Universidade do Minho
Escola de Engenharia

Nuno Miguel De Jesus Andrade

Análise de alterações temporárias e anormalidades nas estradas para veículos de condução autónoma

Dissertação de Mestrado
Mestrado Integrado em Engenharia Eletrónica Industrial e Computadores

Trabalho efetuado sob a orientação do
Professor Doutor António Fernando Macedo Ribeiro

Novembro de 2021

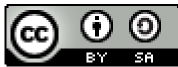
DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

Este é um trabalho académico que pode ser utilizado por terceiros desde que respeitadas as regras e boas práticas internacionalmente aceites, no que concerne aos direitos de autor e direitos conexos.

Assim, o presente trabalho pode ser utilizado nos termos previstos na licença abaixo indicada.

Caso o utilizador necessite de permissão para poder fazer um uso do trabalho em condições não previstas no licenciamento indicado, deverá contactar o autor, através do RepositóriUM da Universidade do Minho.

Licença concedida aos utilizadores deste trabalho



Atribuição-CompartilhaIgual
CC BY-SA
https://creativecommons.org/licenses/by-sa/4.0/

Agradecimentos

Este trabalho é o resultado de anos de esforço e dedicação, tanto de parte individual como coletiva. É com grande satisfação que dedico este projeto a todos os que me ajudaram a concretizá-lo.

Começo por agradecer às pessoas mais importantes da minha vida, os meus pais, José Andrade e Madalena Andrade. Por todo o carinho e suporte durante estes anos importantes para o meu futuro, pela motivação que sempre me transmitem e pela resiliência que me incentiva a lutar pelos meus sonhos.

O meu segundo agradecimento é ao meu avô, Orlando Veiga. A pessoa mais próxima que tenho com a maior experiência de vida, sempre com um conselho para dar em qualquer situação, sempre com um olhar positivo para a vida.

À Eva Silva, pela tua afabilidade e generosidade comigo e por estares sempre pronta para me ajudar, obrigado por seres quem és.

Um agradecimento especial ao meu Orientador, o professor Fernando Ribeiro, por todo o suporte incansável e conhecimento profundo que me transmitiu para a conclusão de um trabalho desta dimensão. Não poderia deixar de mencionar o Tiago Ribeiro, que além de um apoio profissional, foi um amigo que não imaginava aparecer e certamente não o irei esquecer. Quero também mencionar toda a equipa do LAR pelo ambiente positivo proporcionado ao longo de todo o trabalho.

Ao Pipoca, o meu animal de estimação, que merece esta menção por se ter mostrado uma das melhores companhias de trabalho que eu alguma vez poderia pedir. Pelo teu conforto e amor, e por jamais me deixares trabalhar horas a fio, preservas o meu bem-estar à tua maneira.

Por último, a todos os meus amigos em geral e a todas as pessoas que aqui não referi o nome, por estarem do meu lado em todos os momentos e darem o vosso suporte, eu agradeço do fundo do coração.

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

Resumo

"Análise de alterações temporárias e anormalidades nas estradas para veículos de condução autónoma"

A condução autónoma tem vindo cada vez mais a ser utilizada como aplicação prática de métodos de Inteligência Artificial, como *supervised learning* e *reinforcement learning*. A Inteligência Artificial é uma solução bem conhecida para resolver problemas de condução autónoma mas ainda não está estabelecida e suficientemente estudada para lidar com problemas específicos do mundo real que os seres humanos lidam diariamente, como sinalização temporária de obras nas estradas. Esta é a motivação principal para o desenvolvimento de um sistema como o que é apresentado neste projeto. O *YOLOv3-tiny* é utilizado para detetar sinais de obras nas estradas em que um veículo circula. O *Deep Deterministic Policy Gradient* (*DDPG*) é utilizado para controlar o comportamento do veículo no momento de circulação em zona de sinalização temporária. A segurança dos passageiros e do ambiente ao redor é a regra que mais foi tida em consideração ao longo da implementação do sistema. O *YOLOv3-tiny* obteve uma *accuracy* de 94.8% *mAP* e provou ser viável a sua utilização em situações reais. O *DDPG* permitiu o sucesso do comportamento de um veículo mais de 50% dos episódios durante os testes, contudo ainda precisa de melhorias para ser utilizado no mundo real para respeitar uma condução segura e preventiva.

Palavras-Chave

YOLO, Reinforcement Learning, Deep Deterministic Policy Gradient, Condução Autónoma, Sinalização Temporária

Abstract

“Analysis of temporary public roadworks for autonomous driving vehicles”

Autonomous driving is emerging as a useful practical application of Artificial Intelligence algorithms regarding both supervised learning and reinforcement learning methods. Artificial Intelligence is a well-known solution for some autonomous driving problems but it is not yet established and fully researched for facing real world problems regarding specific situations human drivers face every day, such as temporary roadworks and temporary signs. This is the core motivation for the proposed framework in this project. YOLOv3-tiny is used for detecting roadworks signs in the path traveled by the vehicle. Deep Deterministic Policy Gradient (DDPG) is used for controlling the behavior of the vehicle when overtaking the working zones. Security and safety of the passengers and the surrounding environment are the main concern taken into account. YOLOv3-tiny achieved an 94.8% mAP and proved to be reliable in real-world applications. DDPG made the vehicle behave with success more than 50% of the episodes when testing, although still needs some improvements to be transported to the real-world for secure and safe driving.

Keywords

YOLO, Reinforcement Learning, Deep Deterministic Policy Gradient, Autonomous Driving, Public Roadworks

Índice

Agradecimentos ... iii

Resumo ... v

Abstract ... vi

Índice de Figuras ... ix

Índice de Tabelas ... xiii

Nomenclatura ... xiv

1. Introdução ... 15

1.1 Descrição do problema ... 15

1.2. Objetivos ... 15

1.3. Proposta ... 15

1.4. Motivação ... 16

1.5. Estrutura da dissertação de mestrado ... 16

2. Revisão da literatura ... 17

2.1. Inteligência Artificial ... 17

2.1.1. Machine Learning ... 17

2.1.2. Deep Learning ... 19

2.2. Inteligência Artificial na Robótica ... 26

2.2.1. Deteção de objetos ... 27

2.2.2. Condução Autónoma ... 33

3. Fundamentos Teóricos ... 38

3.1. YOLOv3-tiny ... 38

3.2. Deep Deterministic Policy Gradient ... 42

4. Modelo e Simulação ... 51

4.1. YOLOv3-tiny: Otimização do Modelo ... 51

4.2. YOLOv3-tiny: Dataset ... 52

4.3. YOLOv3-tiny: Treino ... 54

4.4. YOLOv3-tiny: Teste ... 57

4.5. Ambiente de simulação: ROS, CoppeliaSim e Pycharm ... 59

4.6. DDPG: Configuração do modelo ... 62

4.7. DDPG: treino 65
4.8. DDPG: teste 68
5. Resultados 69
5.1. Deteção de sinalização temporária 69
5.2. Movimentação na via delimitada por sinalização temporária 80
6. Conclusões 86
Referências Bibliográficas 88

ÍNDICE DE FIGURAS

Figura 1 Representação dos principais 3 campos de Machine Learning. 18

Figura 2 Ilustração do enquadramento temporal dos métodos de Deep learning. 19

Figura 3 Representação de uma Artificial Neural Network e uma Deep Learning Neural Network........ 20

Figura 4 Estrutura de uma rede neuronal artificial. m_ij representa o weight entre o neurónio "i" e o neurónio "j". 21

Figura 5 Representação da arquitetura de uma CNN. Neste exemplo, "ReLU" é a função de ativação implementada na rede neuronal. 22

Figura 6 Representação da aplicação de um kernel numa matriz e respetiva matriz de saída. 22

Figura 7 Exemplo de uma operação max-pooling de stride 2. 23

Figura 8 Estrutura de uma rede de camadas totalmente ligadas. 23

Figura 9 Exemplo da utilização da aplicação móvel para tradução de texto em imagens da Google (Google Translate). 25

Figura 10 Relação entre Tensorflow e Keras. Desde o nível mais baixo, GPU ou CPU, até ao nível mais alto, onde se situa o Keras.. 26

Figura 11 Exemplos de robôs que utilizam inteligência artificial no seu funcionamento. Fonte:[12]..... 26

Figura 12 Protótipo do Robô CHARMIE (desenvolvido em 2017). Fonte:[33]. 27

Figura 13 Arquitetura base de uma RCNN. 28

Figura 14 Arquitetura das redes SSD. Fonte:[40]. 29

Figura 15 Modelo YOLO. 29

Figura 16 Arquitetura da rede YOLO. Fonte:[37]. 30

Figura 17 Comparação das métricas resultantes da implementação de sistemas líderes em deteção de objetos, neste caso pretende-se realizar a comparação entre a YOLO e a R-CNN (Fonte: [37]). 30

Figura 18 Comparação das métricas resultantes da implementação dos sistemas líderes em deteção de objetos, neste caso pretende-se realizar a comparação entre a YOLO e a SSD (Fonte:[16]). 31

Figura 19 Comparação das métricas resultantes da implementação de sistemas líderes em deteção de objetos, neste caso pretende-se realizar a comparação entre a YOLOv2 e as mais conhecidas (Fonte:[41]). 31

Figura 20 Comparação das métricas resultantes da implementação de sistemas líderes em deteção de objetos, neste caso pretende-se realizar a comparação entre a YOLOv3 e as mais conhecidas (Fonte:[42]). 32

Figura 21 Comparação das métricas resultantes da implementação de sistemas líderes em deteção de objetos em tempo real e tempo mais lento, neste caso pretende-se realizar a comparação entre a YOLOv3-tiny e as mais conhecidas (Fonte:[43]). 33
Figura 22 Resultado obtido para o treino em 2 bases de dados diferentes, KITTI e BDD. Fonte:[45]. 33
Figura 23 Esquema de 2 abordagens comuns a problemas de condução autónoma. a) Sistema modular e b) Sistema end-to-end. Fonte [48] 35
Figura 24 Tarefas importantes na condução autónoma que requerem Deep Reinforcement Learning para aprender uma policy ou um comportamento. Fonte: [61]. 35
Figura 25 Captura de ecrã, da simulação em computador,de um veículo autónomo executando seguimento da trajetória das linhas amarelas usando algoritmos de Deep Reinforcement Learning. Fonte:[60]. 36
Figura 26 Comparação de performance entre Q-Learning e DDAC na mesma via. Fonte:[64]. 36
Figura 27 Simulação em TORCS de uma ultrapassagem controlada por métodos de Reinforcement Learning. Fonte:[63]. 37
Figura 28 Diagrama do funcionamento base da rede YOLOv3-tiny. 38
Figura 29 Arquitetura da rede YOLOv3-tiny. 39
Figura 30 Obtenção das coordenadas das predicted bounding boxes. 40
Figura 31 Cálculo do IoU. 41
Figura 32 Cálculo do GIoU. 42
Figura 33 Diferença entre os métodos on-policy e off-policy. 45
Figura 34 Os dois principais tipos de Reinforcement Learning. 46
Figura 35 Dois tipos de algoritmos model-free. 46
Figura 36 Exemplo de processamento de Deep Q-Learning. 47
Figura 37 Estrutura de um algoritmo Actor-Critic. 48
Figura 38 Estrutura do DDPG. 48
Figura 39 Fluxograma com o processo de otimização utilizado para obtenção de um dataset com a melhor qualidade. 51
Figura 40 Exemplos de imagens que constituem o dataset utilizado para treinar o YOLOv3-tiny. 52
Figura 41 Sinais presentes no dataset sujeitos a deteção e classificação. (Cone, “trabalhos na via”, separador de via). 53
Figura 42 Fita de barreira. 53
Figura 43 Exemplo de um ficheiro em formato XML e respetiva comparação com formato YOLO. 55

Figura 44 Visualização do estado do treino, nomeadamente: número de epoch, step, learning rate e losses. 56
Figura 45 Exemplo do mAP calculado a partir do AP de cada classe. 57
Figura 46 Diagrama que representa o comportamento do sistema em situação de teste. 57
Figura 47 Bounding boxes restantes após aplicação de pós-processamento. Pode-se verificar que apenas ficam as bounding boxes com classificação superior a 30%. 58
Figura 48 Representação da aplicação de NMS depois do NMS. Pode verificar-se que, após um processo iterativo de exclusão, permanece aquela que tem melhor classificação. 59
Figura 49 Captura de ecrã do ambiente de simulação construído no CoppeliaSim. 60
Figura 50 Veículo com estrutura semelhante ao real utilizado na prova de Condução de Autónoma que foi utilizado para treino e teste do sistema. 60
Figura 51 Diagrama simplificado (em cima) da comunicação que o ROS deve sustentar, acompanhado do diagrama (em baixo) retirado do rqt_graph que permite visualizar todos os processos em ação necessários para satisfazer os requerimentos propostos pelo diagrama no topo. 61
Figura 52 Diagrama obtido por rqt_graph que mostra a reformulação executada para obedecer aos requisitos da segunda solução. 62
Figura 53 Captura de ecrã do processamento de imagem efetuado pela segunda solução. As 16 linhas estão numeradas da direita para a esquerda em conformidade com a orientação do círculo unitário. 63
Figura 54 Captura de ecrã do ambiente de simulação integrando a segunda solução. Os sensores 8 e 9 detetam o target. 64
Figura 55 Captura de ecrã do processamento de imagem que é feito, pela primeira solução, por cima de uma frame captada pela câmara do veículo. 67
Figura 56 Visualização da loss no 2º treino e na respetiva validação. 72
Figura 57 Visualização da loss obtida na fase de treino e na fase de validação para o 9º treino. 73
Figura 58 Visualização da loss na fase de treino e teste para o 11º treino. 74
Figura 59 Visualização da loss para a fase de treino e validação num teste experimental com data augmentation desligado. 75
Figura 60 Visualização da loss nas fases de treino e validação do 17º treino. 75
Figura 61 Exemplo de imagem inserida no dataset com o separador de via na perspetiva mais comum tendo em conta o ângulo de visão normal de um condutor quando dirige um veículo. 76
Figura 62 Imagens retiradas de um vídeo de exemplificação do funcionamento do sistema na deteção dos sinais de obras na estrada. 77

Figura 63 Gráficos correspondentes à fase de treino e teste do sistema, respetivamente, relativos à primeira solução implementada. 81
Figura 64 Capturas de ecrã do ambiente de simulação para três situações distintas de percursos que o veículo tem de percorrer: Reta, Curva, Contracurva. 82
Figura 65 Gráficos alusivos ao treino e teste do sistema, respetivamente, na situação onde o percurso é uma reta. 83
Figura 66 Gráficos alusivos ao treino e teste do sistema, respetivamente, na situação onde o percurso é uma curva. 83
Figura 67 Gráficos alusivos ao treino e teste do sistema, respetivamente, na situação onde o percurso é uma contracurva. 84
Figura 68 Demonstração do percurso concluído pelo veículo 100% autónomo utilizando o sistema desenvolvido neste projeto. 85

ÍNDICE DE TABELAS

Tabela 1 Principais conceitos importantes de introdução ao Reinforcement Learning. 43

Tabela 2 Exemplo de uma matriz enviada para o DDPG. 64

Tabela 3 Resultados obtidos para o treino inicial. 71

Tabela 4 Resultados obtidos para o segundo treino (com AP obtido para as diferentes classes). 72

Tabela 5 Resultados obtidos no 9° treino (com AP obtido para as diferentes classes)......................... 72

Tabela 6 Resultados obtidos para os 10° e 11° treinos (com AP obtido para as diferentes classes). ... 73

Tabela 7 Resultados obtidos no 12° treino (com AP obtido para as diferentes classes)....................... 74

Tabela 8 Resultados obtidos nos treinos 16 e 17 (com AP obtido para as diferentes classes)............. 75

Tabela 9 Teste a diferentes valores de epochs. 78

Tabela 10 Teste a diferentes valores de Learning Rate. 78

Tabela 11 Teste a diferentes valores de Batch Size. 79

Tabela 12 Teste a diferentes valores de Leaky ReLU. 79

Tabela 13 Teste a diferentes valores de Kernel Regularizer.. 79

Tabela 14 Teste a diferentes valores de divisão de dataset. 80

NOMENCLATURA

API – Application Programming Interface

ANN – Artificial Neural Network

CNN – Convolutional Neural Network

CPU – Central Processing Unit

DDAC – Deep Deterministic Actor Critic

DDPG – Deep Deterministic Policy Gradient

DQN – Deep Q-Network

DNN – Deep Neural Network

FNN – Feed-Forward Neural Network

FPS – Frames Per Second

GIoU – Generalized Intersection over Union

GPU – Graphic Processing Unit

ILSVRC – ImageNet Large Scale Video Recognition Competition

IoU – Intersection over Union

mAP – mean Average Precision

R-CNN – Region Convolutional Neural Network

RNN – Recurrent Neural Network

ROS – Robot Operating System

SSD – Single Shot Multibox Detector

TORCS – The Open Racing Car Simulator

YOLO – You Only Look Once

YOLOv3 – You Only Look Once v3

YOLOv3-tiny – You Only Look Once v3-tiny

1. INTRODUÇÃO

1.1 Descrição do problema

Nos últimos anos têm sido desenvolvidos algoritmos de condução autónoma que visam proporcionar a circulação de veículos autónomos respeitando a regulamentação do código da estrada em condições normais. Porém, em algumas situações, é necessário existir sinalização temporária que modifique o regime normal de utilização da via. Nestes casos, a sinalização temporária sobrepõe-se à sinalização já existente definindo um novo conjunto de regras e indicações que suspendem o regulamento em vigor. Neste contexto, surgiu a necessidade de soluções que permitam a veículos autónomos a identificação e correta atuação face a estas situações de sinalização temporária.

1.2. Objetivos

O objetivo deste projeto consiste na utilização de dois algoritmos de inteligência artificial: o primeiro baseado em *Supervised Learning* e *Deep Learning* que é responsável pela identificação e respetiva classificação das várias sinalizações temporárias; o segundo, baseado em *Deep Reinforcement Learning*, é um sistema que através de aprendizagem por interação com o ambiente permite a movimentação de um veículo em zonas de sinalização temporária. Com a integração dos dois algoritmos pretende-se que o veículo seja capaz de ter um comportamento autónomo, respeitando todas as regras de trânsito numa versão controlada que simula a via pública em condições extraordinárias de sinalização temporária.

1.3. Proposta

Propõe-se o desenvolvimento de software com auxílio de ferramentas de inteligência artificial apropriadas, como o *Tensorflow* e o *Keras*. Neste sentido, o principal objetivo pode ser cumprido, sem a necessidade de dispositivos de hardware para teste em situações reais. Para satisfazer essa condição, são utilizados ambientes de simulação, especificamente construídos para este e mais projetos semelhantes, para darem veracidade aos mesmos.

O primeiro algoritmo, além de testado em simulação, é também testado em situação real porque nada interfere com a segurança da via pública nem a própria segurança do condutor, já que não requer a sua

atenção ou intervenção. O segundo algoritmo, por outro lado, pode interferir com a segurança quer do condutor, quer dos veículos que o rodeiam, pelo que foi estabelecido apenas o seu teste em simulação.

1.4. Motivação

Muito recentemente, começaram a ser desenvolvidos projetos que visam impulsionar soluções para este problema, onde se testaram algoritmos que permitem efetuar a deteção de sinais temporários num ambiente controlado e a atuação do veículo face à presença dos mesmos, nomeadamente veículos de corrida, da competição *Formula Student*. Uma grande parte das soluções não engloba dois tipos de redes neuronais distintas para endereçar o problema ou até mesmo não utilizam *Reinforcement Learning* para o controlo do movimento do veículo. Assim, este projeto torna-se relevante porque, além de permitir aprofundar o conhecimento na área da Robótica, envolve matérias de alto valor científico como a Visão Por Computador e a Inteligência Artificial para complementar o conhecimento necessário no desenvolvimento de projetos com vista à inovação na área promissora da condução autónoma.

1.5. Estrutura da dissertação de mestrado

O documento divide-se em seis capítulos: a introdução, que dá a conhecer a motivação do projeto; a revisão da literatura, para dar a conhecer os temas e matérias aqui utilizados bem como uma revisão e exemplos de aplicações práticas; os fundamentos teóricos, que explicam com detalhe o fundamento das matérias principais abrangidas pelo projeto; o modelo e simulação, que demonstram todo o processo de ligação e configuração dos dois algoritmos utilizados; os resultados, o capítulo mais importante que descreve ao leitor os resultados atingidos pela construção do projeto e onde realmente se discute o comportamento do sistema implementado, os seus pontos fortes e o seus pontos fracos; as conclusões, um breve sumário do que foi possível alcançar com o trabalho desenvolvido.

2. Revisão da literatura

Este capítulo contém uma revisão da literatura na área da inteligência artificial, que é o objeto de estudo principal deste projeto. Os temas relativos à inteligência artificial abordados são *Machine Learning*, *Deep Learning* e as suas aplicações na Robótica. Foram escolhidos sob o critério de serem os mais relevantes para este trabalho e que, portanto, são merecedores de uma revisão. Ao longo do texto, pode-se verificar que a estrutura de apresentação de cada um dos temas segue o seguinte critério: uma breve introdução ao conceito; uma referência aos métodos mais utilizados em cada tema; uma breve referência às áreas ou organizações, onde têm vindo a ser desenvolvidos projetos tendo por base este tema.

2.1. Inteligência Artificial

A inteligência artificial é uma área de estudo dedicada a tornar as máquinas "inteligentes", onde a inteligência é o atributo que possibilita que uma entidade funcione apropriadamente e com precaução no seu meio envolvente [1]. Embora o termo tenha surgido em 1956, só passados vários anos se popularizou sendo parte desse motivo proveniente do avanço progressivo na área das ciências de computação, o que culminou num aumento gradual dos recursos computacionais, indispensáveis para o estudo e implementação dos sistemas de inteligência artificial. Este ramo, das ciências da computação, integra múltiplas tecnologias que foram surgindo ao longo dos anos e aplicações em diversas áreas da ciência, que são a base de gigantes tecnológicas como a Google ou de ferramentas como o Bing da Microsoft [2].

2.1.1. Machine Learning

Machine learning é, segundo Arthur Samuel (1959), um campo da inteligência artificial que dá aos computadores a capacidade de aprenderem a fazer uma tarefa sem serem pré-programados com instruções específicas para o fazerem. Mais especificamente, são autonomamente capazes de reconhecer padrões num conjunto de dados. Uma propriedade que caracteriza os sistemas baseados em *Machine Learning* é a capacidade de aperfeiçoamento da realização das tarefas ao longo do tempo. Este aperfeiçoamento depende de inúmeros fatores, entre eles, da quantidade de dados disponibilizada para o treino da máquina bem como da qualidade dos mesmos e da qualidade e seleção apropriada de características relevantes para o reconhecimento de padrões. Em alguns algoritmos de *Machine Learning* é possível obter previsões a partir do reconhecimento de padrões resultante do treino dos algoritmos, tendo um nível de *accuracy* associado.

Machine Learning pode dividir-se em três principais categorias (Figura 1), *Supervised Learning*, *Unsupervised Learning* e *Reinforcement Learning* [3].

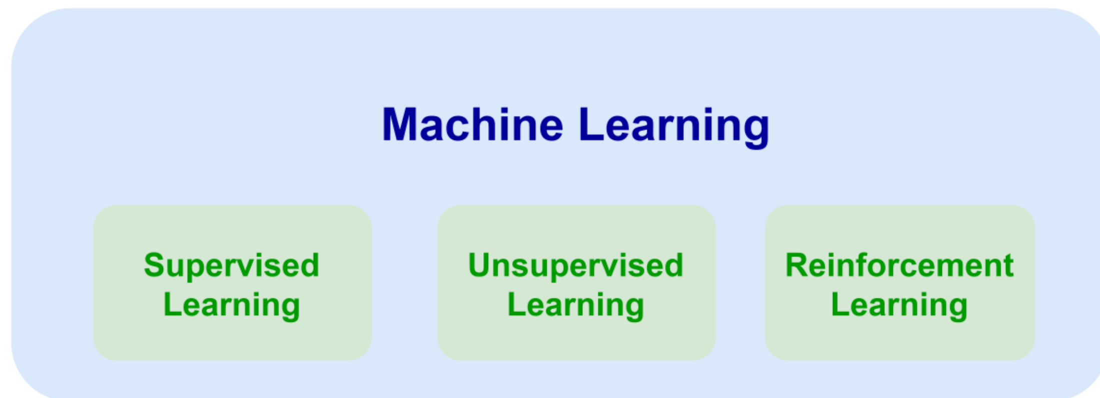


Figura 1 Representação dos principais 3 campos de Machine Learning.

Supervised Learning é um ramo de *Machine Learning* que aprende com exemplos. O nome *Supervised Learning* é fundamentado na ideia de que o treino deste tipo de algoritmos é semelhante a ter um professor que supervisiona todo o processo [4], por outras palavras, são utilizados dados previamente rotulados para treinar os algoritmos. Os dados rotulados constituem a entrada do algoritmo e são um objeto fundamental para se encontrar uma função capaz de rotular os outros dados idênticos a esses com um bom nível de *accuracy*. Este ramo do *Machine Learning* divide-se em duas principais categorias: a classificação, quando o sistema prevê dentro de um número finito de classes qual a classe associada a dados desconhecidos; a regressão, quando a previsão do sistema é um valor numérico real. A sua aplicação abrange áreas como reconhecimento ótico de caracteres, reconhecimento de fala, classificação de imagens, tradução de línguas, geração de sequências, previsão de sintaxe e deteção de objetos [5]. Esta última é a aplicação de maior interesse e relevância para este estudo científico e por esse motivo é bastante abordada nos capítulos seguintes.

Uma base de dados vastamente conhecida na comunidade científica utilizada pelos principiantes em *Supervised Learning* é a *MNIST*. É baseada num conjunto de 60000 imagens para treino juntamente com 10000 imagens para teste, recolhidas pelo *National Institute of Standards and Technology* (*NIST*) nos anos de 1980. É usada para verificar se os algoritmos funcionam como previsto e interpretada como o ponto de partida no mundo do *Deep Learning* [5].

Unsupervised Learning é, por outro lado, uma tecnologia que permite abordar problemas com pouca ou nenhuma ideia do que os resultados devem aparentar e, portanto, não são fornecidas *labels* [6]. Dependendo do tipo de problema em estudo, este processo organiza os dados nas seguintes formas: *Clustering*, *Anomaly detection*, *Association* e *Autoencoders*. É bastante aplicada na descoberta de padrões em grandes bases de dados ou classificação dos dados em categorias para as quais não foram

explicitamente treinados [7]. Por não existirem dados rotulados, torna-se mais complexo medir a *accuracy* deste tipo de algoritmos, o que só por si remete esta solução para outros temas de estudo que não este e, portanto, não é a abordagem usada para este projeto.

Reinforcement Learning, é uma abordagem computacional para a automatização da aprendizagem orientada a um objetivo ou decisão [8]. Esta abordagem difere das restantes por dar ênfase à aprendizagem por intermédio direto de um agente no seu meio ambiente, onde este será mais ou menos recompensado, consoante o seu desempenho. Sucintamente, *Reinforcement Learning* tem um objetivo muito bem definido: maximizar a *reward* total [9], resultante da boa execução das suas tarefas.

No contexto da Robótica, por exemplo, foi desenvolvido um estudo no âmbito de treinar um robô para devolver uma bola de ténis de mesa que esteja junto à rede [10]. Neste caso específico, o robô terá em conta algumas variáveis dinâmicas que especificam a posição e velocidade da bola, bem como a posição das juntas do robô e a sua velocidade. Uma das ações possíveis a executar por parte do mesmo poderia ser definir qual o valor do binário enviado para o motor para atingir a aceleração desejada. Este comportamento é levado a cabo por uma função, à qual se pode chamar de *policy* [11]. Um dos problemas do algoritmo é então encontrar a *policy* que otimize a longo prazo a *reward* obtida, e então pode-se dizer que um algoritmo de *Reinforcement learning* é aquele que é desenhado para encontrar o que seria o mais próximo de uma *optimal policy* [12].

2.1.2. Deep Learning

Deep Learning deriva de *Machine Learning* (Figura 2) e é a mais recente forma de aprender representações a partir de dados, que dá ênfase em aprender por intermédio de neurónios de onde são extraídas essas representações. Uma representação entende-se como uma maneira diferente de olhar, representar ou codificar dados [5]. A estrutura base de *Deep Learning* são conjuntos de neurónios agrupados em camadas sucessivas [13].

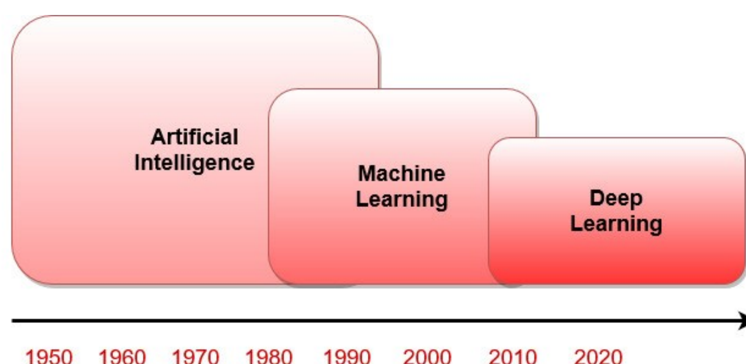


Figura 2 Ilustração do enquadramento temporal dos métodos de Deep learning.

As redes neuronais foram desenhadas no intuito de simular o comportamento do cérebro humano na resolução de problemas [14], quer pela sua estrutura, quer pelo seu método de processar informação. Dentro das redes neuronais existem algumas variantes. As *Deep Neural Networks* (*DNN*), utilizadas em *Deep Learning*, são uma evolução das *Artificial Neural Networks* (*ANN*), que surgiram, na sua primeira versão, em 1948 por Donald Hebb [15]. A diferença entre ambas é o número de *hidden layers* em cada uma (Figura 3) [16].

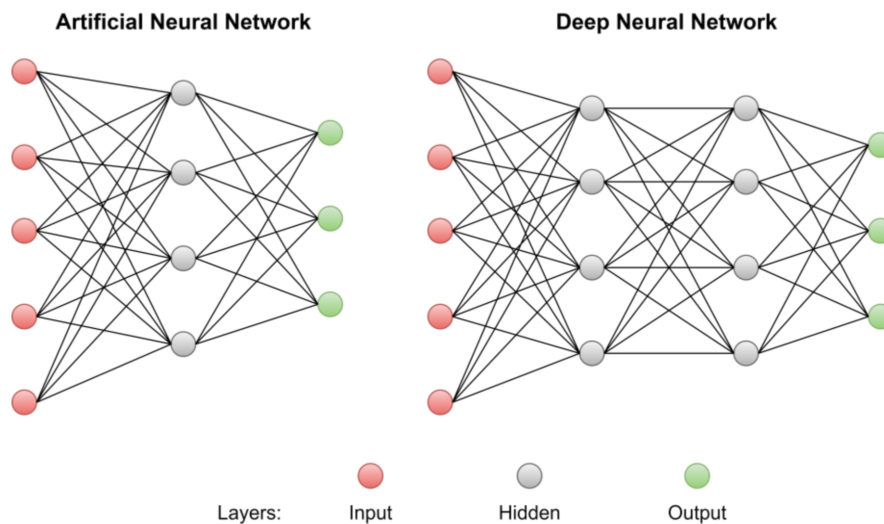


Figura 3 Representação de uma Artificial Neural Network e uma Deep Learning Neural Network.

A unidade de processamento básica de uma rede neuronal é o neurónio. Um agrupamento de neurónios forma aquilo a que se chama de camada e um conjunto de camadas forma a rede neuronal. A soma das variáveis de entrada do neurónio é processada por uma função não-linear, também chamada de função de ativação, que dará origem à variável de saída do neurónio (Figura 4). Algumas funções de ativação conhecidas são *Sigmoid*, *Tansig* e *ReLU*. A força das conexões entre diferentes neurónios das camadas adjacentes é chamada de *weight*, *m* [17]. À medida que a aprendizagem decorre, o valor dos *weights* vai variando, alterando-se o nível de influência que as variáveis de entrada têm na variável de saída dos neurónios. Por exemplo, quanto mais forte o *weight*, maior será a influência que o valor de um neurónio tem no neurónio do outro lado da conexão[18].

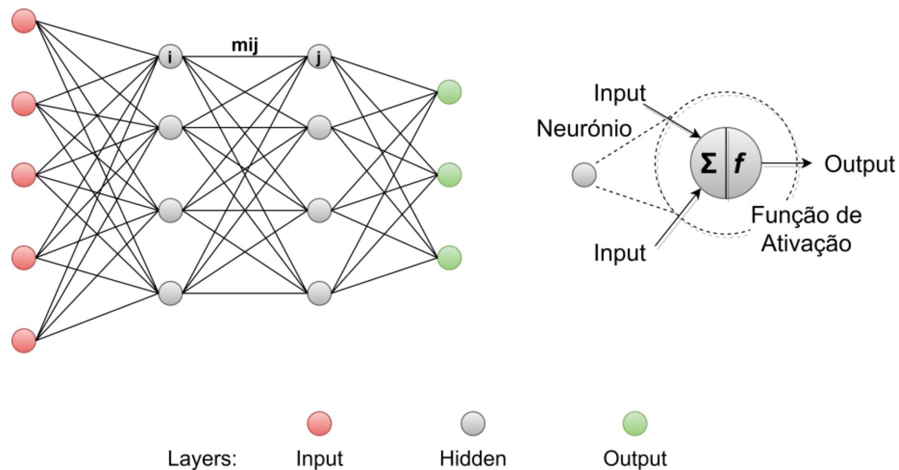


Figura 4 Estrutura de uma rede neuronal artificial. m_ij representa o weight entre o neurónio "i" e o neurónio "j".

As redes neuronais têm sido bastante aplicadas em diversos sistemas para resolução de problemas complexos onde se lidam com grandes quantidades de informação e há pouco conhecimento de uma possível solução generalizada. Nomeadamente em modelos económicos, sistemas de alerta prévio, reconhecimento de texto manuscrito, reconhecimento de imagem, reconhecimento de voz, reconhecimento de padrões, síntese de fala, sistemas biométricos, tradução de línguas, entre outros [18].

Tipicamente as redes neuronais podem ainda ter várias estruturas. Alguns exemplos são *Feed-Foward Neural Networks* (*FNN*), *Recurrent Neural Networks* (*RNN*), *Long Short Term Memory* (*LSTM*) e *Convolutional Neural Networks* (*CNN*) [5]. Para este projeto, as CNN são cruciais para a execução do reconhecimento de objetos de sinalização temporária de obras na estrada.

Em 2011, Dan Giresan do *IDSIA* (*Dalle Molle Institute for Artificial Intelligence Research*), laboratório de inteligência artificial na Suíça, começou a ganhar competições académicas sobre classificação de imagens recorrendo a *DNN* treinadas por unidades de processamento gráficas (*GPU*), tendo sido considerado o primeiro sucesso prático de *Deep Learning* moderno. O melhor momento foi marcado pela entrada do grupo Hintons numa competição anual de larga escala chamada *ImageNet Large Scale Visual Recognition Competition* (*ILSVRC*). Esta era considerada difícil pois consistia na classificação de imagens de resolução alta em 1000 categorias diferentes depois de serem treinadas 1,4 milhões de imagens. A competição foi então, em 2012, dominada pela introdução de *CNN* com um vencedor, em 2015, a obter um nível de *accuracy* de 96,4%, o que culminou no desafio principal da competição ser completamente ultrapassado. Desde então, as *DNN* têm sido mais procuradas para as tarefas de visão por computador. Por exemplo, os cientistas do *CERN*, *European Organization for Nuclear Research*, durante vários anos usou métodos baseados em árvores de decisão para analisar partículas do detetor *ATLAS* no *Large*

Hadron Colider, mas mudaram para as *DNN* pela sua performance e facilidade em treinar grandes bases de dados [5].

As *CNN* são um tipo de modelo de *Deep Learning* utilizado para processar dados que respeitam um padrão matricial, como por exemplo as imagens, e são desenhadas para, automaticamente e de forma adaptativa, aprenderem conjuntos de características, desde padrões de baixo nível a padrões de alto nível. Tipicamente, são constituídas por 3 tipos de camadas: camadas de convolução, camadas de *pooling*, e camadas totalmente ligadas. As primeiras duas são responsáveis pela extração de características e a terceira mapeia, na saída, as características extraídas bem como a respetiva classificação (Figura 5) [19]. *LeNet*, *AlexNet*, *VG-GNet*, *GoogleNet*, *ResNet* e *ZFNet* são alguns exemplos de algoritmos que tiram partido de *CNN* [20].

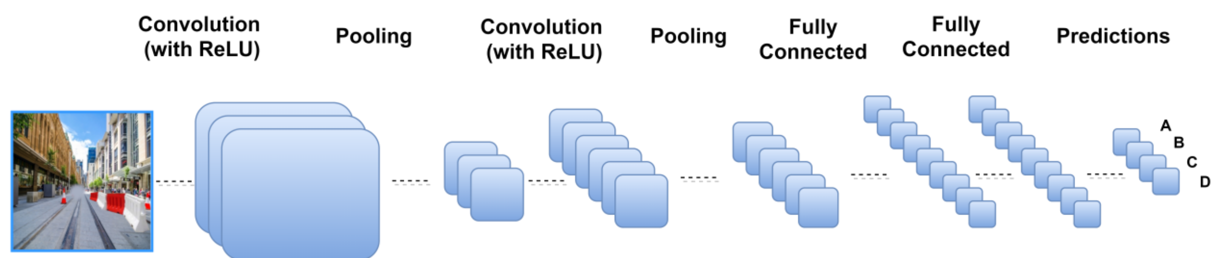


Figura 5 Representação da arquitetura de uma CNN. Neste exemplo, "ReLU" é a função de ativação implementada na rede neuronal.

As camadas de convolução são a parte fundamental das *CNN* por ser onde a maior parte do processamento ocorre [21]. Cada camada possui filtros também conhecidos como *kernels* (Figura 6). Um *kernel* é uma matriz de números inteiros que é aplicada aos valores dos pixéis da imagem de entrada por uma operação de multiplicação. O resultado é somado de forma a obter-se um único valor que representa uma célula, ou um pixel, da matriz da imagem de saída [22].

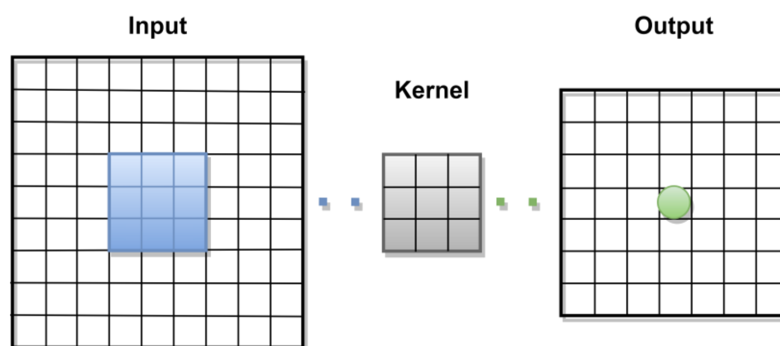


Figura 6 Representação da aplicação de um kernel numa matriz e respetiva matriz de saída.

As camadas de *pooling* são responsáveis por reduzir a dimensão das imagens de entrada por processos de escolha do valor mais ocorrido ou o valor máximo encontrado por exemplo (Figura 7). O objetivo é reduzir, passo a passo, a dimensão dos dados para se obter um número inferior de parâmetros e, consequentemente, uma menor complexidade do modelo permitindo controlar melhor o fenómeno de

overfitting [21] – fenómeno que ocorre quando o modelo é demasiado complexo ou é treinado em demasia, aprendendo informação irrelevante que resultará numa boa performance no treino e numa má performance no processo de avaliação e teste [5].

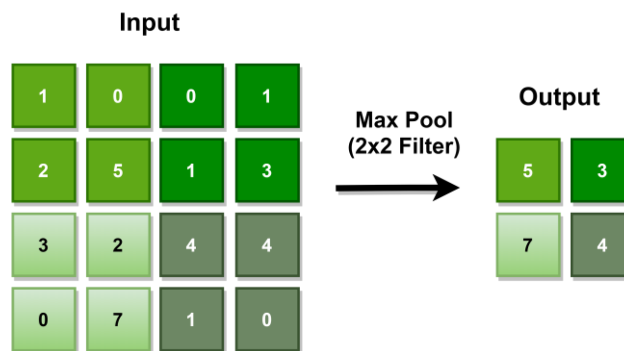


Figura 7 Exemplo de uma operação max-pooling de stride 2.

Em último estão as camadas totalmente ligadas. Num modelo básico de *CNN*, as *features* geradas pela última camada de convolução representam uma porção da imagem de entrada porque o seu campo recetivo não integra toda a dimensão espacial da imagem. Torna-se necessária a introdução de camadas totalmente ligadas e estas podem abranger a maior parte dos parâmetros da rede [23]. Os neurónios nas camadas totalmente ligadas, têm conexões totais com todos os da camada anterior (figura 8). As ativações dos mesmos podem ser processadas por multiplicação de matrizes seguida do *offset* a que se chama de *bias* [21],[24].

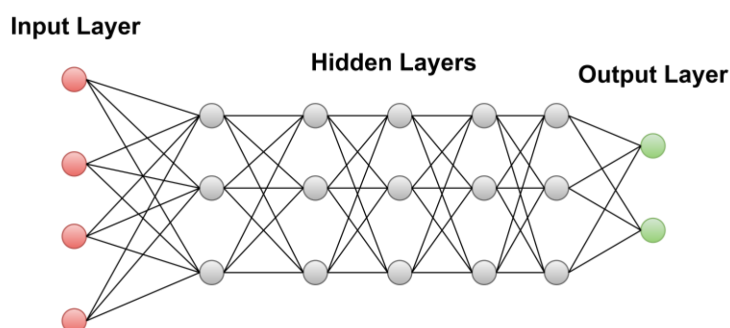


Figura 8 Estrutura de uma rede de camadas totalmente ligadas.

A aplicação das *CNN* é bastante frequente no campo da Visão por Computador. Desde o processamento de linguagem, categorização de texto e classificação de frases, ao reconhecimento de imagem [21]. Para este projeto é explorada esta última aplicação. Como foi visto antes, a base de dados *MNIST* tem sido vastamente utilizada para os que entram no processo de aprendizagem de redes neuronais aplicadas ao reconhecimento de imagem, com especial uso das *CNN* em grande parte dos casos.

Dada a dimensão da utilização destas redes neuronais, têm surgido *frameworks* para simplificar a utilização das redes, *Tensorflow* e *Keras* são exemplos de duas bem conhecidas que podem até funcionar em conjunto.

Tensorflow é uma biblioteca de alto nível da Google utilizada para a implementação de redes neuronais [25] que possui funções nativas que permitem a criação de um modelo de redes neuronais, como por exemplo, a criação de camadas para as *CNN*, com as respetivas variáveis bem como o dimensionamento de *kernels* para as camadas [16]. O treino de alguns modelos complexos pode ser uma tarefa exaustiva e demorosa. Esta dificuldade foi contornada com a entrada do *Tensorflow* e um investimento por parte da *Nvidia* no setor de *Machine Learning*, que levou ao fabrico de *GPUs* com capacidade de serem utilizadas pelo *Tensorflow* [5]. Foi apresentada a opção de executar os programas diretamente a partir do processamento da *GPU*, se esta for *Nvidia* e tiver apta para o fazer, o que significa que a performance de treino de redes neuronais complexas e com enormes bases de dados aumenta significativamente. O *Tensorflow* dispõe ainda de uma *API* chamada *Tensorboard* que é fundamental para a visualização de alguns resultados provenientes dos processos de treino e validação do modelo, como por exemplo, o valor da *accuracy* e o valor da *loss* [16].

Desde que foi lançado, o *Tensorflow* tem sido muito utilizado em diversos domínios e tem ganho popularidade e um suporte consistente na comunidade *open-source*, com as mais variadas contribuições de terceiros. Por exemplo, no domínio do JavaScript, nas aplicações em tempo real, na educação e na capacidade de tornar algoritmos mais acessíveis a outros investigadores. Como tal, tem o potencial de alargar ainda mais a comunidade de investigadores na área de *Machine Learning* [26]. De acordo com Jack Clark [27], mais de 15% de todas as pesquisas feitas em *www.google.com* são novas no sistema. Face a isso, a Google utiliza um algoritmo criado pela mesma, o *RankBrain*, que usa o *Tensorflow* para sugerir aos utilizadores palavras ou frases similares com significado idêntico para as partes desconhecidas da frase que está a ser escrita.

Outra área onde a Google aplica *Deep Learning* e o *Tensorflow* é na escrita inteligente de respostas por e-mail [28]. O sistema capaz de sugerir respostas rápidas inteligentes a um e-mail recebido, usa *RNN* para o processo de perceção da linguagem.

No artigo [29], foi reportado como a Google utiliza *CNN* para o reconhecimento de imagens e tradução de texto automático. É notável a característica de que dispõe a aplicação móvel *Google Translate*, onde permite que os utilizadores capturem uma foto com o telemóvel e é capaz de traduzir em cima da própria imagem qualquer texto presente na mesma. Neste sentido, até sinais de trânsito podem ser traduzidos com a aplicação (Figura 9).

Figura 9 Exemplo da utilização da aplicação móvel para tradução de texto em imagens da Google (Google Translate).

O autor do artigo [30], acredita que não apenas a Google virá a beneficiar com esta ferramenta mas também toda a comunidade científica para abrir novos horizontes para o uso de algoritmos rápidos de inteligência artificial de forma mais generalizada.

Keras, fundada por F. Chollet [5] em 2015, é uma *framework* para *Python* que disponibiliza uma maneira conveniente para definir e treinar grande parte dos modelos de redes neuronais existentes. Permite que o mesmo código seja executado de forma eficiente tanto na *CPU* como na *GPU* e dispõe de uma *API* intuitiva que possibilita a construção de protótipos de modelos de *DNN*. Além disso, dispõe de suporte nativo para as *CNN*, para *RNN* e qualquer combinação de ambas. Suporta também arquiteturas arbitrárias de redes neuronais, querendo isto dizer que o tipo de arquitetura do modelo das redes neuronais fica à escolha do investigador.

À data do lançamento do livro de F. Chollet [5], em 2017, o *Keras*, distribuído por uma licença do *MIT*, já contava com a presença de mais de 200 000 utilizadores, tendo cada vez mais aumentado a sua procura tanto por parte de investigadores académicos como por parte de engenheiros de *start-ups* e grandes empresas como a Google, a Netflix, a Uber, o CERN, a Yelp, a Square, etc.

Este opera sobre o *Tensorflow* [5] (Figura 10), isto é, permite a abstração de grande parte da complexidade que o *Tensorflow* engloba e, portanto, não foi construído para lidar com operações de baixo nível como a manipulação de *tensors* – estrutura de dados utilizada para as entradas e saídas no modelo das redes neuronais – e diferenciação. A vantagem da relação entre os dois está, precisamente, na utilização de ambos, onde se utiliza o *Keras* para abstrair algumas complexidades e se usa o *Tensorflow* quando há necessidade de efetuar algumas operações de mais baixo nível de forma a obter-se a maior eficiência que for possível.

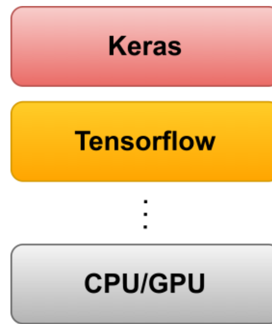
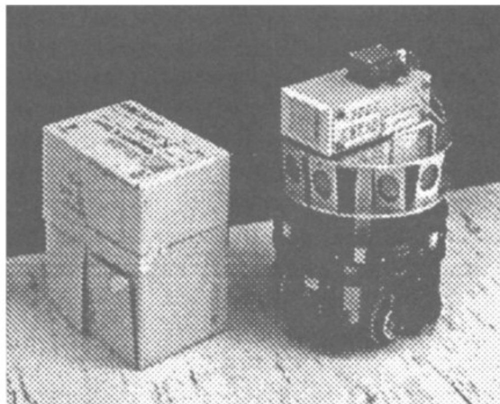


Figura 10 Relação entre Tensorflow e Keras. Desde o nível mais baixo, GPU ou CPU, até ao nível mais alto, onde se situa o Keras.

2.2. Inteligência Artificial na Robótica

A maior parte dos exemplos citados na secção anterior foram desenvolvidos no âmbito da área da robótica e da visão por computador. Desde muito cedo, a inteligência artificial tem um papel fundamental para o avanço da robótica. Na figura 11 encontram-se robôs que são implementados com recurso a métodos de *Reinforcement Learning*. Na alínea a), encontra-se o robô *OBELIX* que é um robô móvel que desempenha a função de empurrar caixas [31]. Na alínea b), o *CMU Yamaha R50* da *Carnegie Mellon University*, onde se integrou um controlador de voo robusto [32].



a) b)

Figura 11 Exemplos de robôs que utilizam inteligência artificial no seu funcionamento. Fonte:[12].

No Laboratório de Automação e Robótica (LAR) da Universidade do Minho têm também sido desenvolvidos robôs que fazem uso de algoritmos de inteligência artificial para atingir os objetivos que lhes foram atribuídos. Ao longo dos últimos anos tem vindo a ser desenvolvido um robô antropomórfico, o *CHARMIE* (figura 12) [33], que é um desafio tecnológico na integração da inteligência artificial para diversas funções que se pretende implementar no robô.

Figura 12 Protótipo do Robô CHARMIE (desenvolvido em 2017). Fonte:[33].

Recentemente, Tiago Ribeiro, aluno de doutoramento e colaborador do LAR, desenvolveu soluções para a navegação de robôs autónomos e desvio de obstáculos, recorrendo a métodos de *Reinforcement Learning*, onde implementou diversos algoritmos [3].

Existem ainda vários projetos desenvolvidos por laboratórios e faculdades internacionais, apresentados nas secções seguintes mais específicos da deteção de objetos e da condução autónoma que são os temas mais relevantes para este projeto.

2.2.1. Deteção de objetos

O principal objetivo da deteção de objetos é detetar todas as instâncias dos objetos de uma classe conhecida como pessoas, carros, animais ou caras numa imagem [34]. Para um objeto de uma determinada classe podem existir múltiplas imagens com as mais variadas características como a tonalidade, a resolução, o ruído, o contraste, a iluminação etc. O desafio é desenvolver algoritmos de deteção que sejam neutros a esse tipo de variações nas imagens e computacionalmente eficientes, de forma a ser possível detetar o objeto pretendido em situações reais, ou seja, não ideais. Parte do sucesso deste método e de outros da área da visão por computador, deve-se à adoção de métodos de *Machine Learning* [35], como por exemplo *Supervised Learning*.

Depois de detetada a instância de um objeto é possível tirar várias conclusões como reconhecer a que classe pertence, prever informação acerca do ambiente ao redor do objeto e muito mais informação contextual [34]. A deteção de objetos tem sido utilizada em interações humano-máquina, robótica de serviço, *smartphones*, segurança, motores de busca e transporte efetuado por veículos autónomos. Neste projeto esta será feita por veículos em movimento e por questões de segurança e de respeito do código da estrada, pretende-se que a deteção de objetos seja a mais rápida possível assemelhando-se a um processamento em tempo real. Como não existem sistemas ideais, é esperada uma redução da

accuracy dos mesmos quanto maior se pretender que seja a sua velocidade de processamento e menor o tempo de resposta. Tendo isto em consideração, investigou-se um sistema que fosse bem conhecido e testado pela comunidade científica na condução autónoma que permitisse um equilíbrio entre a sua *accuracy* e o seu tempo de resposta, mas sempre dando prioridade ao parâmetro tempo de resposta. Após a revisão da literatura foram encontrados algoritmos *state-of-the-art* que já são bastante populares para deteção de objetos em tempo real que são, *R-CNN* (*Region Convolutional Neural Networks*), *SSD* (*Single Shot Multibox Detector*) e *YOLO* (*You Only Look Once*) [16], [37]–[40].

A *R-CNN* [16] foi das primeiras a tirar partido da *CNN* para a deteção de objetos com boa *accuracy*. De uma forma muito breve ela segue o seguinte princípio: gerar potenciais regiões (*region proposals*) executando uma técnica chamada *Selective Search*; cada uma dessas regiões é submetida a uma *CNN* pré-treinada, sendo posteriormente classificada como pertencente ou não ao objeto de interesse; depois de classificada, as potenciais regiões são otimizadas pelo uso de um modelo de regressão linear. A *R-CNN* foi, ao longo do anos, resultando em várias variantes da mesma, *Fast R-CNN* e *Faster R-CNN* como se pode verificar em [39].

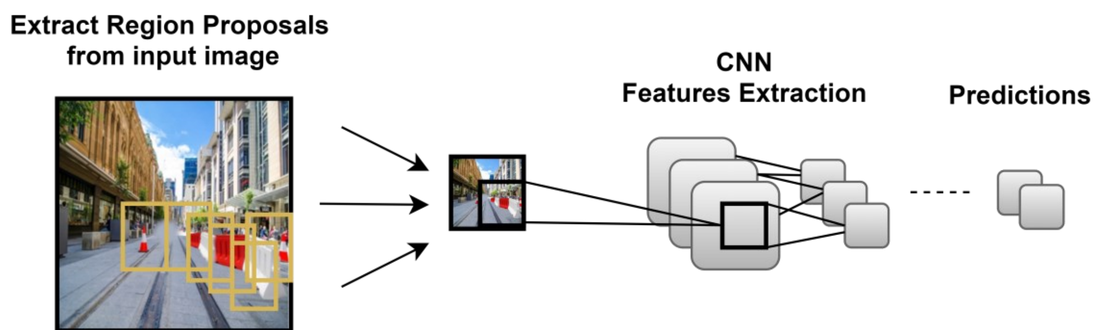


Figura 13 Arquitetura base de uma RCNN.

A *SSD* foi desenvolvida na Google por W. Liu et al (2016) [40]. A sua construção tem como base a arquitetura *VGG-16*, contudo descarta as camadas totalmente ligadas como se pode ver na figura 14. É criado um conjunto de *default boxes* para calcular *offsets* e prever a *confidence* de todos os objetos. Quando um objeto é detetado, é gerada uma pontuação (*score*) e a forma da caixa é ajustada para corresponder à forma do objeto. Durante o treino, as *default boxes* são colocadas a coincidir com as *ground truth boxes* e as perdas do modelo são calculadas como a soma ponderada da *confidence loss* e da *location loss* [40].

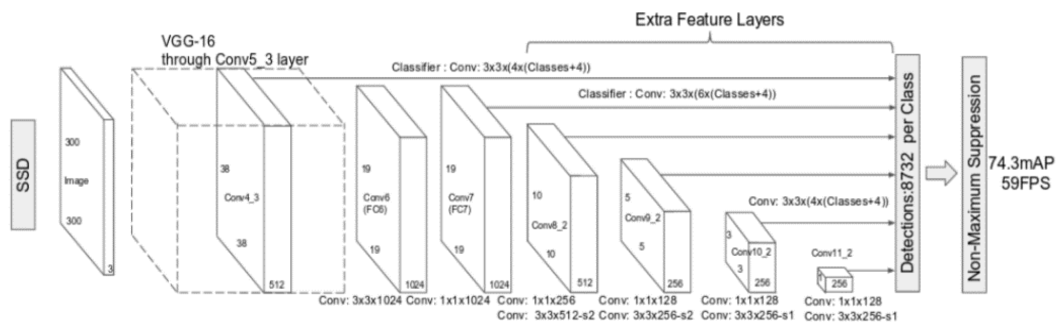


Figura 14 Arquitetura das redes SSD. Fonte:[40].

O modelo *YOLO* foi apresentado por J. Redmon et al.[37] e foi enunciado como uma arquitetura unificada que permite obter valores altos de performance no que diz respeito à velocidade de processamento do mesmo. Neste modelo, uma única rede neuronal prevê *bounding boxes* e *class probabilities* diretamente a partir de imagens completas em apenas uma observação da mesma. A deteção de objetos é feita através de métodos de regressão. A imagem é dividida numa grelha $S \times S$ e para cada célula são previstas *bounding boxes* com a sua respetiva *confidence* e as *class probabilities*, como ilustrado na figura 15.

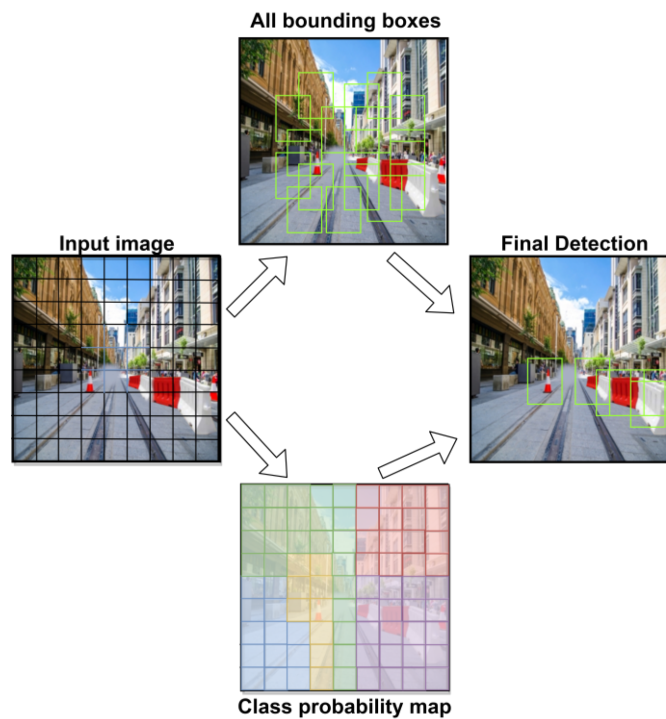


Figura 15 Modelo YOLO.

A arquitetura desta rede (figura 16) é inspirada no modelo *GoogleNet* para classificação de imagens. Possui 24 camadas de convolução e 2 camadas totalmente ligadas, contudo, difere do modelo *GoogleNet* na medida em que não utiliza os *inception modules*, usando, como alternativa, camadas de redução 1 x 1 seguidas de camadas de convolução 3 x 3. A saída da rede corresponde a uma matriz de 7 x 7 x 30 [38].

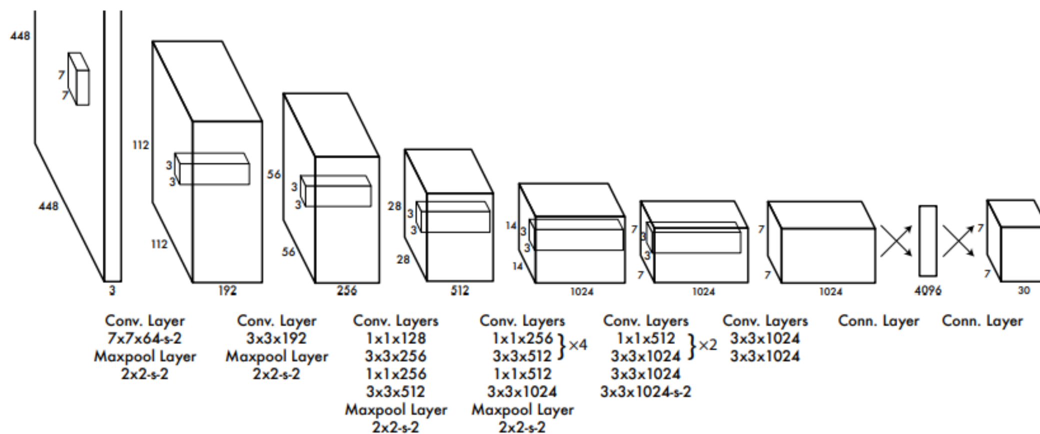


Figura 16 Arquitetura da rede YOLO. Fonte:[37].

Os modelos desta rede disponíveis pelo autor no site, são pré-treinados na competição online *ImageNet* usando como base de funcionamento a *Darknet*, uma ferramenta criada pelo mesmo.

Na publicação de J. Redmon, os resultados apresentados (figura 17) provam que, utilizando o mesmo *dataset*, a versão base da *YOLO* é mais rápida que as variantes da *R-CNN*, obtendo no seu melhor resultado 45 *FPS*. No entanto, a líder em termos de *accuracy* é a a *Faster R-CNN VGG-16*. Foi ainda implementada uma versão da rede *YOLO* com arquitetura *VGG-16* onde se obteve melhor *accuracy*, mas menos velocidade.

Real-Time Detectors	Train	mAP	FPS
100Hz DPM [31]	2007	16.0	100
30Hz DPM [31]	2007	26.1	30
Fast YOLO	2007+2012	52.7	**155**
YOLO	2007+2012	**63.4**	45
Less Than Real-Time			
Fastest DPM [38]	2007	30.4	15
R-CNN Minus R [20]	2007	53.5	6
Fast R-CNN [14]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[28]	2007+2012	73.2	7
Faster R-CNN ZF [28]	2007+2012	62.1	18
YOLO VGG-16	2007+2012	66.4	21

Figura 17 Comparação das métricas resultantes da implementação de sistemas líderes em deteção de objetos, neste caso pretende-se realizar a comparação entre a YOLO e a R-CNN (Fonte: [37]).

Porém quando comparadas as redes *YOLO* e *SSD* , a *YOLO* passa para segundo lugar, com a SSD a obter uma *accuracy* de 74,3% *mAP* com 46 *FPS* (figura 18) [16].

Object detector	*mAP*	*FPS*	*#Boxes*	*Input resolution*
Faster R-CNN (VGG 16)	73.2	7	6000	1000x600
YOLO (VGG 16)	66.4	21	98	448x448
SSD300	74.3	46	8732	300x300
SSD512	76.8	19	24564	512x512

Figura 18 Comparação das métricas resultantes da implementação dos sistemas líderes em deteção de objetos, neste caso pretende-se realizar a comparação entre a YOLO e a SSD (Fonte:[16]).

Tiago Pinto [16], em 2018, verificou que a *SSD* seria então a melhor escolha dadas estas circunstâncias. Contudo, foram posteriormente desenvolvidas variantes da rede *YOLO* que se mostram superiores à concorrência.

A primeira variante da rede *YOLO* original que foi desenvolvida foi a *YOLOv2*. A performance aumentou através da introdução de *Batch Normalization* e mostra-se superior porque permite o aumento da resolução das imagens para o classificador e a previsão de múltiplos objetos por cada célula pela qual a imagem se divide. Além disso, a *YOLOv2* usa um modelo de classificação diferente, o *Darknet-19* [41]. Analisando os resultados obtidos pela comparação do *YOLOv2* com os restantes sistemas (figura 19) pode-se verificar que o YOLOv2 apresenta um bom compromisso entre velocidade e exatidão para vários tipos resolução de imagem. Contudo, para este projeto, a velocidade é o fator mais importante e nesse sentido seria escolhida a resolução de 288 x 288, por apresentar uma *accuracy* de 69 *mAP* e uma velocidade de 91 *FPS*.

Detection Frameworks	Train	mAP	FPS
Fast R-CNN [5]	2007+2012	70.0	0.5
Faster R-CNN VGG-16[15]	2007+2012	73.2	7
Faster R-CNN ResNet[6]	2007+2012	76.4	5
YOLO [14]	2007+2012	63.4	45
SSD300 [11]	2007+2012	74.3	46
SSD500 [11]	2007+2012	76.8	19
YOLOv2 288 × 288	2007+2012	69.0	91
YOLOv2 352 × 352	2007+2012	73.7	81
YOLOv2 416 × 416	2007+2012	76.8	67
YOLOv2 480 × 480	2007+2012	77.8	59
YOLOv2 544 × 544	2007+2012	**78.6**	40

Figura 19 Comparação das métricas resultantes da implementação de sistemas líderes em deteção de objetos, neste caso pretende-se realizar a comparação entre a YOLOv2 e as mais conhecidas (Fonte:[41]).

Junto ao *YOLOv2* foi lançado também o *YOLO9000*, contudo tem algumas limitações na deteção de objetos referidas pelo autor e não é tão utilizado.

O *YOLOv3* é uma versão com melhorias face ao *YOLO*. Melhora a previsão das *bounding boxes*, introduz classificação *multi-label* para ser possível identificar 2 classes num mesmo objeto (exemplo: uma mulher detetada numa imagem é associada à classe mulher e à classe pessoa), melhora a deteção de pequenos

objetos (onde o *YOLO* tinha alguma dificuldade) e usa um modelo de classificação melhorado, o *Darknet-53* [42].

Analisando o gráfico dos resultados obtidos pelo autor (figura 20), verifica-se que o *YOLOv3* para as três resoluções de imagens diferentes em que se apresenta (320, 416 e 608), obtém os tempos de execução mais pequenos e uma *accuracy* similar às restantes redes. Neste quadro o sistema mais apropriado para o projeto seria o *YOLOv3-320* por ter o tempo de execução mais rápido sem perder muita *accuracy*.

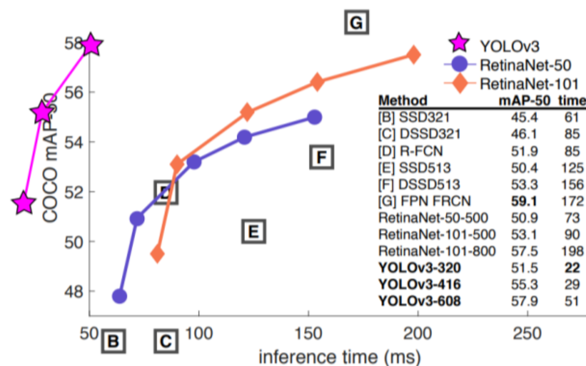


Figura 20 Comparação das métricas resultantes da implementação de sistemas líderes em deteção de objetos, neste caso pretende-se realizar a comparação entre o YOLOv3 e os mais conhecidos (Fonte:[42]).

A pensar na velocidade, o autor foi mais além. Como o *YOLOv3* possui uma *accuracy* aceitável, foi melhorada a sua performance em termos de velocidade para sistemas com pouca capacidade de processamento dando origem a uma outra versão, o *YOLOv3-tiny* [44].

O *YOLOv3-tiny* é uma versão reduzida das restantes versões *YOLO* que consegue velocidades até aproximadamente 442% mais rápidas, contudo, perde alguma *accuracy* em prol deste aumento de velocidade. Esta perda de *accuracy* é pouco relevante comparativamente ao ganho obtido de velocidade que é fundamental para deteção em tempo real, fazendo deste um sistema mais eficiente que o *YOLOv3* na deteção e localização de objetos e respetiva identificação da classe em tempo real [44].

Observando a figura 21 verifica-se que o *YOLOv3-tiny* é capaz de uma velocidade de 220 *FPS* e uma *accuracy* de 33.1 *mAP*, eliminando (quase toda) a concorrência em termos de velocidade. Ainda na figura, encontra-se a rede *Tiny YOLO*, uma versão reduzida da *YOLO* original, que faz frente à *YOLOv3-tiny* obtendo 244 *FPS*, contudo, a *accuracy* é demasiado baixa.

Model	Train	Test	mAP	FLOPS	FPS	Cfg	Weights
SSD300	COCO trainval	test-dev	41.2	-	46		link
SSD500	COCO trainval	test-dev	46.5	-	19		link
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40	cfg	weights
Tiny YOLO	COCO trainval	test-dev	23.7	5.41 Bn	244	cfg	weights
SSD321	COCO trainval	test-dev	45.4	-	16		link
DSSD321	COCO trainval	test-dev	46.1	-	12		link
R-FCN	COCO trainval	test-dev	51.9	-	12		link
SSD513	COCO trainval	test-dev	50.4	-	8		link
DSSD513	COCO trainval	test-dev	53.3	-	6		link
FPN FRCN	COCO trainval	test-dev	59.1	-	6		link
Retinanet-50-500	COCO trainval	test-dev	50.9	-	14		link
Retinanet-101-500	COCO trainval	test-dev	53.1	-	11		link
Retinanet-101-800	COCO trainval	test-dev	57.5	-	5		link
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45	cfg	weights
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35	cfg	weights
YOLOv3-608	COCO trainval	test-dev	57.9	140.69 Bn	20	cfg	weights
YOLOv3-tiny	COCO trainval	test-dev	33.1	5.56 Bn	220	cfg	weights
YOLOv3-spp	COCO trainval	test-dev	60.6	141.45 Bn	20	cfg	weights

Figura 21 Comparação das métricas resultantes da implementação de sistemas líderes em deteção de objetos em tempo real, neste caso pretende-se realizar a comparação entre o YOLOv3-tiny e os mais conhecidos (Fonte:[43]).

Um estudo conduzido pela *Seoul National University* [45], com o objetivo de encontrar o melhor detetor de objetos para a condução autónoma, dentro dos que são considerados os melhores em termos de velocidade (os *One-Stage Detectors*), concluiu que o *YOLOv3* é um algoritmo aceitável para a condução autónoma (Figura 22). Nos mesmos resultados verifica-se ainda que o algoritmo *YOLOv3-tiny* é uma boa alternativa para ser usado em sistemas com pouca capacidade de processamento.

Algorithm	KITTI mAP (%)	KITTI fps	BDD mAP (%)	BDD fps	Input resolution
SSD [5]	61.3	28.9	14.1	23.1	512 × 512
RefineDet [6]	84.4	27.8	17.4	22.3	512 × 512
RFBNet [7]	73.4	39.2	14.5	39.0	512 × 512
YOLOv2 [9]	64.8	85.5	7.11	83.1	512 × 512
YOLOv3 [10]	80.5	43.6	14.9	42.5	512 × 512
YOLOv3-tiny [10]	64.1	217.3	5.94	216.0	512 × 512

Figura 22 Resultado obtido para o treino em 2 bases de dados diferentes, KITTI e BDD. Fonte:[45].

É relevante referir que existe uma nova versão YOLOv4 [46] mas como é bastante recente não é uma opção para este trabalho pelo facto de ainda não ter sido consideravelmente utilizada pela comunidade. Outro aspeto importante é o facto de esta versão não ter sido lançada pelo autor e fundador das originais. Por último, ainda que fosse aqui considerada seria também descartada pelo facto de não apresentar velocidades superiores à versão *YOLOv3-tiny*.

2.2.2. Condução Autónoma

Uma grande parte dos algoritmos desenvolvidos para dar soluções aos problemas da condução autónoma tem uma componente baseada na deteção de objetos. Um veículo só é capaz de se

movimentar com segurança e respeitando o código da estrada se, primeiramente, tiver conhecimento da sinalização de trânsito presente no seu meio envolvente.

A abordagem realizada anteriormente sobre a deteção de objetos, culmina com a primeira componente fundamental do sistema porque algumas dessas tecnologias são bastante aplicadas na área da condução autónoma para resolver um dos seus maiores problemas, nomeadamente a deteção de sinais de trânsito [47]–[57].

A segunda componente fundamental na condução autónoma, é a movimentação do veículo em segurança na via pública. É sabido que os algoritmos de *Reinforcement Learning* estão bem preparados para lidar com problemas encontrados nesta geração de robôs autónomos [60]. Na introdução deste subcapítulo foram revistas algumas aplicações de *Reinforcement Learning* em robôs que lhes permite ter um comportamento autónomo. Da mesma maneira, podem ser introduzidos comportamentos completamente autónomos num veículo, durante toda a sua circulação. Neste sentido, esta componente será também implementada recorrendo a métodos de *Reinforcement Learning* [50], [58]–[69].

Geralmente, em *Reinforcement Learning*, são utilizadas 2 abordagens diferentes que se chamam *model-based* e *model-free*. A abordagem *model-free* consiste no processo de aprendizagem de uma *policy* que se baseia nas recompensas obtidas apenas pela interação no ambiente envolvente. Já a *model-based* diz respeito a um processo onde o agente conhece o modelo de um dado ambiente e o utiliza para planear e melhorar a *policy*. Em ambientes que requerem o uso de *ANN* (para modelos não-lineares) a aprendizagem *model-free* tende a obter maior sucesso [3].

A combinação de *Reinforcement Learning* com *Deep Learning* deu origem ao termo *Deep Reinforcement Learning* [69]. O método de *Deep Reinforcement Learning* aplicado no *AlphaGo*, proposto pela empresa DeepMind (da Google), é uma junção de *Deep Q-Network* (*DQN*) com *Q-Learning* [51].

Marina [61] fez um estudo onde mostrou os modelos que se tornaram o estado da arte na área do *Reinforcement Learning* e têm um futuro promissor para a condução autónoma, destacando os algoritmos baseados em *Q-Networks*.

Yurtsever et. al [48], apresentam duas abordagens à condução autónoma, uma *modular* e outra *end-to-end* (Figura 23). Os autores indicam que a abordagem *modular* como em grande parte de problemas de engenharia torna-se menos complexa de resolver pelo simples facto de que o problema é dividido em subproblemas mais fáceis de resolver. Já a abordagem *end-to-end* consiste em tratar todo o problema como um só. Neste projeto, logo desde ínicio optou-se por uma abordagem *modular*, visto que será utilizado um módulo com um algoritmo de *Supervised Learning* para resolver o problema da deteção de sinalização de trânsito, um módulo para o algoritmo de *Reinforcement Learning* para resolver o problema

do controlo do comportamento do veículo na estrada e um terceiro que diz respeito ao atuador que se situa no ambiente de simulação. Esta abordagem *modular* traz vantagens como a integração de algoritmos e funções, sem exigir uma completa reestruturação de todo o sistema.

Ainda no mesmo estudo, o autor faz referência ao uso de algoritmos de *Deep Reinforcement Learning*, nomeadamente *Deep Q-Networks* para a aprendizagem da melhor forma de condução por parte do veículo.

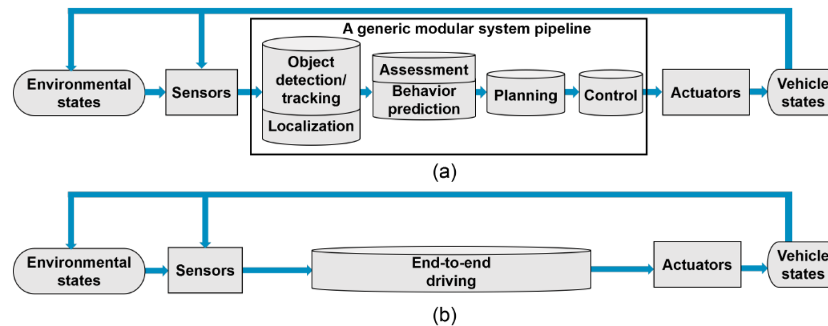


Figura 23 Esquema de 2 abordagens comuns a problemas de condução autónoma. a) Sistema modular e b) Sistema end-to-end. Fonte [48]

Kiran B. et al [61], na sua pesquisa de soluções para o desenvolvimento de sistemas de condução autónoma utilizou também uma abordagem *modular* para executar algumas das tarefas mais importantes na condução autónoma. Na figura 24, encontra-se uma lista dessas tarefas onde a tarefa de seguir a trajetória das linhas da estrada, *Lane Keep*, é a mais coincidente com a segunda componente do trabalho. A diferença será na deteção de objetos porque o sistema terá de seguir a trajetória de cones em vez de linhas e, a partir daí, tomará decisões similares às que tomaria no caso das linhas. Por exemplo, calcular a distância aos cones e manter-se a uma distância mínima de forma a não colidir e/ou ultrapassar os mesmos, seguindo a trajetória formada pelos mesmos. O mesmo autor apresenta uma simulação com um veículo, integrando *Deep Reinforcement Learning*, que percorreu autonomamente um troço de estrada de 250m seguindo a trajetória das linhas da via onde circulou.

AD Task	Description and Utilization of (D)RL
Motion Planning	Learn to plan trajectories dynamically and optimize cost function to provide smooth control behavior of vehicle. Inverse RL is utilized to learn optimal reward function (or shaping) from experts. Authors propose to learn a heuristic function for the A^* algorithm using a DQN over image-based input obstacle map [84]
Overtaking	Authors [85] propose Multi-goal RL (MGRL) framework to learn overtaking policy while avoiding collisions & maintain steady speed.
Intersections/Merging	Ego-vehicle required to negotiate intersections and merges into highways [86], Ramp merging is tackled in [87], where DRL is applied to find an optimal driving policy using LSTM for producing an internal state containing historical driving information and DQN for Q-function approximation.
Lane Change	Learn a policy that decides whether the vehicle performs no operation, lane change to left/right, accelerate/decelerate. Authors [88] use Q-learning, whereas traditional approaches consist in defining fixed way points, velocity profiles and curvature of path to be followed by the ego vehicle.
Lane Keep	Ego-vehicle follows the lane. Authors [89] propose a DRL system for discrete actions (DQN) and continuous actions (DDAC) using the TORCS simulator (see Table V-C), study concludes that continuous actions provide smoother trajectories, while more restricted termination conditions lead to the slower convergence time to learn.
Automated parking	Learn policies to automatically park the vehicle [90].

Figura 24 Tarefas importantes na condução autónoma que requerem Deep Reinforcement Learning para aprender uma policy ou um comportamento. Fonte: [61].

Sallab et al. [60][64] testou algoritmos de *Deep Reinforcement Learning*, nomeadamente *Deep Deterministic Actor Critic* (*DDAC*) e *DQN* para solucionar o problema dos carros autónomos seguirem a trajetória das linhas brancas mantendo-se dentro dos limites da estrada a uma velocidade constante. Na figura 25, encontra-se uma captura de ecrã efetuada num vídeo ilustrativo dos resultados obtidos pelo autor, onde nesse vídeo verifica-se que o carro tem o comportamento mais suave no caso do *DDAC*, até mesmo nas curvas da estrada do ambiente de simulação.

Figura 25 Captura de ecrã, da simulação em computador,de um veículo autónomo executando seguimento da trajetória das linhas amarelas usando algoritmos de Deep Reinforcement Learning. Fonte:[60].

Os mesmos autores fizeram ainda uma comparação de resultados obtidos no teste aos algoritmos *DDAC* e *Q-Learning* (Figura 26). Concluíram que embora os dois algoritmos sejam razoáveis na parte reta da via, há um deles que se destaca novamente pela positiva quer na parte reta quer na parte curva da via, o *DDAC*, que atribuiu ao veículo manobras bastante mais suaves que o *Q-Learning*.

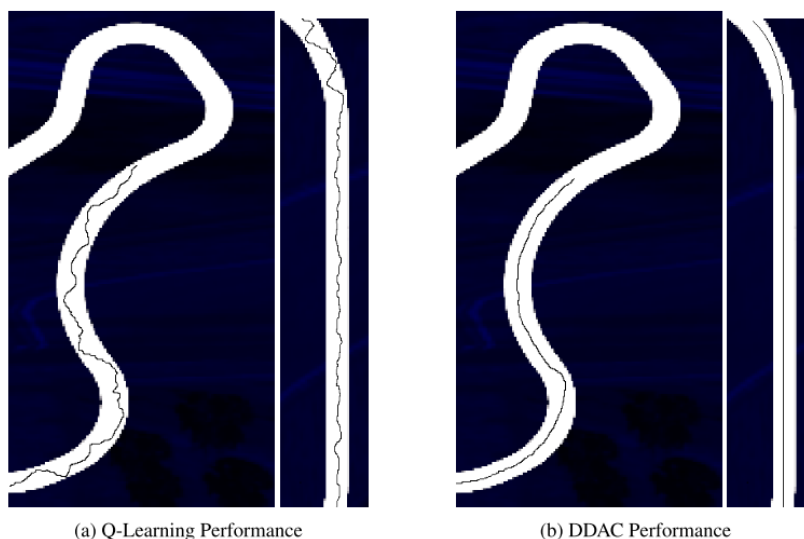


Figura 26 Comparação de performance entre Q-Learning e DDAC na mesma via. Fonte:[64].

Estes testes assim como uma grande parte dos apresentados na literatura são realizados num simulador, o *The Open Racing Car Simulator* (*TORCS*), que permite a simulação de algoritmos de condução

autónoma de forma eficiente por dispor de um *plug-in* que fornece toda a computação gráfica necessária para os testar.

Wang S. et al [63] reconhece que o sucesso recorrente das *DNN* em alguns jogos não é tão trivial de ser implementado na condução autónoma, porque no mundo real a complexidade é maior e é necessário um bom controlo para assegurar a devida segurança. Para dar conta dessas dificuldades, o autor adota o algoritmo *Deep Deterministic Policy Gradient* (*DDPG*) e testa em simulação no *TORCS*, por questões de segurança.

A tarefa principal que o autor pretendia cumprir era a ultrapassagem de um veículo em andamento o que requereu uma capacidade do controlador do agente para aceleração e travagem do veículo quando necessário. No caso da figura 27, o veículo a ser ultrapassado perde algum controlo de tração e o autor descobriu que se deve ao facto de o agente manter sempre a mesma velocidade. Treinou novamente o modelo para assegurar que na curva o agente reduz a velocidade. De notar que o modelo não está preparado para evitar colisões com outros veículos e, portanto, é uma situação possível na simulação e tal acaba por verificar-se em algumas ocasiões.

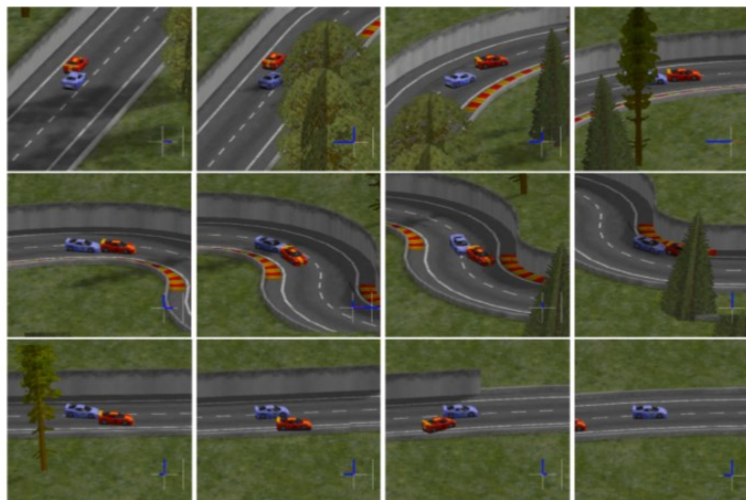


Figura 27 Simulação em TORCS de uma ultrapassagem controlada por métodos de Reinforcement Learning. Fonte:[63].

Após a análise feita à literatura, verifica-se que há algumas alternativas para o cumprimento da tarefa que diz respeito à segunda componente deste trabalho que é o seguimento da trajetória da sinalização temporária. Se na componente de deteção dos objetos foi mais trivial a escolha de um algoritmo, neste caso a escolha dependeu bastante do historial em aplicações na mesma área. Há algoritmos mais adequados para o caso de ações discretas e há algoritmos mais adequados para o caso de ações contínuas no tempo. Tendo isto em mente, foi ponderado o *DDPG* para assegurar um bom controlador do veículo em tempo contínuo que respeite, tanto o código da estrada, como a segurança necessária para todos os agentes no meio envolvente.

3. Fundamentos Teóricos

3.1. YOLOv3-tiny

O *YOLOv3-tiny,* derivado do *YOLOv3*, foi criado com o intuito de se distinguir pelo seu tempo de resposta, descartando uma parte das camadas convolucionais existentes na *Darknet-53*. A classificação é feita em duas escalas, média e grande, deixando de fora a classificação de objetos pequenos. Por outras palavras, a previsão de *bounding boxes* passa a ser feita em duas escalas diferentes, que são 13x13 e 26x26, considerando as imagens de *input* com resolução de 416x416. Esta pequena penalização não é significativa para este projeto, já que as vantagens da mesma são superiores às suas desvantagens. Como foi discutido previamente, um algoritmo que produza bons resultados de classificação com o tempo de resposta mínimo, dando prioridade ao tempo de resposta, é o mais equilibrado para assegurar requisitos importantes na condução autónoma, como a segurança.

A figura 28 descreve o processo de deteção e classificação por parte do YOLOv3-tiny. o sistema recebe as imagens de entrada, realiza a extração de *features* relevantes em escalas diferentes e utiliza-as para proceder à deteção dos objetos e à sua respetiva classificação. No final são esperadas *bounding boxes* com a classe do objeto associada que são colocadas nas *frames* das respetivas imagens de entrada para verificar se o sistema fez a correta identificação dos objetos de interesse lá presentes.

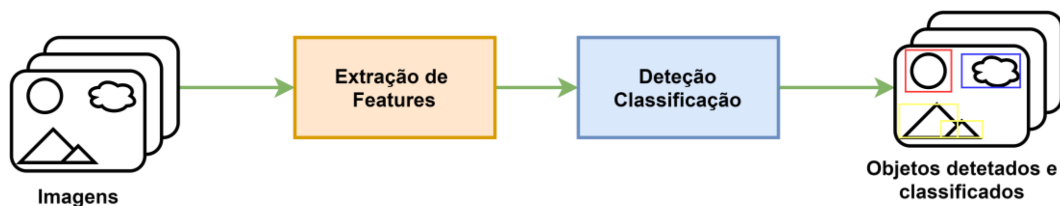


Figura 28 Diagrama do funcionamento base da rede YOLOv3-tiny.

As imagens de entrada podem ser provenientes de um *dataset*, no caso do treino da rede neuronal, ou de uma *webcam* para o teste do algoritmo em tempo real.

Na figura 29 pode-se observar a arquitetura da rede *YOLOv3-tiny*. Para a extração de features, são usadas camadas convolucionais e camadas de max-pooling. Cada camada convolucional é seguida de normalização, usando Batch Normalization, e da função de ativação Leaky Rectified Linear Unit (Leaky ReLU).

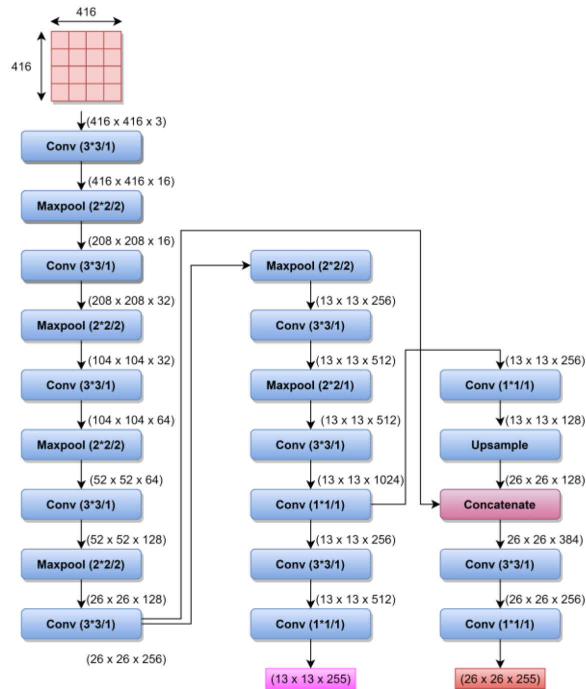


Figura 29 Arquitetura da rede YOLOv3-tiny.

A versão *tiny*, prevê os seguintes atributos, *objecteness score*, *bounding boxes* e as classes para as duas escalas. É considerada a versão compacta do *YOLOv3* por funcionar de forma idêntica, fazendo uso das suas vantagens, mas com algumas restrições dando origem à exclusão da deteção de objetos pequenos e a uma redução significativa do tempo de resposta. Juntamente ao *YOLOv3*, esta usa: deteção a diferentes escalas; os conceitos de *anchor boxes*, *Intersection over Union* (*IoU*); regressão logística para a previsão da *confidence* dos objetos e *multi-labeling*. Nomeadamente, usa *cross-entropy loss function*, uma função introduzida pelos autores do *YOLO* na versão 3 para corrigir algumas falhas das versões anteriores.

A característica que ambas têm em comum e que mais se destaca é a capacidade de *multi-labeling*, por outras palavras, atribuição de cada objeto a diferentes classes numa mesma imagem de forma eficaz e eficiente.

Para a deteção em duas escalas, o algoritmo usa fatorização por dois valores, 32 e 16 os quais são chamados de *stride*. As imagens de entrada com resolução 416x416 são reduzidas num fator de 32, ou seja, ficam com resolução de 13x13. A imagem resultante sendo bastante mais pequena, permite dar ênfase aos objetos grandes nela presentes, por outras palavras, os objetos pequenos e médios ficarão indetetáveis. Numa nota à parte, a resolução de entrada das imagens deve ser divisível por 32, para se poder aplicar os diferentes valores do *stride*. De seguida, a imagem é igualmente reduzida por um fator de 16. Como a imagem de entrada já foi diminuída para 13x13, é realizado *upsampling* por 2x para

obter o segundo tamanho desejado, 26x26, equivalente a reduzir 416x416 por *stride* 16. Desta vez a imagem não se encontra tão pequena e assim os objetos médios já se encontram detetáveis.

Para o algoritmo prever as *bounding boxes* usa *anchor boxes*. A introdução de *anchor boxes* no algoritmo, por Joseph Redman, deve-se ao facto de ao prever *offsets* em vez de coordenadas simplificar o problema, tornando-o mais fácil para a rede neuronal aprender e eliminando gradientes instáveis durante o treino. Na prática, a rede neuronal prevê as 4 coordenadas para cada *bounding box* tx (coordenada x), tw(largura), ty(coordenada y), th(altura). Cx e Cy são as coordenadas do canto superior esquerdo da *anchor box*, em relação à origem que está representada na figura 30 pelo ponto amarelo.

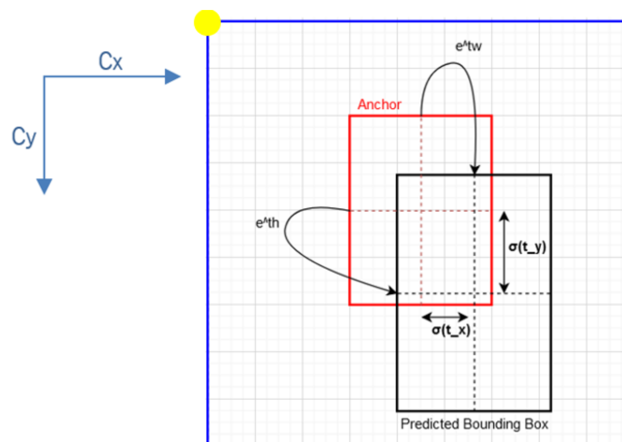


Figura 30 Obtenção das coordenadas das predicted bounding boxes.

Sabendo a altura e largura da *anchor box* (pw, ph), então as previsões são dadas pelas seguintes fórmulas:

$$b_x = \sigma(\mathrm{t_x}) + C_x$$
$$b_y = \sigma(\mathrm{t_y}) + C_y$$
$$b_w = \mathrm{p_w} * e^{tw}$$
$$b_h = \mathrm{p_h} * e^{th} \qquad (3.1)$$

As coordenadas (bx, by) e as dimensões (bw, bh) da *bounding box* são previstas a partir do *offset* em relação ao centro da *anchor box*, sendo que este offset é calculado utilizando a função *sigmoid* σ. Por este motivo, a posição e a dimensão das *anchor boxes* dependem do *dataset* em uso, mais concretamente da localização dos objetos nas imagens do mesmo. Se as imagens do *dataset* forem compostas com objetos no centro das fotos, posicionam-se as *anchor boxes* o mais para o centro possível. Se os objetos se encontrarem mais nas extremidades, as anchors boxes devem estar posicionadas o mais possível para as extremidades. O tamanho dos objetos também é um fator a ter em conta para regular o tamanho das *anchor boxes*. O utilizador dispõe da capacidade de manipular estes

valores podendo até usar *k-means clustering* para os calcular se necessário. No caso específico do *YOLOv3-tiny* existem 6 *anchor boxes*, 3 para cada escala.

Após este processamento, o *IoU* é utilizado para filtrar as *bounding boxes*.

O *IoU* identifica o quanto duas *bounding boxes* se intercetam uma sobre a outra, como se pode ver na figura 31.

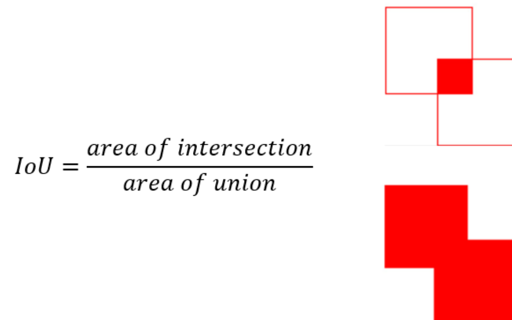
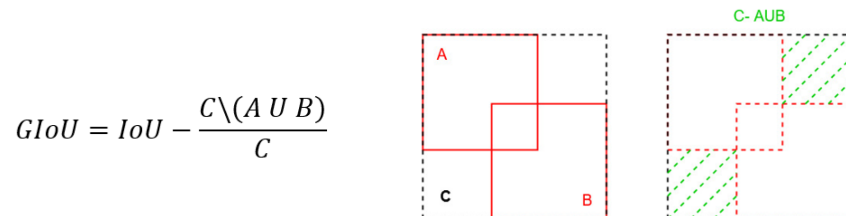


Figura 31 Cálculo do IoU.

O valor resultante da equação varia entre 0, se as *bounding boxes* não tiverem nenhuma sobreposição e 1, se as *bounding boxes* estiverem totalmente sobrepostas, respetivamente. A sua função é avaliar qual o *offset* máximo aceitável das coordenadas da *ground truth bounding box* para as coordenadas da *predicted bounding box* através de um *threshold* definido (para o treino). Este offset indica o quão precisa foi a *predicted bounding box* em detetar um determinado objeto. Da mesma maneira, permite facilitar os cálculos da respetiva *loss* do algoritmo durante o treino, bem como reproduzir resultados mais concisos durante o teste do algoritmo, através da eliminação das que estão abaixo do valor de *threshold* (escolhido para o teste). É por esta razão que se verificam, em alguns testes, o *mAP* denotado com um número à frente, por exemplo, *mAP45*. O valor 0.45 corresponde ao *threshold* do *IoU* utilizado, sendo neste caso igual a 0.45. Este processo de escolha da melhor *bounding box*, de entre várias vizinhas, culmina num processo que se chama *Non Maximum Suppression* que será debatido no capítulo seguinte, na secção de teste do algoritmo.

Contudo, o *IoU* sendo utilizado como métrica de avaliação não é o mais vantajoso. No caso, de duas *bounding boxes* não se intersetarem de todo, não é possível pela fórmula apresentada, saber a que distância estão uma da outra. Por este motivo, é utilizado um método que faz uso das vantagens do *IoU* e complementa as suas falhas.

É conhecido por *Generalized Intersection over Union* (*GIoU*). De forma suncinta, permite que se obtenha um gradiente do quão longe duas *bounding boxes* se encontram, ainda que estas não se intercetem. Por este motivo, é uma mais-valia para ser utilizado como *loss*. O seu cálculo é efetuado pela equação da figura 32.



$$GIoU = IoU - \frac{C \backslash (A \cup B)}{C}$$

Figura 32 Cálculo do GIoU.

Por motivos de interpretação são usadas as letras A, B e C que representam *bounding boxes*, onde U é o símbolo de união, neste caso, entre as caixas A e B. A caixa C pode ser vista como a *frame* que envolve as caixas A e B. Do numerador da equação, resulta o conjunto de todos os elementos de C que não estão na união de A e B. O *GIoU* é obtido pelo resultado da diferença entre o *IoU* e a zona verde dividida por C. Ao contrário do *IoU*, o alcance do valor resultante é entre -1 e 1. Para as *bounding boxes* intersetadas o valor será positivo, entre 0 e 1, caso não se intersetem o valor será de -1 a 0. O valor do *GIoU* é tão mais negativo quanto mais afastadas as caixas se encontrarem.

3.2. Deep Deterministic Policy Gradient

A segunda componente fundamental deste sistema é o algoritmo de *Reinforcement Learning*, necessário para o controlo do movimento do veículo. O objetivo principal de qualquer algoritmo de *Reinforcement Learning* é encontrar a melhor estratégia, ou a estratégia ótima, que defina o comportamento de um agente de modo que este obtenha a melhor recompensa possível num determinado ambiente. Apesar de ser uma área relativamente recente, já se encontram propostos vários algoritmos de diferente natureza, capazes de resolver problemas relativamente complexos, cada um com as suas vantagens e desvantagens. Geralmente, o processo de escolha é por exclusão de partes e procura-se um algoritmo pelo tipo de ambiente onde se pretende inseri-lo, tendo por base experiências anteriores com o mesmo. Antes de rever a origem do algoritmo escolhido é importante enumerar alguns conceitos básicos fundamentais sobre *Reinforcement Learning* para melhor compreender as suas bases. Neste sentido, a tabela 1 descreve os conceitos e termos que requerem uma interpretação prévia para abordar os algoritmos de forma mais simples.

Tabela 1 Principais conceitos importantes de introdução ao Reinforcement Learning.

Agente	A entidade que executa ações num determinado ambiente com o objetivo de receber uma, ou mais, recompensas.
Ambiente	O cenário em que se insere um agente. Geralmente é o local do desafio que o agente tem de ultrapassar.
Estado (s)	Atributo do ambiente num dado instante que é dado a conhecer por parte desse mesmo ambiente.
Ação (a)	Atributo do agente que visa a atuação num dado ambiente e que, por resultado, pode alterar o estado desse mesmo ambiente.
Episódio	Duração do percurso de um agente até que não restem mais instâncias de estados possíveis e, portanto, não restem mais ações que o agente possa tomar. Por outras palavras, é o conjunto de todas as instâncias dos estados entre a primeira e a última.
Espaço de estados (S)	Conjunto dos atributos que representam o estado do ambiente onde a instância de cada um pode ser alterada ao longo do tempo através da execução de ações por parte do agente.
Espaço de ações (A)	Conjunto dos atributos que representam a ação do agente onde a instância de cada um pode ser alterada ao longo do tempo dependendo da *policy* do agente.
***Policy* (π)**	Função que determina o comportamento do agente. Por outras palavras, é uma função responsável pela atuação do agente tendo em conta o estado atual do ambiente em que se insere.
***Reward* (R)**	Retorno imediato atribuído a um agente quando este executa uma ação.
***Return* (G)**	Conjunto das recompensas obtidas ao longo de um episódio, *total reward*.
Model	Comportamento bem conhecido do ambiente que permite conhecer exatamente o estado do mesmo após uma determinada ação por parte do agente.
Value function	A *value function* específica o valor v. O valor v mede o quão bom é um determinado estado tendo em conta o seu retorno.
Q-Function	Especifica o valor Q. Este difere do valor v pela introdução de um parâmetro adicional que é a ação no instante atual, formando um par estado-ação.

O processo de controlo de um agente num ambiente de natureza estocástica é modelado como sendo um *Markov Decision Process*, onde S é o espaço de estados, A é o espaço de ações e o controlo do comportamento do agente é levado a cabo por uma *policy* tendo em consideração a *reward* obtida por uma determinada função de *reward*. O objetivo é otimizar este processo verificando qual o conjunto de ações que resultam num *return* maior, onde para isso se deve treinar no ambiente por processos de tentativa-erro.

A *value function* e a *Q-function*, associadas a uma determinada *policy* respeitam, em vários algoritmos de *Reinforcement Learning*, a equação de Bellman de forma a resolver problemas de otimização complexos:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}\left[r(s_t, a_t) + \gamma\, \mathbb{E}_{a_{t+1} \sim \pi}\left[Q^{\pi}(s_{t+1}, a_{t+1})\right]\right] \tag{3.2}$$

A atuação de um agente num determinado ambiente por processos de tentativa-erro, requer uma gestão do seu comportamento no que diz respeito aos processos de exploração do ambiente e aprendizagem com base na informação que explorou. A este problema chama-se de *explore-exploit dilemma*. *Exploration* corresponde à parte em que o sistema procura e armazena informação do ambiente, por vezes retirada com carácter aleatório com auxílio de ruído. Posteriormente, o sistema passa à parte de *exploitation* que se baseia em tirar partido da informação armazenada durante a *exploration* para aprender com o intuito de no futuro tomar as melhores decisões e, por conseguinte, obter o maior *return* possível.

Uma primeira característica que diverge alguns algoritmos é a *policy* ser *online* ou *offline*, também denotados de *on-policy* e *off-policy*. Nos sistemas *on-policy*, a *policy* que é melhorada é a mesma que é usada para fazer a *exploration* e é recomendada para situações onde o agente tem um processo de *exploration* maior. Já nos sistemas *off-policy* a *policy* relativa ao processo de *exploitation* é diferente da *policy* utilizada para introduzir a *exploration*, o que permite fazer o agente encontrar a melhor *policy* a partir de um determinado conjunto de dados e fazer *exploration* ao mesmo tempo e de forma independente. Aplicando à condução autónoma, é possível simular a aprendizagem como se fosse a partir da condução de um ser humano em vez de uma máquina, o que lhe dá um caracter mais próximo do real. Os sistemas *off-policy* têm um bom desempenho no que toca a prever movimentos na robótica pois tiram partido de amostras retiradas de um *buffer* para treinar a rede neuronal, o *replay buffer* (figura 33). As amostras contêm o estado atual (*s*), a ação (*a*), o estado seguinte (*s'*), e a reward (*r*). O *replay buffer* armazena grandes quantidades de amostras permitindo eliminar eventuais correlações indevidas que possam surgir ao longo do treino. Para utilizar os métodos *on-policy*, seria necessário recolher dados do ambiente a cada mudança de *policy* o que se tornaria menos eficiente. Em suma, são os dois bons motivos para escolher um sistema *off-policy* como o *DDPG*.

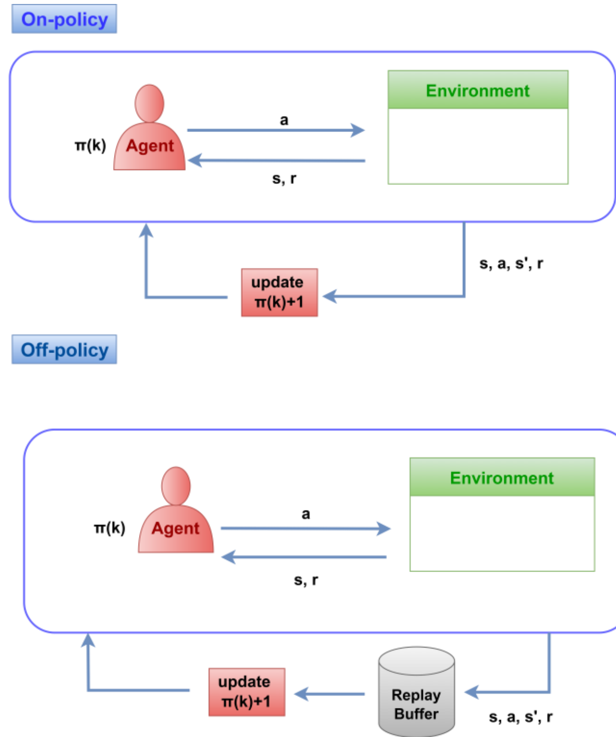


Figura 33 Diferença entre os métodos on-policy e off-policy.

Uma segunda característica intrínseca na *policy* diz respeito ao seu carácter determinístico ou estocástico.

Uma *policy* determinística $\pi(s)$ mapeia estados em ações, da seguinte maneira:

$$s \rightarrow \pi(s) \rightarrow a \tag{3.3}$$

É utilizada em ambientes onde qualquer ação determina o enredo do episódio, isto é, um ambiente onde uma ação especificamente pode levar o agente para um estado que influenciará os próximos estados.

Já uma *policy* estocástica $\pi(a|s)$, retorna uma distribuição probabilística sobre uma ação para um determinado estado:

$$s, a \rightarrow \pi(a|s) \rightarrow P(a_t|s_t) \tag{3.4}$$

Este tipo de abordagem é utilizado em ambientes incertos que requerem uma abordagem puramente aleatória e embora não seja utilizada no algoritmo escolhido para este projeto, não implica que não o pudesse ser, seguindo uma outra abordagem ao problema. A vantagem da *target policy* ser determinística é, segundo o autor do *DDPG*, a possibilidade de evitar o cálculo do valor espectável interno na equação de Bellman:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}\left[r(s_t, a_t) + \gamma\, \mathbb{E}_{a_{t+1} \sim \pi}\left[Q^{\pi}(s_{t+1}, a_{t+1})\right]\right] \tag{3.5}$$

Assim, o valor espectável depende apenas do ambiente e o Q^{μ} já pode ser treinado off-policy:

$$Q^{\mu}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}\left[r(s_t, a_t) + \gamma Q^{\mu}(s_{t+1}, \mu(s_{t+1}))\right] \tag{3.6}$$

Na equação 3.6, $r(s_t, a_t)$ é a função de *reward*, γ é um fator de desconto aplicado ao Q^{μ} do estado seguinte.

Os algoritmos de *Reinforcement Learning* podem ainda diferir quanto à presença ou ausência de modelo, os *model-based* e os *model-free*, respetivamente (figura 34). Os algoritmos *model-based* dependem de um modelo representativo do ambiente onde irá ser inserido. Já os algoritmos *model-free*, não dependem da construção ou aprendizagem a partir de um modelo específico do ambiente onde se encontram inseridos. Podem-se já excluir os algoritmos *model-based* visto que este projeto remete para um ambiente de condução autónoma e nesses casos o ambiente é imprevisível, não sendo definido por nenhum tipo de modelo conhecido.

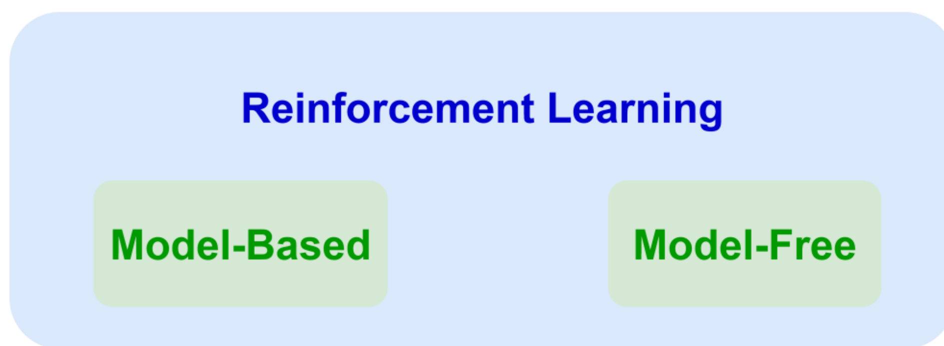


Figura 34 Os dois principais tipos de Reinforcement Learning.

Dentro dos algoritmos *model-free* existe um vasto conjunto de métodos, como por exemplo, os métodos *policy-based* e *value-based* (figura 35). De seguida são descritas algumas diferenças entre esses métodos e as características de cada um que levaram à escolha do *Deep Deterministic Policy Gradient*.

Figura 35 Dois tipos de algoritmos model-free.

Os métodos *value-based* baseiam-se em *Temporal Difference Learning* e têm como objetivo aprender uma função, a *Q-Function*. É considerada uma versão estocástica de um processo iterativo de avaliação da *policy*. Aqui enquadram-se os algoritmos de *Q-Learning* e *Deep Q-Learning*.

Já os métodos *policy*-based têm uma estratégia bem definida que se baseia em ignorar Q e passar a dar ênfase à aprendizagem da *policy* (ótima). O objetivo é encontrar os parâmetros da *policy* que maximizam o *return* e é de onde surgem os algoritmos de *Policy Gradient* e *Deterministic Policy Gradient*.

Os *Actor-Critic* são uma junção dos métodos *value-based* e *policy-based*.

Os conceitos enunciados são a base do *DDPG*, um algoritmo estado da arte, que tem vindo a ser cada vez mais utilizado em ambientes de natureza contínua e complexa, onde não há um modelo específico que defina o ambiente onde se insere o agente. O melhor exemplo de um ambiente desse tipo é o de uma via pública, sendo um dos motivos de começar a ser aplicado em projetos de condução autónoma. É uma composição de métodos de *Deep Q-Learning* e *Deterministic Policy Gradient*. Mais ainda, é uma modificação de *Deep Q-Learning*, um algoritmo que apenas processa ações discretas, para funcionar com ações contínuas. Em *Deep Q-Learning* são utilizadas redes neuronais para mapear os estados nos respetivos pares de ação, para obter o valor *Q*. Sendo *off-policy* este aprende a partir de ações arbitrárias armazenadas no *replay buffer*, de modo que possa ser treinado enquanto outra *policy* é responsável por explorar o ambiente. Desta maneira, o algoritmo de *Deep Q-Learning* aprende em pequenos *batches* retirados do *buffer* à medida que o episodio avança. Ao início, como o *buffer* está vazio, é preenchido por variáveis aleatórias resultantes da *exploration*.

O algoritmo de *Deep Q-Learning* só funciona para simulações em ambientes de natureza discreta, algo que não é viável na condução autónoma. Ainda que se pudesse discretizar todo o processo e o ambiente relativo à condução autónoma, criaria outros problemas de otimização que o autor do DDPG enuncia na sua publicação.

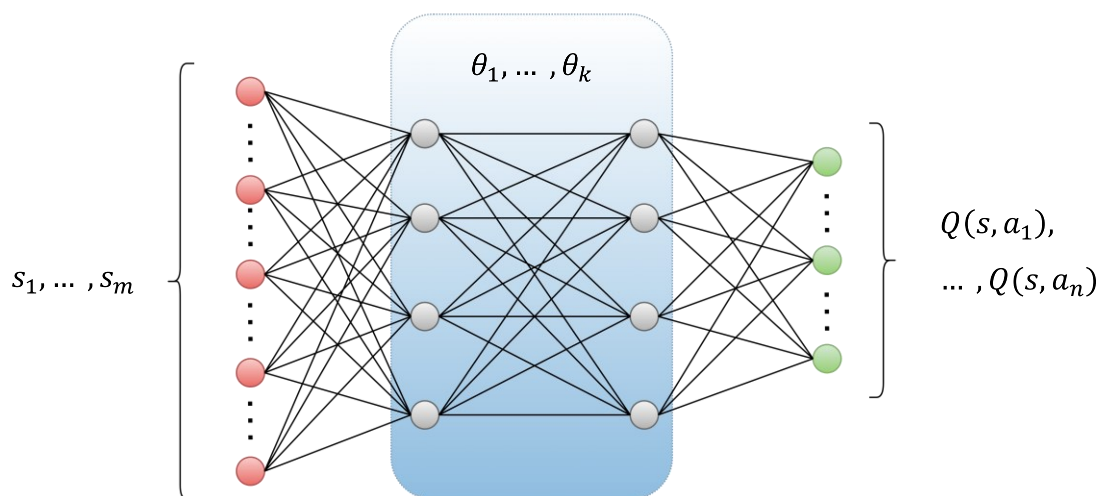


Figura 36 Exemplo de processamento de Deep Q-Learning.

O funcionamento de *Deep Q-Learning* baseia-se na premissa de que existe um estado s que é introduzido na rede neuronal de onde resulta um valor *Q* que é um par de ações possíveis para cada estado, *s*, (figura 36). É um espaço de ações discreto na medida em que é possível enumerar um conjunto de

ações para os respetivos estados. Deste modo, escolher a melhor ação para um determinado estado s é algo trivial pela seguinte equação:

$$\pi(s) = a^* = argmax_a Q(s,a) \quad (3.7)$$

Esta equação é a *policy* responsável por escolher a melhor ação possível num espaço de ações. Este processo torna-se inviável em ambientes de natureza contínua, pelo simples facto de que não é possível enumerar todos os valores *Q* para um espaço de ações onde estas são infinitas.

Os autores do *DDPG* resolveram o dilema ao introduzir uma rede neuronal específica para encontrar a ação ótima, ou seja, a *policy* passou a ser uma rede neuronal, mas com o mesmo objetivo, receber o estado e calcular a ação ótima. Esta rede é treinada tendo por base os valores *Q* mais elevados, resultantes da rede que faz as previsões dos mesmos. O resultado deste método traduz-se num *Actor-Critic*, como se pode ver na figura 37.

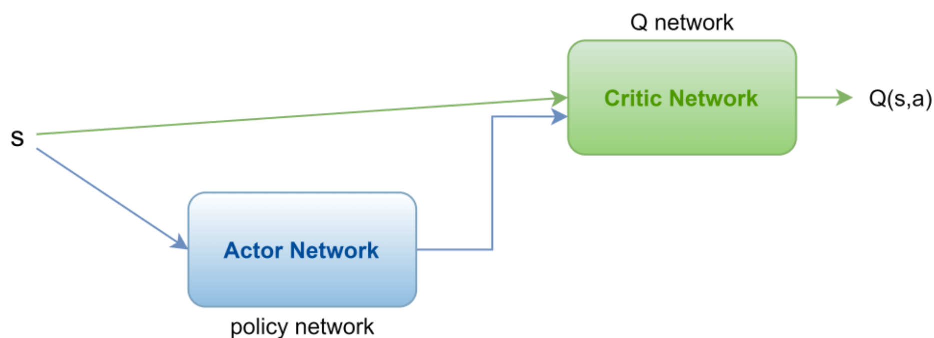


Figura 37 Estrutura de um algoritmo Actor-Critic.

A primeira rede neuronal, *actor*, é responsável por encontrar a melhor ação e a segunda rede neuronal, *critic*, avalia o par de estados e ações (*s*, *a*), traduzindo num valor *Q*.

Um dos desafios que surgiu foi a convergência do valor *Q* para o valor ótimo quando não existe uma referência ou um *dataset* como em *Supervised Learning*, por exemplo. Para resolver este problema, o autor do *DDPG* utiliza uma *target network*, que é uma cópia da original (figura 38).

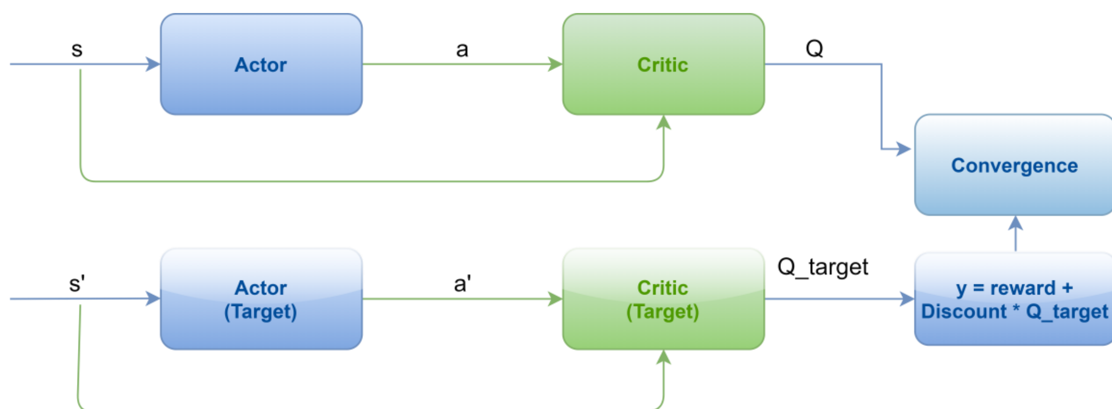


Figura 38 Estrutura do DDPG.

Na figura 38, s' representa o estado seguinte e a' a ação seguinte, de onde resulta o valor Q_target, que é o valor ótimo para o próximo estado e é utilizado para calcular o *target* (*y*) acompanhado de um fator *Discount*, geralmente de 0.99.

$$y = reward + Discount * Q_target \quad (3.8)$$

O *critic* é treinado de forma a atingir a convergência entre o valor Q do estado atual e o valor do *target* que tem em consideração o valor ótimo para o estado seguinte. Para alcançar esta convergência é utilizado *gradient descent* (os *weights* da*target network* têm de ser atualizados a um ritmo menor, caso contrário seria difícil atingir convergência). Depois de o *critic* já ter treinado, o *actor* começa o seu treino. Quando o *actor* se encontra a treinar, o critic suspende e recebe o espaço de estados para encontrar a ação ótima para maximizar o *Q*. O *actor* é treinado aplicando a regra da cadeia no *return* esperado da distribuição J tenda em conta os parâmetros do *actor* $\theta\mu$:

$$\nabla_{\theta\mu} J = E_{s_t \sim \rho^\beta}[\nabla_a Q(s,a|\theta^Q)|_{s=s_t,a=\mu(s_t)} \nabla_{\theta_\mu} \mu(s|\theta^\mu)|_{s=s_t}] \quad (3.9)$$

A regra ficou provada como sendo *Policy Gradient*, segundo os autores do *DDPG*. O treino do *actor* depende do cálculo do gradiente no *critic*, $\nabla_a Q(s,a|\theta^Q)$, com respeito às possíveis ações tendo em conta os estados que são utilizados para o treino (em *batches*). $\mu(s|\theta^\mu)$ é uma função do actor que especifica a *policy* atual. É possível retirar informação sobre qual é o par (*s*, *a*) com valor *Q* mais elevado e os parâmetros do *actor*, são atualizados tendo essa informação em consideração.

Os autores referem também que utilizar redes neuronais diretamente com *Q-Learning* resultava em instabilidade em diversos ambientes. Isto porque, a rede neuronal que estava a ser atualizada, era a mesma usada para calcular o valor do *target* e demonstrou-se difícil para os valores convergirem. De forma a contornar isso, os mesmos introduziram *soft-updates* para atualizar a *target network*. Ou seja, os *weights* introduzidos são sujeitos a um fator de desconto, τ , para não resultar em mudanças bruscas no Q'_{target}:

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta', \tau \ll 1 \quad (3.10)$$

Um outro desafio apresentado diz respeito à parte de exploração nos algoritmos de *Reinforcement Learning* em ambientes de natureza contínua. Nos métodos clássicos de *Reinforcement Learning*, habitualmente usam-se métodos de exploração como *Epsilon-Greedy*, contudo, em algoritmos de *Deep Reinforcement Learning*, onde são introduzidas redes neuronais como função de aproximação, já se utilizam outros métodos. Para explorar uma maior variedade de ações introduziu-se ruído (N):

$$\mu'(s) = \mu_\theta(s) + N \quad (3.11)$$

Da equação resulta a *policy* para fazer *exploration* ou a *behavior policy*, $\mu'(s)$.

Esta exploração no caso do *DDPG*, como foi visto antes, é independente da aprendizagem porque a *policy* de aprendizagem recolhe informação do *replay buffer*, sendo uma vantagem dos algoritmos *off-policy*. O tipo de ruído recomendado pelo autor é o Ornstein-Uhlenbeck.

4. MODELO E SIMULAÇÃO

O sistema foi implementado num computador portátil *Asus X556U* com um processador *Intel Quad-Core i5*, 2.30 GHz, uma memória RAM de 8GB e com placa gráfica *Nvidia GeForce 940M*, no sistema operativo *Ubuntu 18.04.5 LTS* de *64-bit*. Foi utilizado Python como linguagem de programação, juntamente com as bibliotecas necessárias como o OpenCV, o Tensorflow e o Keras.

O *OpenCV* é utilizado para processar as *frames* vindas da *webcam* do computador e introduzi-las no algoritmo para as tarefas de deteção e classificação. O *Tensorflow* é utilizado para a implementação da rede neuronal por ser bastante utilizado pela comunidade científica (incluindo pelos investigadores do LAR da Universidade do Minho). É também uma mais-valia já que possui funções específicas e especializadas para lidar com o processo de construção e dimensionamento da rede neuronal. O *Keras* permite introduzir algumas funções específicas com algum nível de abstração. O otimizador *Adam* é um exemplo importante para o treino da rede neuronal.

Além destas bibliotecas, existem outros programas que foram necessários para tarefas complementares como o *Google Colaboratory*, o *Google Drive*, o *GIMP*, o *LabelImg* e o *Tensorboard*, onde a sua função é explícita nas páginas seguintes.

4.1. YOLOv3-tiny: Otimização do Modelo

Para o sistema desempenhar as funções previstas, requereu primeiro uma otimização contínua do modelo até se encontrarem soluções que satisfazem as condições necessárias. O sucesso do mesmo dependeu de vários fatores como a qualidade do *dataset* e a correta personalização dos hiperparâmetros. Para assegurar este sucesso, foram realizadas as ações presentes no fluxograma da figura 39.

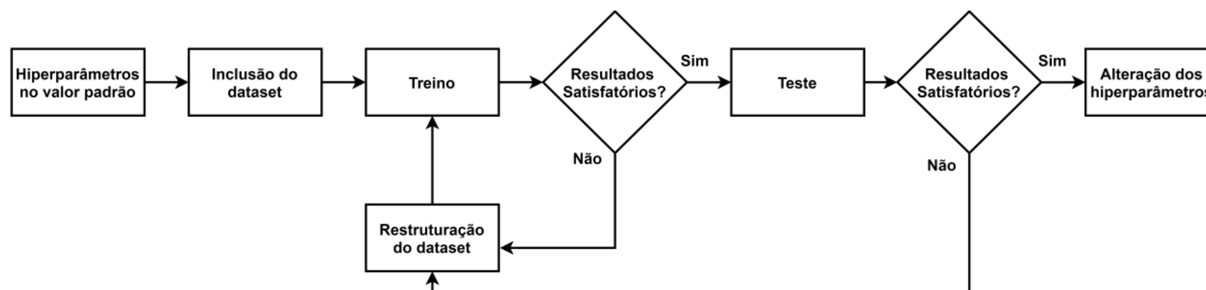


Figura 39 Fluxograma com o processo de otimização utilizado para obtenção de um dataset com a melhor qualidade.

Numa fase inicial, foi necessário estabelecer um valor padrão para os hiperparâmetros para se poder verificar, de forma individual, o impacto da restruturação contínua do *dataset*. De outra forma, não seria possível saber se o aumento da performance foi devido à otimização do *dataset* ou à otimização dos hiperparâmetros. Depois de encontrados resultados aceitáveis para o *dataset*, passou-se então à

alteração dos hiperparâmetros, treinando de novo o algoritmo e testando-o. A alteração dos hiperparâmetros também seguiu uma norma para não ser completamente aleatória. Tanto a norma como os resultados dos testes são apresentados na secção dos Resultados.

4.2. YOLOv3-tiny: Dataset

O *dataset* foi inteiramente desenvolvido de raiz por dois importantes motivos. O primeiro reside no facto de não se conhecerem ferramentas para anotar de forma eficiente as imagens necessárias para o treino. Até porque se o contrário se verificasse, aquilo que é a primeira parte do problema deste projeto estava automaticamente solucionado e deixaria de ser um problema. O segundo motivo diz respeito à escassez de qualidade que se verifica em grande parte dos *datasets* existentes. De forma sucinta, há grandes *datasets* que contribuem menos para o bom funcionamento do sistema do que pequenos datasets onde cada imagem é pensada e estudada com cautela para maior eficiência.

Uma imagem existente no *dataset* difere de inúmeras características das restantes. Estas características podem ser, por exemplo, o número de cones de estrada presentes na mesma, a posição dos cones, o tamanho dos cones, a presença dos sinais de estrada a classificar na mesma foto em simultâneo, o brilho, a saturação, o ruído, o desfoque e até o fundo da própria imagem. A acrescentar a estas características todas, a mais importante é cada sinal de estrada ser de diferente tipo e cor (dentro dos padrões normais e previstos no código da estrada). Na figura 40 encontram-se três imagens aleatórias retiradas do *dataset*. Cada imagem possui características bastante diferentes incluindo objetos que não fazem parte da classificação, propositadamente, para o algoritmo os ignorar quando estes aparecerem.

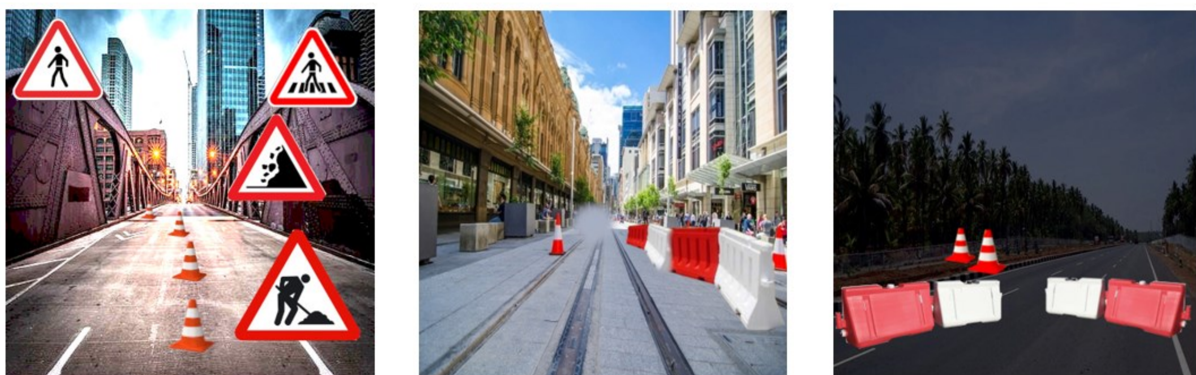


Figura 40 Exemplos de imagens que constituem o dataset utilizado para treinar o YOLOv3-tiny.

No total, para os três sinais de construção na via, foram desenvolvidas 1252 imagens recorrendo a um *software open-source* que se chama *GIMP*. Todas estas imagens foram desenvolvidas de forma manual para assegurar que não induzem o algoritmo em erro, nem o viciam com imagens repetidas. Uma das vantagens deste controlo manual é a oportunidade de introduzir objetos idênticos aos que se pretendem

classificar numa mesma imagem, sendo que estes não são anotados porque não fazem parte da classificação. Com isto, o algoritmo não confunde tanto os objetos a classificar com outros semelhantes, resultando numa classificação mais rigorosa.

O número total de anotações é indefinido porque, como visto na figura anterior, existem imagens que contêm várias anotações e a contagem do número de ficheiros *XML* (1252) não corresponde à contagem total de anotações. Estas anotações foram efetuadas, também manualmente, recorrendo ao software *LabelImg* e guardadas em ficheiros *XML* (formato padrão deste tipo de anotações). Posteriormente, foram convertidas para o formato *YOLO* para poderem ser interpretadas pelo algoritmo *YOLOv3-tiny*.

Para este projeto, a deteção de objetos baseou-se nos seguintes sinais: cones de estrada, sinais de trabalhos na via e separadores de via. Na figura 41 encontra-se uma imagem representativa de cada um dos três objetos.

Figura 41 Sinais presentes no dataset sujeitos a deteção e classificação. (Cone, "trabalhos na via", separador de via).

Esta é a versão dos três sinais mais comum em Portugal, no entanto podem surgir variações, como por exemplo o cone de estrada ser todo laranja ou todo vermelho, o sinal de trabalhos na via ter o fundo branco e o separador de via pode ser branco ou até mesmo de diferente forma ou material. O algoritmo está preparado para lidar com todas essas variações.

Previamente, fora introduzido no *dataset* a classe de fitas de barreira. Na figura 42 encontra-se o exemplo de uma fita de barreira.

Figura 42 Fita de barreira.

Este tipo de objeto é maleável e facilmente distorcido daquilo que é a sua forma, posição e ângulos considerados normais e a sua deteção verificou-se extremamente complexa, ou até mesmo fora do âmbito deste projeto. Na secção dos resultados será feita referência à performance do algoritmo com a

utilização deste objeto. Como forma de substituição do mesmo, foi introduzido o separador de via, um objeto com alguma presença em autoestradas, representando a terceira classe deste *dataset*.

No *YOLOv3-tiny*, a variação da resolução das imagens de entrada resulta numa mudança decisiva de velocidade e capacidade de classificação. A deteção de imagens em tempo real com resolução reduzida traduz-se num aumento da taxa de amostragem (*FPS*). Por outro lado, se a resolução das imagens for aumentada, verifica-se uma queda acentuada na taxa de amostragem mas um aumento na capacidade de deteção e classificação. Por esse motivo, de modo a atingir um equilíbrio, ficou estipulado que a resolução das imagens de treino é de 416x416.

Em cada ficheiro *XML* estão presentes as classes dos objetos com as respetivas coordenadas de todas as *ground truth bounding boxes*, relativas a cada objeto existente na imagem que se pretende classificar. O ficheiro contém ainda a diretoria e o nome da imagem a que as anotações dizem respeito para identificar se é uma imagem para treino ou teste.

Foram efetuados múltiplos treinos onde se acrescentaram (ou modificaram) gradualmente as imagens no *dataset*, até se atingir um nível de classificação desejado. As imagens devem ser tendencialmente semelhantes às situações que o algoritmo poderá encontrar em fase de teste de modo a ser mais fácil a este reconhecer as suas características.

Quando o *dataset* é processado durante o treino do mesmo, é acionado um método bastante útil e eficaz para aumentar o desempenho do algoritmo, o *data augmentation*. O *data augmentation* executa várias transformações nas imagens do *dataset* dando-lhe mais viabilidade e reduzindo o fenómeno de *overfitting*. As transformações realizadas são as seguintes: reflexão, translação e corte da imagem. Estas são aleatórias e o fator de transformação é limitado por um máximo de 50%, caso contrário, poderia deformar completamente a imagem.

4.3. YOLOv3-tiny: Treino

O treino da rede neuronal foi executado no *Google Colaboratory* dado o seu poder computacional. Este funciona de forma similar ao computador pessoal, estando ao dispor do utilizador para executar qualquer tipo de processamento necessário como o treino de redes neuronais. A vantagem de ser um processo feito na *Cloud* é o facto de ter permitido utilizar o computador pessoal enquanto o treino decorreu, requerendo apenas alguma memória RAM disponível.

Para se iniciar o processo foi necessário carregar todos os ficheiros relevantes para o *Google Drive*, que é a nuvem da Google ao serviço do utilizador. Depois de carregados os ficheiros, procedeu-se à instalação das bibliotecas de *Python* para se poder efetuar o processo de treino. O primeiro passo foi testar se a

placa gráfica, do *Google Colaboratory*, estava disponível para o uso durante o treino. Após isso, deu-se a conversão dos ficheiros *XML* para *YOLO*. Esta conversão é apenas uma mudança de formato da disposição das anotações e das suas características, para o programa as ler. Na figura 43 pode-se verificar a diferença entre os 2 formatos.

Figura 43 Exemplo de um ficheiro em formato XML e formato YOLO.

No formato *YOLO*, o valor no final representa a classe do objeto. Neste caso o 0 diz respeito ao separador de via, para os restantes dois objetos estão atribuídos os valores 1 e 2.

De seguida é criado o modelo e os *weights* pré-treinados da *darknet* são carregados no modelo. O treino inicia-se e só termina depois de percorridas todas as *epochs* previamente estabelecidas pelo utilizador. Cada epoch divide-se em *steps* uma vez que a informação é fornecida ao modelo por conjuntos de amostras. Este conjunto de amostras chamam-se *batches* e o seu tamanho, *batch size*, é definido pelo utilizador. O número de *steps* depende do número total de amostras (*samples*) e do *batch size* da seguinte forma:

$$steps = \frac{samples}{batch\ size} \quad (4.1)$$

Durante o treino, é utilizado um método que substitui o clássico *Stochastic Gradient Descent*, chama-se *Adam* e é considerado um otimizador que reúne as vantagens dos clássicos, como o *AdaGrad* e o *RMSProp*. A atualização dos *weights* no modelo é feita no final de cada *epoch* depois de introduzidos todos os *batches* com o seu conjunto de amostras de treino. No final de cada *epoch* existe um passo de validação. Para otimizar o processo só são guardados os melhores *weights* do treino. Por outras palavras, se a *validation loss* for mais pequena que a anterior os *weights* são guardados, caso contrário, segue-se o treino sem guardar *checkpoint*.

Na figura 44 é possível visualizar o *learning rate* e o processo de decaimento do valor total da *loss*, que é composto pela soma das três *losses* utilizadas no *YOLOv3-tiny*: *GIoU loss*, *confidence loss* e *probability loss*.

```
Epoch:99 step:  490/503, Learning Rate:0.000001 / Loss -> GIoU:  0.48, Confidence:  0.00, Probability:  0.22, Total:  0.70
Epoch:99 step:  491/503, Learning Rate:0.000001 / Loss -> GIoU:  0.22, Confidence:  0.00, Probability:  0.17, Total:  0.39
Epoch:99 step:  492/503, Learning Rate:0.000001 / Loss -> GIoU:  0.19, Confidence:  0.00, Probability:  0.13, Total:  0.32
Epoch:99 step:  493/503, Learning Rate:0.000001 / Loss -> GIoU:  0.43, Confidence:  0.00, Probability:  0.30, Total:  0.74
Epoch:99 step:  494/503, Learning Rate:0.000001 / Loss -> GIoU:  0.11, Confidence:  0.00, Probability:  0.13, Total:  0.24
Epoch:99 step:  495/503, Learning Rate:0.000001 / Loss -> GIoU:  0.21, Confidence:  0.00, Probability:  0.13, Total:  0.34
Epoch:99 step:  496/503, Learning Rate:0.000001 / Loss -> GIoU:  0.09, Confidence:  0.00, Probability:  0.09, Total:  0.18
Epoch:99 step:  497/503, Learning Rate:0.000001 / Loss -> GIoU:  0.24, Confidence:  0.00, Probability:  0.17, Total:  0.41
Epoch:99 step:  498/503, Learning Rate:0.000001 / Loss -> GIoU:  0.34, Confidence:  0.00, Probability:  0.26, Total:  0.60
Epoch:99 step:  499/503, Learning Rate:0.000001 / Loss -> GIoU:  1.83, Confidence:  0.07, Probability:  1.21, Total:  3.11
Epoch:99 step:  500/503, Learning Rate:0.000001 / Loss -> GIoU:  0.76, Confidence:  0.01, Probability:  0.51, Total:  1.29
Epoch:99 step:  501/503, Learning Rate:0.000001 / Loss -> GIoU:  0.62, Confidence:  0.02, Probability:  0.56, Total:  1.20
Epoch:99 step:  502/503, Learning Rate:0.000001 / Loss -> GIoU:  0.19, Confidence:  0.00, Probability:  0.09, Total:  0.27
Epoch:99 step:    0/503, Learning Rate:0.000001 / Loss -> GIoU:  0.14, Confidence:  0.00, Probability:  0.13, Total:  0.27
Epoch:99 step:    1/503, Learning Rate:0.000001 / Loss -> GIoU:  0.68, Confidence:  0.00, Probability:  0.21, Total:  0.89
```

Figura 44 Visualização do estado do treino, nomeadamente: número de epoch, step, learning rate e losses.

O *IoU*, segundo o autor [70], não tem tanta relevância em ser utilizado como métrica de avaliação de *loss*, pelo simples facto de no caso de dois objetos não se intersetarem este não conseguir informar a que distância os mesmos se encontram. Como o gradiente é zero, o algoritmo não é otimizado com tanta precisão. Por este motivo, o *GioU* foi utilizado como métrica avaliadora da *loss* pois permite avaliar a performance na deteção de objetos de forma mais rigorosa.

A *confidence loss* mede a certeza que o algoritmo tem em determinar se numa *bounding box* existe, ou não, um objeto. Se a *bounding box* prevista contiver objetos alinhados com o centro da *bounding box* então tem *confidence*=1, caso contrário, tem *confidence*<1 (este valor é variável conforme o objeto estiver, ou não, mais perto do centro da *bounding box*).

A *class probability loss*, mede a probabilidade de um determinado objeto pertencer a uma classe. Tanto a confidence loss como a *class probability loss* usam *logistic regression*. Estes cálculos executados de forma iterativa, permitem a construção de gráficos no *Tensorboard* (*API* do *Tensorflow*) com informação relativa ao treino dos algoritmos depois de este ter sido realizado.

O *mAP* é outra métrica de avaliação da performance do *YOLOv3-tiny*. Resulta do cálculo de dois parâmetros importantes do *YOLO* que são: *Precision* e *Recall*. *Precision*, diz respeito à proporção de Positivos Verdadeiros de entre todos os Positivos (Verdadeiros e Falsos). *Recall*, diz respeito à proporção de Positivos Verdadeiros encontrados de entre todos os Positivos realmente existentes. Por outras palavras, *Precision* permite ver a proporção de quantos dos positivos encontrados são realmente verdadeiros e, *Recall*, permite verificar se alguns Positivos Verdadeiros foram deixados de fora. As fórmulas são as seguintes:

$$Precision = \frac{TP}{TP+FP} \qquad Recall = \frac{TP}{TP+FN} \tag{4.2}$$

TP = True Positive, TN= True Negative, FP= False Positive, FN= False Negative

Estes dois parâmetros têm intervalos entre 0 e 1 e, portanto, a equação do *AP* (*Average Precision*) também tem limites 0 e 1. Destas duas variáveis resulta um gráfico, *p(r)*, com uma curva e o *AP* é a área da mesma.

$$AP = \int_0^1 p(r)dr \tag{4.3}$$

Para cada classe é calculado o *AP* e, para melhor avaliação geral da performance, introduziu-se o *mAP* que corresponde à média do *AP* de cada classe. A performance do sistema no seu todo, é avaliada pelo *mAP*, como se pode ver no exemplo da figura seguinte.

```
86.658% = cone AP
98.486% = divider AP
99.259% = sign AP
mAP = 94.801%, 26.43 FPS
```

Figura 45 Exemplo do mAP calculado a partir do AP de cada classe.

No entanto, não foi justificável implementar o cálculo do progresso do *mAP*, ao longo do treino, pelo seu custo computacional, que resultaria num tempo de treino muito superior ao desejado bem como uma capacidade de memória muito superior à disposta pelo *Google Drive*. O cálculo do *mAP* exige o seguinte processo: criação de um modelo, introdução dos *weights* e classificação das imagens da parte da validação do *dataset*. Este processo efetuado iterativamente ao longo do treino, é computacionalmente dispendioso, e mesmo no *Google Colaboratory* o tempo de execução aumenta decisivamente. Por este motivo, o seu cálculo é feito no final do treino, tanto para o treino como para a validação, que é o fundamental para analisar o sucesso do treino. É importante referir que para se detetar *overfitting*, apenas é necessário visualizar o gráfico da evolução da *loss*, tanto para o treino como para a validação.

Depois de treinada a rede neuronal, resultam dois ficheiros (no *Tensorflow* de versão superior a 2.0) relativos aos novos *weights*, bem como as *logs* para se poder visualizar no *Tensorboard* os gráficos da *loss*. Além disso, resultam ainda ficheiros com os resultados do *mAP*.

4.4. YOLOv3-tiny: Teste

Depois de treinado, o sistema deve ser testado. O teste pode ser descrito por um conjunto de ações sequenciais, como se verifica no fluxograma da figura 46.

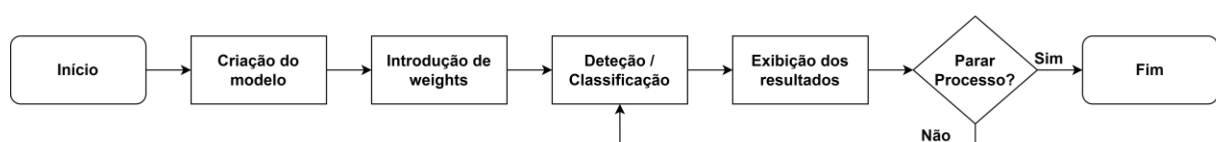


Figura 46 Diagrama que representa o comportamento do sistema em situação de teste.

O modelo deve ser criado indicando as classes dos objetos e introduzindo os respetivos *weights* que resultaram do processo de treino. Usando o *OpenCV*, todas as *frames* captadas pela *webcam* em tempo real são processadas. Cada *frame* é sujeita a um pré-processamento que nada mais é que um *resizing* da imagem para a resolução pretendida. Este pré-processamento é relevante na medida em que, a resolução de qualquer *webcam* conectada ao computador não afeta a eficiência do algoritmo porque todas as *frame*s são convertidas para o tamanho desejado, 416x416. Após isto, a imagem é introduzida no modelo para este prever as *bounding boxes* na imagem. Como o algoritmo gera múltiplas *bounding boxes* para um objeto, é executado um pós-processamento para eliminar as bounding boxes com pouca *confidence*, usando um *score threshold*. Na figura 47, encontra-se uma amostra do resultado do algoritmo depois de realizado o pós-processamento com *score threshold*=0.3.

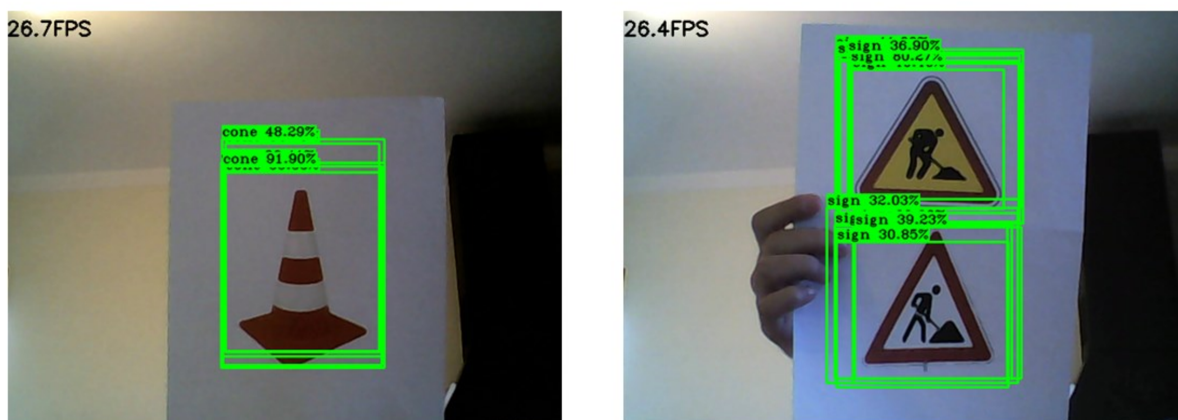


Figura 47 Bounding boxes restantes após aplicação de pós-processamento. Pode-se verificar que apenas ficam as bounding boxes com classificação superior a 30%.

Após as boxes com *confidence* mais baixa serem eliminadas, podem ainda ficar *bounding boxes* duplicadas por terem *confidence* elevada. Estas são eliminadas pelo *Non Maximum Supression* (*NMS*), um processo que escolhe a *bounding box* com a maior *confidence* e calcula o *IoU* das restantes *bounding boxes* em relação a essa. Se o *IoU* for superior a um *threshold* estabelecido então são eliminadas uma a uma, e faz isto até permanecer a mais correta. Na figura 48 encontra-se o resultado depois de aplicado *NMS* ao algoritmo de deteção, com iou_threshold=0.45.

Figura 48 Representação da aplicação de NMS.

Por fim, permanecem as melhores *bounding boxes* para o utilizador ter uma perceção visual do comportamento do algoritmo bem como do seu desempenho. Juntamente com a *frame*, aparece o número de *frames* por segundo que o algoritmo é capaz de obter no computador, bem como o nome da classe e a *confidence* de cada respetiva *bounding box*. Neste exemplo a deteção está a ser realizada de duas em duas *frames* porque a captação de vídeo mostrou-se demasiado lenta, devido a limitações de hardware. Ou seja, a taxa de amostragem de deteção é cerca de 13 *FPS* e a de captação é aproximadamente 26 *FPS*.

4.5. Ambiente de simulação: ROS, CoppeliaSim e Pycharm

Para poder testar a integração do *YOLOv3-tiny* o *DDPG*, foi necessário escolher um *software* para desenvolver um ambiente de simulação. O *CoppeliaSim* foi escolhido por ser bastante conhecido e utilizado pela comunidade científica em inúmeras aplicações de Robótica.

Na figura 49, pode ver-se uma imagem retirada do cenário de simulação construído. É constituído por um ponto de partida, duas fileiras de cones estrategicamente posicionadas e um ponto de chegada. O objetivo é o veículo mover-se do ponto de partida para o ponto de chegada sem violar as regras.

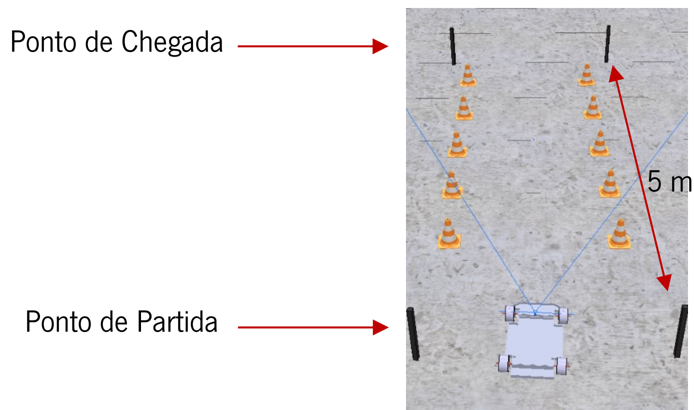


Figura 49 Captura de ecrã do cenário de simulação construído no CoppeliaSim.

Foi também necessário conceber um modelo do veículo, de preferência semelhante ao exigido na prova de condução autónoma. O modelo do veículo foi fornecido pelo LAR e pode ver-se na figura 50 o veículo utilizado com cerca de 0,80 metros de comprimento.

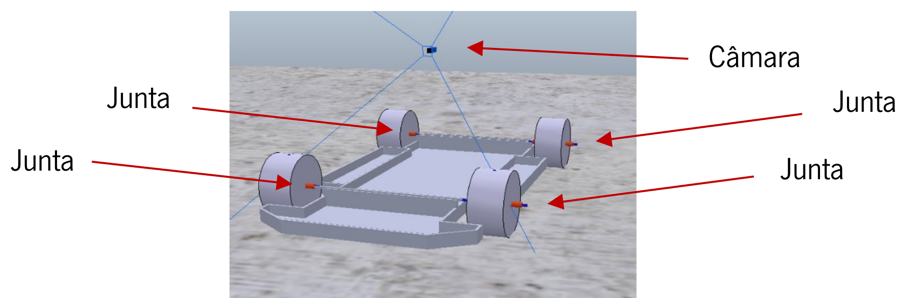


Figura 50 Modelo do veículo utilizado para treino e teste do sistema.

É maioritariamente constituído por um chassis com quatro juntas que possibilitam o acoplamento das quatro rodas ao veículo. Dispõe ainda de uma câmara numa posição aproximadamente coincidente com a frente do tejadilho de um veículo para se obter uma perspetiva mais abrangente da via pública. De forma a tornar os processos de treino e teste mais fidedignos, a orientação inicial do veículo em cada episódio, varia num intervalo de ± 30 graus.

Após reunidas as condições mínimas necessárias para o ambiente de simulação, foi necessário um protocolo de comunicação para os módulos comunicarem entre si. O *Robot Operating System*(*ROS*) é um conjunto de bibliotecas que permite o estabelecimento de um protocolo de comunicação entre duas entidades por meio de nós que podem publicar ou receber mensagens entre eles. É um sistema *open-source* utilizado por todo o mundo, quer em projetos de investigação, quer na conceção de produtos específicos.

É uma mais-valia neste projeto na medida em que permite o envio de imagens captadas pela câmara do veículo, bem como a receção das mesmas pelo algoritmo *YOLOv3-tiny*, para este fazer a deteção da

respetiva sinalização temporária. É também responsável pelo envio do espaço de ações comandado pelo *DDPG* para o veículo no *CoppeliaSim*. Os programas em *ROS* podem ser desenvolvidos em duas linguagens de programação, *C++* e *Python*. Foi utilizado o *Python* pelo facto de o projeto ter sido desenvolvido nessa linguagem, com a exceção dos *scripts* do *CoppeliaSim* que são, por definição, programados em linguagem *Lua*.

Na figura 51, encontra-se um diagrama que simplifica todo o processo efetuado pelo sistema e um diagrama do *rqt_graph* que mostra os processos em uso.

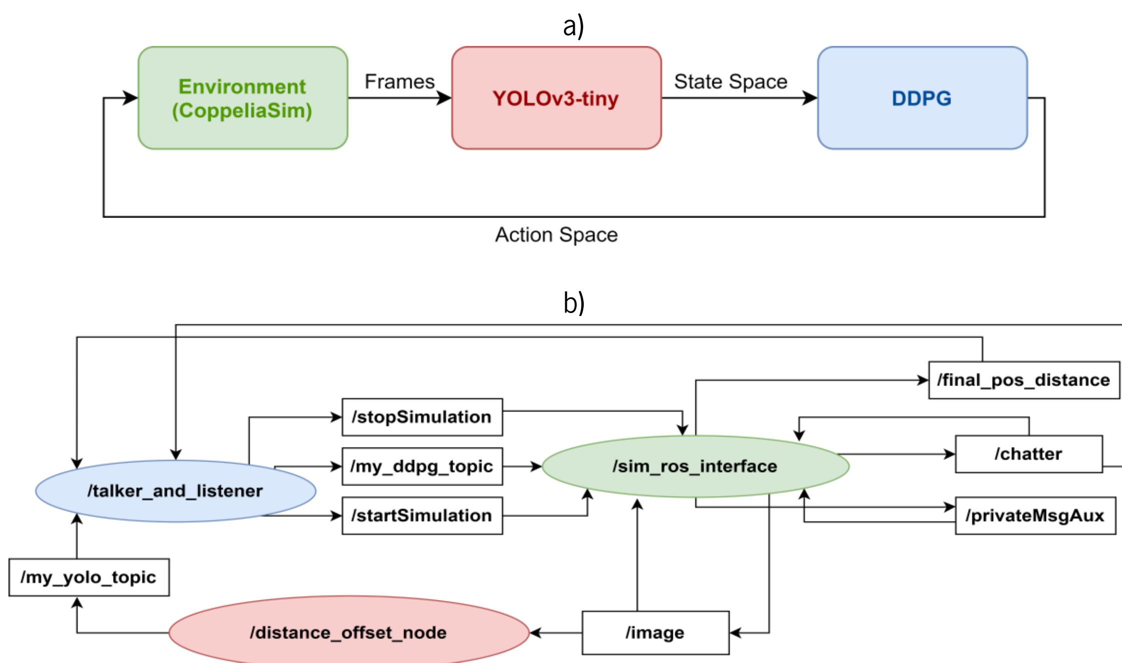


Figura 51 a) diagrama simplificado da comunicação que o ROS deve sustentar. b) diagrama adapatado do rqt_graph que permite visualizar todos os processos em uso.

Analisando o diagrama retirado do *rqt_graph*, verificam-se os nós em forma de elipse e os tópicos em forma retangular. O nó *sim_ros_interface envia imagens* do *CoppeliaSim* para o *distance_offset_node*, através do tópico */image*. De seguinda, são enviadas pelo */my_yolo_topic* duas componentes da função de *reward* para o nó *talker_and_listener* – neste nó encontra-se o DDPG. São elas a *center_offset* e a *cone_distance* necessárias para o correto funcionamento do *DDPG*. O *talker_and_listener* recebe também a velocidade atual do veículo no *CoppeliaSim*, pelo *chatter*, bem como a distância ao ponto de chegada, pelo *final_pos_distance*. De seguida, envia o espaço de ações através do *my_ddpg_topic*, para comandar o veículo.

O exemplo anterior diz respeito à primeira solução encontrada para o sistema. Contudo, com o objetivo de encontrar melhores resultados foi introduzida uma outra solução que teve como efeito reduzir a amplitude do efeito negativo que alguns obstáculos poderiam causar à primeira solução.

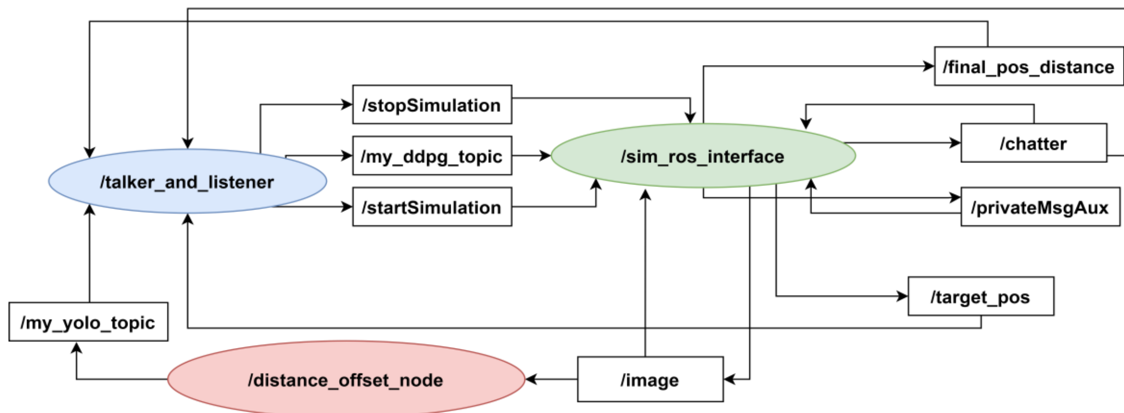


Figura 52 Diagrama adaptado do rqt_graph que mostra a reformulação executada para obedecer aos requisitos da segunda solução.

É uma solução semelhante à primeira, mas o conteúdo enviado pelo tópico */my_yolo_topic* é diferente. Além disso, pode-se visualizar um novo tópico */target_pos* que envia para o DDPG a posição do *target*, um novo paradigma no sistema que é mencionado no subcapítulo 4.6. Devido ao elevado tempo de processamento do *YOLOv3-tiny*, a frequência do ROS estipulada é de 5 Hz.

4.6. DDPG: Configuração do modelo

Como foi visto nos Fundamentos Teóricos, o algoritmo requer um espaço de estados, S, para poder ter informação relevante sobre o ambiente de modo a indicar ao agente as próximas ações a tomar, através do espaço de ações, A. Assim, para a primeira solução definiu-se o espaço de estados e o espaço de ações da seguinte forma:

$$S = \{center_{offset}, speed_{instant}, distance_{cone}\}$$
$$A = \{steering_{applied}, speed_{applied}\} \quad \textbf{(4.4)}$$

As variáveis do espaço de ações dizem respeito às manobras de direção aplicadas pelo veículo e à velocidade aplicada pelo mesmo, respetivamente.

As variáveis do espaço de estados são o desvio veículo ao centro da via, a sua velocidade instantânea e a sua distância ao cone mais próximo da via, respetivamente. Estas são explicadas ao pormenor no subcapítulo alusivo ao treino (subcapítulo 4.7.) por estarem presentes no sistema de *reward* utilizado para o treino do algoritmo.

Já para a segunda solução testada, o espaço de ações mantém-se o mesmo, mas o espaço de estados recebe outras informações sobre o ambiente:

$$S = \{intersection_{matrix}, distance_{finish_line}\}$$
$$A = \{steering_{applied}, speed_{applied}\} \quad \textbf{(4.5)}$$

A variável $all_{lines_{intersections}}$ é uma matriz de dimensões 16x3 e a $distance_{finish_line}$ é o valor da distância à linha final. Esta matriz surge devido ao processamento, diferente da primeira solução, efetuado nas imagens captadas pelo veículo, como se pode ver na figura 53.

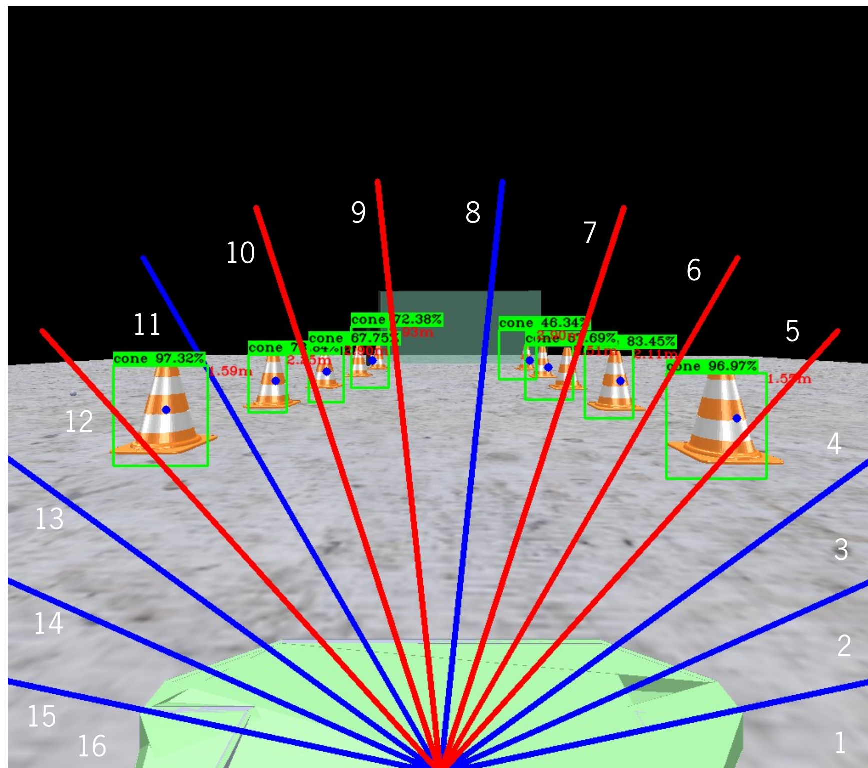


Figura 53 Captura de ecrã do processamento de imagem efetuado pela segunda solução. As 16 linhas estão numeradas da direita para a esquerda em conformidade com a orientação do círculo unitário.

Existe um ponto único que une 16 linhas de igual comprimento que se separam entre elas por um ângulo de 12° ao longo de 180°. Estas linhas servem como orientação para o DDPG perceber onde se situam os cones e responder de forma rápida. Sempre que um cone interseta uma linha, esta fica a vermelho para melhor perceção do que está a acontecer. Cada interseção coloca uma *flag* a 1 correspondente à linha. Essa *flag* é armazenada na matriz ocupando a primeira a coluna e a linha correspondente. No caso de haver mais do que um cone a intersetar a mesma linha, é considerado o cone mais perto por ser o mais relevante. A segunda coluna da matriz indica a que distância do veículo se encontra esse cone, em percentagem. As linhas que não são intersetadas são acompanhadas de uma distância ao cone de valor 1, porque ficou estipulado que 100% representa uma distância inalcançável (um valor alto à qual o *YOLOv3-tiny* já não deteta cones e, portanto, não interfere com as distâncias dos cones que intersetam as linhas). A última coluna da matriz diz respeito à direção do *target*. Para se saber a direção do target foi necessário introduzir um *array* de sensores no ambiente de simulação que o permitisse detetar como se pode ver na figura 54.

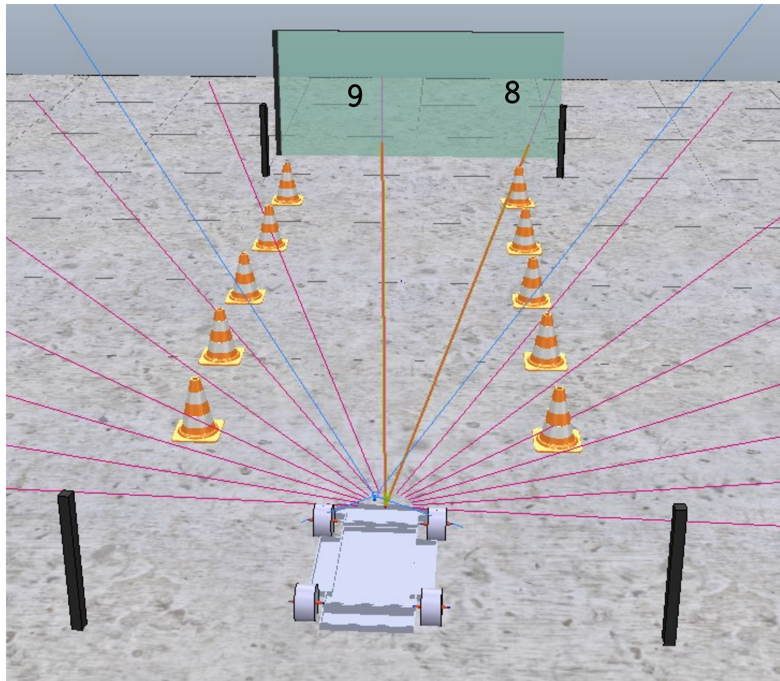


Figura 54 Captura de ecrã do ambiente de simulação integrando a segunda solução.

Os sensores foram colocados estrategicamente a coincidir com as linhas desenhadas nas *frames* captadas pela câmara de forma a mapear na matriz as linhas que intersetam o *target*, usando uma *flag* a 1. Na tabela 2, encontra-se um exemplo de como é constituída a matriz final.

Tabela 2 Exemplo de uma matriz enviada para o DDPG.

	Interseção	Distância (%)	Target
Linha 1	0	1.	0
Linha 2	0	1.	0
Linha 3	0	1.	0
Linha 4	0	1.	0
Linha 5	1	0.39	0
Linha 6	1	0.53	0
Linha 7	1	0.63	0
Linha 8	0	1.	1
Linha 9	1	0.73	1
Linha 10	1	0.725	0
Linha 11	0	1.	0
Linha 12	1	0.4	0
Linha 13	0	1.	0
Linha 14	0	1.	0
Linha 15	0	1.	0
Linha 16	0	1.	0

Este novo paradigma pretende dar um sentido de orientação ao veículo, principalmente nas curvas acentuadas, onde a câmara por vezes não capta o caminho livre para o veículo percorrer e este acaba

por não chegar à linha final porque só encontra cones (aos quais não pode passar no meio senão é penalizado pelo sistema).

4.7. DDPG: treino

A função de *reward* do *DDPG* é responsável por calcular a recompensa resultante do comportamento do algoritmo a cada instante de um episódio num determinado treino. Neste projeto, pretende-se que o veículo chegue ao final da via delimitada pela fileira de cones sem colidir com nenhum deles e muito menos ultrapassar os limites da via. Após alguns testes considerou-se a seguinte metodologia para a primeira solução:

$$reward = A * speed_{instant} - B * center_{offset} - C * distance_{finish_{line}} + D * distance_{cone} \quad \textbf{(4.6)}$$

A primeira componente é a velocidade do veículo num dado instante, que retorna um valor positivo para o veículo pois pretende-se que faça o percurso o mais rápido possível dentro dos limites de velocidade estabelecidos e respeitando a segurança. Pretende-se também que o veículo não se efetue marcha-atrás e para isso tem retorno negativo no caso de a velocidade ter sinal negativo. A segunda componente diz respeito ao *offset* entre o centro da *frame* da câmara e o centro da via pública calculado através da distância entre os cones mais próximos. A terceira componente corresponde à distância do veículo ao ponto de chegada e a última componente é a distância do veículo ao cone mais próximo. Para todas as componentes da expressão que são negativas, o agente deve fazer os possíveis para minimizar o impacto negativo das mesmas. Para as componentes positivas, o agente é recompensado por aumentar o seu valor. As letras A, B, C e D representam coeficientes sujeitos a alterações conforme os resultados obtidos assim o exijam.

Todas as componentes foram definidas após um estudo do ambiente. A velocidade instantânea do veículo é enviada pelo *CoppeliaSim* por motivos de segurança, uma vez que a velocidade comandada diretamente pelo DDPG pode não ser aplicada em algumas situações. O algoritmo vai comandando velocidades para o veículo, mas pode não corresponder à velocidade real do mesmo, como por exemplo, no momento em que o veículo colide contra um cone. Por questões de segurança, está previsto no *CoppeliaSim* um comando automático que trava o veículo levando a sua velocidade até 0 km/h. Esta norma de segurança permite indicar ao *DDPG* que o episódio acabou, independentemente de quaisquer ações que esteja a comandar para o veículo. Assim, se a velocidade na função de *reward* fosse a mesma do espaço de ações o cálculo da *reward* estaria automaticamente errado porque não se estaria a utilizar a verdadeira velocidade do veículo naquele instante. O comando de segurança que leva a velocidade a

zero ativa quando o veículo: chega ao ponto de chegada, está fora dos limites da via, pára ou colide contra um cone.

O ambiente de simulação envia esse valor de velocidade acompanhado de uma nota para o *DDPG* encerrar o episódio descrevendo o motivo. O *DDPG* toma as medidas necessárias de modo a restaurar o bom funcionamento do sistema. Qualquer um dos motivos presentes para o veículo estar parado, cobre as principais situações possíveis em que o veículo se pode encontrar parado e resulta num recomeço de um novo episódio. A diferença reside apenas no valor de *reward* obtido. Se, por exemplo, o veículo estiver parado porque chegou ao ponto de chegada será recompensado positivamente, mas se estiver parado porque colidiu contra um cone será recompensado negativamente. A quantidade de recompensa associada a cada situação é ajustada no sistema de recompensas tendo em conta os resultados obtidos.

A segunda componente da função de *reward*, $center_{offset}$, é a segunda variável existente no espaço de estados e é obtida pela diferença entre a posição do centro da *frame*, $center_{frame}$, subtraída da posição do centro da via, $center_{road}$, em módulo.

$$center_{offset} = |\, center_{frame} - center_{road}| \tag{4.7}$$

As coordenadas do centro da *frame* são obtidas pela seguinte expressão.

$$center_{frame_x} = \frac{length(image_x)}{2} - 1$$

$$center_{frame_y} = \frac{length(image_y)}{2} - 1 \tag{4.8}$$

$center_{frame_x}$ e $center_{frame_y}$ são as coordenadas do centro nos eixos x e y, respetivamente. $length(image_x)$ e $length(image_y)$ são as dimensões da *frame*.

Na figura 55, pode-se observar o centro da *frame* representado por um ponto vermelho. O centro da via representado pelo ponto azul é obtido através de processamento adicional sobre a *frame*, visto que não existe uma referência visível na própria via. É o ponto médio da uma linha imaginária vermelha que separa os cones de uma fileira para a outra. Para traçar essa linha, o algoritmo identifica os cones mais perto de si através da implementação de um filtro de cor laranja dentro de cada *bounding box* para evidenciar a estrutura do cone na *frame*. Com este filtro foi possível calcular-se o ponto mais alto e o ponto mais baixo de cada cone, descobrindo-se a altura do cone, em pixéis. Assumindo que todos os cones presentes no ambiente têm o mesmo tamanho, torna-se evidente que os cones maiores na *frame* serão os cones mais perto do veículo. O algoritmo traça uma linha entre eles e esta fica estipulada como sendo a largura da via. Assim, calcula-se o seu ponto médio bem como o seu offset ao centro da frame.

Previamente, experimentou-se calcular o tamanho do cone pelo tamanho da sua respetiva *bounding box*, contudo, mostrou-se um método pouco viável pelo facto de o tamanho da *bounding box* variar constantemente e não coincidir sempre com as extremidades do cone.

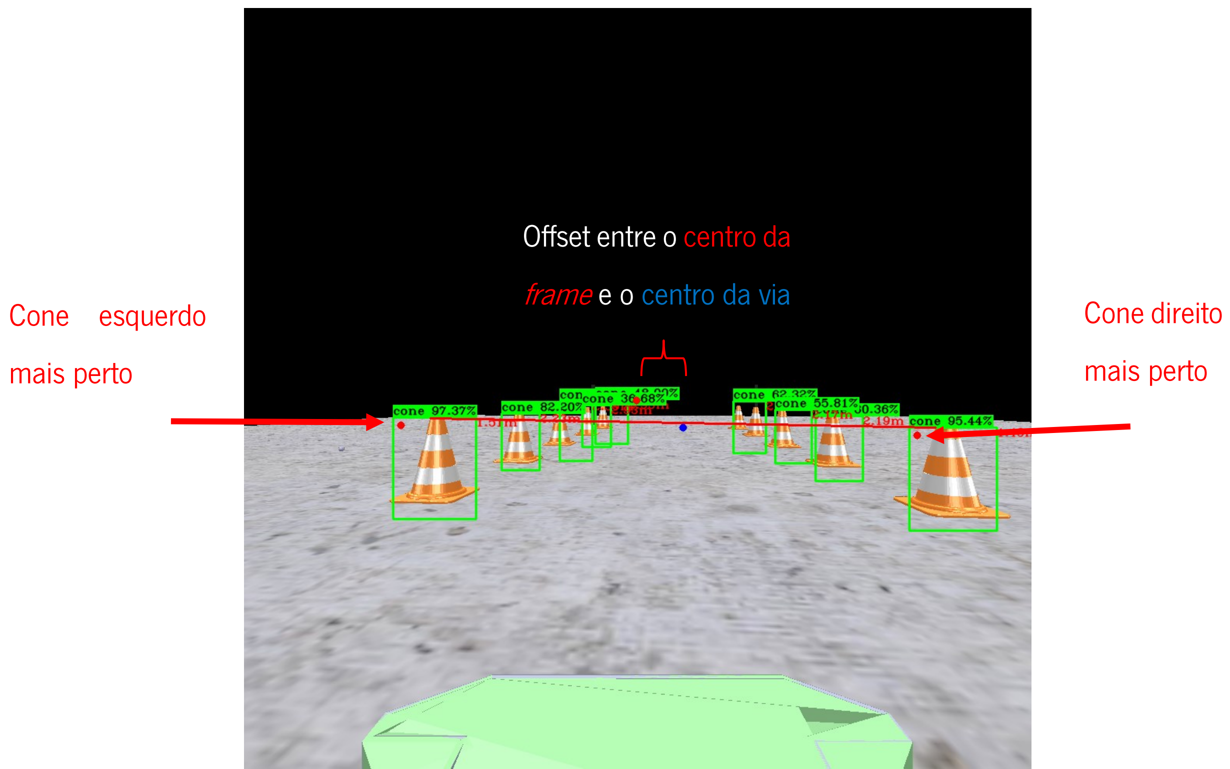


Figura 55 Captura de ecrã do processamento de imagem que é feito, pela primeira solução, por cima de uma frame captada pela câmara do veículo.

Pretende-se que o veículo minimize ao máximo o *offset* pois significa que está próximo do centro da via. Caso contrário, significa que o veículo está prestes a colidir contra os cones, algo que não é de todo desejável.

A terceira componente da função de *reward* é calculada no *CoppeliaSim* com auxílio de duas barras de referência. Através do cálculo da posição do centro entre as duas barras ao final da via, juntamente com a posição do centro do veículo, calcula-se a distância entre os dois pontos em tempo real que corresponde à distância a que o veículo se encontra do ponto de chegada.

Por fim, a distância do veículo ao cone é uma outra componente que o agente deve maximizar o mais possível. Esta é obtida pela mesma medição da altura dos cones para a variável anterior. Durante o processamento dos cones na *frame* é calculado o tamanho de todos, como já foi visto. O tamanho de todos é, posteriormente, armazenado num buffer. Percorrendo o buffer, o maior cone é identificando como o cone mais próximo do veículo. Foi necessário estimar a distância do carro ao cone em metros, utilizando para isso, a respetiva função criada e calibrada para esse efeito. Criou-se um pequeno *dataset* com algumas dezenas de pontos retirados de uma amostragem experimental no ambiente de simulação,

onde se fez a associação entre a distância real e o tamanho do cone na *frame*, em pixéis. Utilizando regressão não-linear, foi possível encontrar a seguinte função que prevê, com um erro médio de 30cm, a distância do carro ao cone dentro dos limites estabelecidos.

$$y = 768.2 - 131.5 * \log x \quad \textbf{(4.9)}$$

y é a distância real em metros e x é a distância em píxeis.

Para a segunda solução implementada, a função de *reward* é a seguinte:

$$reward = A * speed_{instant} - B * distance_{finish_line} - C * \Delta angle_{direction} - D * step \quad \textbf{(4.10)}$$

As duas primeiras variáveis são as mesmas da solução anterior. O $\Delta angle_{direction}$ diz respeito à variação da direção do veículo e quanto mais brusca for maior será a penalização. É calculado pela diferença entre a direção no instante atual e a direção no instante anterior em módulo. O *step* é um contador de *steps* de cada episódio e penaliza o algoritmo à medida que este vai demorando mais tempo para terminar o episódio. Se o veículo andar muito devagar ou parar, acumulará uma *reward* bastante negativa porque o número de *steps* vai aumentado gradualmente, aumentando também o impacto negativo na equação.

As normas de segurança e situações que fazem terminar o episódio são as mesmas da solução anterior.

4.8. DDPG: teste

O teste do algoritmo foi executado no mesmo ambiente de simulação. Para isso, introduziu-se os weights resultantes do treino nas redes neuronais. A diferença reside no facto de que os parâmetros do processo de exploração são configurados a zero, incluindo o encerramento do replay buffer. O algoritmo efetua o teste utilizando exclusivamente informação que reteve no processo de aprendizagem ou *exploitation*.

5. Resultados

Neste capítulo são apresentados os resultados obtidos pelos treinos e testes consecutivos efetuados ao sistema. Nomeadamente, na secção da deteção de sinalização temporária, os treinos e testes foram feitos em duas partes como foi explicado na secção 4.1. Na primeira parte, foi escolhido um valor padrão para os hiperparâmetros do sistema de forma a ser possível otimizar o *dataset*, acompanhando o impacto que as alterações ao mesmo produziam. Na segunda parte, depois do *dataset* estar concluído, foram alterados os hiperparâmetros seguindo uma norma explicada nas páginas seguintes.

Na secção da movimentação na via delimitada por sinalização temporária, foram registados os resultados obtidos pelos treinos e testes consecutivos do algoritmo de *DDPG*. Mais especificamente, a quantidade de treino necessária para se obterem os resultados apresentados utilizando as configurações de *reward* revistas e os hiperparâmetros indicados.

5.1. Deteção de sinalização temporária

Os hiperparâmetros são variáveis que controlam o processo de treino e podem determinar o seu sucesso ou insucesso, o que torna a sua configuração importante. Escolher uma configuração para todos é algo baseado no conhecimento e experiência do utilizador em lidar com os mesmos. Isto significa que não existe um modelo específico para os determinar que não a prática. O objetivo é otimizar ao máximo o treino produzindo os melhores resultados possíveis.

Os respetivos valores padrão foram escolhidos com base na pesquisa de outros projetos semelhantes de forma a criar uma referência que serve como base para o treino. Alguns são mesmo recomendados pelos autores e outros são estipulados por *frameworks* como o *Keras*. Na lista seguinte estão apresentados os hiperparâmetros bem como os seus valores padrão:

- *Epochs* = 100

As *epochs* são as iterações de um treino. O número de *epochs* de um treino representa o número de ciclos a que a informação vinda do *dataset* introduzida para treino. Se este número for muito baixo o sistema fica aquém daquilo que é capaz de aprender, se for muito alto o sistema fica mais vulnerável ao fenómeno de *overfitting*.

- **Divisão do *dataset* = 80% (treino)/20% (teste)**

A divisão do *dataset* diz respeito às parcelas de treino e teste de um *dataset*. É esperado que haja mais amostras para treinar o algoritmo do que para o testar, caso contrário o algoritmo não fica bem treinado.

- ***Learning Rate*** **= 0.0001**

O *learning rate* diz respeito à velocidade com o que o algoritmo aprende. Valores baixos geram treinos lentos, valores altos, geram treinos rápidos. Esta variável permite ao otimizador *Adam* ajustar a velocidade do treino.

- ***Batch Size*** **= 4**

O *Batch Size* corresponde ao número de amostras a serem introduzidas no algoritmo em cada *step* do seu treino. Valores baixos desta variável permitem treinos mais estáveis e valores altos permitem treinos mais rápidos, mas aprendizagens mais instáveis.

- ***Kernel Regularizer*** **= 0.001**

O *Yolov3-tiny* inclui um regularizador L2 nas camadas convolucionais para prevenir o modelo de fazer *overfitting*. Este regularizador introduz uma componente que mede a complexidade do modelo, calculada através da soma do quadrado de todos os *weights* das *features*. Este hiperparâmetro introduz uma penalização para controlar a atividade das *layers* e o seu contributo para a soma.

- ***Leaky ReLU*** **(*alpha*) = 0.3**

A função de ativação das camadas convolucionais é a *Leaky ReLU*. Esta função possui um declive para x<0, *alpha*, que é variável e o seu valor padrão definido pelo *Keras* é 0.3.

- ***Anchors*** **= [[[10, 14], [23, 27], [37, 58]],**
  [[81, 82], [135, 169], [344, 319]],
  [[0, 0], [0, 0], [0, 0]]]

As *anchor boxes* já foram discutidas previamente. Os valores padrão escolhidos são do *COCO dataset*, um dos maiores *datasets* utilizado na maior parte dos testes efetuados ao algoritmo *YOLOv*3 pela comunidade científica. A ultima linha diz respeito às *anchor boxes* para a deteção de objetos pequenos. Como não são detetáveis nesta arquitetura ficam a zero. Os restantes valores são as dimensões das *anchor boxes* para objetos médios e grandes.

- ***Data Augmentation*** **= On**

O *data augmentation* tem como função prevenir o *overfitting* realizando múltiplas transformações às imagens introduzidas pelo *dataset*. Este hiperparâmetro deve estar ligado. Quando desligado é de esperar que o *overfitting* se verifique mais acentuado.

- **Resolução de entrada = 416x416**

A resolução das imagens de entrada ficou estipulada pelo autor em 416x416, que é um valor que se traduz num bom compromisso entre capacidade de deteção e velocidade.

- *IoU threshold* = 0.5

O *threshold* do *IoU*, utilizado para calcular a *GiOU loss* e para o processamento do *NMS*, é decisivo para determinar se uma *bounding box* contém um objeto ou não. O valor 0.5 estipula a presença de objeto no caso de interseção mínima de 50% entre a *predicted bounding box* e *a ground truth bounding box*.

- *Score threshold* = 0.3

O *score threshold* permite mostrar apenas as *bounding boxes* com uma *confidence* acima dele mesmo. Um *score* de 0.3 significa que as *bounding boxes* com *confidence* abaixo de 30% serão descartadas.
Os valores estipulados para os hiperparâmetros utilizados durante os testes do algoritmo são iguais aos atribuídos para os treinos. Após todos os hiperparâmetros estarem configurados com um valor padrão, é introduzido o *dataset* e são efetuadas as suas sucessivas otimizações.
O tempo de duração de cada treino rondou em média 5 horas utilizando o *Google Colaboratory*. Pode-se concluir que os 17 treinos realizados durariam cerca de 85 horas no total, se não existissem pausas. Na prática, todo o processo de treino demorou substancialmente mais tempo. Os testes realizados entre cada treino acompanhados do trabalho manual no *GIMP* para criar e modificar as imagens, resultou em cerca de 1 mês de trabalho. Foi um processo de treino do sistema para verificar a performance do mesmo, com o intuito de desenvolver e otimizar o *dataset* desejado.
No início, começou-se por desenvolver um *dataset* com cerca de 553 imagens de cones de estrada e respetivas anotações. O *YOLOv3-tiny* foi sujeito a um treino tendo-se obtido os seguintes resultados:

Tabela 3 Resultados obtidos para o treino inicial.

mAP	FPS
88%	10

No segundo treino foi efetuada uma otimização ao modelo e introduziram-se as fitas de barreira e o sinal de trabalhos na via, no *dataset*. Os resultados foram os seguintes:

Tabela 4 Resultados obtidos para o segundo treino (com AP obtido para as diferentes classes).

mAP	FPS
74%	29

Objeto	Cone	Sign	Barrier
AP	85.4%	100%	35.4%

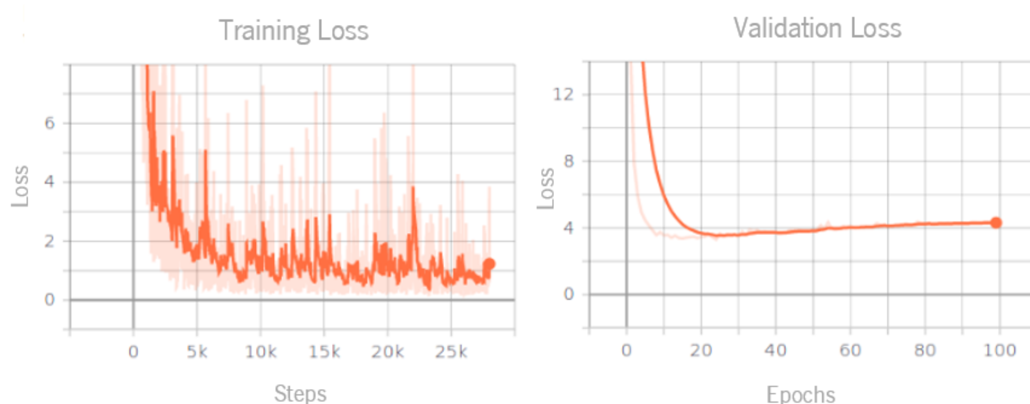


Figura 56 Visualização da loss no 2º treino e na respetiva validação.

Pode-se verificar que ocorreu uma descida de 88% para 74% do *mAP*, resultante da introdução das fitas de barreira que obtiveram um AP de 35%. O *frame rate* aumentou para 29 *FPS*, resultado da otimização efetuada no modelo. Além disso, pode-se verificar que o modelo tem um pouco de overfitting pela análise do gráfico da validation loss.

De seguida foram efetuados 8 treinos consecutivos, nos quais o *dataset* foi alterado de várias formas. As fotos foram alteradas no *GIMP*, foi aumentado o número de fotos no *dataset* e ainda foram retiradas as fitas de barreira. A tabela 5 e a figura 57 mostram os resultados do 9° treino.

Tabela 5 Resultados obtidos no 9º treino (com AP obtido para as diferentes classes).

Treino	mAP	FPS	Dataset (Fotos)
9	93.1%	30	1090

Objeto	cone	sign
AP	86.2%	99.9%

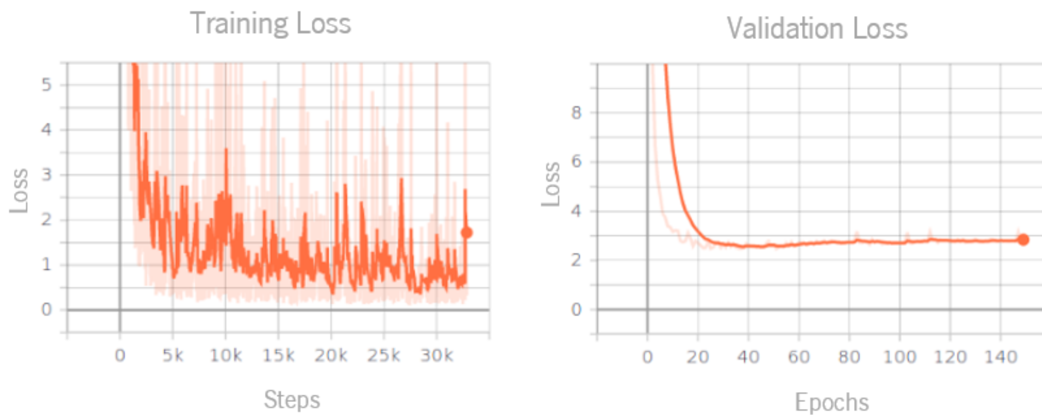


Figura 57 Visualização da loss obtida na fase de treino e na fase de validação para o 9º treino.

Depois de retiradas as fitas de barreira, o *mAP* aumentou perante os treinos anteriores e o *frame rate* aumentou para 30 *FPS*, resultando nos melhores valores obtidos até ao momento. Também se observa uma melhoria na *validation loss*. O sistema foi testado e notou-se que o algoritmo confundia um pouco os sinais de trabalhos na via com outros sinais triangulares pela sua semelhança.

Face a isto foram efetuados 2 novos treinos, mas desta vez acrescentando imagens com sinais triangulares que não fazem parte da classificação com o intuito do algoritmo aprender a distinguir as diferenças entre eles e ignorá-los. Os resultados são os seguintes:

Tabela 6 Resultados obtidos para os 10º e 11º treinos (com AP obtido para as diferentes classes).

Treino	mAP	FPS	Dataset (Fotos)
10	93.0%	26.9	1090
11	93.7%	27.2	1110

	AP	
Treino	Cone	Sign
10	86.8%	99.2%
11	87.4%	100.0%

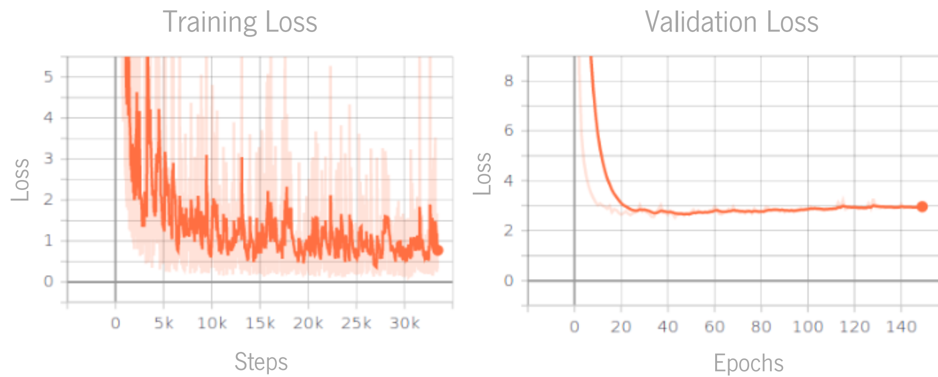


Figura 58 Visualização da loss na fase de treino e teste para o 11º treino.

De um treino para o outro nem sempre se aumentou o número de imagens. No caso do treino 9 para o 10, foi efetuada uma alteração às imagens concedendo-lhes mais distinção entre elas mas não adicionando novas.

Após análise das tabelas, percebe-se uma quebra de performance do treino 9 para o 10. No entanto, apesar do *mAP* descer 0.1% pela descida do *AP* do sinal, verifica-se que o *AP* do cone aumentou 0.6% face ao treino anterior. No treino 11, houve uma melhoria em geral. Este foi o melhor resultado até ao momento para os dois sinais. O sistema foi testado e demonstrou-se capaz de uma deteção rápida e rigorosa. Foi concluído que para não pôr em causa o sucesso na deteção destes dois objetos, as fitas de barreira tinham de ser substituídas por outro objeto. Os separadores de via foram adicionados ao dataset. Por motivos de experiência, foi efetuado um treino com as mesmas características do treino 11, mas com o *data augmentation* desligado. Os resultado estão descritos na tabela 7 e na figura 59.

Tabela 7 Resultados obtidos no 12º treino (com AP obtido para as diferentes classes).

Treino	mAP	FPS	Dataset (Fotos)
12	91.5%	26.2	1110 (sem alteração)

	AP	
Treino	Cone	Sign
12	83.7%	99.2%

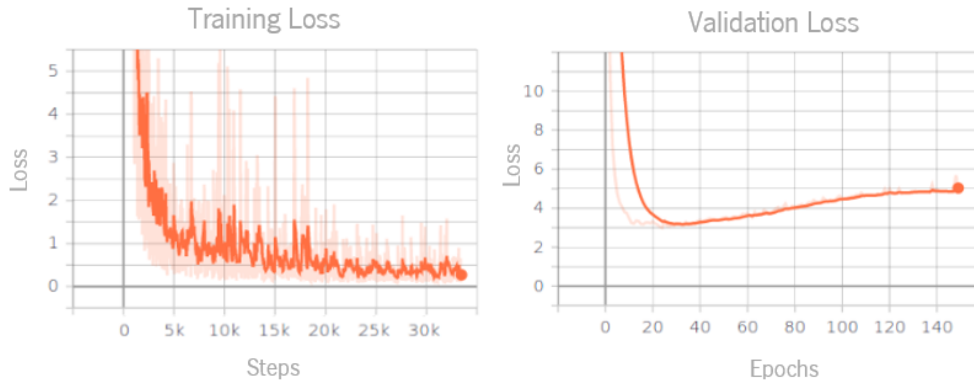


Figura 59 Visualização da loss para a fase de treino e validação num teste experimental com data augmentation desligado.

Este treino sem *data augmentation* demonstrou um impacto de 2.2% na quebra de *mAP*. Assim verifica-se que a presença de *data augmentation* influencia o sucesso do algoritmo, porque no patamar dos 90%, 2.2% de diferença é um valor alto. Verifica-se ainda a ocorrência de *overfitting* na fase de validação. Com base nestes dados, confirma-se o impacto real do *data augmentation* no sucesso do treino.

Foi acrescentado o separador de via e foram efetuados 5 treinos para verificar o desempenho do *dataset* depois da sua introdução. Para compactar a informação estão apresentados apenas os resultados dos melhores treinos, 16 e 17.

Tabela 8 Resultados obtidos nos treinos 16 e 17 (com AP obtido para as diferentes classes).

Treino	mAP	FPS	Dataset (Fotos)
16	93.1%	26.5	1231
17	**94.8%**	26.4	1252

Treino	Cone AP	Sign AP	Divider AP
16	84.1%	99.3%	95.8%
17	86.7%	99.3%	98.5%

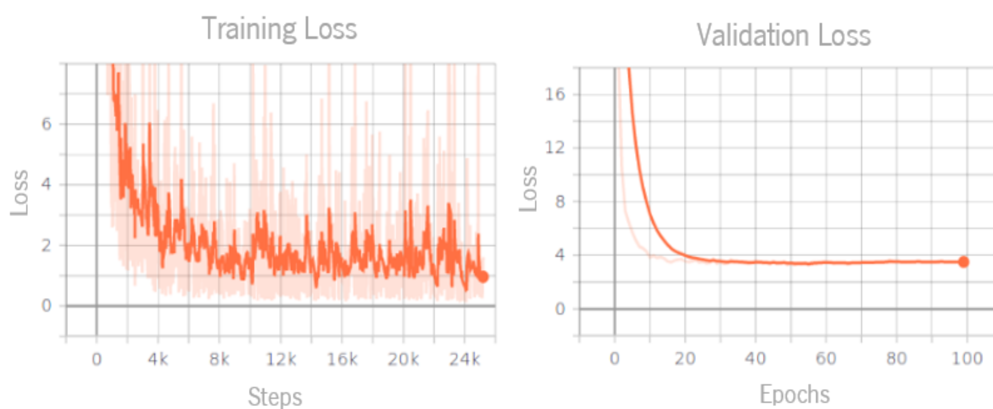


Figura 60 Visualização da loss nas fases de treino e validação do 17º treino.

Estes foram os melhores resultados obtidos para as três classes. Os valores da accuracy são os mais altos de todos os treinos efetuados, o valor de loss é o mais baixo e por fim, não se verifica *overfitting*. Neste momento, o *dataset* foi considerado apto a satisfazer as necessidades do sistema.

Na introdução do separador de via no *dataset* existiu um cuidado redobrado pelo facto de este, numa situação real, não aparecer em ângulos favoráveis ao seu reconhecimento. O ângulo mais favorável ao seu reconhecimento é aquele em que aparece na figura 41. Porém, nas estradas reais o carro encontra o separador de via numa posição diferente, como se pode ver na figura 61.

Figura 61 Exemplo de imagem inserida no dataset com o separador de via na perspetiva mais comum tendo em conta o ângulo de visão normal de um condutor quando dirige um veículo.

Para contornar este problema, as imagens do *dataset* seguiram o ângulo mais parecido com o encontrado em situações reais e não o ângulo considerado ideal. Após esse cuidado, o sistema mostrou-se mais capaz de detetar os separadores de via.

Na figura 62, encontra-se um grupo de imagens do algoritmo em funcionamento. Foi usada a *webcam* do portátil para identificar fotos de cones aleatoriamente posicionados num ambiente real semelhante ao da prova de condução autónoma e sinais de rua encontrados aleatoriamente.

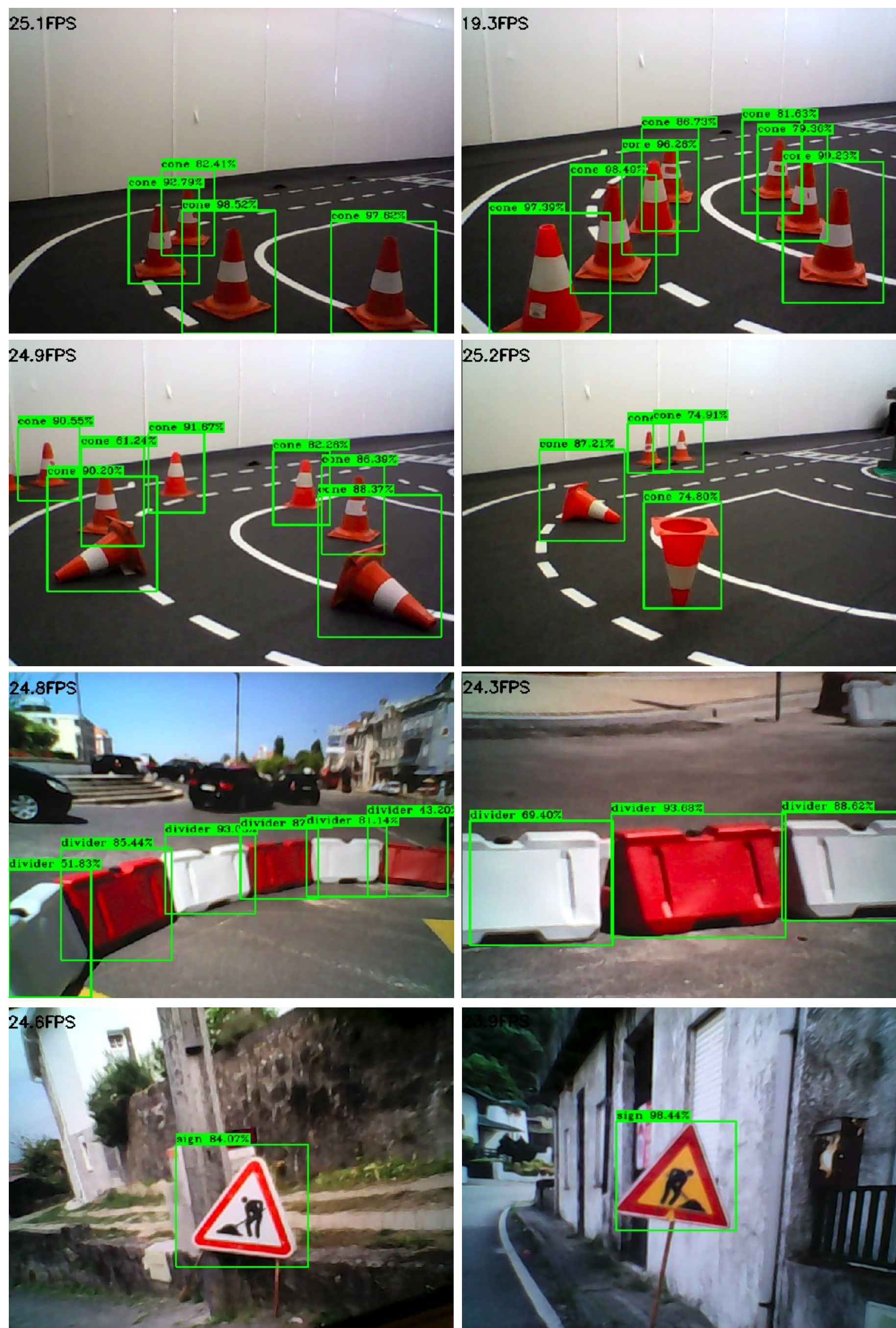


Figura 62 Imagens retiradas de um vídeo de exemplificação do funcionamento do sistema na deteção dos sinais de obras na estrada.

Como se pôde observar, o algoritmo tem um bom desempenho e deteta os sinais sem dificuldade.

Depois de o *dataset* estar concluído restou testar quais os valores dos hiperparâmetros que fazem o sistema ter o melhor desempenho.

A maneira ideal para se encontrar a melhor combinação de valores dos hiperparâmetros seria o teste de todas as combinações possíveis desses valores, o que seria inconcebível para este projeto. Assim, foram escolhidos para teste, os hiperparâmetros mais suscetíveis a variação e com um impacto menos

previsível no treino. Por exemplo, o impacto do *data augmentation* é previsível e foi testado no treino 12, neste sentido, não é suscetível a variação e fica inalterado.

De seguida apresentam-se os testes realizados aos hiperparâmetros mais suscetíveis a variação e que podem ser alterados em simultâneo para se encontrar um equilíbrio.

- Número de *epochs*

Tabela 9 Teste a diferentes valores de epochs.

Epochs	mAP	Total Loss (Valor final)
25	92.9%	3.43
50	94.3%	3.30
100	**94.8%**	**3.50**
150	93.9%	3.80
200	94.8%	3.73

Analisando a tabela 9, observa-se que o número de *epochs* mais equilibrado é 100 porque apresenta o melhor compromisso entre *mAP* e *loss*. O tamanho do *dataset* pode influenciar a configuração deste hiperparâmetro.

- *Learning Rate*

Tabela 10 Teste a diferentes valores de Learning Rate.

Learning Rate	mAP	Total Loss (Valor final)
0.00001	90.6%	3.94
0.0001	**94.8%**	**3.50**
0.001	94.1%	3.57

Pela análise da tabela 10, percebe-se que 0.0001 é o melhor valor de *Learning Rate* por obter o maior *mAP* e a menor *loss*.

- *Batch Size*

Tabela 11 Teste a diferentes valores de Batch Size.

Batch Size	mAP	Total Loss (Valor final)
2	94.7%	3.43
4	**94.8%**	**3.50**
16	93.4%	3.31
32	92.0%	3.67

Nos testes da tabela 11 verifica-se que o *mAP* é mais alto para o *batch size* de 4. Já a *loss* obtém o valor mais baixo para um *batch size* de 16, contudo, o *mAP* baixa bastante e não é desejável.

- *Leaky ReLU*

Tabela 12 Teste a diferentes valores de Leaky ReLU.

Leaky ReLU	mAP	Total Loss (Valor final)
0.1	94.3%	3.61
0.3	**94.8%**	**3.50**
0.5	94.5%	3.41

Para o *Leaky ReLU* (tabela 12), os valores são próximos mas o 0.3 tem maior *mAP*.

- *Kernel regularizer*

Tabela 13 Teste a diferentes valores de Kernel Regularizer.

Kernel Regularizer	mAP	Total Loss (Valor final)
0.0001	94.9%	3.53
0.001	**94.8%**	**3.50**
0.01	94.0%	3.53

Para o *Kernel Regularizer* (tabela 13) o melhor equilíbrio de *mAP* e *loss* é o valor 0.001.

- *Divisão do dataset*

Tabela 14 Teste a diferentes valores de divisão de dataset.

Divisão do dataset	mAP	Total Loss (Valor final)
70/30	92.5%	3.37
80/20	**94.8%**	**3.50**
90/10	92.4%	4.31

Para a divisão do *dataset* (tabela 14) o *ratio* 80/20 é o mais apropriado devido ao elevado valor de *mAP* e valor intermédio de *loss*. Novamente, é um caso onde se obtém um valor inferior de *loss*, 3.37, mas a diferença de décimas não justifica o decaimento de 2.3% de *mAP*.

Com estes resultados verifica-se que a alteração dos valores dos hiperparâmetros não origina uma diferença decisiva na performance porque os valores divergem num pequeno intervalo. Como os valores nas extremidades desses intervalos não se mostraram melhores que os do centro, não se justificou o teste de outros valores.

Os restantes hiperparâmetros não foram modificados porque a mudança dos mesmos tem um resultado previsível. Por exemplo, ao modificar o *threshold* do *IoU* está a alterar-se o limite para o qual uma *bounding box* é aceitável através da sua interseção com a *ground truth bounding box*. Abaixo desse valor a deteção será pouco rigorosa, mas acima desse valor começa a ser demasiado rigorosa podendo, em casos extremos, resultar na não deteção de um objeto. Estas métricas dependem do objetivo do utilizador em termos de precisão das deteções e qual a margem que este aceita para o seu algoritmo. O projeto que o utilizador está a desenvolver é um fator decisivo na escolha dos valores para os hiperparâmetros.

5.2. Movimentação na via delimitada por sinalização temporária

O treino do *DDPG* é mais complexo que o do *YOLOv3-tiny*. A correta configuração dos hiperparâmetros está dependente de inúmeros fatores presentes no ambiente de simulação tornando o algoritmo mais imprevisível. Um facto que foi observado durante os treinos do algoritmo para este projeto é que se forem executados 10 treinos diferentes com a mesma configuração de hiperparâmetros, irão obter-se 10 resultados diferentes devido ao carácter aleatório do algoritmo. Ainda assim, os hiperparâmetros têm sempre um intervalo de valores recomendados ou normalmente usados por autores que usam o mesmo algoritmo. Os autores do *DDPG* mencionaram os valores que utilizaram para testar o algoritmo no *TORCS* e no *MUJoCo*. Com base neles, os valores estipulados para a implementação da primeira solução são os seguintes:

Number of epochs = 500; *Actor Learning Rate* = 0.001; *Critic Learning Rate* = 0.0001; *OU theta* = 0.15; *OU sigma* = 0.2; *Minibatch size* = 64; *Buffer size* = 10000; *Tau* (utilizado para atualizar as *target networks*) = 0.001; *Gamma*= 0.99

A primeira solução implementada demonstrou-se pouco eficaz em lidar com os diferentes percursos apresentados. A Figura 63 mostra os resultados obtidos para a fase de treino e teste em percursos de linha reta. Embora no gráfico de treino em linha reta se verifique uma pequena aprendizagem, esta não é suficiente para atingir resultados consistentes na fase de teste.

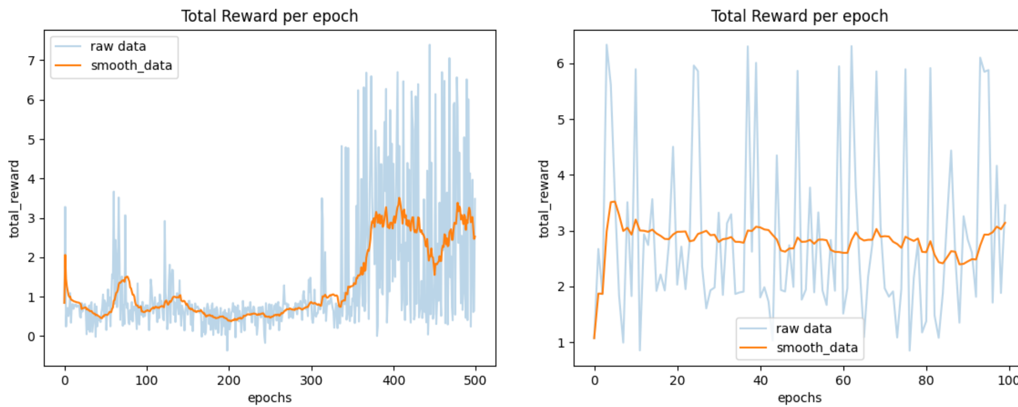


Figura 63 Gráficos correspondentes ao treino e teste com percurso em linha reta, respetivamente, para a primeira solução implementada.

Já nos percursos onde se inserem curvas, a aprendizagem não aconteceu, como se pode ver na figura 64. Um dos motivos deve-se ao facto de, nesta solução, não existir orientação para o ponto de chegada.

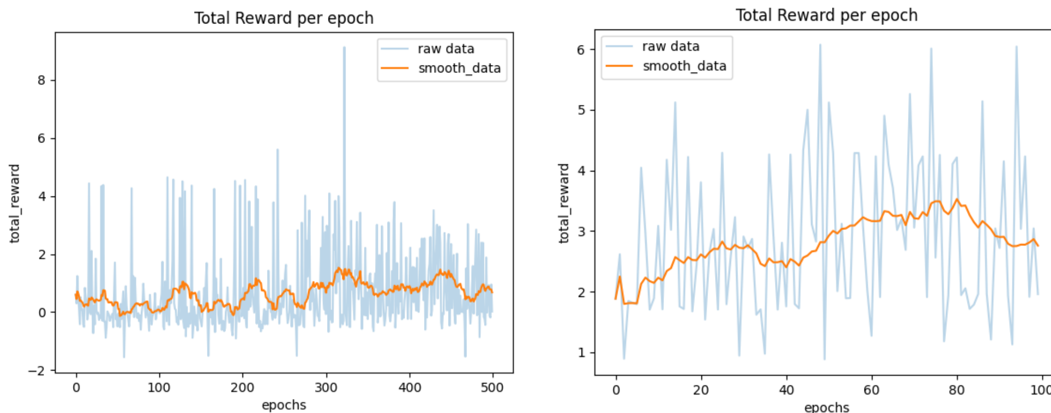


Figura 64 Gráficos correspondentes ao treino e teste com percurso em curva, respetivamente, para a primeira solução implementada.

Após muitas tentativas, a aprendizagem não foi bem-sucedida e passou-se de imediato para a experiência com a segunda solução implementada.

Durante os múltiplos treinos efetuados pelo algoritmo, este verificou-se instável no que diz respeito à consistência de aprendizagem ao longo do treino. Mais especificamente, o algoritmo desaprendia determinados comportamentos que lhe garantiam uma boa *reward* final. Este fenómeno não se mostrou

esporádico, mas sim muito frequente no decorrer dos demais treinos e mostrou-se penalizador para a aprendizagem, uma vez que os *weights* finais eram uma junção de bons e maus comportamentos. Por este motivo foram guardados *checkpoints* dos *weights* aquando dos seus bons comportamentos. Desta forma, a partir do episodio cinquenta, foi verificado se nos últimos cinquenta episódios a média da *reward* total seria maior do que a dos cinquenta equivalentes aos *weights* guardados anteriormente. Este processo foi iterativo até ao fim do treino e permitiu guardar os *weights* que correspondem ao melhor comportamento possível num determinado intervalo de tempo de um treino. Estes *weights* foram inseridos nos treinos seguintes de modo que o algoritmo obtivesse logo de início um bom ponto de partida. O processo mostrou-se imprescindível para acelerar a aprendizagem do *DDPG* e descartar o mais rápido possível os comportamentos que não fazem parte da solução do problema.

Outro fator tido em consideração foi o ruído. Como foi dito anteriormente, o ruído diminui gradualmente ao longo do treino e é estipulado um valor inicial e um valor final. Estes valores não foram fixos e estavam dependentes do objetivo e do comportamento do veículo. Quando o veículo fez um bom treino, não foi aplicado ruído elevado no treino seguinte porque iria mudar drasticamente o comportamento do veículo. Em vez disso, aplicou-se ruído com menor amplitude para otimizar a aprendizagem que adquiriu.

Os hiperparâmetros utilizados são semelhantes aos da primeira solução com a exceção do número de *epochs* que foi reduzido para 100 porque se verificou que o algoritmo ia desaprendendo com um elevado número de *epochs*.

Como efetuado para a solução anterior, o algoritmo foi primeiramente treinado e testado num percurso reto. Após uma boa aprendizagem iniciou-se o treino e teste no percurso da curva e, devido ao sucesso da segunda solução, introduziu-se um percurso em contracurva. Na figura 65 estão apresentados os três percursos construídos lado a lado.

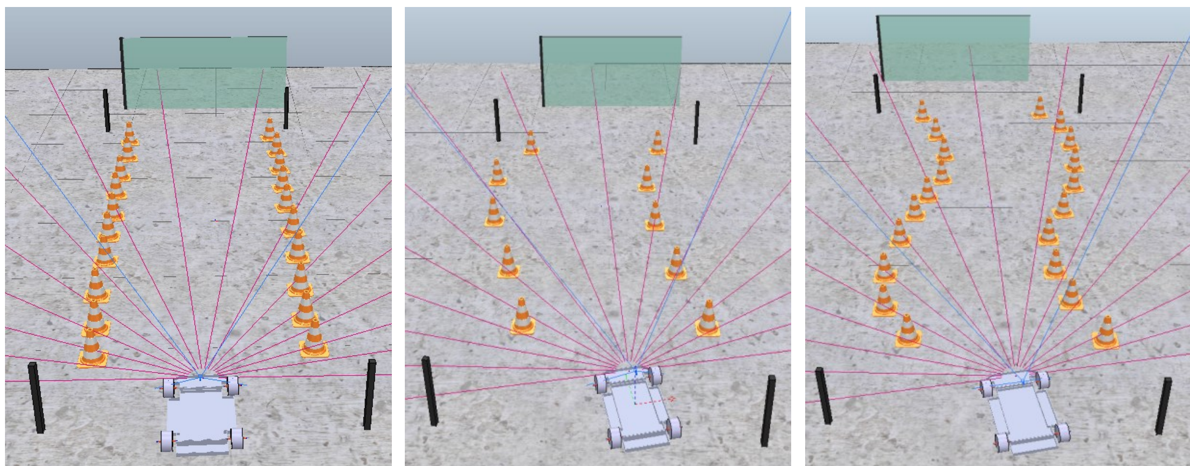


Figura 65 Capturas de ecrã do dos três percursos que o veículo percorreu: Reta, Curva, Contracurva.

Foi registada a evolução da *reward* cumulativa obtida em cada episódio para as três situações diferentes, tanto na fase de treino como de teste. Na figura 66 estão os resultados obtidos no percurso em linha reta.

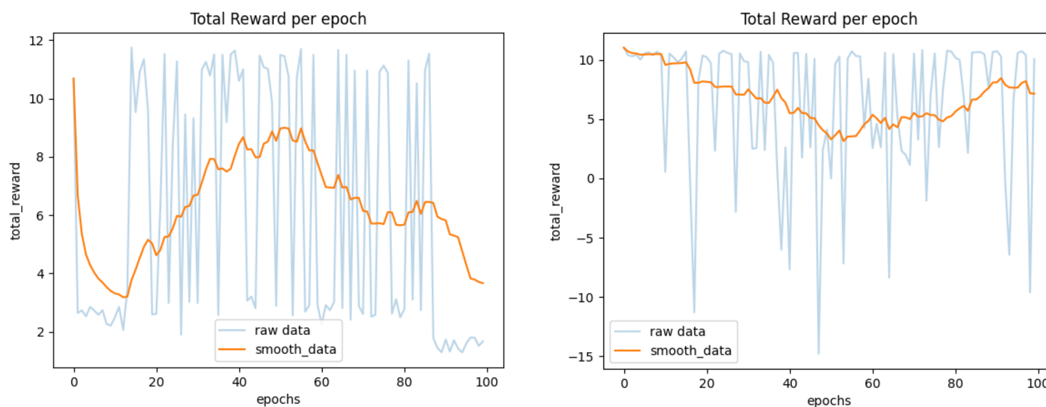


Figura 66 Gráficos alusivos ao treino e teste do sistema, respetivamente, na situação onde o percurso é uma reta.

Na fase de treino verifica-se um intervalo de aprendizagem positiva do episodio 15 ao episodio 55. Este é o intervalo de melhor aprendizagem e os *weights* correspondentes são automaticamente guardados. A aprendizagem foi submetida a uma fase de teste e, analisando o segundo gráfico, observa-se que o algoritmo começa com um comportamento exemplar e perde um pouco o ritmo entre, aproximadamente, o episodio 20 e 55. A partir daí volta a recuperar e pode concluir-se que, tirando essas perdas, tem um melhor resultado que a primeira solução. Na figura 67 estão os resultados obtidos no percurso em curva.

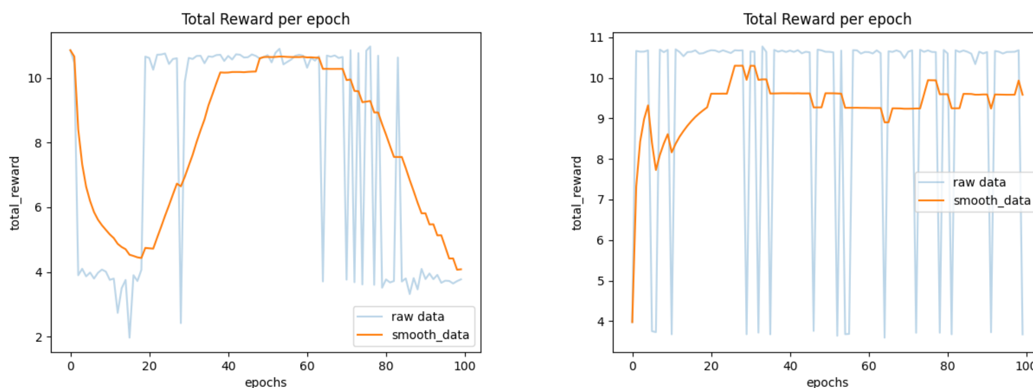


Figura 67 Gráficos alusivos ao treino e teste do sistema, respetivamente, na situação onde o percurso é uma curva.

Pela análise do gráfico do treino, verifica-se um intervalo entre os episódios 20 e 70 onde o algoritmo teve um comportamento quase ótimo. Isso resultou em fases de teste mais fidedignas, como se pode verificar no gráfico alusivo ao teste que conta com uma evolução bastante positiva. De notar que, em ambos os gráficos, o valor acima de 9 de *reward* total corresponde à chegada do veículo com sucesso ao ponto de chegada sem comportamentos impróprios. Um valor acima de 5 indica que o veículo chegou com sucesso ao final, mas não teve um comportamento tão otimizado (um comportamento otimizado

significa ir de forma ininterrupta até ao ponto de chegada, sem paragens ou movimentos desnecessários). No caso de executar comportamentos impróprios e chegar na mesma à linha final, a *reward* será muito baixa e não significa que o veículo não chegou à linha final, mas o seu comportamento não cumpre movimentos de segurança para o transporte de passageiros. Na figura 67 estão os resultados obtidos no percurso em contracurva.

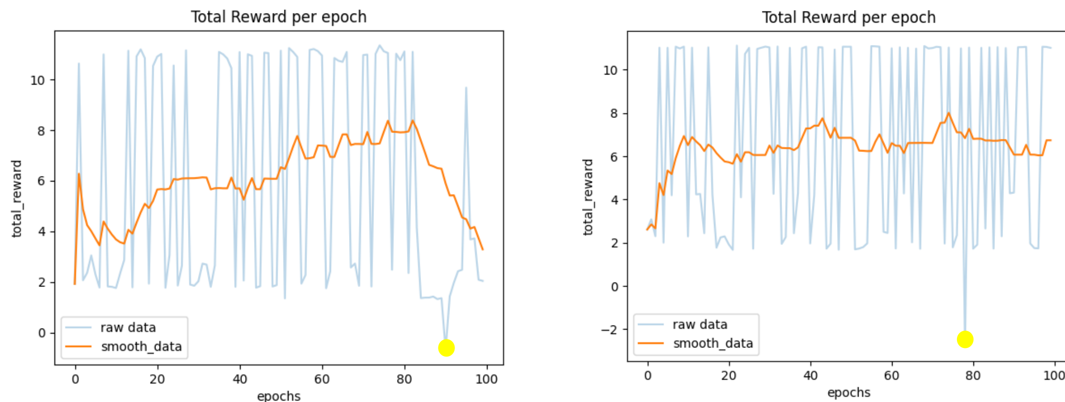


Figura 68 Gráficos alusivos ao treino e teste do sistema, respetivamente, na situação onde o percurso é uma contracurva.

No treino do percurso em contracurva verifica-se uma evolução positiva de aprendizagem. Como no episodio 80 ocorre o pico do valor da média da *reward* cumulativa, o último *checkpoint* foi guardado nesse ponto. A fase de teste com um valor médio acima de 5 indica que o veículo chegou um número elevado de vezes ao ponto de chegada com sucesso e segurança.

É possível ainda observar um pico negativo (a amarelo) nos gráficos do percurso em contracurva. Este pico surge da penalização dada pela variável de contagem de *steps*, aquando do aumento da demora por parte do veículo para terminar o percurso. Esta demora deve-se na generalidade a comportamentos menos úteis, como por exemplo andar a uma velocidade quase nula. É um fenómeno que não deve ser ignorado pois só assim o sistema aprenderá que não pode ter esse comportamento.

Estes gráficos foram o resultado de uma sequência extensiva de treinos e testes. Neste sentido, foram efetuadas centenas de tentativas e uma gestão cautelosa dos *weights* para se produzirem resultados consistentes. Como foi possível observar, existe um intervalo em cada gráfico que regista uma média de *reward* cumulativa obtido a evoluir positivamente, que corresponde à parte do treino em que o sistema está a tirar partido da aprendizagem e a convergir para o objetivo. O reaproveitamento de *weights* por *checkpoints* foi fundamental para a aceleração dos treinos do sistema que, de outro modo, não convergiria para o objetivo tão rápido.

Para concluir, na figura 69 encontram-se sucessivas imagens retiradas de um vídeo que mostra o veículo 100% autónomo a fazer o percurso do início ao fim, utilizando o sistema desenvolvido neste projeto, sem qualquer intervenção humana e sem ruído introduzido nas ações.

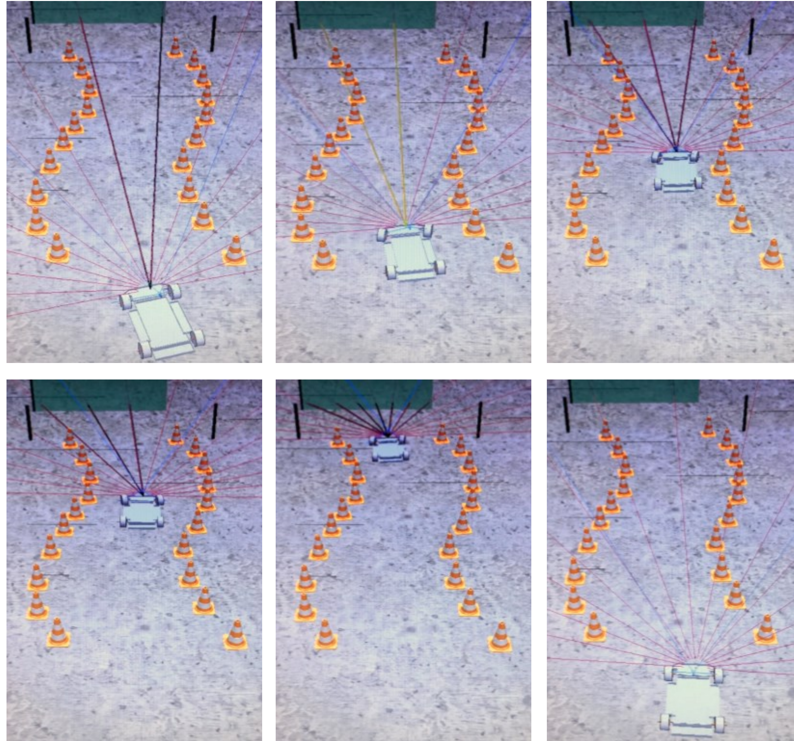


Figura 69 Demonstração do percurso concluído pelo veículo 100% autónomo utilizando o sistema desenvolvido neste projeto.

Esta sequência de imagens demonstra um episódio concluído com sucesso. Pode-se observar que o veículo não tem um comportamento orientado pelo centro da via. Esta característica é esperada visto que a segunda solução implementada para satisfazer os percursos com curvas não tem nenhuma referência de onde é o centro da via, ao contrário da primeira solução.

Foi possível demonstrar que, embora o veículo chegue na maior parte das vezes ao final com sucesso, existe uma minoria de episódios onde tem comportamentos menos desejáveis o que põe em causa a segurança do mesmo na perspetiva de o transportar para o mundo real.

6. CONCLUSÕES

Este projeto demonstrou uma prova de conceito de como a inteligência artificial pode ser utilizada em aplicações reais que dizem respeito a situações complexas ao nível da condução autónoma, como obras na estrada. Como o sistema baseou-se em satisfazer dois objetivos de natureza diferente, foram necessários dois algoritmos de natureza diferente.

O *YOLOv3-tiny* foi utilizado para cumprir o primeiro objetivo que foi a deteção de sinalização temporária de obras na estrada e obteve um *mAP* superior a 90% com um tempo de resposta mínimo. Foi aplicado e testado em ambiente real e ambiente de simulação, onde se registaram todos os resultados obtidos pelo mesmo e provou-se a sua eficiência. Sendo considerada uma versão compacta do *YOLOv3*, mostrou-se menos preciso mas bastante mais rápido, o que o torna imprescindível para a condução autónoma uma vez que o tempo de resposta é um elemento fundamental para garantir a segurança durante a condução na via pública.

O *DDPG* foi utilizado para cumprir o segundo objetivo que envolveu a movimentação do veículo nas situações de obras na estrada e fez uso do algoritmo do primeiro objetivo para se orientar no espaço. Uma das suas principais vantagens é a *exploration* do ambiente ser independente da *exploitation*, com auxílio de um *replay buffer*. Esta característica permitiu, não só mais fluidez e velocidade no treino, como também eliminou eventuais correlações indesejáveis no ambiente em que se situou. Foi treinado e testado em ambiente de simulação e registaram-se os resultados para duas soluções implementadas no processamento de imagem. A primeira solução mostrou-se pouco eficaz para lidar com diversos percursos. A segunda solução mostrou-se promissora em todos os percursos. Assim, o *DDPG* mostrou-se bastante qualificado quando lida com ambientes de natureza complexa e contínua, uma vez que atingiu o objetivo com sucesso em cerca de 73% das vezes para o percurso em reta, 83% para o percurso em curva e 55% para o percurso em contracurva.

Neste momento, não é recomendado a utilização do sistema no mundo real dado que existe uma margem de erro que poderia pôr em causa a segurança dos passageiros do veículo ou dos veículos que o rodeiam.

A utilização de dois algoritmos de inteligência artificial de natureza distinta, para a construção de um sistema como este, permitiu dar a conhecer o poder destes algoritmos a interagir em conjunto e os resultados extraordinários que são capazes de atingir. Desta forma, impulsiona a conceção e o design de novos sistemas que também utilizem vários algoritmos de inteligência artificial em conjunto, usando bibliotecas como o *ROS* para garantir a comunicação de todas as entidades.

Para trabalho futuro está prevista a integração da orientação do centro da via na segunda solução para ter um comportamento mais suave. Assim pode satisfazer-se os vários tipos de percursos e estar minimamente habilitado para uma condução mais preventiva. Se o resultado for sólido, passa a ser possível a hipótese de transportar o sistema para o mundo real onde ocorre a otimização do sistema para aplicações reais.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] N. J. Nilsson, *The quest for artificial intelligence: A history of ideas and achievements*, Cambridge University Press, 2009.

[2] S. Das, A. Dey, A. Pal, and N. Roy, "Applications of Artificial Intelligence in Machine Learning: Review and Prospect," *Int. J. Comput. Appl.*, vol. 115, no. 9, pp. 31–41, 2015, doi: 10.5120/20182-2402.

[3] T. Ribeiro, "Deep Reinforcement Learning for Robot Navigation Systems," Dept. Elect. Eng., University of Minho, Guimarães, 2019.

[4] A. Wilson (2019), *A Brief Introduction to Supervised Learning* [Online]. Available: https://towardsdatascience.com/a-brief-introduction-to-supervised-learning-54a3e3932590 (accessed Nov. 22, 2020).

[5] F. Chollet, *Deep Learning with Phyton*, MANNING, 2018.

[6] J. Delua (2021), *Supervised Learning vs. Unsupervised Learning: What's the Difference?* [Online]. Available: https://www.ibm.com/cloud/blog/supervised-vs-unsupervised-learning (accessed Apr. 1, 2021).

[7] L. Wang, "Discovering phase transitions with unsupervised learning," *Phys. Rev. B*, vol. 94, no. 19, pp. 2–6, 2016, doi: 10.1103/PhysRevB.94.195105.

[8] S. Sutton, R. and G. Barto, A., *Reinforcement Learning: An Introduction*, 2nd ed., The MIT Press, 2014.

[9] J. Oh *et al.*, "Discovering Reinforcement Learning Algorithms," *34th Conference on Neural Information Processing Systems*, 2021

[10] K. Mülling, J. Kober, O. Kroemer, and J. Peters, "Learning to select and generalize striking movements in robot table tennis," *Int. J. Rob. Res.*, vol. 32, no. 3, pp. 263–279, 2013, doi: 10.1177/0278364912472380.

[11] J. M. D. Pessoa, "Análise Funcional Comparativa de Algoritmos de Aprendizagem por Reforço," Dept. Elect. Eng., Instituto Superior De Engenharia De Lisboa, 2011.

[12] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *Int. J. Rob. Res.*, vol. 32, no. 11, pp. 1238–1274, 2013, doi: 10.1177/0278364913495721.

[13] I. Cloud Education (2020), *AI vs. Machine Learning vs. Deep Learning vs. Neural Networks: What's the Difference?* [Online]. Available: https://www.ibm.com/cloud/blog/ai-vs-machine-learning-vs-deep-learning-vs-neural-networks. [Accessed: 29- Apr- 2021].

[14] A. Zayegh and N. Al Bassam, "Neural Network Principles and Applications," *Digital Systems*, 2018, doi: 10.5772/intechopen.80416.

[15] G. L. Shaw, "Donald Hebb: The Organization of Behavior," *Brain Theory*, 1986, pp. 231–233, doi: 10.1007/978-3-642-70911-1_15.

[16] T. Pinto, "Object detection with artificial vision and neural networks for service robots,"Dept. Elect. Eng., University of Minho, Guimarães, 2018.

[17] Y. Chen, H. Chang, J. Meng, and D. Zhang, "Ensemble Neural Networks (ENN): A gradient-free stochastic method," *Neural Networks*, vol. 110, pp. 170–185, Feb. 2019, doi: 10.1016/j.neunet.2018.11.009.

[18] M. M. Mijwel, A. Esen and A. Shamil, ''Overview of Neural Networks'', 2019.

[19] R. Yamashita, M. Nishio, R. K. G. Do, and K. Togashi, "Convolutional neural networks: an overview and application in radiology," *Insights Imaging*, vol. 9, no. 4, pp. 611–629, 2018, doi: 10.1007/s13244-018-0639-9.

[20] A. Mathew, P. Amudha, and S. Sivakumari, "Deep learning techniques: an overview," *Adv. Intell. Syst. Comput.*, vol. 1141, no. August 2020, pp. 599–608, 2021, doi: 10.1007/978-981-15-

3383-9_54.

[21] S. Sakib, Ahmed, A. Jawad, J. Kabir, and H. Ahmed, "An Overview of Convolutional Neural Network: Its Architecture and Applications," *ResearchGate*, no. November, 2018, doi: 10.20944/preprints201811.0546.v1.

[22] C. Thomas (2019), *An introduction to Convolutional Neural Networks* [Online]. Available: https://towardsdatascience.com/an-introduction-to-convolutional-neural-networks-eb0b60b58fd7 (accessed Nov. 27, 2020).

[23] S. H. S. Basha, S. R. Dubey, V. Pulabaigari, and S. Mukherjee, "Impact of fully connected layers on performance of convolutional neural networks for image classification," *Neurocomputing*, vol. 378, pp. 112–119, 2020, doi: 10.1016/j.neucom.2019.10.008.

[24] A. Karpathy (2014), *CS231n Convolutional Neural Networks for Visual Recognition* [Online]. Available:https://web.stanford.edu/class/cs379c/archive/2018/class_messages_listing/content/Artificial_Neural_Network_Technology_Tutorials/KarparthyCONVOLUTIONAL-NEURAL-NETWORKS-16.pdf

[25] Tensorflow, *TensorFlow* [Online]. Available: https://www.tensorflow.org/. [Accessed: 30-Nov-2020].

[26] D. Smilkov *et al.*, "Tensorflow.JS: Machine learning for the web and beyond," in *SysML Conference*, Palo Alto, CA, USA, 2019.

[27] J. Clark (2015), *Google Turning Its Lucrative Web Search Over to AI Machines* [Online]. Available: https://www.bloomberg.com/news/articles/2015-10-26/google-turning-its-lucrative-web-search-over-to-ai-machines (accessed Nov. 30, 2020).

[28] G. Corrado (2015), *Computer, respond to this email* [online]. Available: http://googleresearch.blogspot.de/2015/11/computer-respond-to-this-email.html (accessed Dec. 01, 2020).

[29] O. Good (2015), *How google translate squeezes deep learning onto a phone* [Online]. Available: https://ai.googleblog.com/2015/07/how-google-translate-squeezes-deep.html.

[30] P. Goldsborough, "A Tour of TensorFlow" *arXiv:1610.01178*, 2016

[31] S. Mahadevan and J. Connell, "Automatic programming of behavior-based robots using reinforcement learning," *Artif. Intell.*, vol. 55, no. 2–3, pp. 311–365, Jun. 1992, doi: 10.1016/0004-3702(92)90058-6.

[32] J. A. Bagnell and J. G. Schneider, "Autonomous Helicopter Control Using Reinforcement Learning Policy Search Methods." in *Robotics and Automation*, 2001, doi: 10.1109/ROBOT.2001.932842.

[33] MinhoTeam, *Laboratory of Automation and Robotics* [Online]. Available: http://lar.dei.uminho.pt/index.php (accessed Dec. 03, 2020).

[34] Y. Amit and P. Felzenszwalb, "Object Detection," *Comput. Vis.*, pp. 537–542, 2014, doi: 10.1007/978-0-387-31439-6_660.

[35] R. Verschae and J. Ruiz-del-Solar, "Object detection: Current and future directions," *Front. Robot. AI*, vol. 2, no. NOV, 2015, doi: 10.3389/frobt.2015.00029.

[36] Z. Zhao, P. Zheng, S. Xu and X. Wu, "Object Detection with Deep Learning : A Review," *IEEE Trans. Neural Netw. Learn. Syst.*, pp. 1–21, 2019.

[37] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection.", in *CVPR*, 2016.

[38] L. Jiao *et al.*, "A survey of deep learning-based object detection," *IEEE Access*, vol. 7, no. 3, pp. 128837–128868, 2019, doi: 10.1109/ACCESS.2019.2939201.

[39] G. Chandan, A. Jain, H. Jain, and Mohana, "Real Time Object Detection and Tracking Using Deep Learning and OpenCV," *Proc. Int. Conf. Inven. Res. Comput. Appl. ICIRCA 2018*, no. July 2018, pp. 1305–1308, 2018, doi: 10.1109/ICIRCA.2018.8597266.

[40] W. Liu *et al.*, "SSD: Single shot multibox detector," *LNCS*, vol. 9905, pp. 21–37, 2016, doi: 10.1007/978-3-319-46448-0_2.
[41] J. Redmon and A. Farhadi, "YOLO9000: Better, Faster, Stronger." in *CVPR*, 2017.
[42] J. Redmon and A. Farhadi, "YOLOv3: An Incremental Improvement," *arXiv*, 2018.
[43] J. Redmon and A.Farhadi, *YOLO: Real-Time Object Detection.* [Online]. Available: https://pjreddie.com/darknet/yolo/ (accessed Dec. 07, 2020).
[44] P. Adarsh, P. Rathi, and M. Kumar, "YOLO v3-Tiny: Object Detection and Recognition using one stage improved model," *2020 6th Int. Conf. Adv. Comput. Commun. Syst. ICACCS 2020*, pp. 687–694, 2020, doi: 10.1109/ICACCS48705.2020.9074315.
[45] D. Chun, J. Choi, H. Kim, and H. J. Lee, "A Study for Selecting the Best One-Stage Detector for Autonomous Driving," *34th ITC-CSCC 2019*, pp. 4–6, 2019, doi: 10.1109/ITC-CSCC.2019.8793291.
[46] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and *Accuracy* of Object Detection," *arXiv*, 2020.
[47] Á. Arcos-García, J. A. Álvarez-García, and L. M. Soria-Morillo, "Evaluation of deep neural networks for traffic sign detection systems," *Neurocomputing*, vol. 316, pp. 332–344, 2018, doi: 10.1016/j.neucom.2018.08.009.
[48] E. Yurtsever, J. Lambert, A. Carballo, K. Takeda, and S. Member, "A Survey of Autonomous Driving : Common Practices and Emerging Technologies," *IEEE Access*, vol. 8, 2020.
[49] L. Liu *et al.*, "Computing Systems for Autonomous Driving: State-of-the-Art and Challenges," *IEEE Internet of Things*, 2020.
[50] M. Ružička and P. Mašek, "Real time object tracking based on computer vision," *Mechatronics 2013 Recent Technol. Sci. Adv.*, pp. 591–598, 2014, doi: 10.1007/978-3-319-02294-9-75.
[51] Y. Huang and Y. Chen, "Autonomous Driving with Deep Learning: A Survey of State-of-Art Technologies," in *IEEE 20th International Conference (QRS-C)*, 2020, doi:10.1109/QRS-C51114.2020.00045.
[52] S. Kaplan Berkaya, H. Gunduz, O. Ozsen, C. Akinlar, and S. Gunal, "On circular traffic sign detection and recognition," *Expert Syst. Appl.*, vol. 48, pp. 67–75, 2016, doi: 10.1016/j.eswa.2015.11.018.
[53] K. Lim, Y. Hong, Y. Choi, and H. Byun, "Real-time traffic sign recognition based on a general purpose GPU and deep-learning," *PLoS One*, vol. 12, no. 3, pp. 1–22, 2017, doi: 10.1371/journal.pone.0173317.
[54] N. A. Rosenberg, "K . O . L . T .: Known Object Localization and Mapping," *Worcester Polytechinc Institute*, 2019.
[55] Á. Arcos-García, J. A. Álvarez-García, and L. M. Soria-Morillo, "Deep neural network for traffic sign recognition systems: An analysis of spatial transformers and stochastic optimisation methods," *Neural Networks*, vol. 99, pp. 158–165, 2018, doi: 10.1016/j.neunet.2018.01.005.
[56] A. Shustanov and P. Yakimov, "CNN Design for Real-Time Traffic Sign Recognition," *Procedia Eng.*, vol. 201, pp. 718–725, 2017, doi: 10.1016/j.proeng.2017.09.594.
[57] Y. Zhu, C. Zhang, D. Zhou, X. Wang, X. Bai, and W. Liu, "Traffic sign detection and recognition using fully convolutional network guided proposals," *Neurocomputing*, vol. 214, pp. 758–766, 2016, doi: 10.1016/j.neucom.2016.07.009.
[58] G. Konidaris and G. M. Hayes, "An Architecture for Behavior-Based Reinforcement Learning," *Adaptive Behavior*, 2005, doi:10.1177/105971230501300101.
[59] L. Marina and A. Sandu, "Deep Reinforcement Learning for Autonomous Vehicles-State of the Art," *Bull. Transilv. Univ. Brașov*, vol. 10, no. 59, 2017.
[60] A. El Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *S&T Electronic Imaging, Autonomous Vehicles and Machines*, 2017.

[61] B. R. Kiran *et al.*, “Deep Reinforcement Learning for Autonomous Driving : A Survey,” *IEEE*, 2021.
[62] A. Miglani and N. Kumar, “Deep learning models for traffic flow prediction in autonomous vehicles: A review, solutions, and challenges,” *Veh. Commun.*, vol. 20, p. 100184, 2019, doi: 10.1016/j.vehcom.2019.100184.
[63] S. Wang, “Deep Reinforcement Learning for Autonomous Driving,” *arXiv*, 2017.
[64] M. L. Dec, “End-to-End Deep Reinforcement Learning for Lane,” *NIPS*, 2016.
[65] X. Li, X. Xu, and L. Zuo, “Reinforcement Learning Based Overtaking Decision-Maling for Highway Autonomous Driving,” in *ICICIP*, 2015, doi:10.1109/ICICIP.2015.7388193.
[66] X. Pan, Y. You, Z. Wang, and C. Lu, “Virtual to Real Reinforcement Learning for Autonomous Driving,” *BMVC*, 2017.
[67] X. Li, X. Xu, and L. Zuo, “Reinforcement Learning Based Overtaking Decision-Maling for Highway Autonomous Driving,” in *ICICIP*, 2015, doi:10.1109/ICICIP.2015.7388193.
[68] T. Ribeiro, F. Gonçalves, I. Garcia, G. Lopes, and A. F. Ribeiro, “Q-Learning for Autonomous Mobile Robot Obstacle Avoidance,” *19th IEEE Int. Conf. Auton. Robot Syst. Compet.* ICARSC 2019, 2019, doi: 10.1109/ICARSC.2019.8733621.
[69] Y. Li, “Deep Reinforcement Learning,” *arXiv*, 2019.
[70] R. Hamid et al., "Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression," in *CVPR*, 2019.