



PONTIFÍCIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS  
Programa de Graduação em Engenharia de Computação

Wenderson Júnio de Souza

**ALGORITMO DEEP Q-NETWORK PARA PERCEPÇÃO E  
MOVIMENTAÇÃO DE UM VEÍCULO AUTÔNOMO NO SIMULADOR  
CARLA**

Belo Horizonte  
2020

Wenderson Júnio de Souza

**ALGORITMO DEEP Q-NETWORK PARA PERCEPÇÃO E  
MOVIMENTAÇÃO DE UM VEÍCULO AUTÔNOMO NO SIMULADOR  
CARLA**

Monografia apresentada ao Programa de Graduação em Engenharia de Computação da Pontifícia Universidade Católica de Minas Gerais, como requisito parcial para obtenção do título de Bacharel em Engenharia de Computação.

Professor: Prof. Dr. Henrique Cota de Freitas

Área de concentração: Sistemas de Computação

Belo Horizonte  
2020

FICHA CATALOGRÁFICA

Elaborada pela Biblioteca da Pontifícia Universidade Católica de Minas Gerais



Wenderson Júnio de Souza

**ALGORITMO DEEP Q-NETWORK PARA PERCEPÇÃO E  
MOVIMENTAÇÃO DE UM VEÍCULO AUTÔNOMO NO SIMULADOR  
CARLA**

Monografia apresentada ao Programa de Graduação em Engenharia de Computação da Pontifícia Universidade Católica de Minas Gerais, como requisito parcial para obtenção do título de Bacharel em Engenharia de Computação.

Área de concentração: Sistemas de Computação

---

Prof. Dr. Henrique Cota de Freitas (Orientador) – PUC Minas

---

Prof. Dr. Zenilton Kleber Gonçalves do Patrocínio Júnior – PUC Minas

---

Prof. MSc. Marcos Wander Rodrigues – PUC Minas

Belo Horizonte, 04 de Dezembro de 2020

*Dedico este trabalho aos meus pais, Edilene e Wilson, pelos cuidados, suporte e incondicional apoio. À minha família pelos encorajamento aos estudos e à minha namorada Carolina, pelo carinho, incentivos e paciência ao ouvir sobre carros autônomos.*

## **AGRADECIMENTOS**

Agradeço a todos que puderam dedicar algum tempo, diretamente ou indiretamente, a me auxiliar em alguma etapa antes e durante este trabalho. Em especial ao professor orientador Henrique Cota, com suas opiniões, sugestões, incentivos e acreditar no trabalho poderia e foi realizado. Ao professor Zenilton K., por seus conselhos, sugestões e críticas, quando necessárias, além das brincadeiras quando oportunas. Ao professor Sandro J., pelo primeiro contato com Algoritmos Genéticos, que me levou a aguçar a curiosidade pelo problema de carros autônomos, e sugestões de trabalho. A todos os colegas, professores(as) e coordenadores do curso de Engenharia de Computação. Aos colegas do laboratório de pesquisa CArT, pelas dicas e recepção, e a PUC Minas, pelo auxílio na pesquisa sobre o tema.

## RESUMO

O transporte é uma necessidade intrínseca da humanidade e visando melhorar a qualidade de vida, reduzindo os custos e ineficiência proporcionados pelo intenso tráfego de veículos no dia a dia das grandes cidades, o desenvolvimento dos veículos autônomos vêm sendo cada vez mais estudados, em busca de soluções que atendam a essa demanda. Este trabalho consiste no desenvolvimento, exploração, simulação, testes e avaliação de um carro autônomo com abordagem de aprendizado *end-to-end* utilizando aprendizado por reforço com o algoritmo Deep Q-Network, baseado nas imagens de uma câmera frontal e da velocidade do carro, sendo utilizado o simulador CARLA para simulação do ambiente virtual. O carro realiza o controle lateral e longitudinal, de forma a movimentar-se no trânsito e não colidir, recebendo recompensas de acordo com o seu comportamento e possui sensores virtuais de colisão e invasão de faixa, exclusivos do simulador e dos sensores de câmera e velocidade. O veículo tem como objetivo movimentar-se livremente pelo trânsito, recebendo recompensas pela sua velocidade e posição em relação ao centro da faixa. Neste trabalho a experimentação foi muito importante para implementar, incrementar e aprimorar o algoritmo DQN com a estrutura que recebe os dados e controla o carro no simulador Carla. Com o progresso na implementação do sistema, inicia-se o treinamento do carro autônomo, utilizando 200 episódios limitados a 501 passos por episódio, seguido por uma avaliação de 100 episódios, com ações ilimitadas, desde que não sejam ações que imobilizem (ação "freio" e "sem-ação") o veículo por mais de 100 passos sequenciais. Os resultados mostram que o veículo consegue aprender comportamentos básicos, mas falha por não ser capaz de percorrer longas distâncias, ainda que não seja um dos objetivos de suas recompensas. Na análise de resultados é possível perceber que o veículo aprende sobre o controle lateral, ao permanecer dentro de sua faixa, mas apresenta traços de ter se tornado muito conservador em lidar com as colisões com outros veículos que se aproximam do sentido contrário e tende a tomar ações mais cautelosas como ir para a frente, mas virando o volante para a direita (ação "frente-direita"). Essas ações levam o veículo a invadir as faixas da direita, que limitam a sua via de rolamento do seu sentido, uma ação que tem menor punição do que causar uma colisão e invadir a faixa do sentido oposto. Além disso, o veículo também encontrou brechas nas recompensas, explorando recompensas recebidas por estar no centro da faixa, mas parado. As ações no geral do agente mostram quem é capaz de aprender as ações básicas no trânsito, mas necessitam de melhoria, requerendo mais horas de treinamento e possíveis mudanças nas recompensas.

Palavras-chave: Carro autônomo, CARLA, *End-to-End*, DQN, Aprendizado por Reforço.

## ABSTRACT

Transportation it is a intrinsic human kind's necessity and aiming to improve life quality, reducing costs and inefficiency proportioned by day-to-day intense human traffic of vehicles from big cities, the autonomous vehicle development comes to be every time more studied, searching for solutions that answers that demand. This work consists of developing, exploration, simulation, tests and evaluation of a self-driving car with an end-to-end approach using reinforcement learning with Deep Q-Network algorithm, based in a frontal camera and vehicle's speed, using CARLA simulator to simulate the virtual environment. The car perform a lateral and longitudinal control, capable of moving through traffic and not collide, receiving rewards according to its behaviour and posses virtual sensor to detect collision and lane invasions, exclusives from the simulator, and camera and speed sensors. The vehicle's objectives are move freely trough the traffic, receiving rewards by its speed and position related to the lane center. In this work the experimentation was very important to implement, increment and enhance the DQN algorithm with the structure that receives the data and controls the car in the simulator Carla. With the implementation's progress, begins the self-driving car's training using 200 episodes limited to 501 steps per episode, followed by an evaluation of 100 episodes, with unlimited actions, since it's not 100 sequential immobilization actions actions (action "break"and "no-action"). The results shows that the vehicle can learn basic behaviours, but fails to go long distances yet it's not its reward's objective. In the result analyses it's possible to notice that the vehicle learn about lateral control, remaining in its lane, but show conservative traces on dealing with collision with other vehicles that comes from the opposite lane and tends to take cautious actions such as go front, but turning the wheels to the right (action "front-right"). Those actions takes the vehicle to invade the right lanes, that limits its way, an action that has a small punishment that collide and invade an two-way lane on the opposite side. Besides, the vehicle also found a breach in the rewards, exploiting received rewards from being the center of the lane, but not moving. In general, the agent's actions show that it's capable of learning the basic traffic actions, but needs improvement, requiring more training hours and possible reward changes.

Keywords: Self-driving-car, CARLA, End-to-End, DQN, Reinforcement Learning.

## LISTA DE FIGURAS

|   |    |
|---|----|
| FIGURA 1 – Níveis de automação - SAE . . . . .  | 19 |
| FIGURA 2 – Arquitetura para veículos autônomos . . . . .                                    | 20 |
| FIGURA 3 – Redes Neurais . . . . .  | 21 |
| FIGURA 4 – Processo de Aprendizagem por Reforço . . . . .                                   | 22 |
| FIGURA 5 – Algoritmo <i>Q-Learning</i> . . . . .  | 23 |
| FIGURA 6 – Algoritmo DQN . . . . .  | 24 |
| FIGURA 7 – Exemplo de Simulação do Carla . . . . .  | 25 |
| FIGURA 8 – Esquema de recompensas . . . . .   | 26 |
| FIGURA 9 – Arquitetura do ambiente Carla com DQN . . . . .                                  | 29 |
| FIGURA 10 – Rede Neural da DQN . . . . .  | 32 |
| FIGURA 11 – Funções das máquinas PC1 e PC2 . . . . .  | 34 |
| FIGURA 12 – Experimentos com o simulador Carla . . . . .                                    | 34 |
| FIGURA 13 – Treinamento - Quantidade de Colisões por Episódio . . . . .                     | 38 |
| FIGURA 14 – Treinamento - Tendência da Quantidade de Colisões por Episódio . . . . .        | 39 |
| FIGURA 15 – Treinamento - Invasão de Faixas Seccionadas . . . . .                           | 39 |
| FIGURA 16 – Treinamento - Invasão de Faixa Contínua . . . . .                               | 40 |
| FIGURA 17 – Treinamento - Contador de Ações do Episódio 200 . . . . .                       | 40 |
| FIGURA 18 – Treinamento - Imagem da Câmera Frontal do Ep. 200 . . . . .                     | 40 |
| FIGURA 19 – Treinamento - Distância Total Percorrida de Ré em Metros por Episódio . . . . . | 41 |
| FIGURA 20 – Treinamento - Distância Total Percorrida em Metros por Episódio . . . . .       | 41 |
| FIGURA 21 – Treinamento - Velocidade por passos (Tacógrafo) do Ep. 200 . . . . .            | 41 |
| FIGURA 22 – Gráfico da Função <i>Loss</i> do Treinamento por Episódio . . . . .             | 42 |
| FIGURA 23 – Treinamento - Quantidade de passos por Episódio . . . . .                       | 42 |
| FIGURA 24 – Treinamento - Recompensa Total e Média por Episódio . . . . .                   | 43 |
| FIGURA 25 – Quantidade de Colisões por Episódio (Avaliação) . . . . .                       | 44 |
| FIGURA 26 – Invasão de Faixas Seccionadas por Episódio (Avaliação) . . . . .                | 44 |
| FIGURA 27 – Gráfico de Invasão de Faixa Contínua por Episódio (Avaliação) . . . . .         | 44 |

|   |    |
|---|----|
| FIGURA 28 – Contador de Ações do Episódio 100 (Avaliação) . . . . .                       | 45 |
| FIGURA 29 – Velocidade por Passos (Tacógrafo) do Ep. 100 (Avaliação) . . . . .            | 45 |
| FIGURA 30 – Imagem da Câmera Frontal do Ep. 100 (Avaliação) . . . . .                     | 46 |
| FIGURA 31 – Distância Total Percorrida de Ré em Metros por Episódio (Avaliação) . . . . . | 46 |
| FIGURA 32 – Distância Total Percorrida em Metros por Episódio (Avaliação) . . . . .       | 46 |
| FIGURA 33 – Quantidade de Passos por Episódio (Avaliação) . . . . .                       | 47 |
| FIGURA 34 – Recompensa Total e Média por Episódio (Avaliação) . . . . .                   | 47 |

## **LISTA DE TABELAS**

|   |    |
|---|----|
| TABELA 1 – Variáveis utilizadas no código . . . . . | 31 |
|---|----|

## **LISTA DE SIGLAS**

CPU – *Central Processor Unit*

GPU – *Graphics Processor Unit*

DQN – *Deep Q-Learning*

IMU – *Inertial Measurement Unit*

MDP – *Markov Decision Process*

ACC – Controle de velocidade adaptativo

SAE – *Society of Automotive Engineers*

RL – Aprendizado por reforço (*Reinforcement Learning*)

API – *Application Programming Interface*

## SUMÁRIO

|   |           |
|---|-----------|
| <b>1</b> Introdução . . . . .   | <b>14</b> |
| 1.1 Problema . . . . .  | 15        |
| 1.2 Objetivos . . . . .   | 15        |
| 1.2.1 <i>Objetivos específicos</i> . . . . .  | 15        |
| 1.3 Justificativa . . . . .   | 16        |
| 1.4 Organização do texto . . . . .  | 17        |
| <b>2</b> Referencial Teórico . . . . .  | <b>18</b> |
| 2.1 Veículos autônomos . . . . .  | 18        |
| 2.1.1 <i>Níveis de automação</i> . . . . .  | 18        |
| 2.1.2 <i>Percepção</i> . . . . .  | 18        |
| 2.1.3 <i>Arquitetura de veículos autônomos</i> . . . . .                              | 19        |
| 2.2 Aprendizado de Máquina . . . . .  | 20        |
| 2.2.1 <i>Aprendizado Profundo</i> . . . . .   | 20        |
| 2.3 Aprendizado por reforço . . . . .   | 21        |
| 2.4 <i>Q-Learning</i> . . . . .   | 22        |
| 2.5 Algoritmo DQN (Deep Q-Network) . . . . .  | 23        |
| 2.6 Simulador Carla . . . . .   | 24        |
| 2.7 Trabalhos Correlatos . . . . .  | 25        |
| <b>3</b> Metodologia . . . . .  | <b>28</b> |
| 3.1 Materiais . . . . .   | 28        |
| 3.2 Estudo do contexto do Simulador Carla e ferramentas . . . . .                     | 28        |
| 3.3 Desenvolvimento do ambiente . . . . .   | 28        |
| 3.4 Implementação da etapa de percepção e movimentação do veículo . . . . .           | 28        |
| 3.5 Implementação do algoritmo DQN . . . . .  | 29        |
| 3.6 Experimentos, testes e correções . . . . .  | 29        |
| 3.7 Treinamento do agente . . . . .   | 29        |
| 3.8 Validação do agente . . . . .   | 30        |
| 3.9 Métricas de avaliação . . . . .   | 30        |
| <b>4</b> Proposta Desenvolvida . . . . .  | <b>31</b> |
| 4.1 Desenvolvimento do Código DQN . . . . .   | 31        |
| 4.2 Classes de Ações . . . . .  | 32        |
| 4.3 Sensores . . . . .  | 33        |
| 4.4 Desenvolvimento do ambiente e utilização de mapas . . . . .                       | 33        |
| 4.5 Experimentos . . . . .  | 34        |
| 4.6 Função de recompensa . . . . .  | 35        |
| <b>5</b> Análise e discussão dos resultados . . . . .                                 | <b>37</b> |
| 5.1 Experimentos . . . . .  | 37        |
| 5.2 Treinamento . . . . .   | 38        |
| 5.3 Avaliação do teste final . . . . .  | 43        |
| <b>6</b> Conclusão . . . . .  | <b>48</b> |
| 6.1 Trabalhos Futuros . . . . .   | 48        |
| Referências Bibliográficas . . . . .  | 49        |
| <b>APÊNDICE A - Trecho de Código DQN para Treinamento do carro</b> . . . . .          | <b>51</b> |
| <b>APÊNDICE B - Trecho de código para AVALIAÇÃO da Agente</b> . . . . .               | <b>53</b> |
| <b>ANEXO A - Trecho de Código Auxiliar para criar atores - Spawn Actors</b> . . . . . | <b>54</b> |

**ANEXO B - Trecho de código para controle manual de um veículo . . . . 56**

## 1 INTRODUÇÃO

Nos dias atuais, o transporte é uma necessidade intrínseca da humanidade e está presente em todos os momentos da vida do ser humano, seja direta ou indiretamente, ainda que os avanços da computação permitam a comunicação com pessoas distantes, realizar compras e recebê-las sem sair de casa, etc. A locomoção de pessoas, utilizando um transporte pessoal, é movida pela necessidade que elas têm de chegar mais rápido e de forma confortável aos seus destinos. No entanto, um dos problemas causados por essas necessidades é o aumento das frotas de veículos nas cidades. De acordo com os dados fornecidos pelo DENATRAN (2020), em janeiro de 2018 havia 74.712.393 de automóveis e motocicletas. Em janeiro de 2020 havia 80.020.977, o que representa um aumento de, aproximadamente, 7,10% de automóveis e motocicletas. Esse aumento da frota implica em maiores taxas de acidentes, tendo o fator humano na condução de um veículo como uma das causas, já que os humanos são suscetíveis à diversos fatores, tais como emocionais, erros de sinalização, etc (LíDER-DPVAT, 2019).

Os veículos autônomos contam com o auxílio da tecnologia para serem capazes de automatizarem o controle do veículo que seriam realizadas por um ser humano, podendo permitir melhorar a qualidade do trânsito e podendo ser capazes de lidar com os problemas e adversidades que o trânsito proporciona, sem o desgaste físico e mental que impactam o ser humano. O objetivo final da automação dos veículos pode ser considerado como um veículo que leve passageiros ou cargas de um ponto ao outro, de escolha do usuário, sem que esta realize o controle direto do veículo e o veículo possa realizar o controle em qualquer terreno, ambiente, condições de trânsito, tempo, etc. Permitindo um transporte autônomo para o usuário em qualquer situação (equivalente ao nível 5 de autonomia, classificado pela SAE (INTERNATIONAL; SAE, 2018), e gerando um maior conforto e segurança aos usuários.

Há muitos anos têm-se automatizado determinados controles dos veículos, como por exemplo o controle de cruzeiro (a central de comando do veículo mantém uma velocidade pré-determinada, realizando a aceleração do motor quando abaixo da velocidade determinada); frenagem de emergência, etc. Estes controles têm como objetivo melhorar segurança do trânsito, aumentar o conforto do motorista e passageiros, etc. Tais automações fazem parte dos níveis básicos de automação e precedem os veículos autônomos (INTERNATIONAL; SAE, 2018; Coichesci; Filip, 2020).

Vários artigos propõem soluções para resolver parcialmente os problemas ou tarefas dos veículos autônomos, usando algum método de aprendizado de máquina. Entre as propostas de soluções, têm-se as propostas de soluções de movimentação por meio do controle lateral ou longitudinal do veículo, que são, respectivamente, o controle do volante, utilizado em soluções que mantém o veículo dentro da faixa ou segue linhas, e da aceleração e frenagem do veículo, como o controle de velocidade de cruzeiro adaptativo (ACC), que busca manter o veículo a uma distância segura do veículo da frente, alterando sua velocidade conforme necessário (Kuutti et al., 2020).

Além de publicações acadêmicas, há também o exemplo dos veículos utilizados pela empresa Waymo, que se classificam em um nível 4 de 5 níveis de automação (classificados pela SAE) equivalente a um veículo quase totalmente autônomo, mas que ainda requer um motorista em situações específicas (Coichesci; Filip, 2020).

Dentre as diferentes técnicas computacionais que podem ser usadas para construir um veículo autônomo, os algoritmos de aprendizado por reforço (RL), que estão sendo cada vez mais utilizados para realizar as mais diferentes tarefas (LIU et al., 2017). Estes algoritmos permitem que o veículo possa aprender a comportar-se, por meio de sua própria ação e

recompensas recebidas por estas ações, que irão incentivar ou desencorajar a repetição destas ações em outros momento.

Portanto, visando explorar a resolução do problema de controle simultâneo de um veículo autônomo, este trabalho irá inserir um veículo no ambiente do simulador Carla, devido a seu alta capacidade de customização e interação, e utilizar um algoritmo de RL chamado de *Deep Q-Network* (DQN), para que o veículo possa aprender a movimentar-se sem colidir, levando em consideração que o uso da simulação é conveniente junto ao uso de RL, pois permite o veículo interagir sem colocar vidas em risco e não ter o custo de protótipos físicos. O algoritmo DQN já mostrou-se capaz de se adaptar a diversos ambientes, sendo capaz de inferir as ações apropriadas para diferentes agentes, como visto por Mnih et al. (2015).

## 1.1 Problema

Um veículo autônomo pode ser implementado com base em diferentes modelos de arquitetura de componentes que divide as tarefas do veículo. Como a divisão em camadas ou etapas para a captura de dados, percepção, planejamento, controle e movimentação do veículo (JO et al., 2013; GUO et al., 2019).

Para que um veículo autônomo possa realizar a sua movimentação, de forma autônoma, é preciso que seja capaz de realizar o seu controle lateral e longitudinal, de forma simultânea. Para tal, é preciso que o veículo possa ser capaz de identificar o cenário que está à sua volta e reagir. Dessa forma, é capaz de movimentar-se sem colisões.

A tarefa de movimentar-se sem colidir de um veículo autônomo pode ser custosa por demandar muitas sub-tarefas, devido ao pós-processamento necessário dos dados dos sensores, como, por exemplo, a detecção e delimitação das faixas de trânsito por meio de processamento de imagens, que podem envolver diferentes sub-tarefas. Reduzir esse custo de subdivisão de tarefas é um problema atual na área de veículos autônomos e podem ser empregadas as soluções e arquiteturas *end-to-end*, um campo ainda em estudo, que utiliza os dados, sem uma implementação de módulos/componentes específicos para a realização de sub-tarefas, para definir a política de comportamento do veículo.

Para a realização de soluções do tipo *end-to-end*, os algoritmos de RL podem ser utilizados na implementação de veículos autônomos, mas ainda são pouco explorados na literatura, representando um grande problema a ser explorado e analisado.

## 1.2 Objetivos

O presente trabalho tem como objetivo explorar a utilização do algoritmo de DQN e verificar se é possível utilizar este algoritmo para a identificação e controle da movimentação lateral e longitudinal do veículo, evitando colisões no trânsito de uma cidade, por um agente autônomo, na forma de um carro autônomo, utilizando o simulador Carla e, por fim, análise dos resultados.

### 1.2.1 *Objetivos específicos*

Os objetivos específicos são:

- Implementar a camada de percepção e movimentação, junto dos seus sensores e atuadores no simulador Carla;
- Implementar o algoritmo DQN para realizar o aprendizado do veículo;
- Implementar uma função de recompensa para o veículo, tendo como foco a sua movimentação evitando colisões;

- Desenvolver um sistema de testes e avaliação do desempenho alcançado com o carro autônomo;
- Realizar o treinamento do veículo utilizando o algoritmo DQN;
- Realizar o uso da simulação do carro autônomo com processadores *multicore* e *GPU*;
- Avaliar o comportamento do veículo com base nas métricas e avaliações estabelecidas na metodologia para as simulações e resultados obtidos.

### 1.3 Justificativa

A implementação das diferentes etapas e tarefas que podem ser usadas na construção de um veículo autônomo podem ser realizadas com diferentes algoritmos ou técnicas. Como por exemplo, a modelagem matemática, que descrevem o comportamento do veículo com equações de cinemática e modelos de probabilidades (THRUN et al., 2006). No entanto, a utilização destas técnicas em todas as etapas diminui a flexibilidade do veículo e podem diminuir a capacidade de atuação do veículo em diferentes localidades. Em contrapartida, também podem ser utilizadas técnicas de aprendizado de máquina, sendo estas o aprendizado supervisionado, não-supervisionado e aprendizado por reforço (RL).

O aprendizado supervisionado depende de um conhecimento especialista que, por meio de exemplos, ensine a máquina o que é certo ou errado. Ou seja, é dependente de supervisão e de grande quantidade de dados de exemplo. Porém, em ambientes desconhecidos e estocásticos (probabilísticos), obter exemplos de comportamentos adequados e que abrangam toda a dimensão de todas as possíveis ações para um estado do ambiente é inviável. Portanto, métodos como o aprendizado supervisionado não são adequados, dado que para obter um bom desempenho, devem ser treinados com uma grande abrangência de dados, os quais devem ser capturados e rotulados. A rotulagem de dados em um ambiente com diversas variáveis, como o de um carro autônomo apresenta grande complexidade devido a diversidade de situações possíveis, aumentando o custo e complexidade do projeto.

Já o método de aprendizado não supervisionado, que, em linhas gerais, identifica estruturas em uma base de dados, não sendo capaz de aprender qual a ação deve ser realizada em um dado estado do ambiente. Portanto, não adequado ao treinamento de um veículo autônomo.

Os algoritmos de RL, que estão sendo cada vez mais utilizado (LIU et al., 2017), podem ser muito bem aproveitados, tendo em vista que a iteratividade é possível utilizando simuladores sem colocar em risco a vida de terceiros e realizar a interação com diversos cenários. Os agentes autônomos que utilizam algoritmos de RL conseguem aprender, sem que haja um envolvimento direto de seres humanos ou máquinas para apontar, diretamente, qual ação realizar em um grande (ou infinito) número de situações. Portanto, este trabalha busca a vantagem do RL, o agente não aprende com um especialista, mas sim com a própria experiência (KIRAN et al., 2020).

Assim, o uso dos algoritmos de RL permite a construção de veículos autônomos mais flexíveis, sem a necessidade de modelagens de cinemática ou equações formais, específicas para cenários e movimentações pré-definidos, já que são algoritmos que aprendem à medida que exploram os ambientes, permitindo que o agente possa adaptar-se às diferentes situações, em que são guiados pelas recompensas recebidas. Portanto, esse treinamento não necessita de captura e rotulagem de dados, como no caso de algoritmos supervisionados.

O algoritmo DQN, que, apesar de não ser o mais recente ou o mais complexo, mas sendo um dos algoritmos mais conhecido do RL, possui características que o permite ser usado para o treinamento de veículos autônomos. Dentro estas pode-se destacar que o algoritmo usa uma política de comportamento que explora novas possibilidades de ações aleatórias, mas rapidamente começa a tirar vantagem da experiência do agente. Possui melhorias, em relação ao algoritmo *Q-Learning*, no qual é baseado e é um algoritmo que ficou muito conhecido por alcançar bons resultados no aprendizado de diversos jogos de Atari, em diferentes ambientes, inclusive atingindo resultados melhores que humanos utilizando os milhões de *frames* tela do jogo (Mnih et al., 2015).

A escolha pelo uso do simulador Carla se deve pela ampla interação da comunidade nos fóruns do GitHub, por ser um simulador de código totalmente aberto e permitir uma ampla variedade de customização da simulação, objetos e fluxo de dados.

Portanto, o presente trabalho poderá ser utilizado pela academia como ponto de partida para a exploração de um algoritmo que consiga realizar, sem a decomposição em outros algoritmos ou tarefas secundários, os controles lateral e longitudinal, simultaneamente, em um ambiente simulado e com trânsito, podendo ser aprimorado e complementado com outras tarefas, como a mudança de arquitetura envolvendo o planejamento de rotas, alterações de recompensas, utilização de diferentes sensores, etc.

#### **1.4 Organização do texto**

Este trabalho se divide da seguinte forma: no Capítulo 1 é apresentada a introdução, problema e justificativa para a proposta de solução do problema de controle simultâneo dos veículos autônomos. No Capítulo 2 é feito o referencial teórico, incluindo os trabalhos correlatos. No Capítulo 3 é apresentada a metodologia que este trabalho segue. No Capítulo 4 é apresentado o desenvolvimento do trabalho, incluindo códigos e os experimentos realizados. O Capítulo 5 apresenta os resultados alcançados pelo trabalho. O Capítulo 6 apresenta a conclusão dos resultados e do trabalho e, por fim, as possibilidades de trabalhos futuros.

## 2 REFERENCIAL TEÓRICO

Neste Capítulo será apresentada a revisão teórica utilizada para a realização do trabalho.

### 2.1 Veículos autônomos

Veículos autônomos dependem de seus programadores para entender o mundo real, pois precisam de uma programação que os levem a reconhecer e atuar sobre as situações do trânsito do mundo real. Assim, pode ser utilizado um vasto acervo sobre algoritmos computacionais, equações matemáticas complexas e probabilísticas, dependendo da sua implementação.

#### 2.1.1 Níveis de automação

O relatório publicado pela SAE International (INTERNATIONAL; SAE, 2018), apresenta práticas recomendadas para veículos autônomos, classificou os níveis de automação que são possíveis, como é apresentado na Figura 1. São classificados os níveis de 0 a 5, do menos automatizado ao totalmente autônomo, respectivamente. Nos níveis apresentados, são categorizados do 0 ao 3 os que possuem nenhuma ou alguma automação, mas que ainda requerem que o veículo seja dirigido por um condutor. As tarefas que esses níveis de automação executam estão relacionadas à assistências ao motorista, como frenagem automática, controle de velocidade de cruzeiro adaptativo e centralização de linha, entre outras. No caso desses níveis, o modo autônomo pode ser interrompido pelo veículo, caso este perceba que não pode conduzir, e os níveis 4 e 5 já são totalmente autônomos suas tarefas e condução.

Os níveis 4 e 5 se diferem na limitação de onde são totalmente autônomos. Sendo que no nível 4, o veículo é autônomo em locais e condições restritos, realizando tarefas como estacionar sozinho e dirigir entre trechos restritos. No entanto, no nível 5, o uso de volantes e pedais já não é mais necessário e o veículo é autônomo em todos os cenários, locais e condições.

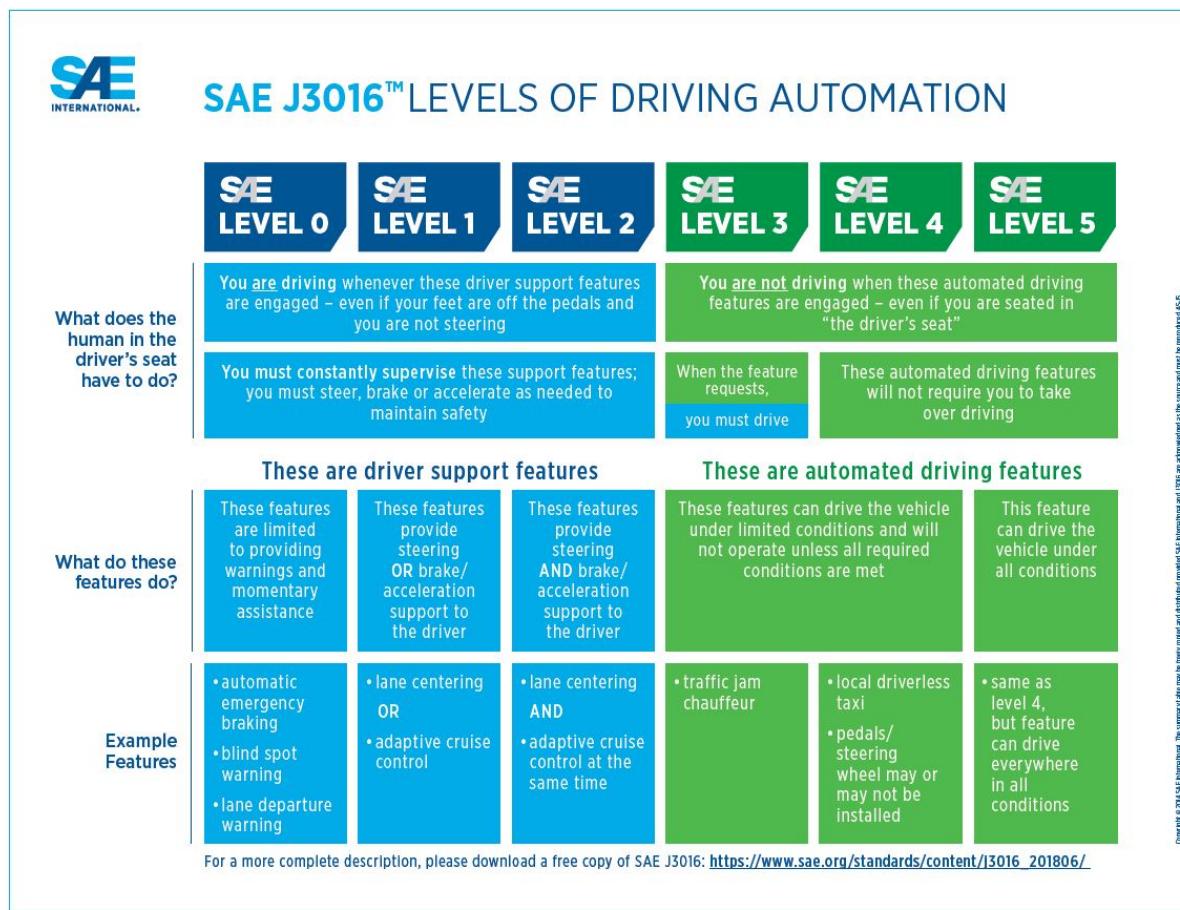
Para alcançar o nível 5 de automação, um veículo deve ser capaz de perceber o seu ambiente, reagir apropriadamente às diversas situações de risco que possam se apresentar e locomover-se, independente dos pontos de origem e destino, sendo capaz de traçar rotas que permitam uma viagem segura aos seus ocupantes.

#### 2.1.2 Percepção

Atualmente os veículos autônomos utilizam de diversas técnicas para perceberem o mundo à sua volta, interpretá-lo e agir em seu ambiente. Pode-se citar as técnicas de visão computacional para reconhecimento de faixas ou a definição do local que o veículo pode ou não dirigir, detecção de objetos, entre outros. Para enxergar o mundo à sua volta, estes veículos utilizam os mais variados sensores, como câmeras de vídeo, radares, lidares, IMU, etc. Cada sensor tem seu objetivo, seu ponto fraco e forte. Portanto, a união destes melhora a capacidade de percepção do veículo e permite que sejam observados o ambiente sobre diferentes perspectivas e aumenta sua confiabilidade (LIU et al., 2017).

Em virtude do alto custo computacional envolvido no processamento dos dados dos sensores e geração de ações dos veículos autônomos, estes só eram desenvolvidos em laboratórios, como tema de pesquisas. No entanto, em 2005, o desafio realizado pela instituição DARPA dos Estados Unidos, chamado *The Grand Challenge for Autonomous Vehicles* mudou o panorama e pesquisas que eram realizados na área, pois esse desafio premiou os veículos capazes de realizar um percurso longo no meio de um deserto, sem

Figura 1 – Níveis de automação - SAE



Fonte: Disponível em  
<https://www.sae.org/news/2019/01/sae-updates-j3016-automated-driving-graphic>.  
 Acesso em 21 de maio de 2020.

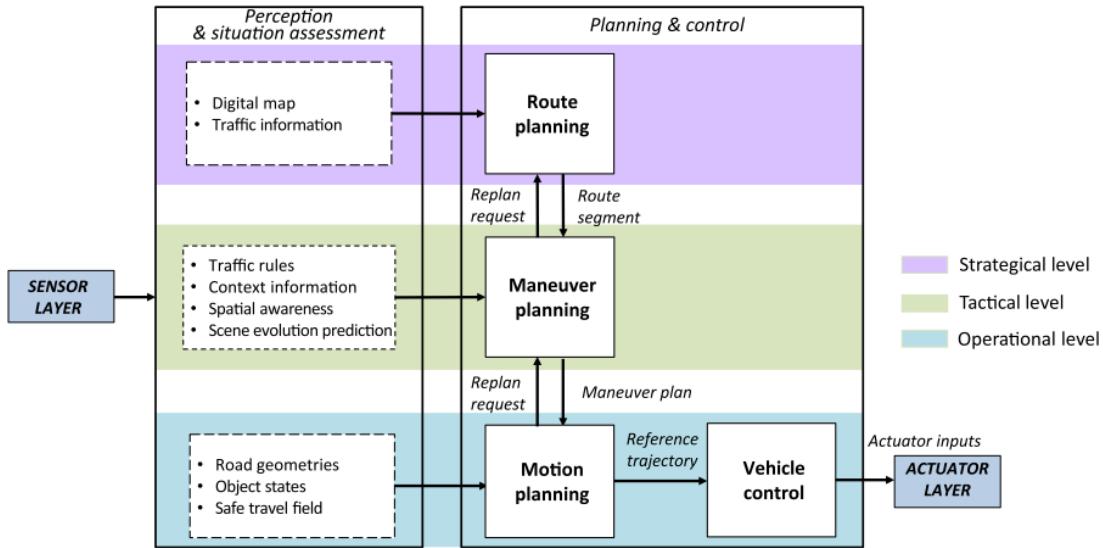
interferência humana. Assim, esse desafio gerou conhecimento e referências bibliográficas, causando um grande avanço na área de veículos autônomos (THRUN et al., 2006).

### 2.1.3 Arquitetura de veículos autônomos

O avanço das tecnologias e literatura sobre a construção de um veículo autônomo ocasionou uma evolução da capacidade dos veículos, o que levou à divisões das tarefas, dadas que essas passaram a se tornar cada vez mais complexas. Por conta disto foram propostas diversas arquiteturas que dividem as tarefas do veículo em diferentes etapas de implementação e processamento, permitindo a melhoria da construção, organização e implementação do veículo (JO et al., 2013; GUO et al., 2019).

De acordo com Guo et al. (2019), uma arquitetura frequentemente utilizada é a que as etapas se dividem em captura de dados, percepção, planejamento de rota e controle do veículo, como é apresentado na Figura 2.

**Figura 2 – Arquitetura para veículos autônomos**



Fonte: (GUO et al., 2019).

## 2.2 Aprendizado de Máquina

O crescimento do volume de dados e a necessidade por resolução de problemas cada vez mais complexos em nossa sociedade ocasionaram a criação de novas técnicas para resolução. Uma destas foram os algoritmos aprendizado de máquina, que conseguem realizar tarefas de resolução de problemas de forma mais autônoma e pode interagir com grande volumes de dados. O aprendizado tende a generalizar e aproximar funções ou hipóteses a partir de exemplos. Sua atuação atualmente é branda, realizando tarefas como reconhecimento de falas, diagnóstico de câncer, etc. (FACELI et al., 2011).

O aprendizado de máquina pode ser subdividido em várias áreas. Entre elas, o aprendizado supervisionado, que é o aprendizado que realiza classificação por meio da aproximação de uma função com base em dados de entrada e dados de saída rotulados.

O aprendizado não supervisionado consiste em encontrar estruturas em uma coleção de dados, podendo realizar o agrupamento, summarização, associação e descrição desses. A coleção de dados não é identificada, ou seja, o algoritmo deve classificar e encontrar os grupos, de acordo com suas características em comum (MARQUES, 2019; SUTTON; BARTO, 2018).

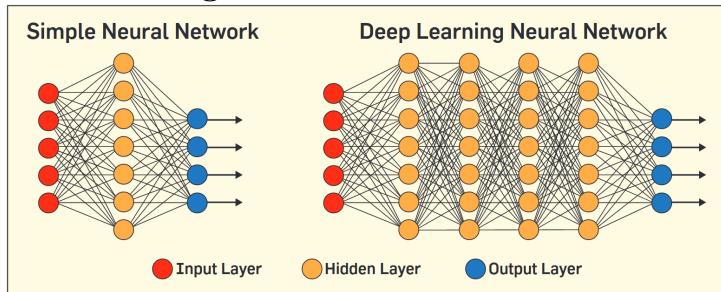
O aprendizado por reforço é o aprendizado iterativo, em que um agente aprende por meio de sua própria experiência, ao realizar ações em um ambiente. Assim, o aprendizado se baseia em relacionar as ações adequadas para o estado que o agente se encontra. Este tende a maximizar a recompensa recebida, o qual irá indicar o quão próximo se está do objetivo definido (SUTTON; BARTO, 2018). Mais detalhes são apresentados na seção 2.3.

### 2.2.1 Aprendizado Profundo

O aprendizado profundo é utilizado no aprendizado de máquina e permite aumentar o poder de computação de uma rede neural simples, com apenas o processamento das entradas por uma camada de inferência, seguida de sua saída. O aprendizado profundo

se dá pela composição de várias camadas de neurônios computacionais, que são utilizados para o processamento de dados. Essas estruturas recebem dados de entradas, as processam e geram uma função, que, por fim, gera um resultado. A estrutura aprende corrigindo a sua função de aproximação, de acordo com os dados de saída e comparando-os com os resultados esperados (GOODFELLOW; BENGIO; COURVILLE, 2016). A Figura 3 apresenta um modelo de estrutura chamada de redes neurais profundas, que é composta de diversas camadas de neurônios. Uma rede neural simples é formada por apenas uma camada entrada de uma camada intermediária de neurônios, além das camadas de entrada e saída.

**Figura 3 – Redes Neurais**



Fonte: (EDWARDS, 2018).

### 2.3 Aprendizado por reforço

O aprendizado por reforço consiste em um agente que pode perceber o ambiente e realizar ações neste ambiente, em busca de alcançar um ou mais objetivos. As suas ações são realizadas visando maximizar a recompensa recebida, em que estas ações são mapeadas, de acordo com as situações, para um ambiente. Estas situações são os chamados **estados**, que são as informações que estão disponíveis para o agente, com base na situação atual do ambiente e o efeito de suas ações neste. No aprendizado por reforço, o agente não é ensinado diretamente quais as ações deve tomar, mas aprende ao explorar (*explore*) e/ou tirar vantagens de sua experiência (*exploit*), conseguindo realizar as ações que o aproximam de um objetivo pré-estabelecido, sendo guiado pelas recompensas recebidas.

Um algoritmo de aprendizado por reforço deve contar com os seguintes elementos, além do agente e ambiente:

A **política** define o comportamento (qual ação deverá realizar) do agente para um dado estado no ambiente.

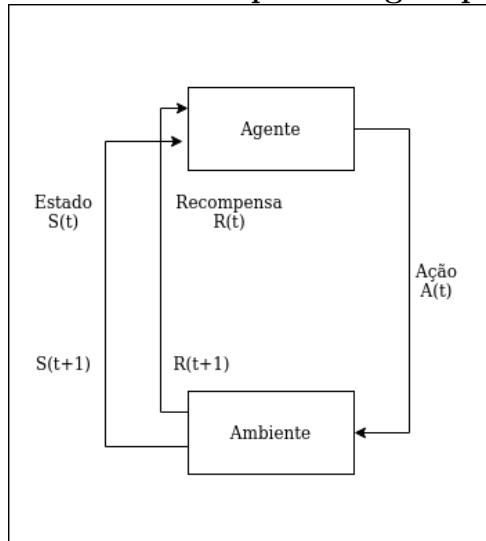
A **recompensa** é o valor da ação imediata de uma ação no ambiente, que é recebida pelo agente. Baixas recompensas para uma dada ação afetam a política do aprendizado, permitindo que o agente possa evoluir e utilizar outra ação com melhor possibilidade de recompensa. Altas recompensas, à princípio, são as melhores opções, dado que estas aproximam o agente de atingir um objetivo. Entretanto, recompensas baixas, em virtude de ações específicas, podem levar a recompensas mais altas futuramente, portanto, é necessário que o agente consiga explorar diversas possibilidades para continuar a evoluir. O valor das recompensas recebidas pelo agente são definidas por uma política de recompensas, que é uma função definida durante a implementação do algoritmo utilizado. A definição dos valores de recompensa da função de recompensas devem punir as ações indesejadas e incentivar as desejadas.

A **função de valor** é a aquela que define o valor das ações que serão tomadas, baseadas em um estado, e na "visão" de quais estados se seguiram e quais ações deverão ser

tomadas, para que sua recompensa futura seja alta. Ainda que a recompensa para uma ação, em um dado momento, seja menor que outra imediata. A função de valor de recompensa é de alta importância para o aprendizado do agente, pois é este que irá permitir o agente escolha as ações que o aproximam do objetivo e não apenas escolher as ações de maior recompensa. Os valores da função afetam a política de comportamento, assim como as recompensas imediatas. Os valores das ações tomadas são constantemente atualizadas com base na experiência do agente em busca do objetivo. Esse é um dos fatores mais importantes nos métodos de aprendizado por reforço.

O **modelo do ambiente** é a representação do ambiente e seu comportamento. Esses modelos podem ser utilizados para que um agente possa "prever" quais as consequências poderão ocorrer de suas ações, o que possibilita a capacidade de planejamento de um agente. Os métodos de aprendizado por reforço que utilizam dessa característica são chamados de "métodos baseados em modelos". Entretanto, nem todos os métodos utilizam esses modelos, caracterizando modelos puramente de tentativa-e-erro.

**Figura 4 – Processo de Aprendizagem por Reforço**



Fonte: Adaptado de (SUTTON; BARTO, 2018).

A Figura 4 demonstra o ciclo de aprendizado iterativo de um agente em um ambiente, conhecido como Processo de Decisão de Markov (MDP). O MDP é a forma matemática utilizada no aprendizado por reforço. Neste, o agente, que observará um estado  $S(t)$  do ambiente, interage com o ambiente por meio de uma ação  $A(t)$ . Esta irá alterar o estado do ambiente, portanto, o agente observará um novo estado  $S(t+1)$  e recebe uma recompensa  $R(t+1)$  por seu ato.

A recompensa é atribuída ao agente, que irá impactar em sua política e a sua função de valor. Assim, iterativamente, o agente é punido ou encorajado a realizar ações, até atingir seu objetivo, quando terminará o processo (SUTTON; BARTO, 2018).

## 2.4 Q-Learning

O *Q-learning* (WATKINS, 1989) é um método que teve um papel muito importante no desenvolvimento do estado-da-arte do aprendizado por reforço. Este método é utilizado para o aprendizado do agente, em que aprende a reconhecer quais as melhores ações para os estados em que se encontra. Esta situação é compreendida na função  $Q$ , composta pelo par estado-ação( $S,a$ ), chamado de *ação-valor*. A atualização dessa função lhe confere

o caráter evolutivo, onde o agente aprende qual a melhor ação realizar em determinado cenário (SUTTON; BARTO, 2018).

**Figura 5 – Algoritmo *Q-Learning***

**Q-learning (off-policy TD control) for estimating  $\pi \approx \pi_*$**

```

Algorithm parameters: step size  $\alpha \in (0, 1]$ , small  $\varepsilon > 0$ 
Initialize  $Q(s, a)$ , for all  $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$ , arbitrarily except that  $Q(\text{terminal}, \cdot) = 0$ 
Loop for each episode:
    Initialize  $S$ 
    Loop for each step of episode:
        Choose  $A$  from  $S$  using policy derived from  $Q$  (e.g.,  $\varepsilon$ -greedy)
        Take action  $A$ , observe  $R, S'$ 
         $Q(S, A) \leftarrow Q(S, A) + \alpha[R + \gamma \max_a Q(S', a) - Q(S, A)]$ 
         $S \leftarrow S'$ 
    until  $S$  is terminal

```

Fonte: (SUTTON; BARTO, 2018)

A Figura 5 apresenta o algoritmo *Q-Learning*, em que o método tem como base a inicialização com valores aleatórios de todos os possíveis estados do ambiente e as possíveis ações para cada um desses estados. Portanto, têm-se uma explosão combinatória da combinação entre os estados que podem ser alcançados e as ações que podem ser feitas. Os valores que são utilizados no par estado-ação indica o quanto bom é realizar uma ação em um determinado estado. Quando o valor do par é nulo, ou seja, quando o agente encontra-se em um estado terminal, é que o agente atingiu ou falhou em seu objetivo, terminando o algoritmo.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \times \max_a [Q(S_{t+1}, a)] - Q(S_t, A_t)]. \quad (2.1)$$

A atualização da função  $Q$  é realizada pela Equação 2.1, em que  $R$  é a recompensa recebida pela ação em um estado  $S$ , a variável  $\alpha$ , que é a taxa de aprendizado, irá determinar o passo da atualização da função  $Q$  e a variável  $\gamma$  é a responsável por determinar qual a importância das recompensas futuras - mais próxima de 0, maior a importância da recompensa imediata e próxima de 1, maior a recompensa futura. A função  $\max_a$  busca pela melhor recompensa no próximo estado. Um dos fatores marcantes do *Q-learning* é o uso da política que divide-se em explorar (*exploration*) e tirar vantagem do aprendizado (*exploitation*), que é caracterizado pela variável  $\epsilon$  com valor  $0 < \epsilon \leq 1$ . O algoritmo *Q-learning* utiliza dessa para decidir qual o tipo de ação deverá realizar, sendo que este valor tende a diminuir com o tempo, indicando que o algoritmo deixa de realizar ações aleatórias e de exploração, para utilizar os conhecimentos já adquiridos (MARQUES, 2019).

## 2.5 Algoritmo DQN (Deep Q-Network)

Em ambientes em que os estados são contínuos e infinitos, o algoritmo *Q-Learning* é insuficiente, já que é preciso guardar uma grande quantidade de relações para cada estado. Logo, a ideia do algoritmo DQN é utilizar as redes neurais profundas para aproximar a função ação-valor do melhor valor possível, permitindo que mesmo que todos os estados não sejam visitados, seja possível identificar qual a ação mais adequada, com base na aproximação da função (RAVICHANDIRAN, 2018; MNIIH et al., 2015). Este algoritmo foi proposto por Mnih et al. (2015) e é apresentado na Figura 6.

**Figura 6 – Algoritmo DQN**

**Algorithm 1: deep Q-learning with experience replay.**

Initialize replay memory  $D$  to capacity  $N$

Initialize action-value function  $Q$  with random weights  $\theta$

Initialize target action-value function  $\hat{Q}$  with weights  $\theta^- = \theta$

**For** episode = 1,  $M$  **do**

    Initialize sequence  $s_1 = \{x_1\}$  and preprocessed sequence  $\phi_1 = \phi(s_1)$

**For**  $t = 1, T$  **do**

        With probability  $\varepsilon$  select a random action  $a_t$

        otherwise select  $a_t = \text{argmax}_a Q(\phi(s_t), a; \theta)$

        Execute action  $a_t$  in emulator and observe reward  $r_t$  and image  $x_{t+1}$

        Set  $s_{t+1} = s_t, a_t, x_{t+1}$  and preprocess  $\phi_{t+1} = \phi(s_{t+1})$

        Store transition  $(\phi_t, a_t, r_t, \phi_{t+1})$  in  $D$

        Sample random minibatch of transitions  $(\phi_j, a_j, r_j, \phi_{j+1})$  from  $D$

        Set  $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$

        Perform a gradient descent step on  $(y_j - Q(\phi_j, a_j; \theta))^2$  with respect to the network parameters  $\theta$

        Every  $C$  steps reset  $\hat{Q} = Q$

**End For**

**End For**

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

Os algoritmos que utilizam a atualização do *Q-learning* utilizando redes profundas para aproximações não-lineares da função ação-valor, em geral, podem possuir problemas de instabilidade, não-convergência e correlações causadas pelas atualizações e sequência de observações. Portanto, os autores de Mnih et al. (2015) propuseram a utilização de duas características no algoritmo DQN para resolver esses problemas.

A primeira característica é a utilização de memórias de experiências passadas (*replay experience*), que altera a sequência dos dados obtidos de uma iteração - removendo a correlação e suavizando da distribuição dos dados. A experiência é formada pela tupla  $(S_t; a_t; R_t; S_{t+1})$ , sendo, respectivamente, o estado atual, a recompensa recebida, a ação realizada e o próximo estado da iteração.

A segunda é a atualização iterativa que altera a equação  $Q$ , usando um alvo para atualizar, periodicamente, ao invés de a cada iteração. A segunda iteração traduz-se, na prática, ao utilizar um alvo na forma de uma rede neural auxiliar que é treinada em um momento específico e seus pesos são repassados à rede principal.

Assim, foi possível verificar um melhor desempenho do algoritmo utilizando essas características.

## 2.6 Simulador Carla

O ambiente de simulação Carla (*Car Learning to Act*) é um simulador gratuito de código e materiais (e.g. veículos e prédios) abertos para o desenvolvimento e pesquisa de veículos autônomos. Tem em seu núcleo o motor gráfico *Unreal Engine 4*, o que permite maior flexibilidade para a comunidade desenvolver novas funções, materiais, entre outros.

O funcionamento do Carla é feito por meio da arquitetura de cliente e servidor, em que o cliente comunica com o servidor por meio *sockets* e uma biblioteca de interface de programação de aplicação (*API*) na linguagem Python.

O cliente utiliza de *scripts* para controlar os atores (veículos, pedestres, sensores, etc.) comunicando com o servidor quais ações estão sendo realizadas. O servidor recebe essas informações e retorna, por meio dos sensores, os dados relativos aos seus estados. Por exemplo, o sensor de câmera pode ser acoplado a um ator, como um veículo, que se desloca pela cidade e o sensor irá retornar imagens da sua visão no simulador e retornar ao cliente, que poderá processar os dados recebidos.

Em seu ambiente dinâmico, é possível simular cidades com pedestres e veículos, gerenciar o clima e horário, etc. O foco do simulador é realizar o controle de ambiente, implementar e controlar um veículo específico, acoplar sensores neste e recuperar seus dados, para serem livremente processados. Portanto, esse simulador permite ao desenvolvedor a implementação de várias funções e permite um grande controle do ambiente, que é tão necessário no controle e desenvolvimento de um modelo de aprendizado por reforço (PALANISAMY, 2018; DOSOVITSKIY et al., 2017).

**Figura 7 – Exemplo de Simulação do Carla**



Fonte: Elaborada pelo autor.

A Figura 7 apresenta uma execução do Carla utilizando o *script* de controle manual A, disponibilizada pela documentação do Carla, de um veículo como cliente, no canto da tela inferior direito; A execução do servidor com exibição, na cidade *Town03*, a qual é exibida na tela superior da imagem; e no canto inferior esquerdo têm-se os terminais do Linux fazendo a chamada aos *scripts* e iniciando o servidor.

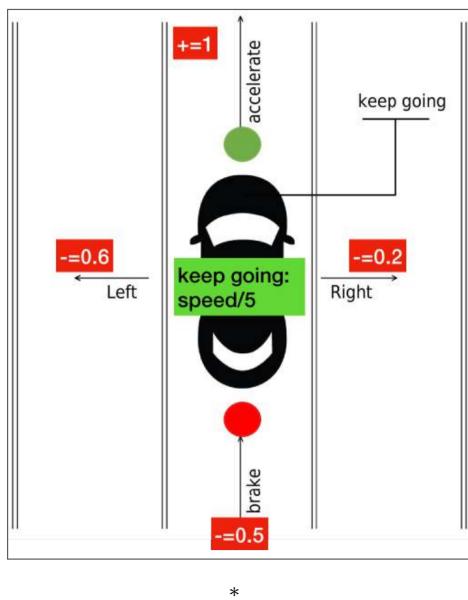
## 2.7 Trabalhos Correlatos

Os autores Dosovitskiy et al. (2017) apresentam um artigo relativo ao uso de diferentes abordagens utilizando o simulador Carla, além de introduzirem e explicarem as informações relativas ao uso do simulador e suas características. No artigo são comparados três técnicas para diferentes tarefas orientadas um objetivo de destino, sendo as técnicas *modular pipeline*, que realiza um fluxo de identificação, identificação de características e controle do veículo baseados em regras, *imitation learning* que usa controles de alto nível com entradas de percepção, e usando RL, com o algoritmo A3C. A análise feita pelos autores mostram que técnica de *imitation learning*, em relação ao modular pipeline possui poucas diferenças na pontuação entre as tarefas, de forma geral. Entre o *imitation learning* e o RL, teve-se grande diferença, sendo o RL tendo um desempenho muito inferior, e é apresentado que o RL tem grande dificuldade em encontrar bons parâmetros de otimização de decisão, pois realiza uma grande exploração deste parâmetros, devido

a complexidade do simulador. Pelo artigo é possível perceber a grande capacidade de customização e possibilidades de realização e comparação de experimentos.

Os autores de Fayjie et al. (2018) apresentam a construção de um carro autônomo em um simulador criado com a ferramenta Unity e utilizam o algoritmo DQN para realizar aprendizado de percepção e movimentação do carro. As recompensas definidas são atribuídas ao deslocamento do veículo, beneficiando a movimentação do veículo em linha reta e manter-se em movimento, enquanto as demais recompensas punem o agente (ir para os lados ou freiar). A Figura 8 apresenta as recompensas da movimentação do veículo. O carro utiliza sensores de câmera e LIDAR para a alimentação e funcionamento da DQN. De acordo com os autores, o algoritmo de DQN se mostrou capaz de realizar um comportamento adequado para o ambiente e desempenhar com sucesso o aprendizado de um carro autônomo em um ambiente simulado.

**Figura 8 – Esquema de recompensas**



O trabalho realizado por Guckiran e Bolat (2019) consiste no aprendizado de um carro autônomo no ambiente simulado *Open Racing Car Simulation* (TORCS) e utiliza uma variação do algoritmo DQN, o *Rainbow DQN*. Os autores utilizam uma função de recompensa baseada no posicionamento do veículo e métricas que observam se o veículo está reagindo como esperado. O posicionamento do veículo é utilizado para gerar equações que identificam a posição na pista do veículo e seu comportamento. Ao fim do experimento, o veículo conseguiu, com sucesso, ser capaz de perceber e se orientar em uma pista de corrida simulada.

O artigo apresentado por Bojarski et al. (2016) utiliza um algoritmo de aprendizado supervisionado, que consiste em utilizar dados pré-processados e rotulados, contendo imagens de vídeo de um veículo em movimento com os rótulos sendo o ângulo do volante, nível de intensidade do acelerador e freio. Os autores utilizaram as estruturas de redes neurais convolucionais para processamento dos dados e reconhecimento dos padrões, permitindo que o modelo possa incorporar qual ação realizar para uma determinada entrada. Foram utilizadas horas de filmagens e dados coletados para o aprendizado do veículo, mas este conseguiu realizar seu aprendizado e circular em vias públicas sem o auxílio de um motorista.

Em (Jaritz et al., 2018) foi desenvolvido um carro autônomo para percorrer as pistas

de rali do jogo *World Rally Championship 6 (WRC6)* e utiliza o algoritmo A3C, *Asynchronous Advantage Actor-Critic*, que foi desenvolvido por Mnih et al. (2016). O trabalho teve como objetivo criar um veículo que seja capaz de percorrer de pistas de rali, realizando um controle lateral e longitudinal utilizando apenas a imagem de uma câmera na parte frontal do carro e sua velocidade. Um dos objetivos dos autores era que o carro aprendesse a utilizar técnicas de *drift* sempre que possível, por meio do uso do freio de mão e para guiar o veículo em seu aprendizado o agente foi recompensado pela distância que está do centro da faixa e, se está fora da faixa, também há punição. De acordo com os autores, utilizar a inicialização aleatória mostrou melhores resultados em generalizar a capacidade de atuação do agente. Uma das particularidades do artigo foi o uso de imagens de vídeos reais para testar o aprendizado do veículo, mostrando que é possível extrapolar o conhecimento adquirido em simulação para o mundo real. Entretanto, os resultados foram parciais, já que o veículo não pôde agir, mas apenas informar qual era a ação a ser feita e não realizá-la. Por fim, os autores concluíram que o carro teve bons resultados, conseguindo bons níveis de generalização e capacidade de percorrer em pistas que não havia treinado na simulação.

O artigo feito por Krishnaswami Sreedhar e Shunmugam (2020) teve como objetivo treinar um carro autônomo usando duas técnicas computacionais, sendo a clonagem de comportamento com o uso de aprendizado supervisionado e aprendizado por reforço, com três câmeras frontais e verificar se era possível utilizar computadores de baixo poder computacional para realizar o aprendizado do carro. Para a primeira técnica foi usado o simulador Udacity CarND *self-driving* e os autores afirmam que foi possível treinar o carro para percorrer as pistas do simulador em pistas que não conhecia. Neste primeiro caso, os autores afirmaram que o uso do simulador Carla não seria viável devido ao custo computacional da simulação junto ao treinamento necessário para treinar com máquina disponível. Para a segunda técnica foi usado o simulador Carla e, de acordo com os autores, os resultados não foram tão bons como esperados, com um carro que aprendeu apenas alguns comportamentos básicos e insuficientes para uma condução autônoma, sendo capaz de conduzir em linha reta e realizar poucas curvas. Eles apresentam que a simulação teve forte impacto no custo computacional, além do treinamento. Os autores citam a falta de recursos computacionais para melhorar o treinamento e sugerem também um pré-treinamento para a inteligência.

De acordo com os trabalhos correlatos, é possível notar que simulações são comumente usadas para o treinamento de carros autônomos e as recompensas dadas podem interferir diretamente no comportamento do veículo e a câmera o sensor mais utilizado. Buscar resolver métodos complexos por meios mais simples, como o uso de *end-to-end* é uma das áreas do estado da arte dos veículos autônomos e necessário maior exploração. Por este motivo, é possível perceber que há espaço para explorar as técnicas de aprendizado por reforço.

### 3 METODOLOGIA

A proposta deste trabalho é realizar a implementação da etapa de percepção e movimentação de um veículo autônomo utilizando o simulador Carla, por meio de um algoritmo capaz de se adaptar ao ambiente, controlar o veículo e reduzir a complexidade de controle e percepção de um veículo autônomo. Portanto, serão utilizados os seguintes materiais e as seguintes etapas:

#### 3.1 Materiais

Os materiais que serão utilizados neste trabalho são:

- Simulador Carla versão 0.9.9.4;
- Mapas do simulador *Town03* e *Town05*;
- Computador 1 (PC1) - Dell Vostro 5481 i7-8556U, 8GB RAM, GPU NVIDIA 130MX;
- Computador 2 (PC2) - Samsung RF511 i5-2450M, 8GB RAM, GPU NVIDIA 730M;
- Linguagem de programação Python;
- Biblioteca Python para inteligência artificial *Tensorflow* com o *framework* Keras;
- Sistema Operacional Linux Mint 20.

#### 3.2 Estudo do contexto do Simulador Carla e ferramentas

Para a realização das demais etapas deste trabalho se faz necessário o estudo da documentação do Simulador Carla, verificando o seu funcionamento e disponibilidade de códigos pronto, e das ferramentas que foram apresentadas na Seção 3.1, visando a sua praticidade, aplicação e disponibilidade de recursos.

#### 3.3 Desenvolvimento do ambiente

Nesta etapa, serão implementados o ambiente para a comunicação entre o veículo e o simulador e da estrutura para o algoritmo de aprendizado por reforço.

O ambiente irá permitir o treinamento em diversas sessões, possibilitando o funcionamento da estrutura para o algoritmo de aprendizado por reforço. Esta estrutura será formada por um processo de captura de dados pelos sensores, processamento (adaptação do tamanho, formatação ou processos similares), atuação do agente (veículo) e atribuição de uma recompensa que serão definidas de forma a permitir uma movimentação que priorize a não-colisão com obstáculos.

#### 3.4 Implementação da etapa de percepção e movimentação do veículo

Serão implementados os sensores e atuadores necessários para que o veículo possa perceber o ambiente à sua volta e reagir. Sendo o sensor de percepção uma câmera e um sensor de colisão (este é um subproduto do simulador Carla). Os atuadores são os controles do veículo, que são realizados por comandos no código do cliente.

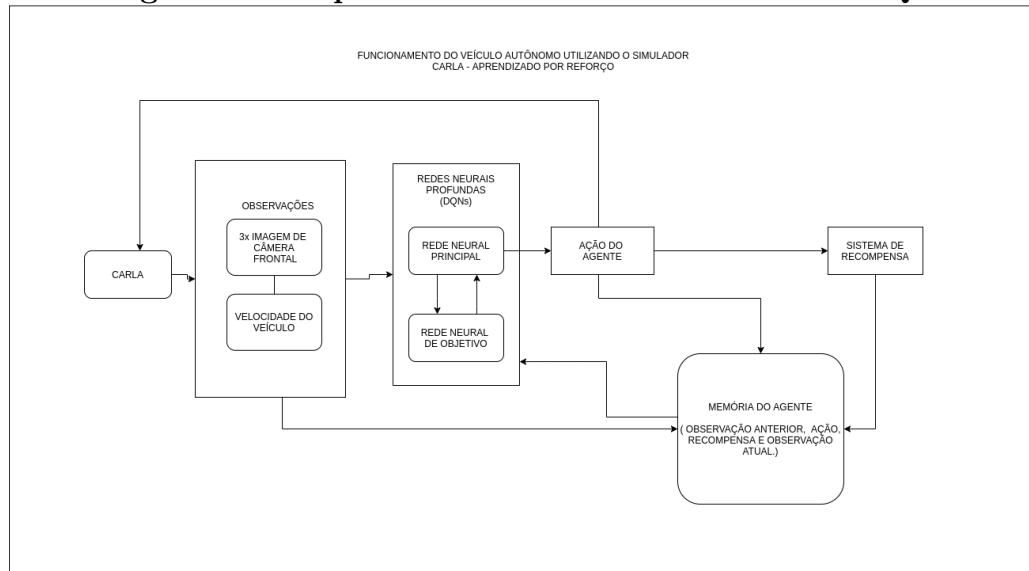
A movimentação do veículo consiste em realizar as ações locais de mover se para frente, virar o volante, ir para trás e ficar parado, de acordo com a necessidade determinada pela DQN para que o veículo possa se movimentar sem colidir com obstáculos.

O controle lateral e longitudinal do veículo será feito com base nas informações extraídas de seus sensores e deverá realizar a inferência de escolha, para o melhor controle possível, por meio do processamento de padrões, que é feito pela DQN.

### 3.5 Implementação do algoritmo DQN

Será implementado o algoritmo de aprendizado por reforço DQN, para realizar o aprendizado do veículo, junto aos sensores e atuadores, formando a camada de percepção e movimentação do veículo autônomo. A Figura 9 apresenta o esquema no qual será baseado a implementação dos algoritmos e a etapa de percepção e movimentação.

**Figura 9 – Arquitetura do ambiente Carla com DQN**



Fonte: Elaborada pelo autor.

### 3.6 Experimentos, testes e correções

Para um funcionamento adequado do ambiente e do veículo, serão realizados experimentos e testes para verificar o correto funcionamento e se as implementações são apropriadas para atingir o objetivo de movimentação de um veículo autônomo, que foram previamente estabelecidos, e que serão avaliadas de acordo com as métricas estabelecidas na Seção 3.9. No caso de inconsistências ou falhas durante os testes com o ambiente, com o algoritmo de aprendizado ou com o veículo, estes serão corrigidos.

### 3.7 Treinamento do agente

Nesta etapa será realizado o treinamento do agente (carro-ego) utilizando os produtos das etapas anteriores, com o objetivo de ensinar e verificar se o veículo é capaz de se movimentar sem colidir com obstáculos. O treino é baseado em repetição das ações no ambiente simulado Carla.

Para ser possível realizar a simulação, serão utilizadas duas máquinas, PC1 e PC2, as quais terão como tarefas serem o cliente e servidor, respectivamente. A comunicação entre estas será feita via protocolo TCP. O PC1 ficará responsável pelo processamento e controle referentes ao carro autônomo e tarefas do cliente. O PC2 será utilizado para disponibilizar o servidor com o ambiente simulador Carla e demais funções auxiliares, que sejam necessárias.

Será realizado o treinamento com 200 episódios, sendo possível que o veículo realize até 501 ações, por episódio. Neste treinamento, será utilizada o mapa *Town05*. Durante o treino, um segundo *script* será executado pelo PC2. Esse terá a função de inserir outros veículos e pedestres (ambos são considerados como atores, pelo Carla) no cenário e realizar o seu gerenciamento. Os atores não possuem um controle inteligente, mas sim um controle pré-definido pelo próprio simulador, tais como destino, ações, etc. Para explorar uma melhor generalização do agente, os locais iniciais do carro-ego, de cada episódio, serão aleatórios.

### **3.8 Validação do agente**

Para validar e avaliar o treinamento será feita uma avaliação final com 100 episódios, sem um limite de iterações, no mapa *Town03*. Serão usadas as métricas da Seção 3.9, assim como são usadas para o treinamento.

### **3.9 Métricas de avaliação**

Nesta seção serão apresentadas as métricas de avaliação, as quais serão usadas para verificar o comportamento do veículo, desde as etapas de experimentos até a etapa de avaliação final do veículo. Estas métricas seguirão as diretrivas de avaliar o comportamento, por meio das ações feitas, e recompensas recebidas pelo veículo, ao longo dos experimentos, treino e avaliação. As métricas são apresentadas na lista a seguir:

1. Medir o progresso da recompensa recebida pelo agente ao longo do treinamento e avaliação;
2. Medir a quantidade de passos, sem colisões, ao longo do treinamento e avaliação do veículo;
3. Medir a quantidade de colisões ao longo do treinamento e avaliação do veículo;
4. Medir a quantidade de invasões de faixas ao longo do treinamento e avaliação do veículo;
5. Medir a distância percorrida do veículo, sem colisões, no decorrer do treinamento e avaliação;
6. Avaliar qualitativamente a movimentação do veículo no mapa da cidade, com base na quantidade de ações e movimentação geral;

## 4 PROPOSTA DESENVOLVIDA

Neste capítulo são apresentadas as seções relativas ao desenvolvimento deste trabalho.

### 4.1 Desenvolvimento do Código DQN

O código desenvolvido para a implementação do veículo autônomo se encontra no Apêndice A. Na Tabela 1 são apresentados os parâmetros para a configuração do algoritmo DQN. A escolha dos valores foram escolhidos ou ajustados de forma arbitrária, tendo como base os valores inciais utilizados por Mnih et al. (2015).

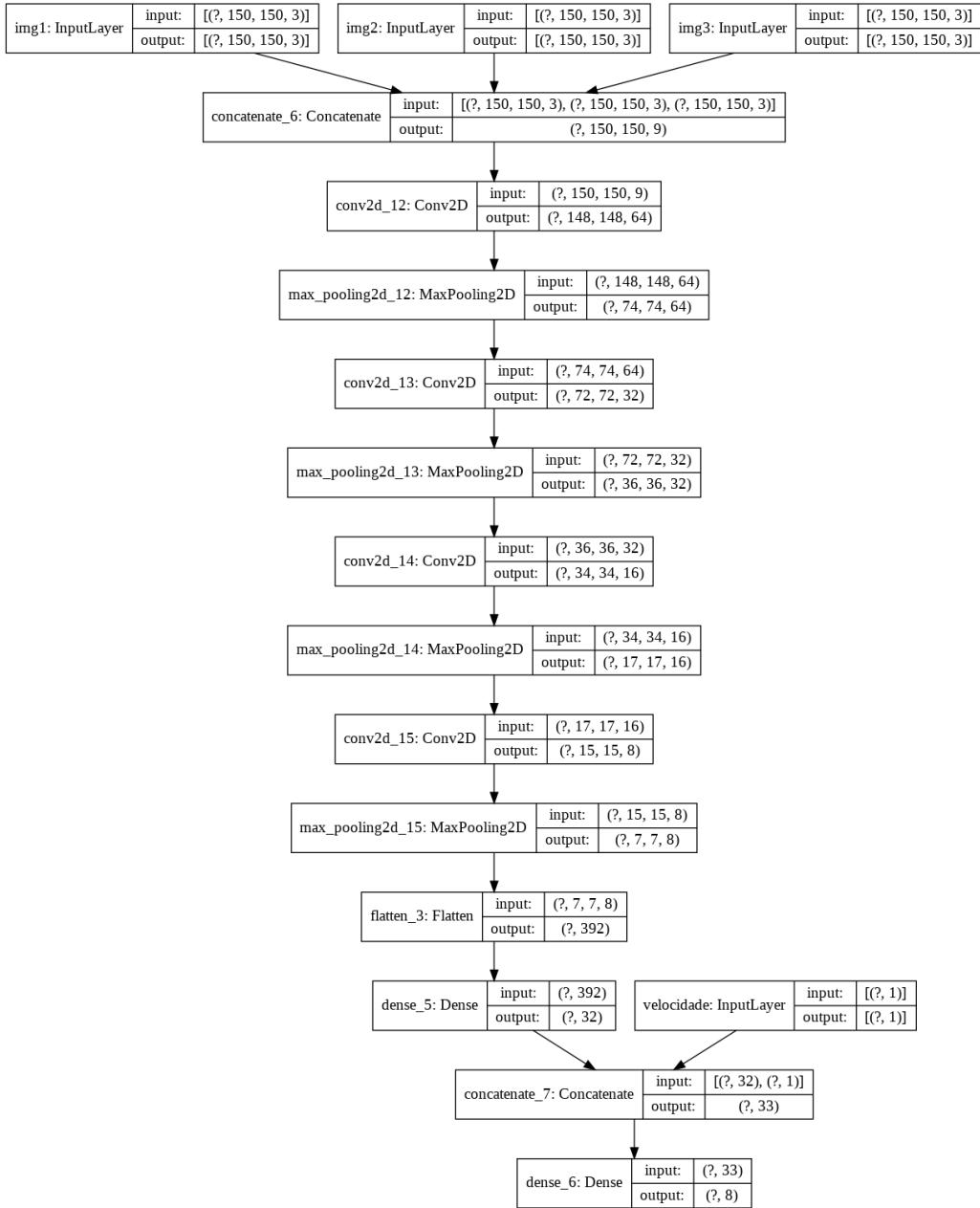
**Tabela 1 – Variáveis utilizadas no código**

| variável                 | função                                      | valor     |
|--------------------------|---|-----------|
| $\alpha$                 | taxa de aprendizado                         | 0.001     |
| $\gamma$                 | desconto de recompensa                      | 0.9       |
| passos p/ treinamento    | da rede principal                           | 250       |
| passos p/ cópia de pesos | da rede principal para a alvo               | 250       |
| qtd. episódios           | para o fim do treino                        | 200       |
| buffer de memória        | estados e transições armazenados            | 10000     |
| $\epsilon$               | probabilidade de usar uma ação aleatória    | [1 - 0.1] |
| decaimento $\epsilon$    | redução do $\epsilon$                       | 50000     |
| <i>batch_size</i>        | quantidade de lembranças usadas para treino | 32        |

**Fonte:** Elaborada pelo autor.

Para o algoritmo DQN, foram utilizados duas redes neurais, sendo a primeira a rede principal e a segunda a rede alvo (ambas possuem a mesma configuração). Para o processamento da rede, foram utilizados intervalos de 250 passos para o treino da rede principal, 250 passos para a cópia dos pesos da rede principal para a rede alvo e 500 passos para a quantidade de passos mínimos para iniciar o treino. Os passos dos intervalos são referentes à quantidade de passos totais do treinamento entre os episódios. Para cada episódio é permitido que o agente realize até 501 ações (passos). Ao longo dos passos (totais), o valor do  $\epsilon$  diminui, fazendo com que as ações do agente deixem de ser aleatórias e passem a ser mais provável ações baseadas no conhecimento obtido. Para o treinamento foram utilizadas um conjunto de memórias (*buffer*) com as últimas 10.000 observações feitas e que são selecionadas um *batch* de 32 imagens para treinar a rede principal e posteriormente serem copiadas para a rede alvo. O algoritmo utiliza uma taxa de aprendizado ( $\alpha$ ) lenta de 0,001, o que causa uma convergência em relação ao ótimo global mais lenta e focando em recompensas futuras com um desconto de recompensa ( $\gamma$ ) alto de 0.9. O algoritmo de DQN utilizará a rede neural profunda, baseada na rede apresentada por (RAVICHANDIRAN, 2018), com as características apresentadas na Figura 10 para a rede principal e alvo. Além de modificações nas camadas e parâmetros da rede, foram utilizados como entradas os dados de imagem e velocidade do veículo. Estes dados são recebidos em momentos diferentes, sendo a imagem recebida no início e processada, para, em seguida, receber o valor de velocidade. O efeito esperado é que a rede possa inferir o conceito de velocidade relacionando com a percepção de movimento.

**Figura 10 – Rede Neural da DQN**



**Fonte:** Elaborada pelo autor.

## 4.2 Classes de Ações

Para realizar o controle do veículo, existem 8 classes de ações que podem ser realizadas pelo veículo. Sendo elas:

- sem ação;
- frente;
- frente-direita;
- frente-esquerda;
- freio;

- ré;
- ré-direita;
- e ré-esquerda.

Cada ação representa um conjunto de controles que o veículo realiza, com os nomes representando o que é feito, ex.: ré-esquerda: o veículo dá ré e vira o volante para a esquerda por um *frame*. Ao nível de programação, o que é feito é passar para a API do Carla quais os comandos do veículo devem ser usados e em qual intensidade, caso este seja necessário. A intensidade padrão é de 0,5, tanto para a aceleração, quanto para o volante. Para a simplificação da implementação, foram utilizados controles discretos, e, por consequência, com valores fixos.

### 4.3 Sensores

Foi utilizado um sensor de câmera, que é posicionado na parte frontal superior do veículo e esta possui sistema de cores RGB com resolução  $150 \times 150$  pixels. Para integrar a observação do ambiente foram recebidas três imagens consecutivas, assim como é utilizado por Mnih et al. (2015), para emular a percepção de movimento. Também foram utilizados sensores virtuais (que são exclusivos da simulação Carla), como um sensor de velocidade, um sensor de colisão com objetos e um sensor de detecção de faixas de rolamento. O primeiro é feito recebendo as informações do agente e processando-as para extrair a velocidade, como um velocímetro. O segundo detecta a colisão com outros objetos do mapa, como veículos, pedestres, calçada, etc. O sensor de detecção de faixas, permite identificar as faixas de rolamento que mais estão próximas ao veículo e, como o agente sempre inicia em uma faixa no sentido correto e ao centro, é possível utilizar estes sensores para detectar as faixas laterais e, por consequência, o centro da faixa, que é utilizado para recompensar o veículo.

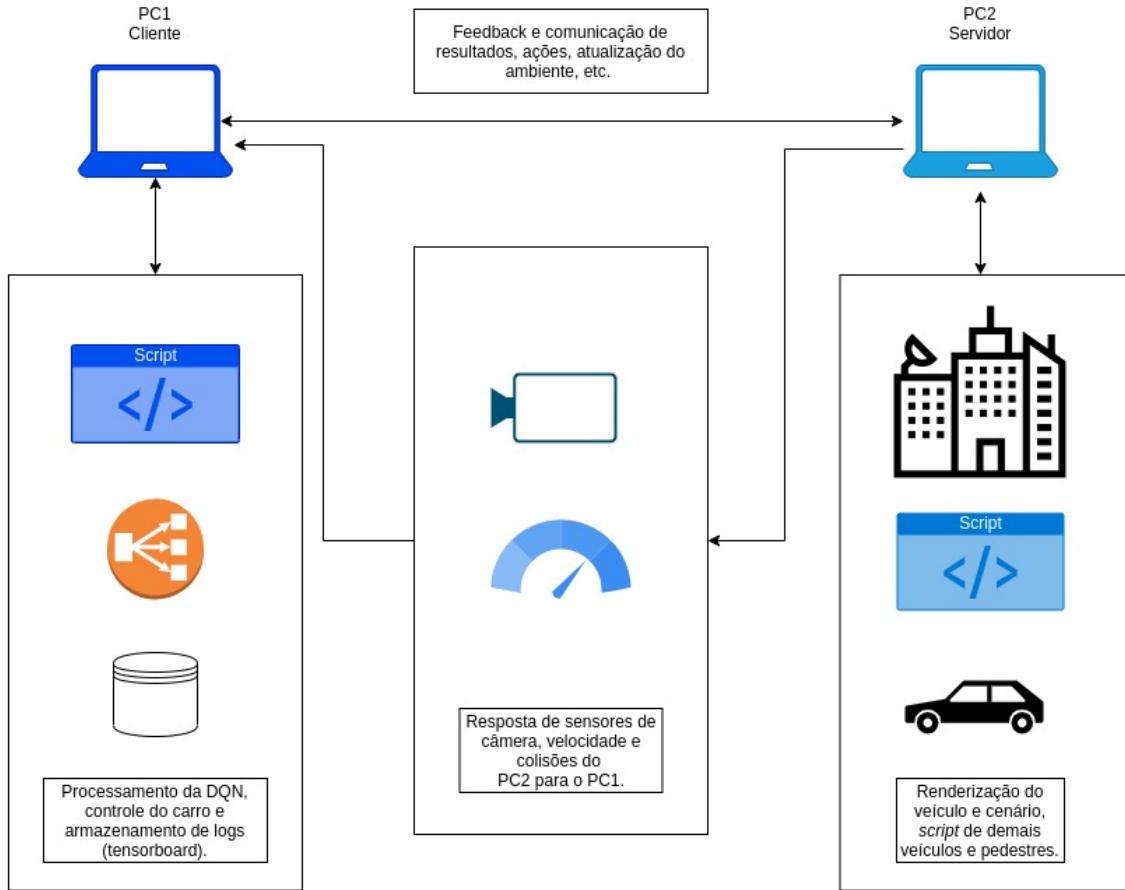
### 4.4 Desenvolvimento do ambiente e utilização de mapas

A união entre o código que interage com a API do simulador Carla, realizando toda a comunicação do cliente com o servidor, enviando comandos para o veículo e recuperando informações de observação do ambiente, por exemplo, e o código com o fluxo da DQN, criando assim um ambiente de aprendizado por reforço, baseado na estrutura de código aberto da (AI, 2020), foram feitas por um *script* primário. Um *script* secundário foi utilizado para a seleção de mapas entre os mapas *Town05* e *Town03* (sendo ambos utilizados para experimentos, mas o primeiro foi usado para o treinamento e o segundo avaliação) e integrando novos atores no cenário, como outros veículos e pedestres. O código para este *script* é apresentado no Apêndice A e são inseridos 50 veículos e 100 pedestres. Estes atores possuem um comportamento fixo, programados para se deslocarem pelo ambiente de forma rígida e de forma pré-definida, ou seja, codificados sem inteligência artificial.

A simulação está rodando a 20 FPS (*frames* por segundo) e a cada *tick* (atualização de frame) serão simulados os 20 FPS. Essa condição permite um controle mais rígido da simulação, o que garante que os dados dos sensores não terão problemas ao serem gerados e armazenados; permite que o processamento da DQN seja feito com a simulação parada, ou seja, todos os processos podem ser feitos, sem o impacto da simulação rodando e interferindo nos dados. A simulação foi ajustado para que simulasse com qualidade baixa, causando menor impacto no processamento do PC2 e deixasse o servidor mais fluído.

A Figura 11 apresenta uma visão geral da comunicação e funções de cada uma das máquinas utilizadas no desenvolvimento da proposta.

**Figura 11 – Funções das máquinas PC1 e PC2**



Fonte: Elaborada pelo autor.

#### 4.5 Experimentos

Realizadas as primeiras simulações, foi possível utilizar o simulador Carla por meio *scripts* de funcionamento e interação do ambiente com o veículo simulado, como pode ser visto na Figura 12.

**Figura 12 – Experimentos com o simulador Carla**



Fonte: Elaborada pelo autor.

Os primeiros contatos com a simulação foram realizados utilizando *scripts* prontos disponibilizados pela documentação do simulador, que podem ser editados conforme o necessário para o desenvolvedor. Estes experimentos permitiram visualizar melhor o am-

biente e o tipo de estrutura que o Carla utiliza em seus códigos e interações.

Para os primeiros experimentos, até a criação de uma versão inicial de ambiente integrando o código de controle do veículo e DQN, foi inicializada a cidade principal disponível no simulador Carla e *scripts* secundários foram utilizados para inserir veículos e pedestres no ambiente, que circularam pela cidade, mas não se preocupando com destino ou colisões, já que são disponibilizados com uma programação sem inteligência pelo simulador. Também foram utilizados *scripts* para controlar um veículo pela cidade, podendo ser inseridos comandos pelas teclas do teclado para movimentá-lo ou utilizar linhas de código para informar a movimentação. Neste último *script* foi possível visualizar como são utilizados e implementados os sensores, possibilitando a visão mais clara do funcionamento sensorial do veículo. Este primeiro momento permitiu analisar o funcionamento básico do simulador e avaliar como poderia ser implementados os veículos e outros atores, sensores e como interagir com o ambiente em conjunto com o algoritmo DQN. Com estes experimentos iniciais, foi possível gerar a arquitetura apresentada na Figura 9, que comporta o funcionamento geral do algoritmo em integração com o simulador Carla.

Após gerados os primeiros *scripts* contendo o carro-ego implementado no cenário e toda a estrutura e fluxo das informações para o treinamento da DQN, foram realizados os experimentos que permitiram descobrir falhas no código e realizar alterações na recompensas, que são apresentadas na sua versão final na seção 4.6 e foram utilizadas no treinamento final do carro. Dentre os diferentes experimentos realizados, são apresentados os resultados dos mais relevantes na seção 5.1.

#### **4.6 Função de recompensa**

O intuito do uso da função de recompensas é ensinar o veículo quais ações devem ser reforçadas e quais desencorajadas. Portanto, na lista a seguir são apresentadas as recompensas propostas que foram elaboradas e definidas ao fim dos experimentos e que foram utilizadas para o treinamento final do agente. Estas foram definidas, respectivamente, em +1, -1 ou ausência de valor de recompensa e tiveram o intuito de priorizar a movimentação livre, realizando um controle lateral e longitudinal do veículo pelas cidades do simulador, evitando colisões, mas sem se preocupar com todas as normas e regras de trânsito.

1. Recompensa por velocidade:

- (a) Recompensa de +1, para velocidades maiores que 3 km/h e menores que 30 km/h;
- (b) Não há recompensa por estar com uma velocidade entre 0 e 3 km/h;
- (c) Recompensa de -1 enquanto permanecer em velocidades maiores que 30km/h;

2. Recompensas por distância do centro da faixa central:

- (a) Recompensa de +1 por estar a menos de 0,4 metros do centro das faixas da via do seu sentido, a cada passo;
- (b) Recompensa de -1 por estar a mais de 0,4 metros e menos de 2 metros do centro da faixa;
- (c) Recompensa de -1 por estar a mais de 2 metros do centro da faixa, com o fim do episódio (como em caso de invasão de faixa contínua).

3. Recompensa de -1 para invasão de faixa contínua;

4. Recompensa de -1 ao andar mais que 3 metros de ré.
5. Recompensa de -1 para colisões com outros objetos ou atores do mapa;
6. Recompensa de -1 a cada fim de episódio;

As recompensas do veículo foram definidas com o intuito de desencorajar ultrapassar o limite de velocidade arbitrário de 30 km/h; forçar o veículo a agir, tendo como punição ficar parado ou não tomar ações que receba recompensa. Como ficar imóvel, pisando no freio, por exemplo.

## 5 ANÁLISE E DISCUSSÃO DOS RESULTADOS

Nesta capítulo são apresentados os resultados obtidos durante os experimentos, treinamento e testes.

### 5.1 Experimentos

Realizada a experimentação de diferentes abordagens, de forma a explorar as possibilidades de aprendizado e comportamento com diferentes características e alterações de recompensas que permitiram entender como veículo se comporta ao receber incentivos e punições diferentes. Além disso, as experimentações também possibilitaram, de forma geral, encontrar erros no código e nos parâmetros utilizados da DQN. Nesta seção são abordados alguns dos experimentos realizados e suas implicações nos resultados do trabalho.

Ao longo dos experimentos ocorreram modificações no valor das recompensas, em que, em um primeiro momento possuíam valor decimal fixo, de acordo com a situação do veículo. Posteriormente, foi utilizada uma função matemática, tentando explorar melhor o espaço de recompensa. Entretanto, os valores se tornavam de difícil análise e imprecisos. Tentando contornar esta situação, utilizou-se valores inteiros para os experimentos seguintes, até finalmente serem utilizados, no treinamento e avaliação, os valores normalizados, por meio do *clipping*, que consiste em tornar a recompensa em um valor fixo de +1 ou -1. Após estas modificações, tornou-se mais produtivo a análise do valor das recompensas e de melhor visualização do progresso do agente.

Logo no início dos experimentos, após a versão integrando a DQN com veículo, foram alcançados resultados com o veículo que não eram os esperados, tendo em vista que o que se esperava era a sua movimentação contínua no trânsito. O veículo comportou-se de forma inesperado, realizando uma movimentação simplória, apenas indo para trás e para frente e movimentando o volante para ambos os lados, sem sair do lugar. Assim, foi revisto a função de recompensa que incentivava o veículo a manter-se com uma velocidade limitada, superior a zero e punindo-o ao colidir-se, enquanto recebia como informação do ambiente apenas uma imagem da câmera frontal do veículo. Como tentativa de forçar o veículo a agir, ao invés de permanecer parado, foi adicionada uma punição ao fim de cada episódio.

Tentando melhorar o entendimento do agente sobre o seu arredor, foram modificadas as informações de observação que recebe do ambiente (visão e percepção), ou seja, os dados que alimentam a sua rede neural, passando a receber três imagens sequenciais da câmera frontal e a velocidade do veículo. As imagens sequenciais são usadas para que o agente possa perceber e entender o fluxo do ambiente, dando a ele a percepção de movimento.

Em outro experimento, ainda utilizando as recompensas com valores inteiros fixos não-normalizados, o agente descobriu que poderia ganhar recompensas movimentando-se apenas de ré. Portanto, para desencorajar, mas sem reprimir esse comportamento, foi definida uma recompensa que, caso a distância percorrida de ré for maior que 5 metros, o agente era punido com uma recompensa de -10 e o episódio terminava, posteriormente a recompensa passou a ter valor -1.

Em seguida, o agente começou a evoluir, percorrendo maiores distâncias e recebendo mais recompensas. Entretanto, foi observado movimentações ilegais e irregulares, que seriam as invasões de faixa contínua da via. Ou seja, o veículo estava entrando na contramão em diferentes ocasiões. Para evitar tal comportamento, foram definidas duas novas recompensas. Sendo a primeira recompensando em -10 a invasão da faixa contínua, com

o fim do episódio. A segunda recompensa com valor de -1, para cada troca de faixa. Esta última recompensa foi pensada para ensinar o veículo a manter-se no centro da faixa (controle lateral). Posteriormente estas recompensas também foram normalizadas para -1.

Porém, com estas novas modificações, o veículo criou um novo comportamento de manter-se imóvel, sem ação, ainda que haja a punição ao fim de cada episódio. Vale ressaltar que também foi experimentado o aprendizado sem a recompensa de punição ao fim do episódio e o veículo tendia a manter-se imóvel, realizando ações do tipo "sem-ação". Neste caso, é provável o agente entender que, ao manter-se com velocidade inferior a 3km/h, não seria punido e poderia ganhar velocidade o suficiente para cobrir a punição ao fim do episódio e não ser punidos por outros erros.

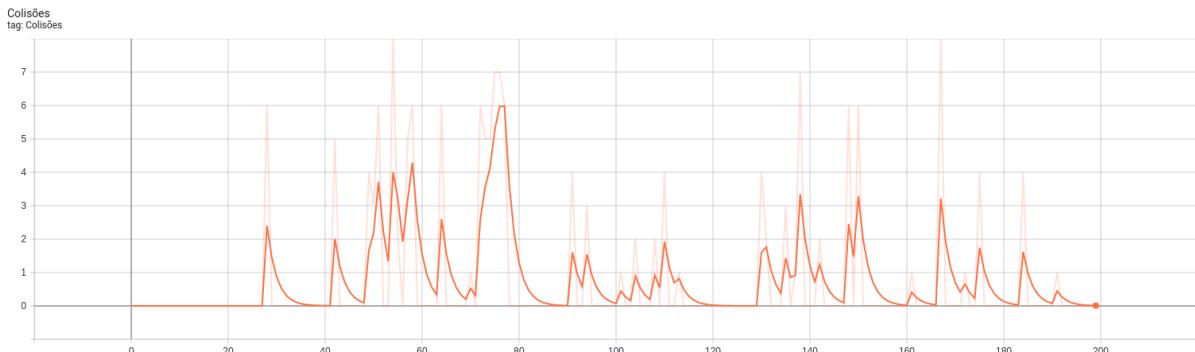
Após diferentes experimentos, foi possível ter uma melhor noção do que precisava ser analisado e os ajustes foram feitos para que o veículo fosse encorajado a realizar um controle lateral e longitudinal para alcançar o objetivo por meio do treinamento.

## 5.2 Treinamento

Após a realização do treinamento principal foram gerados os gráficos que são apresentados nas figuras abaixo e que demonstram o desenvolvimento do veículo.

A análise da Figura 13 permite verificar qual foi a quantidade de colisões causados pelo veículo, sendo estas causadas por subir na calçada, colidir com outros veículos, pedestres e objetos do cenário. Assim, é possível verificar que há uma tendência da redução de colisões ao longo dos episódios de treinamento, como pode-se melhor evidenciar ao verificar a Figura 14, por meio da suavização da curva da quantidade de casos de colisão. Entretanto, a tendência se aproxima de uma reta com pequeno grau de inclinação e, teoricamente, seria necessário uma grande quantidade de episódio de treinamento para aproximá-la de zero.

**Figura 13 – Treinamento - Quantidade de Colisões por Episódio**

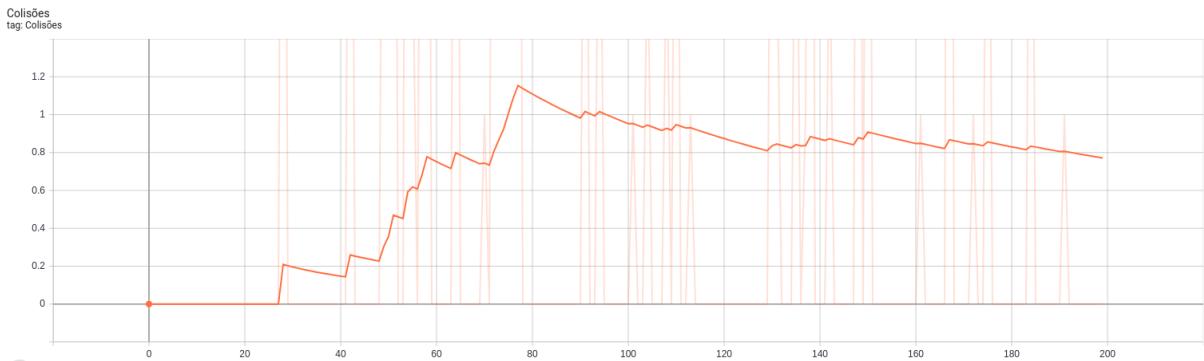


**Fonte:** Elaborada pelo autor.

Para verificar o comportamento do veículo em relação ao seu controle lateral, as Figuras 15 e 16 apresentam a quantidade de invasões de faixas que ocorreram. Sendo as contínuas as faixas que limitam o sentido e o limite da faixa e as faixas seccionadas a divisão da faixa de rolamento no mesmo sentido. Assim, de acordo com a Figura 15, o veículo consegue realizar uma evolução do seu controle lateral, ao invadir menos faixas seccionadas, ao longo dos episódios, mantendo-se por mais tempo dentro da sua faixa.

Entretanto, analisando a Figura 16, há um aumento das invasões de faixas contínuas. De acordo com os experimentos realizados, existem dois pontos que podem ocorrer a invasão de faixa contínua que não são discriminados neste gráfico: a invasão da pista do

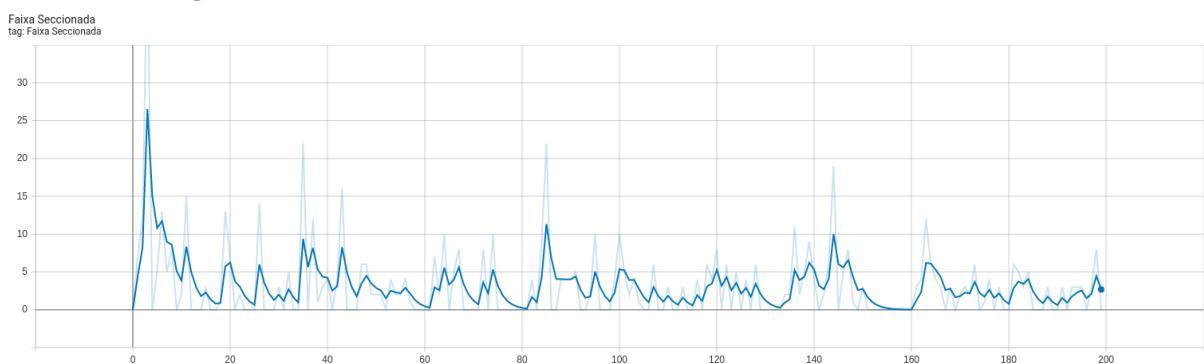
**Figura 14 – Treinamento - Tendência da Quantidade de Colisões por Episódio**



Fonte: Elaborada pelo autor.

sentido contrário (contra-mão) ou do lado da calçada. Assim, apenas com esta figura não seria possível dizer qual é o comportamento que o carro está tomando. Para isso, a Figura 17 contém a contagem de ações realizadas pelo carro no último episódio de treinamento. Pode-se observar que há um grande número de ações "frente-direita", o que nos indica que o carro possui um comportamento de dirigir para frente e para a direita. Este comportamento pode ter sido causada por diversos fatores, entre eles, o agente pode ter percebido que, ao invadir a faixa contínua da esquerda, tem maior probabilidade de colisão com veículos no sentido contrário e ser punido duas vezes, tanto pela colisão quanto pela invasão da faixa. Portanto, ao invadir a faixa apenas da direita, pode ter percebido que é punido apenas pela invasão e não por uma possível colisão. Por fim, a Figura 18 apresenta a imagem do último passo, antes do fim do episódio, apresentando a invasão da faixa contínua da direita pelo carro.

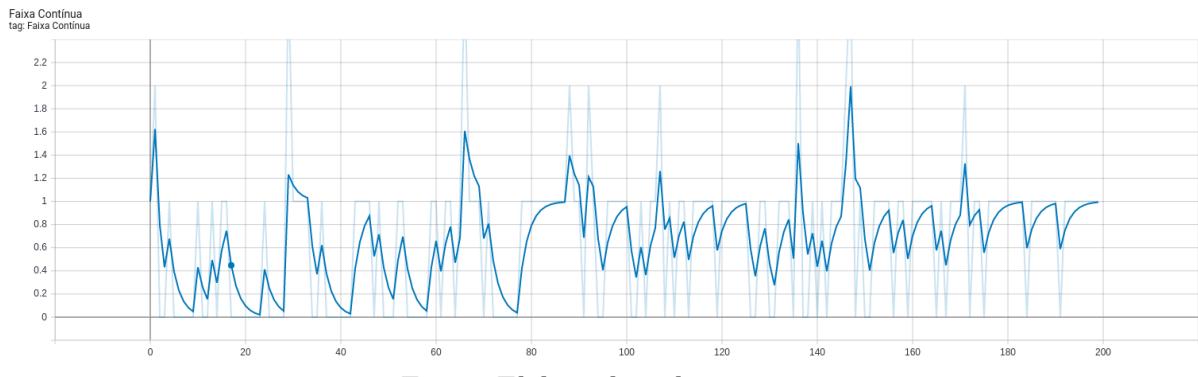
**Figura 15 – Treinamento - Invasão de Faixas Seccionadas**



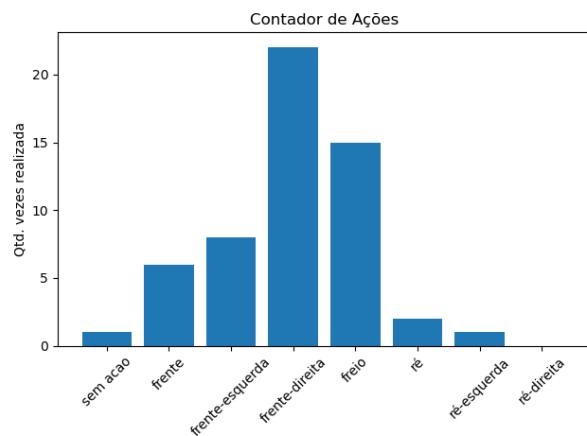
Fonte: Elaborada pelo autor.

A Figura 19 apresenta a distância em metros, percorrido de ré, ao longo do tempo e a Figura 20 apresenta a distância percorrida total (de frente e ré) do carro no ambiente. Com estas duas Figuras, pode-se visualizar que há uma tendência de diminuição do comportamento de andar de ré, uma ação que não pode ser realizada a todo momento por um carro autônomo, em especial se estiver no meio de um fluxo de veículos. Portanto, uma ação que foi desencorajada, por limitação, nas recompensas.

Nas Figuras 19 e 20 pode-se verificar que há uma diminuição da distância percorrida e isto pode ter sido causada pelos vícios do veículo. Em contrapartida, no começo há um veículo que realiza diversas ações aleatórias e ao diminuir a probabilidade de uso de ações aleatórias com o decremento do  $\epsilon$  (responsável pela probabilidade da DQN escolher uma

**Figura 16 – Treinamento - Invasão de Faixas Contínua**

**Fonte:** Elaborada pelo autor.

**Figura 17 – Treinamento - Contador de Ações do Episódio 200**

**Fonte:** Elaborada pelo autor.

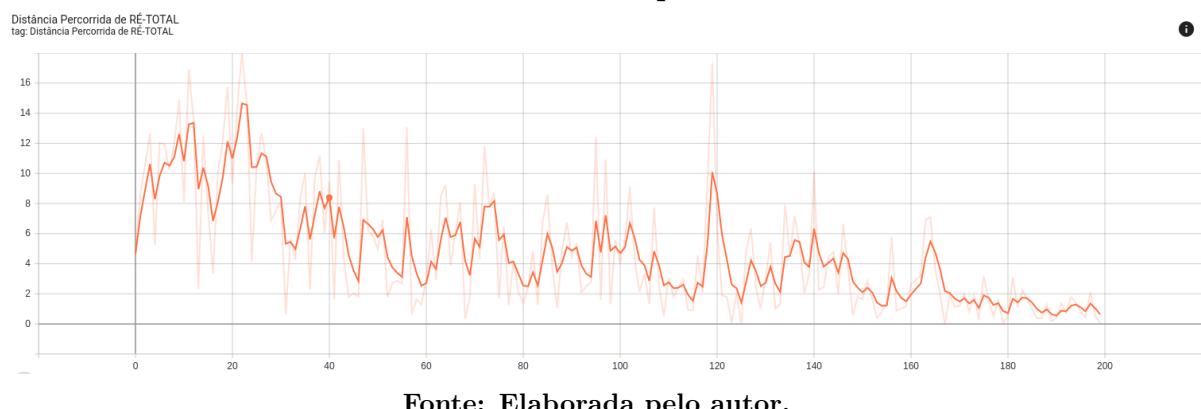
**Figura 18 – Treinamento - Imagem da Câmera Frontal do Ep. 200**

**Fonte:** Elaborada pelo autor.

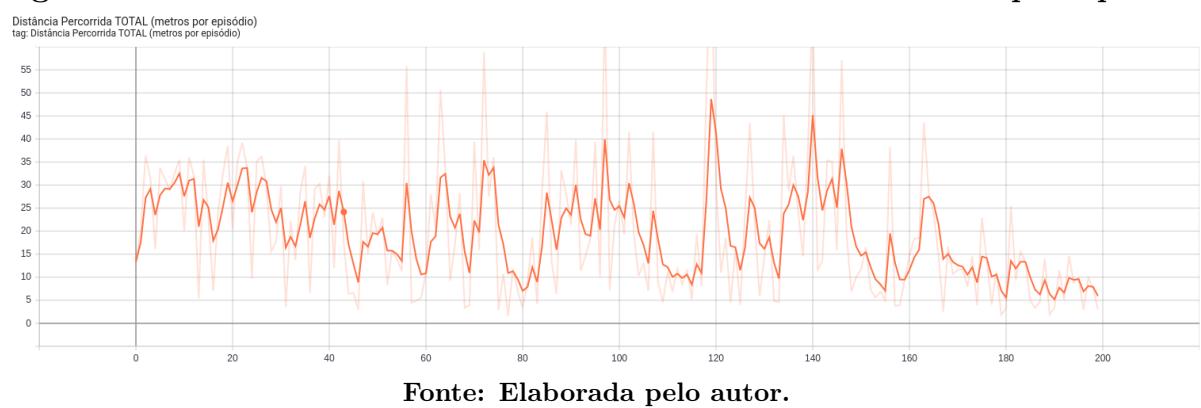
ação aleatória), o veículo provavelmente tende a tomar ações que diminuem sua distância percorrida, já que andar grandes distâncias não o recompensa e não é o seu objetivo

diretamente, pois sua recompensa está ligada, em suma, à manter-se no centro da faixa e atingir velocidades superiores a 3km/h e menores que 30km/h. É importante frisar que o veículo teve o comportamento de andar de ré desencorajado com sucesso, ao receber uma punição por percorrer grandes distâncias de ré, como é possível verificar na diminuição de distância da Figura 19.

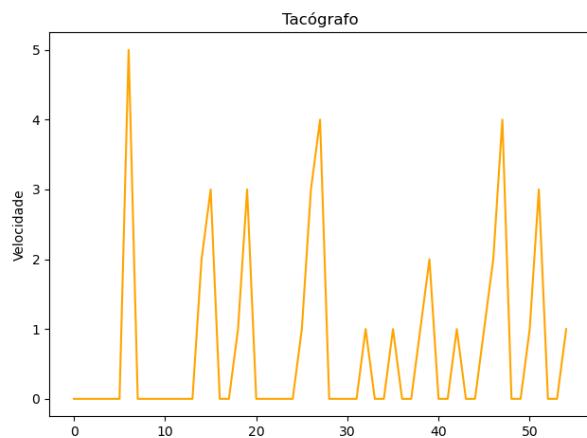
**Figura 19 – Treinamento - Distância Total Percorrida de Ré em Metros por Episódio**



**Figura 20 – Treinamento - Distância Total Percorrida em Metros por Episódio**

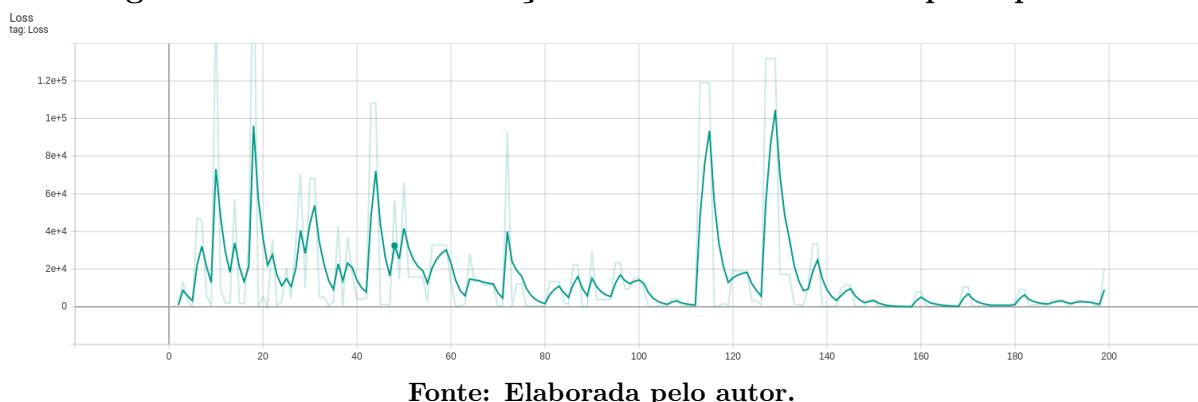


**Figura 21 – Treinamento - Velocidade por passos (Tacógrafo) do Ep. 200**



A distância entre o que foi predito pela rede principal e o que seria a melhor ação, baseado na rede alvo, é calculada pela função de perda (*loss*), que calcula a raiz quadrada da diferença entre o predito (pela rede principal) e o valor esperado (predito pela rede alvo), para uma mesma observação, pode ser observada pela Figura 22. Com base nesta Figura, pode-se observar que a rede principal cada vez mais se aproxima do comportamento da rede alvo, o que indica que há uma convergência entre as escolhas das ações, de acordo com o que aprendeu em iterações passadas.

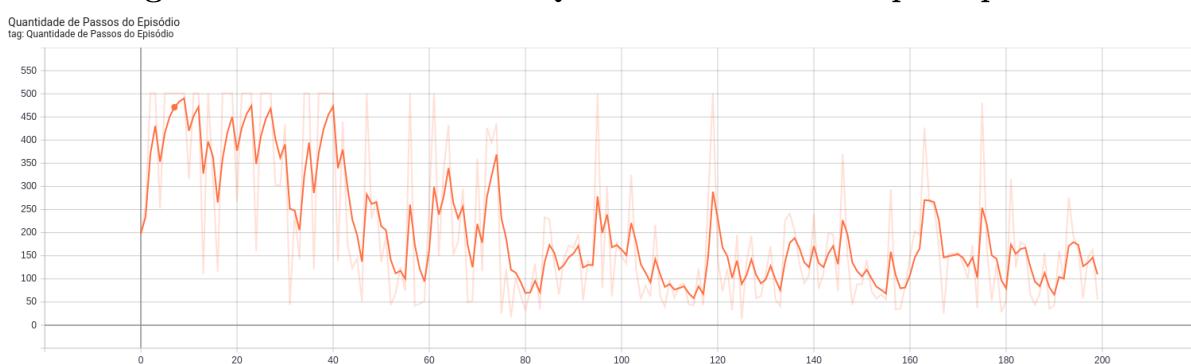
**Figura 22 – Gráfico da Função *Loss* do Treinamento por Episódio**



**Fonte:** Elaborada pelo autor.

A soma das informações anteriores podem ser agrupadas na visão da recompensa final recebida, Figura 24, e pela quantidade de passos totais realizados, Figura 23, a cada episódio, pelo agente.

**Figura 23 – Treinamento - Quantidade de Passos por Episódio**



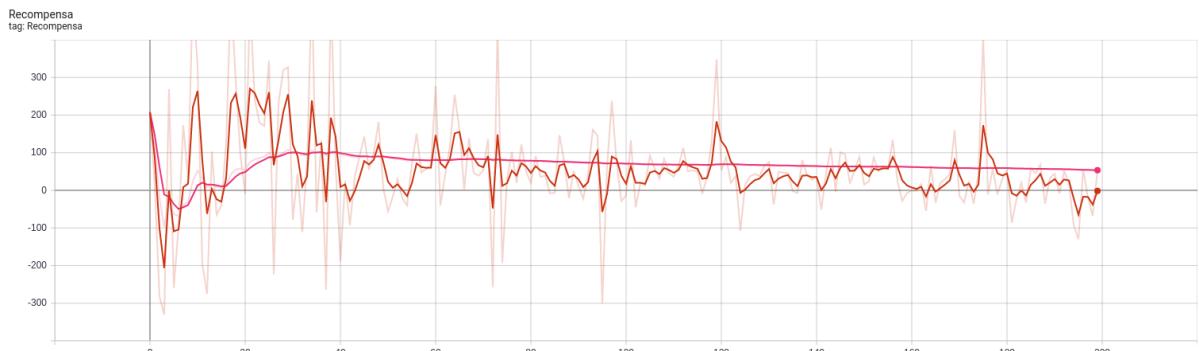
**Fonte:** Elaborada pelo autor.

Avaliando as Figuras 23 e 24 há uma queda dos valores de passos e de recompensas e para avaliar a causa desta queda, deve-se considerar as informações das figuras anteriores. Um dos fatores mais relevantes é o fato do carro ter uma tendência de comportamento de ir para a "frente-direita" e pode causar a baixa de passos e recompensas, se realizada sequencialmente, pois leva o veículo a invadir uma faixa contínua. Junto a esta ação, outro fator importante que pode-se avaliar, é a quantidade de ações de freio e, por consequência, a perda de velocidade, como verifica-se na Figura 21. Ações de freio acompanhadas por "frente-direita" contínua, podem encurtar o tempo de vida (como usar os passos do episódio para frear, possivelmente não recebendo recompensas) e ação do agente, pois, como explicado anteriormente, tendem a levar o veículo a realizar uma invasão de faixa contínua ocasionando o fim do episódio e punição.

Por fim, sabendo que o objetivo do agente é maximizar suas recompensas, a Figura 24 apresenta os valores de recompensa absoluta e média por episódio. Com a média

é possível observar uma tendência de valores e verificar que o agente está realizando comportamentos que estão diminuindo sua recompensa, o que é possível confirmar pelas Figuras anteriores que apresentaram um comportamento conservador do agente, em que é possível que este agente tenha encontrado um ótimo local para evitar colisões, mas receber alguma recompensa.

**Figura 24 – Treinamento - Recompensa Total e Média por Episódio**



**Fonte:** Elaborada pelo autor.

Fica disponível um vídeo\* com trechos do treinamento, em que são utilizados a ferramenta CarlaViz (XU, 2020), para visualizar a simulação e o carro, e o *log* utilizado para *debug* do treinamento. Após o fim do treinamento e análise dos dados encontrados, foi iniciada a avaliação do agente, utilizando a rede DQN treinada.

### 5.3 Avaliação do teste final

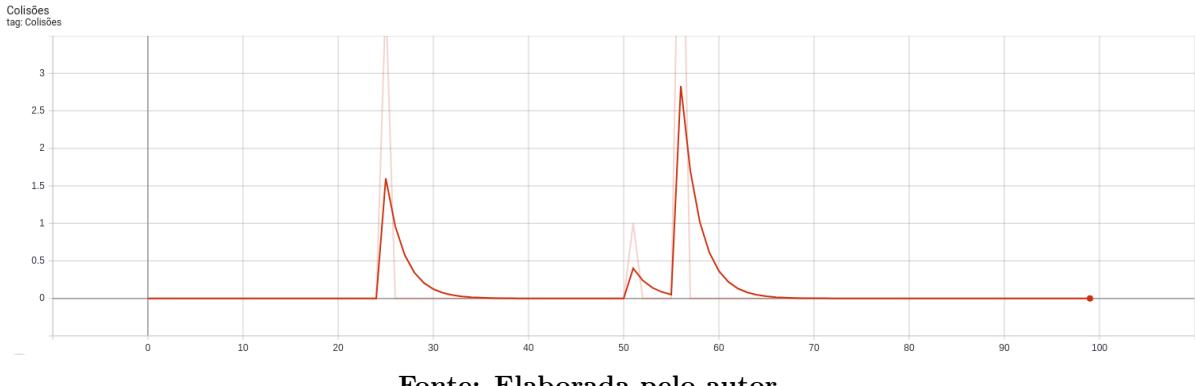
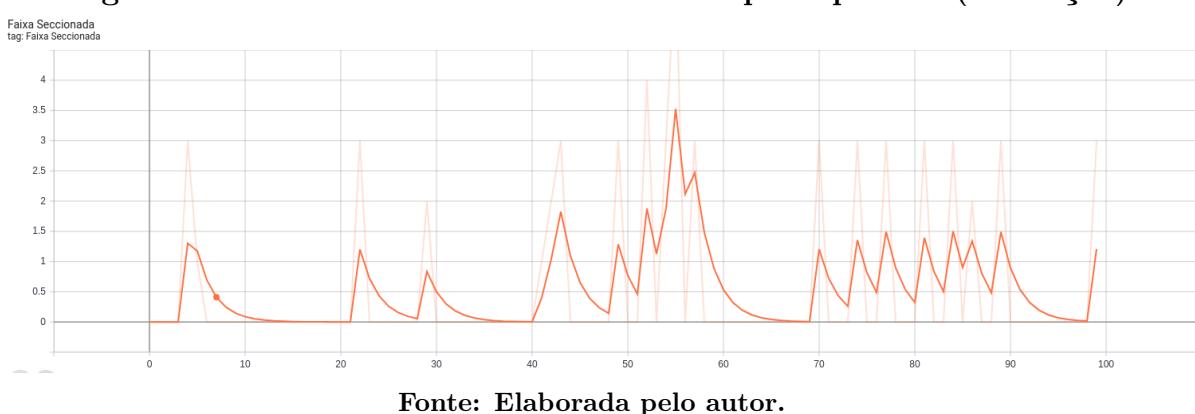
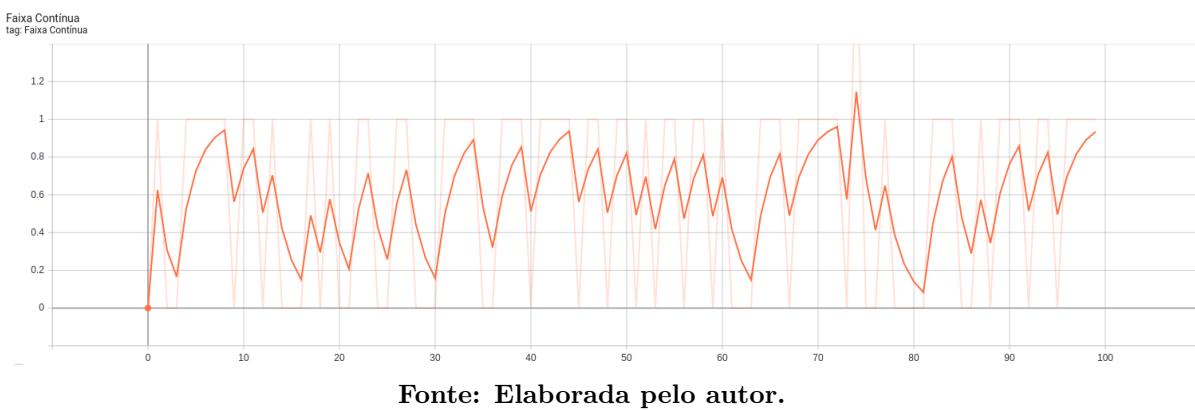
A avaliação final foi realizada com o modelo da DQN do treinamento com 100 episódios, sem limite de ações por episódio, no mapa *Town03*. Nesta avaliação, o agente deixa de aprender e usa apenas o conhecimento adquirido, realizando ações de acordo com a previsão de sua rede neural principal para uma observação. Logo, é possível avaliar o comportamento aprendido pelo agente.

Durante a avaliação, o agente encontrou uma brecha para explorar as recompensas, tendo em vista que não havia limite de ações por episódio, ao permanecer no centro da faixa e usando o freio. Portanto, foi necessário reiniciar a avaliação, limitando a quantidade de ações "sem-ação" e "freio" consecutivas que poderiam ser feitas em 100 ações. Assim, foi possível realizar a avaliação sem que o agente ficasse preso em um ótimo local ou reagisse a observação com passos infinitos usando o freio e impossibilitasse o fim da avaliação em tempo hábil.

É possível comprovar a tendência de redução de colisões encontrada no treinamento, Figura 14, durante a avaliação como mostra a Figura 25, em que tem-se um gráfico de colisões com predominância de episódios sem colisões, com exceções de alguns picos. Esse gráfico traduz que o comportamento do agente de ser conservador, tentando evitar colidir-se. Entretanto, como explicado anteriormente, como há um uso excessivo do freio, o excesso do seu uso pode ser o responsável pela diminuição das colisões. Pode-se verificar o uso excessivo do freio pela Figura 28 e pela variação de velocidade na Figura 29.

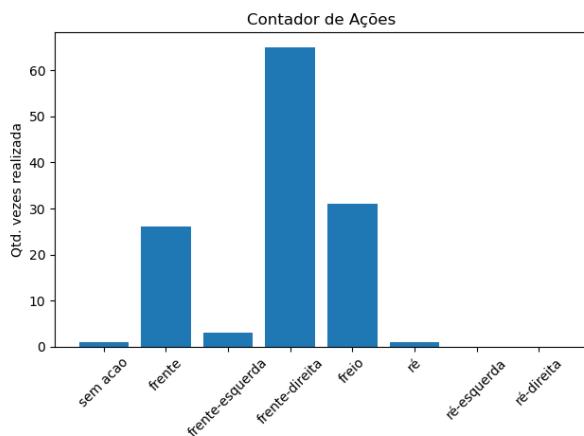
O comportamento de controle lateral do veículo na avaliação segue o padrão do que foi percebido no treinamento, mostrando que o veículo continua tentando realizar um controle lateral, mas não consegue evitar a tendência que tem de invadir ou atingir a faixa contínua, baseado na Figura 27, e que provavelmente seja a invasão da faixa do lado

\* Vídeo do trecho de treinamento <[https://www.youtube.com/watch?v=lVo\\_dXCn7Mk](https://www.youtube.com/watch?v=lVo_dXCn7Mk)>

**Figura 25 – Quantidade de Colisões por Episódio (Avaliação)****Figura 26 – Invasão de Faixas Seccionadas por Episódio (Avaliação)****Figura 27 – Gráfico de Invasão de Faixas Contínua por Episódio (Avaliação)**

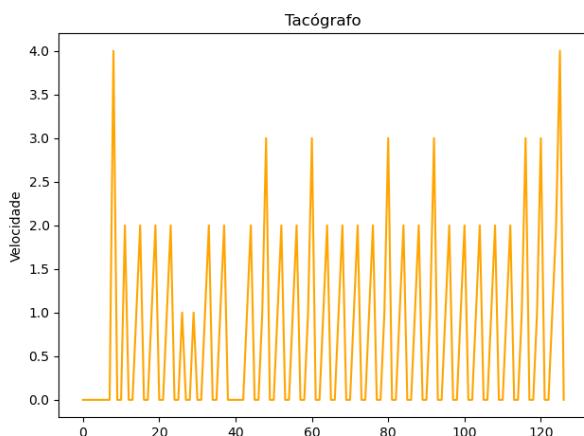
direito, de acordo com a alta quantidade de ações "frente-direita" apresentado na Figura 28. Em contrapartida, há uma redução considerável da quantidade de invasões de faixas seccionadas, como é possível verificar na Figura 26 em relação à Figura do treinamento 15. É possível que essa diminuição esteja ligada ao uso excessivo do freio, ao impedir que o veículo se desloque por grandes distâncias, como pode ser observado na Figura 29 a variação de velocidade entre os passos do episódio ou ainda a um maior número de ações usando a ação "frente". É importante lembrar que o veículo inicia o seu episódio no sentido correto e no centro da pista, permitindo que este possa deslocar-se, sem usar o volante, apenas usando a ação "frente" e receber maior recompensa por estar no centro da faixa e aumentando sua velocidade.

**Figura 28 – Contador de Ações do Episódio 100 (Avaliação)**



**Fonte:** Elaborada pelo autor.

**Figura 29 – Velocidade por Passos (Tacógrafo) do Ep. 100 (Avaliação)**



**Fonte:** Elaborada pelo autor.

Reforçando a ideia de que o veículo é viciado em ações "frente-direita", a Figura 30 apresenta a última imagem da câmera frontal do carro na avaliação, mostrando como foi o momento em que o episódio terminou e observa-se que foi invadindo uma faixa contínua à direita.

A avaliação comprova que a ação de usar a ré foi desencorajada, com base nos poucos metros percorridos por episódio apresentado na Figura 31 e, analisando a Figura 32, assim como no treinamento, o agente percorre pequenas distâncias (inferior a 20 metros na maior parte dos episódios) na avaliação, antes de provavelmente terminar em uma invasão de faixa.

É nítido perceber pela Figura 33 que a pouca distância percorrida está diretamente ligada aos poucos passos realizados, apesar de serem ilimitados, desde que não seja realizado, continuamente, ações de imobilização ("sem-ação" e "freio"). Por sua vez, os poucos passos não permitem que o veículo aumente sua recompensa, tendo uma recompensa média formando quase uma reta para a recompensa de valor 50, aproximadamente.

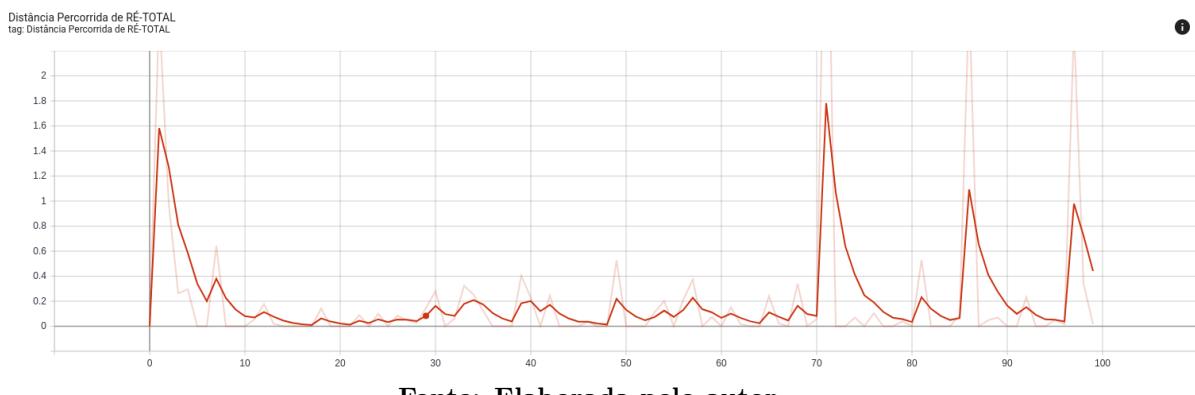
Por fim, foi realizado um vídeo\* com os episódios da avaliação, que apresentam as imagens da câmera frontal durante a avaliação.

---

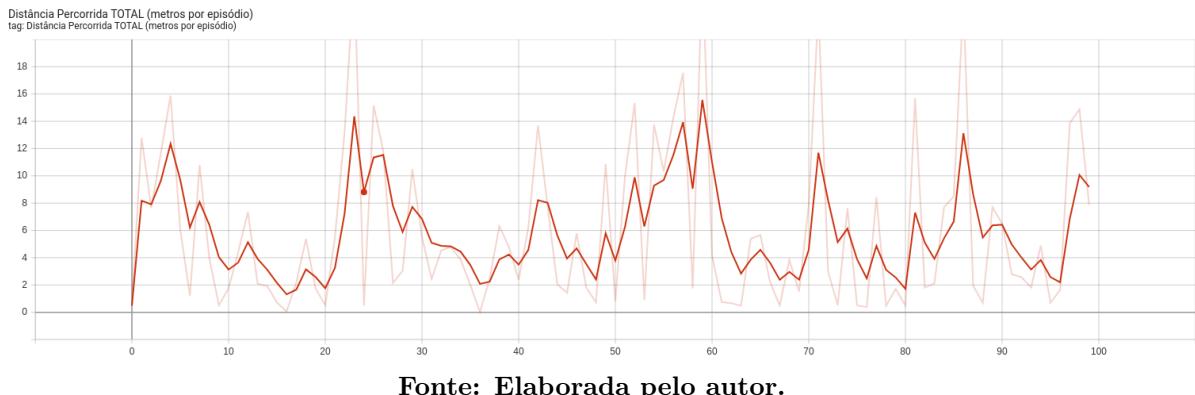
\* Vídeo dos episódios da avaliação: <[https://www.youtube.com/watch?v=lVo\\_dXCn7Mk](https://www.youtube.com/watch?v=lVo_dXCn7Mk)>

**Figura 30 – Imagem da Câmera Frontal do Ep. 100 (Avaliação)**

Fonte: Elaborada pelo autor.

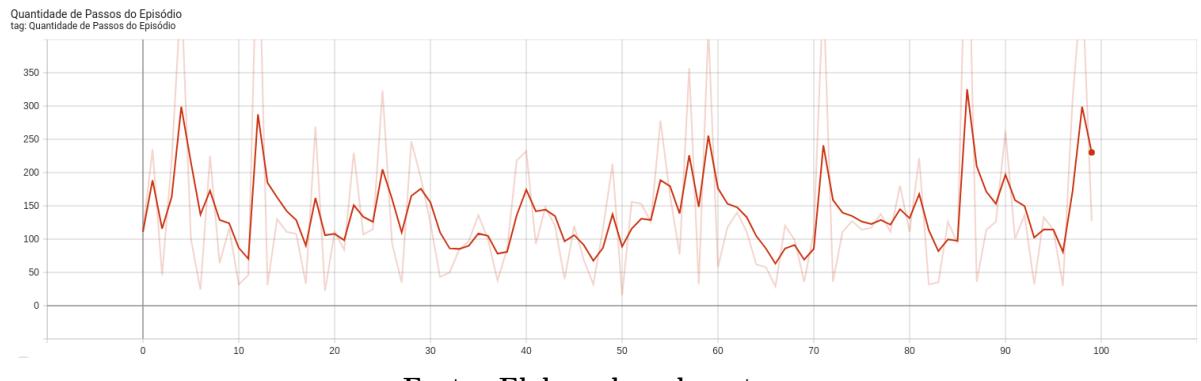
**Figura 31 – Distância Total Percorrida de Ré em Metros por Episódio (Avaliação)**

Fonte: Elaborada pelo autor.

**Figura 32 – Distância Total Percorrida em Metros por Episódio (Avaliação)**

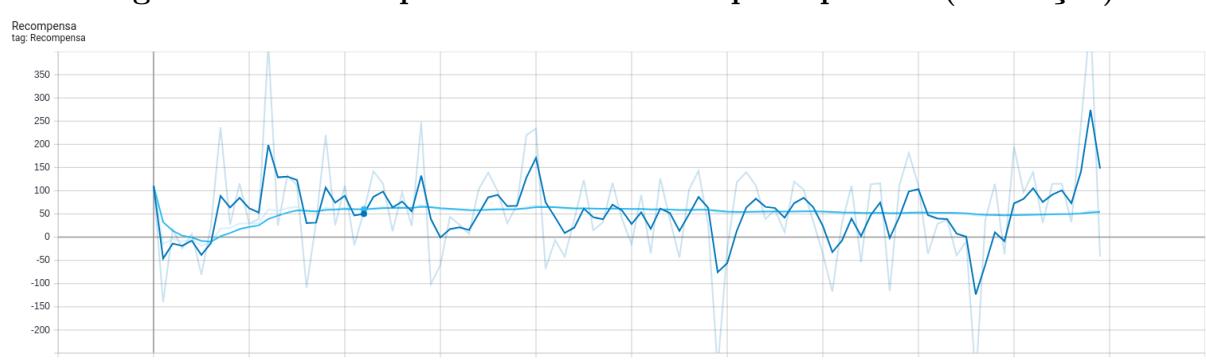
Fonte: Elaborada pelo autor.

**Figura 33 – Quantidade de Passos por Episódio (Avaliação)**



**Fonte:** Elaborada pelo autor.

**Figura 34 – Recompensa Total e Média por Episódio (Avaliação)**



**Fonte:** Elaborada pelo autor.

## 6 CONCLUSÃO

Ao fim do treinamento foi possível perceber que o agente conseguiu evoluir, realizando um melhor controle lateral, mas insuficiente para a realização de um controle apropriado para um carro autônomo. Seu controle longitudinal está diretamente ligado ao controle lateral, pelo fato de realizar uma ação de aceleração (frente) e virar o volante (direita), de acordo com a Figura 17, referente ao treinamento e da Figura 28, referente a avaliação. Apesar disto, o treino foi insuficiente para que o veículo percorresse maiores distâncias, no sentido correto da via. Os resultados obtidos demonstram um agente capaz de aprender os comportamentos básicos de direção, mas que necessita de mais tempo de treinamento e possíveis modificações nas recompensas, de acordo com a necessidade de evolução e tendências de comportamento que possam ser desenvolvidas. Se comparado ao estado da arte, seriam necessárias dezenas de milhões e até bilhões de passos, que, em comparação ao realizado neste trabalho foram feitas apenas 40535 passos em um período de 4 horas.

Apesar da implementação e realização dos experimentos consumirem a maior parte do tempo para a realização deste trabalho, o aprendizado obtido foi valioso para o entendimento da construção de um veículo autônomo e cuidados com uma inteligência artificial (em especial os comportamentos que podem ser desenvolvidos). É importante frisar que apesar das inúmeras tentativas falhas nos experimentos e treinamento, estas foram muito importantes para a que o veículo pudesse evoluir, ao receber melhorias e incrementos.

Um dos grandes desafios para a realização deste trabalho foi a definição das recompensas e quais informações ou dados utilizar para avaliar a situação do carro para recompensá-lo. Outro grande desafio foi o alto tempo de processamento necessário para a realização dos episódios de simulação.

O trabalho foi desafiante ao lidar com tantos problemas, mas provou-se recompensador ao possibilitar descobrir o que é preciso, os desafios e possibilidades envolvidos no desenvolvimento de um veículo autônomo e poder contribuir com a produção de conteúdo acadêmico sobre o estado da arte de veículos autônomos.

### 6.1 Trabalhos Futuros

Os estudos sobre veículos autônomos estão em constante desenvolvimento e aprimoramento e, portanto, é possível utilizar diversas modificações neste trabalho para melhorar o comportamento do veículo. A partir do trabalho realizado, pode-se citar a produção de trabalhos futuros que utilizem novos sensores, alterem-se os objetivos e recompensas do veículo, como adicionar um objetivo de localização, visando que o veículo aprenda a dirigir-se para uma posição determinada; É possível realizar trabalhos mais aprofundados com a variação dos parâmetros do algoritmo DQN, verificando e avaliando quais resultados serão atingidos e compará-los com o presente trabalho produzido; etc. Podem ser realizadas experimentações com novos tipos de arquitetura, deixando de utilizar uma arquitetura do tipo *end-to-end* ao realizar pós-processamento nos dados e alterar como esses dados são fornecidos ao aprendizado do agente; É possível realizar a utilização de outros algoritmos de aprendizado por reforço; e, por fim, a análise dos custos computacionais envolvidos no processamento de qualquer uma das variações dos trabalhos futuros sugeridos, além de comparações entre custo de diferentes algoritmos de treinamento e tempos de tomada de decisão, após o treinamento.

## Referências Bibliográficas

- AI, O. *OpenAI Gym*. 2020. Disponível em: <<https://gym.openai.com/docs/>>. Acesso em: 15 abril de 2020. 33
- BOJARSKI, M. et al. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016. Disponível em: <<http://arxiv.org/abs/1604.07316>>. 26
- Coichecki, S.; Filip, I. Self-driving vehicles: current status of development and technical challenges to overcome. In: *2020 IEEE 14th International Symposium on Applied Computational Intelligence and Informatics (SACI)*. [S.l.: s.n.], 2020. p. 000255–000260. 14
- DENATRAN. *Frota de Veículos*. 2020. Disponível em: <<https://infraestrutura.gov.br/component/content/article/115-portal-denatran/8552-estat%C3%ADsticas-frota-de-ve%C3%ADculos-denatran.html>>. Acesso em: 15 abril de 2020. 14
- DOSOVITSKIY, A. et al. CARLA: An Open Urban Driving Simulator. *CoRR*, 2017. ISSN 1938-7228. Disponível em: <<http://arxiv.org/abs/1711.03938>>. 25
- EDWARDS, C. Deep learning hunts for signals among the noise. *Communications of the ACM*, v. 61, p. 13–14, 05 2018. 21
- FACELI, K. et al. *Inteligência artificial: uma abordagem de aprendizado de máquina*. [S.l.]: LTC, 2011. 20
- Fayjie, A. R. et al. Driverless car: Autonomous driving using deep reinforcement learning in urban environment. In: *2018 15th International Conference on Ubiquitous Robots (UR)*. [S.l.: s.n.], 2018. p. 896–901. 26
- GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. [S.l.]: MIT Press, 2016. <<http://www.deeplearningbook.org>>. 21
- GUCKIRAN, K.; BOLAT, B. Autonomous Car Racing in Simulation Environment Using Deep Reinforcement Learning. In: *2019 Innovations in Intelligent Systems and Applications Conference (ASYU)*. [S.l.: s.n.], 2019. p. 1–6. 26
- GUO, C. et al. Driver–vehicle cooperation: a hierarchical cooperative control architecture for automated driving systems. *Cognition, Technology and Work*, Springer London, v. 21, n. 4, p. 657–670, 2019. ISSN 14355566. 15, 19, 20
- INTERNATIONAL; SAE. *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. [S.l.], 2018. 14, 18
- Jaritz, M. et al. End-to-end race driving with deep reinforcement learning. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. [S.l.: s.n.], 2018. p. 2070–2075. 26
- JO, K. et al. *Overall reviews of autonomous vehicle a1 - System architecture and algorithms*. [S.l.]: IFAC, 2013. v. 8. 114–119 p. ISSN 14746670. ISBN 9783902823366. 15, 19

KIRAN, B. R. et al. *Deep Reinforcement Learning for Autonomous Driving: A Survey*. 2020. 16

Krishnaswami Sreedhar, B.; Shunmugam, N. Deep learning for hardware-constrained driverless cars. In: *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. [S.l.: s.n.], 2020. p. 26–29. 27

Kuutti, S. et al. A survey of deep learning applications to autonomous vehicle control. *IEEE Transactions on Intelligent Transportation Systems*, p. 1–22, 2020. 14

LIU, S. et al. *Creating Autonomous Vehicle Systems*. [S.l.]: Morgan and Claypool, 2017. 14, 16, 18

LíDER-DPVAT, S. *Relatório Anual*. 2019. Disponível em: <<https://www.seguradoralider.com.br/Documents/Relatorio-Anual-2019.pdf>>. Acesso em: 15 abril de 2020. 14

MARQUES, R. F. *Avaliação de Diferentes Estratégias Deaprendizado por Reforço Aplicadas Em Jogos da Plataforma Atari*. Monografia (Graduação) — Pontifícia Universidade Católica de Minas Gerais, 2019. 20, 23

MNIH, V. et al. *Asynchronous Methods for Deep Reinforcement Learning*. 2016. 27

MNIH, V. et al. Human-level control through deep reinforcement learning. *Nature*, Nature Publishing Group, v. 518, n. 7540, p. 529–533, 2015. ISSN 14764687. 15, 17, 23, 24, 31, 33

PALANISAMY, P. *Hands-On Intelligent Agents with OpenAI Gym: Your guide to developing AI agents using deep reinforcement learning*. [S.l.]: Packt, 2018. ISBN 9781788836579. 25

RAVICHANDIRAN, S. *Hands-on Reinforcement Learning with Python. Master Reinforcement and Deep Reinforcement Learning using OpenAI Gym and TensorFlow*. [S.l.]: Packt, 2018. ISBN 9781788836524. 23, 31

SUTTON, R. S.; BARTO, A. G. *Reinforcement Learning: An Introduction*. Second. [S.l.]: The MIT Press, 2018. 20, 22, 23

THRUN, S. et al. Stanley: The robot that won the darpa grand challenge: Research articles. *J. Robot. Syst.*, John Wiley and Sons Ltd., Chichester, UK, v. 23, n. 9, p. 661–692, set. 2006. 16, 19

WATKINS, C. J. C. H. *Learning from Delayed Rewards*. Tese (Doutorado) — King's College, Cambridge, UK, May 1989. 22

XU, M. *CarlaViz*. 2020. Disponível em: <[https://carla.readthedocs.io/en/latest/plugins\\_carlaviz/](https://carla.readthedocs.io/en/latest/plugins_carlaviz/)> .Acesso em : 25de outubro de 2020. 43

## APÊNDICE A - TRECHO DE CÓDIGO DQN PARA TREINAMENTO DO CARRO

```

1  '''
2  Wenderson Júnio de Souza
3  Algoritmo Deep Q-Network para percepção e movimentação de carro autônomo
   → no simulador Carla
4  Monografia 2020 - PUC Minas
5
6  '''
7 SCRIPT DE TREINAMENTO
8 'dqn_sync.py'
9 '''
10
11 Créditos do código base da DQN retirado do livro : 'Hands-On
   → Reinforcement Learning with Python: Master reinforcement
12 and deep reinforcement learning using OpenAI Gym and TensorFlow', 2018.
   → Ravichandiran, Sudharsan.
13
14 DQN modificada por Wenderson Souza para utilizar a framework Keras.
15 Ambiente preparado por Wenderson Souza, baseado no código base da
   → OpenGymIA e códigos de exemplos fornecidos pela
16 documentação do simulador Carla
17
18 Referências:
19 https://carla.readthedocs.io/en/latest/python\_api/
20 https://carla.readthedocs.io/en/latest/core\_concepts/
21 https://pythonprogramming.net/reinforcement-learning-agent-self-driving-autonomous-cars-carla-python/
22 '''
23
24 action = epsilon_greedy(action, global_step)
25
26 # Realiza a ação escolhida e recupera o próximo estado, a recompensa,
   → info e se acabou(se houve colisão).
27 next_obs, reward, done, info = env.step(action)
28
29 # Armazenamento das experiências no buffer de replay
30 print('> Guardando experiências buffer de replay... ')
31 exp_buffer.append([obs, action, next_obs, reward, done])

```

```
32 print('> Terminou o PASSO com recompensa: ', reward, '')  
33  
34 # Treino da rede principal, após uma qtd de passos, usando as  
→ experiências do buffer de replay.  
35 if global_step \% steps_train == 0 and global_step > start_steps:  
36     print('> Atualização da Q-Network \%', steps_train, 'passos.')  
37     # Retira uma amostra de experiências  
38     obs, act, next_obs, reward, done = sample_memories(BATCH_SIZE)  
39  
40     # valor de probabilidade da ação mais provável  
41     targetValues = targetQ.predict(x=next_obs.retornaObs())  
42     print('> Valores alvo: ', targetValues)  
43  
44     bestAction = np.argmax(targetValues)  
45  
46     print('> Melhor Ação (prediction): ', env.DICT_ACT[bestAction],  
→      '(',bestAction,').')
```

## APÊNDICE B - TRECHO DE CÓDIGO PARA AVALIAÇÃO DA AGENTE

```
1  '''
2  SCRIPT DE AVALIAÇÃO
3  'avaliacao_dqn_sync.py'
4  '''
5
6 while not env.done:
7     print( '\n> Passo ', env.passos_ep, ' (ep ', episodio, ')  [Passo
8         → Global:', global_step, ']:' )
9     # Prediz uma ação (com base no que possui treinada) com base na
10    → observação - por enquanto, apenas uma imagem de câmera
11    actions = mainQ.predict(x=obs.retornaObs())
12    # A rede produz um resultado em %. Logo, escolhe a posição do
13    → vetor(ação) com maior probabilidade (argmax())
14    action = np.argmax(actions) # neste caso, argmax retorna a posição
15    → com maior probabilidade
16
17    # Realiza a ação escolhida e recupera o próximo estado, a
18    → recompensa, info e se acabou(se houve colisão).
19    next_obs, reward, done, info = env.step(action)
20
21
22    print('> Terminou o PASSO com recompensa: ', reward, '')
23
24    obs = next_obs # troca a obs
25    passos_ep += 1
26    global_step += 1
```

## ANEXO A - TRECHO DE CÓDIGO AUXILIAR PARA CRIAR ATORES - SPAWN ACTORS

```

1  '''
2  SPAWN ACTORS
3  'spawn_npc.py'
4  '''
5  # -----
6  # Spawn vehicles
7  # -----
8 batch = []
9 for n, transform in enumerate(spawn_points):
10     if n >= args.number_of_vehicles:
11         break
12     blueprint = random.choice(blueprints)
13     if blueprint.has_attribute('color'):
14         color = random.choice(blueprint.get_attribute('color')
15             \ .recommended_values)
16         blueprint.set_attribute('color', color)
17     if blueprint.has_attribute('driver_id'):
18         driver_id = random.choice(blueprint.get_attribute('driver_id')
19             \ .recommended_values)
20         blueprint.set_attribute('driver_id', driver_id)
21     blueprint.set_attribute('role_name', 'autopilot')
22
23     # prepare the light state of the cars to spawn
24     light_state = vls.NONE
25     if args.car_lights_on:
26         light_state = vls.Position | vls.LowBeam | vls.LowBeam
27
28     # spawn the cars and set their autopilot and light state all
29     # together
30     batch.append(SpawnActor(blueprint, transform,
31         .then(SetAutopilot(FutureActor, True,
32             \ traffic_manager.get_port())))
33         .then(SetVehicleLightState(FutureActor, light_state)))
34
35 for response in client.apply_batch_sync(batch, synchronous_master):
36     if response.error:

```

```
35     logging.error(response.error)
36 else:
37     vehicles_list.append(response.actor_id)
```

## ANEXO B - TRECHO DE CÓDIGO PARA CONTROLE MANUAL DE UM VEÍCULO

```
1  '''
2  MANUAL CONTROL
3  'manual_control.py'
4  '''
5
6 def game_loop(args):
7     pygame.init()
8     pygame.font.init()
9     world = None
10
11    try:
12        client = carla.Client(args.host, args.port)
13        client.set_timeout(2.0)
14
15        display = pygame.display.set_mode(
16            (args.width, args.height),
17            pygame.HWSURFACE | pygame.DOUBLEBUF)
18
19        hud = HUD(args.width, args.height)
20        world = World(client.get_world(), hud, args)
21        controller = KeyboardControl(world, args.autopilot)
22
23        clock = pygame.time.Clock()
24        while True:
25            clock.tick_busy_loop(60)
26            if controller.parse_events(client, world, clock):
27                return
28            world.tick(clock)
29            world.render(display)
30            pygame.display.flip()
```