é o *faster\_rcnn\_inception\_v2\_coco* e foi o modelo escolhido para o treinamento do novo modelo.

Tabela 7 - Quadros por segundo máximo esperado para cada modelo

|  |  |  |
| --- | --- | --- |
| Nome do modelo | Velocidade (ms) | QPS |
| [*faster\_rcnn\_inception\_v2\_coco*](http://download.tensorflow.org/models/object_detection/faster_rcnn_inception_v2_coco_2018_01_28.tar.gz) | 58 | 17,24137931 |
| [*faster\_rcnn\_resnet50\_coco*](http://download.tensorflow.org/models/object_detection/faster_rcnn_resnet50_coco_2018_01_28.tar.gz) | 89 | 11,23595506 |
| [*faster\_rcnn\_resnet50\_lowproposals\_coco*](http://download.tensorflow.org/models/object_detection/faster_rcnn_resnet50_lowproposals_coco_2018_01_28.tar.gz) | 64 | 15,625 |
| [*faster\_rcnn\_resnet101\_coco*](http://download.tensorflow.org/models/object_detection/faster_rcnn_resnet101_coco_2018_01_28.tar.gz) | 106 | 9,433962264 |
| [*faster\_rcnn\_resnet101\_lowproposals\_coco*](http://download.tensorflow.org/models/object_detection/faster_rcnn_resnet101_lowproposals_coco_2018_01_28.tar.gz) | 82 | 12,19512195 |

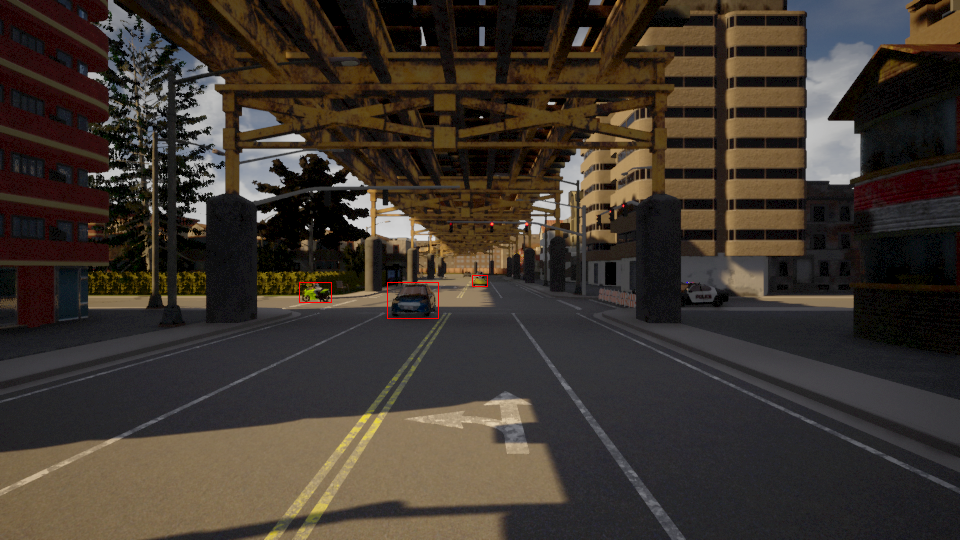
Fonte: TensorFlow Object Detection API

### Obtendo imagens para o *dataset*

A coleta de imagens para a construção do *dataset* foi realizada pelo aluno com o auxílio da ferramenta Carla-2DBBox desenvolvida por (ADIB, M., 2020). Através da combinação dos dados de sensores de profundidade, RGB e API do Carla os algoritmos conseguem determinar as caixas de anotação para os veículos no ambiente da simulação.

Os resultados dessa coleta são dispostos em duas pastas, uma contendo as imagens RGB conforme figura 28 e outra contendo as anotações referentes a cada um dos frames obtidos. Essa anotação encontra-se no formato JSON contendo as posições das caixas de cada elemento e a classe na qual o objeto pertence. Essa anotação automática faz diferenciação entre 4 classes: *passenger\_car, truck, motorcycle, bicycle.*

Figura 28 - Coleta de imagens com o CARLA-2DBBox



Fonte: ADIB, M. 2020

Ao total o *dataset* coletado é composto por 3193 imagens sendo elas divididas entre 2 grupos: Treinamento contendo 2804 imagens com um total de 4688 objetos anotados nestas imagens. O conjunto de teste é contém 389 imagens com 710 objetos anotados. Os números finais do *dataset* são após as etapas manuais que são explicadas na seção 7.3.3, onde são revisados os dados coletados e a anotação manual dos pedestres.

### Pré-processamento e classificação manual

Os dados coletados com o uso do algoritmo descrito no capítulo 7.3.1 possuem um padrão de anotação diferente do necessário para o uso de uma ferramenta de anotações manual na qual o conjunto deve ser revisado e ajustado manualmente.

Sendo o padrão para anotações de objetos em imagens o formato PascalVOC que se trata de uma proposta de padronização de anotações para *datasets.* Este formato é definido por um arquivo xml para cada imagem contendo informações sobre a imagem, e nós para cada um dos objetos na imagem, um exemplo da estrutura do formato pode ser observado conforme figura 29.

Figura 29 - Modelo de anotação PascalVOC

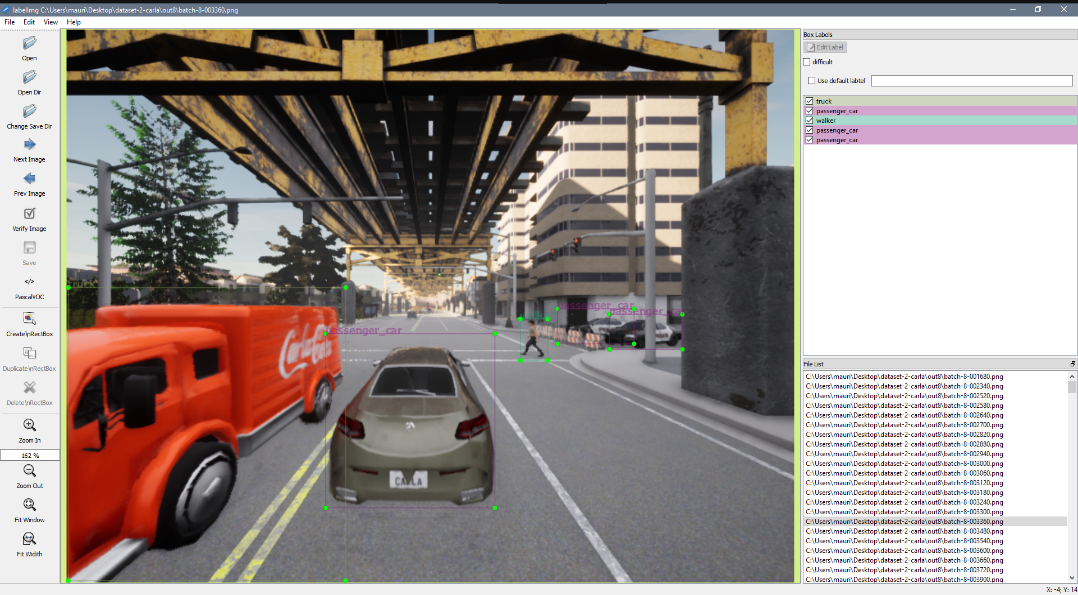


Fonte: KHANDELWAL RENU KHANDELWA, R. 2019

O pré-processamento e conversão dos arquivos foi realizado na linguagem Python com o auxílio da biblioteca *pascal-voc-writter* que permite fácil adição dos objetos lidos no formato de saída do algoritmo Carla-2DBBox. A figura 30 apresenta a interface gráfica do software utilizado para a revisão manual das imagens do *dataset* com um exemplo de anotação realizada.

Após a conversão entre os formatos, os arquivos podem ser abertos na ferramenta *“labelimg”*, uma aplicação gráfica para anotações desenvolvida por Tzutalin e disponível no github tendo sua primeira versão em 2015.

Figura 30 - Interface da ferramenta labelimg



Fonte: Autor

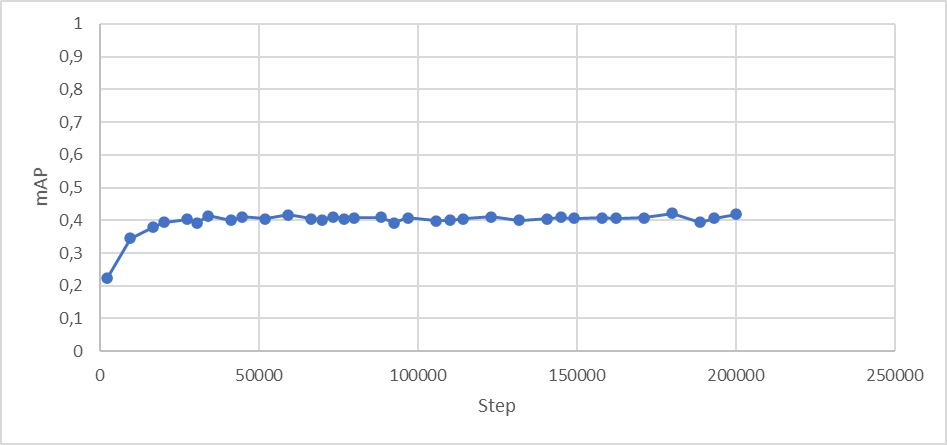
O processamento manual envolve a avaliação individual de cada imagem do conjunto, ajuste das caixas de anotações e adicão dos pedestres. Também foram adicionadas anotacões para semaforos, porém, conforme é discutido nos resultado do modelo e considerando a resolução das imagens no qual o modelo foi treinado a adição dos semáforos se mostrou responsável por uma grande perda na precisão do algoritmo.   
 O *dataset* está disponível em: <https://www.kaggle.com/mauriciobonetti/carla-0994-object-detection>.

### Resultados do modelo

Com o *dataset* descrito anteriormente o treino da rede neural foi iniciado utilizando como modelo de base o *faster\_rcnn\_inception\_v2\_coco.* Ao total foram realizados 200.000 passos de treinocom tempo total de 8 horas e 15 minutos em um computador as seguintes especificações de processador e placa de vídeo: *Ryzen 5 2600x@3.6GHz turbo 4.1GHz e GPU Nvidia 1660ti com 1.536 núcleos CUDA, clock de 1.550MHz a 1770 MHz e 6GB de memória GDDR6@12Gbp/s.*

Alguns resultados relacionados as precisões do modelo estão dispostas nas figuras 31, 32, 33 e 34*.*

Figura 31 - mAP do modelo treinado



Fonte: Autor

Conforme pode ser observado na figura 31, a precisão média do modelo, desde o passo número 40.000 avançou pouco no quesito de precisão, as variações eram compensadas em aumentos na precisão para objetos menores ou maiores. Alcançando ao final uma mAP (mean Average Precision) de 41,38%.

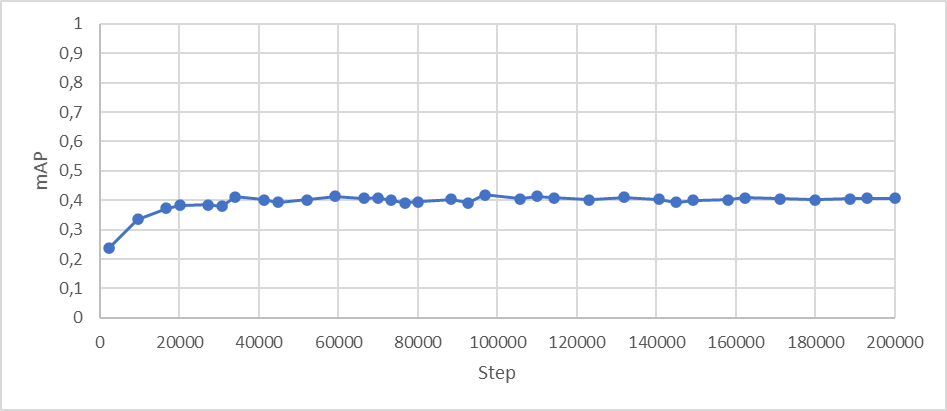
Figura 31 - mAP para objetos grandes



Fonte: Autor

Para objetos grandes, conforme pode ser observado na figura 32 a mAP final foi de 69.8%. A métrica deve compreender veículos próximos a câmera, por exemplo: diretamente a frente.

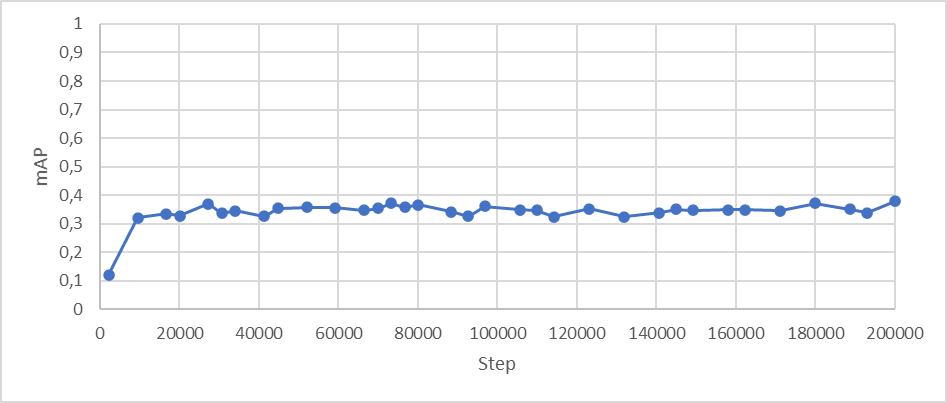
Figura 32 - mAP para objetos médios



Fonte: Autor

Em objetos médios, de acordo com a figura 33 a mAP alcançada foi de 40,71%, essa métrica deve compreender normalmente veículos a uma distância razoável e pedestres próximos.

Figura 33 - mAP para objetos pequenos



Fonte: Autor

Para objetos pequenos, que devem compreender pedestres e veículos em distancias maiores a mAP alcançada foi de 39.75% conforme figura 34.

Avaliando as métricas obtidas em relação a precisão do modelo e comparando-as com o modelo base treinado com o dataset COCO, a mAP obtida se mostrou superior. Idealmente seria necessária uma precisão média maior para evitar problemas relacionados a detecção de agentes no ambiente da simulação.

Ao executar o modelo treinado aplicando ao codigo do agente definido no capitulo 6 o resultado anterior parece mais promissor, demarcando de forma eficiente boa parte dos elementos dispostos na imagem obtida pelo sensor RGB conforme figura 34.

Figura 34 - Modelo treinado aplicado na simulação



Fonte: Autor

Como pode ser observado na figura 34, o modelo treinado consegue detectar com certa eficiencia os elementos dispostos na imagem. A partir destes dados das demarcações é feita uma tentativa de rastreamento dos objetos na tela.

Esse rasteamento é definido através das comparação entre os quadros obtidos, considerando uma colisão entre as áreas para cada quadro. Para melhorar essa continuidade, as áreas são acrescidas de 5%. A partir dessa area detectada são definidos com a variação do ponto central de cada objeto anotado qual seria uma tendência de movimento do mesmo.

Com relação ao desempenho no quesito quadros por segundo do agente da simulação, considerando a detecção dos objetos combinada com o rastreamento dos elementos a taxa de quadros obtida foi de 5 FPS.

### Considerações referentes ao modelo e a simulação

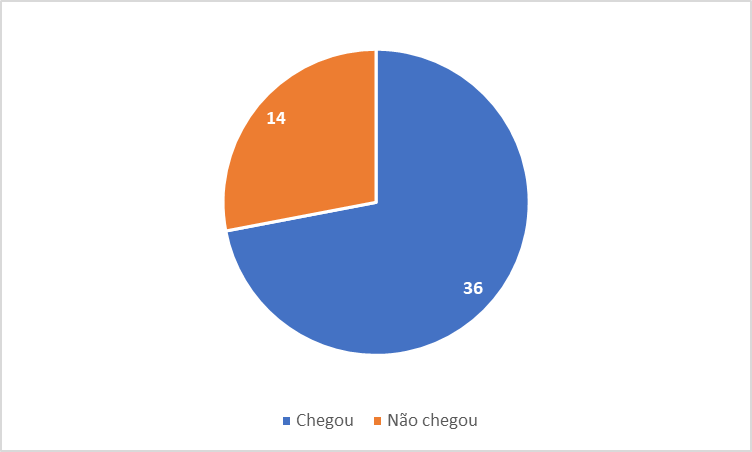
Conforme descrito neste capítulo, inicialmente foi realizado a anotação manual dos semáforos para o modelo treinado, porém, em uma versão anterior a apresentada nesse capítulo, houve grandes problemas na detecção dos mesmos por se tratar de áreas muito pequenas, ainda mais considerando a distância das mesmas em alguns cruzamentos do mapa da simulação, onde a área de um semáforo tinha dimensões menores que 10x10.

Sendo necessário considerar os semáforos no ambiente da simulação para a conclusão de um cenário houve a necessidade do uso de recursos disponíveis na classe do veículo do simulador CARLA, através do uso de uma função que retorna se o agente se encontra próximo a um semáforo e qual sua situação e com isso realizar a parada no mesmo.

# Resultados do agente

Após a integração do modelo de detecção de objetos ao agente foram realizadas algumas avaliações conforme descritas na metodologia considerando o seguinte cenário: utilizando o mapa 3 do simulador com outros 70 veículos e 35 pedestres e 50 execuções do agente. a distribuição dos resultados encontra-se na figura 35.

Figura 35 - Sucessos x Falhas na simulação do agente



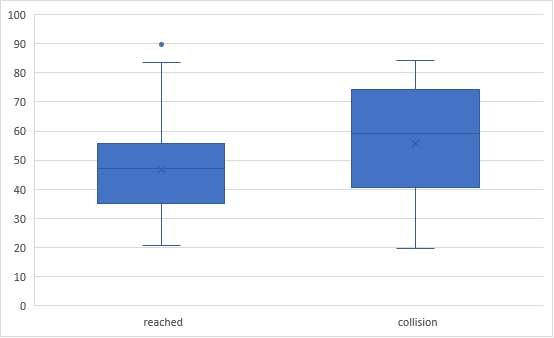
Fonte: Autor

Dentre os resultados observados em 72% das execuções o agente alcançou seu objetivo sem colisões, nos 28% restantes houveram colisões. Também foram coletados dados de velocidade média para cada execução, através dessa métrica pode ser possível entender alguns dos problemas encontrados nesse percentual de falhas.

Conforme capítulo 7 o desempenho em QPS alcançado pelo agente foi de 5 QPS, resultado aquém do ideal para a simulação, que em diversas situações por falta de passos o controlador PID referente ao controle lateral tenha problemas ao suavizar curvas ou mudanças de faixas, levando algumas vezes a perda de controle do veículo.

Outro problema dos cenários que poderia ser mitigado com mais desempenho é a aplicação do modelo de detecção de objeto em mais câmeras distribuídas pelo agente, por questões de performance já relatadas, para que fosse possível a execução do mesmo não possível essa inclusão.

Figura 36 - Velocidade média do agente

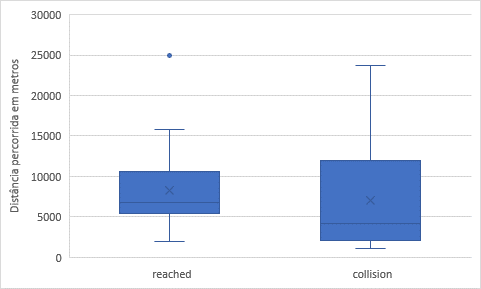


Fonte: Autor

Conforme figura 36, durante a execução do agente para cada segundo decorrido foi realizado a anotação da velocidade média do veículo. Pode-se observar que nos cenários onde houveram falhas, a velocidade média tende a ser maior, indicando possivelmente uma dificuldade na deteção de alguns elementos a uma distância maior e eventualmente impossibilitando uma redução a tempo de evitar a colisão.

Em relação a distancia percorrida em cada cenário executado, a distribuição das médias é definida conforme figura 37.

Figura 37 - Distância percorrida pelo agente



Fonte: Autor

Pode-se observar que a média da distancia percorrida é similar para situações onde houveram colisões e de sucesso, considerando também a diferença no número de observações de falhas 14 em 50.

# CONCLUSÃO

Por tratar-se de uma área em pleno desenvolvimento, diversos modelos de direção autônoma estão em desenvolvimento e testes atualmente, tanto por parte de fabricantes de veículos a estudos da área, buscando novos avanços e possíveis melhorias para essa área que podem revolucionar a forma como é feita a interação com toda a infraestrutura de transportes.

O desenvolvimento como pode ser acompanhando no decorrer do trabalho engloba diversas áreas do conhecimento da ciência da computação como o aprendizado de máquina, visão computacional e composição através de outras técnicas e recursos visando o desenvolvimento e proposta de modelo para navegação autônoma a ser validado no simulador CARLA.

O modelo proposto no trabalho necessita de mais avanços em aspectos de precisão e velocidade, porém, consegue demonstrar que através do uso de recursos mais simples, como uma câmera RGB e sensores de obstáculo com um modelo de detecção de objetos pode alcançar um resultado inicial interessante para a aplicação referida.

Apesar do resultado ficar aquém de diversos parâmetros de segurança necessários, demonstra também, de certa forma, um avanço grande na capacidade computacional para redes neurais aceleradas por placas gráficas. Ainda que o desempenho para a aplicação tenha ficado abaixo do ideal com o hardware disponível, existem opções comerciais que permitiriam a execução do agente a uma taxa de quadros próxima da ideal.

Considerando os resultados de segurança do agente desenvolvido, em 28% dos 50 cenários executados houve algum tipo de colisão que encerrou a execução. Alguns avanços podem ser realizados tanto nos algoritmos que fazem o controle lateral e longitudinal do veículo e no modelo de detecção de objetos. Levando em conta que o agente faz uso de apenas uma câmera frontal o sistema de controle lateral fica prejudicado visto que em uma troca de faixa não é possível determinar se há um veículo realizando uma ultrapassagem.

Conforme descrito nos resultados do agente uma das limitações para o desenvolvimento foi a falta de desempenho para a execução do modelo em condições ideais, porém, considerando a baixa taxa de quadros, os controladores PID responsáveis pelo controle do veículo foram capazes de determinar os valores de controle lateral e longitudinal de forma satisfatórias.

Os problemas apareceram principalmente em situações em que há curvas acentuadas, devido ao tempo de resposta entre as ações que dificulta a execução de um movimento mais suave. Dificultando a compensação do movimento pelo controlador PID.

Apesar dos resultados obtidos não serem os ideais, há um potencial de segurança adicional que poderia ser alcançado através do uso massivo de sistemas de navegação autônoma, visto que boa parte dos acidentes de trânsito poderiam ser evitados.

Ainda há um longo caminho para a massificação e certificação de sistemas capazes de realizar a navegação autônoma de nível 5 onde há plenas condições de operação em qualquer ambiente e qualquer situação. Sistema de nível 4 já estão sendo testados em alguns locais, porém o caminho para a adoção massificada depende de muito de uma redução de custos, mas os resultados certamente podem revolucionar todo o sistema viário do mundo.

Em trabalhos futuros podem ser exploradas possibilidades com o uso de dados dos sensores de LIDAR e melhorias nos algoritmos de detecção do ambiente. Assim como um planejamento local de desvios e tratamento de situações adversas permitindo que o agente tenha maior alcance de aplicação prático.

Referências Bibliográficas

ADIB, M. Automatic Vehicle 2D Bounding Box Annotation Module for CARLA Simulator. 2020. Disponível em: <https://mukhlasadib.github.io/CARLA-2DBBox/>. Acesso em: 31 out. 2020.

AUGUSTO CHAVES, A.; JAMILSON FREITAS SOUZA, M. **Modelagens Exata e Heurística para Resolução do Problema do Caixeiro Viajante com Coleta de Prêmios**. 2003. [s. l.], 2003.

BOJARSKI, M. et al. End to End Learning for Self-Driving Cars. [s. l.], 2016. Disponível em: <http://arxiv.org/abs/1604.07316>. Acesso em: 14 mar. 2020.

DENG, Z. et al. Multi-scale object detection in remote sensing imagery with convolutional neural networks. ISPRS Journal of Photogrammetry and Remote Sensing, [s. l.], v. 145, n. May, p. 3–22, 2018. Disponível em: <https://doi.org/10.1016/j.isprsjprs.2018.04.003>

DOSOVITSKIY, A. et al. CARLA: An Open Urban Driving Simulator. In: PROCEEDINGS OF THE 1ST ANNUAL CONFERENCE ON ROBOT LEARNING 2017, **Anais**... Disponível em: http://proceedings.mlr.press/v78/dosovitskiy17a/dosovitskiy17a.pdf

FERNEDA, E. Redes neurais e sua aplicação em sistemas de recuperação de informação. **Ciência da Informação**, [s. l.], v. 35, n. 1, p. 25–30, 2006. Disponível em: <http://www.scielo.br/pdf/ci/v35n1/v35n1a03.pdf>. Acesso em: 03 abr. 2020.

HOWARD, D.; DAI, D. Public Perceptions of Self-driving Cars. **93rd Annual Meeting of the Transportation Research Board**, [s. l.], v. 2014, p. 21, 2013. Disponível em: https://www.ocf.berkeley.edu/~djhoward/reports/Report - Public Perceptions of Self Driving Cars.pdf

HUANG, J. et al. **Speed/accuracy trade-offs for modern convolutional object detectors**. 2016. Disponível em: https://arxiv.org/abs/1611.10012

HUANG, J. Accelerating AI with GPUs: A New Computing Model. 2016. Disponível em: <https://blogs.nvidia.com/blog/2016/01/12/accelerating-ai-artificial-intelligence-gpus/>. Acesso em: 30 out. 2020.

KHANDELWA, R. COCO and Pascal VOC data format for Object detection. [s.d.]. Disponível em: <https://towardsdatascience.com/coco-data-format-for-object-detection-a4c5eaf518c5>. Acesso em: 20 out. 2020.

KOLLER, D.; LANDSHUT, H. Moving Object Recognition and Classification based on Recursive Shape Parameter Estimation. [s.l: s.n.]. Disponível em: <https://www.researchgate.net/publication/2451473>. Acesso em: 30 maio. 2020.

LIND, E. et al. **A performance comparison between CPU and GPU in TensorFlow**. 2019. Disponível em: <https://www.diva-portal.org/smash/get/diva2:1354858/FULLTEXT01.pdf>.

MIHAJLOVIC, I. **Everything You Ever Wanted to Know About Computer Vision.** 2019. Disponível em: <https://towardsdatascience.com/everything-you-ever-wanted-to-know-about-computer-vision-heres-a-look-why-it-s-so-awesome-e8a58dfb641e>. Acesso em: 3 abr. 2020.

NATIONAL CENTER FOR STATISTICS AND ANALYSIS. Summary of motor vehicle crashes: 2016 data. **Traffic Safety Facts. Report No. DOT HS 812 580**, [s. l.], v. 2018, n. March, p. 1–10, 2018.

OSTROWSKI, D. et al. Comparitive Analysis of the Algorithms for Pathfinding in GPS Systems. **ICN 2015, the Fourteenth International Conferences on Networks**, [s. l.], n. c, p. 115–121, 2015.

PROVDANOV, C. C.; FREITAS, E. C. De. **Metodologia do trabalho científico: métodos e técnicas da pesquisa e do trabalho acadêmico**. [s.l: s.n.]. Disponível em: <http://www.feevale.br/Comum/midias/8807f05a-14d0-4d5b-b1ad-1538f3aef538/E-book Metodologia do Trabalho Cientifico.pdf>

RIESENHUBER, M.; POGGIO, T. Models of object recognition. **Nature Neuroscience**, [s. l.], v. 3, n. 11s, p. 1199–1204, 2000.

SAE INTERNATIONAL. Surface Vehicle. SAE International, [s. l.], v. 4970, n. 724, p. 1–5, 2018.

TZUTALIN. LabelImg - a graphical image annotation tool and label object bounding boxes in images. 2015. Disponível em: <https://github.com/tzutalin/labelImg>. Acesso em: 10 out. 2020.

WITTEN, I. H.; EIBE, F.; HALL, M. A. Data Mining. In: **Data Mining: Practical Machine Learning Tools and Techniques**. 3a Edição ed. [s.l.] : Elsevier, 2011. p. 605.

ZHOU, X.; ANGELOV, P. Autonomous visual self-localization in completely unknown environment using evolving fuzzy rule-based classifier. In: PROCEEDINGS OF THE 2007 IEEE SYMPOSIUM ON COMPUTATIONAL INTELLIGENCE IN SECURITY AND DEFENSE APPLICATIONS, CISDA 2007 2007, **Anais**... : IEEE, 2007. Disponível em: <http://ieeexplore.ieee.org/document/4219092/>. Acesso em: 20 mar. 2020.