UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA E CIÊNCIAS MECÂNICAS

Cleverson Maxwell Santos

DEEP REINFORCEMENT LEARNING ALGORITHM FOR BASIC AUTONOMOUS EMERGENCY BRAKING SYSTEM DEVELOPED IN A SIMULATED ENVIRONMENT

Joinville
2023

Cleverson Maxwell Santos

DEEP REINFORCEMENT LEARNING ALGORITHM FOR BASIC AUTONOMOUS EMERGENCY BRAKING SYSTEM DEVELOPED IN A SIMULATED ENVIRONMENT

Master thesis submitted to the Pos-Graduation Program of Engineering and Mechanical Sciences of Federal University of Santa Catarina to obtain the title of Master in Engineering and Mechanical Sciences.

Advisor: Thiago Antonio Fiorentin, Dr. Eng.

Joinville

2023

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Santos, Cleverson Maxwell
   Deep reinforcement learning algorithm for basic autonomous emergency braking system developed in a simulated environment / Cleverson Maxwell Santos ; orientador, Thiago Antonio Fiorentin, coorientador, Harald Göllinger, 2023.
   63 p.

   Dissertação (mestrado) - Universidade Federal de Santa Catarina, Campus Joinville, Programa de Pós-Graduação em Engenharia e Ciências Mecânicas, Joinville, 2023.

   Inclui referências.

   1. Engenharia e Ciências Mecânicas. 2. Autonomous Emergency Braking. 3. Reinforcement Learning. 4. Driving Simulator. I. Fiorentin, Thiago Antonio. II. Göllinger, Harald. III. Universidade Federal de Santa Catarina. Programa de Pós-Graduação em Engenharia e Ciências Mecânicas. IV. Título.

Cleverson Maxwell Santos

DEEP REINFORCEMENT LEARNING ALGORITHM FOR BASIC AUTONOMOUS EMERGENCY BRAKING SYSTEM DEVELOPED IN A SIMULATED ENVIRONMENT

The present Master level work was evaluated and approved by the committee composed by the following members:

Andrea Piga Carboni, Dr. Eng.
Universidade Federal de Santa Catarina

Marcos Alves Rabelo, Dr. Eng.
Universidade Federal de Santa Catarina

Harald Göllinger, Dr. Eng.
Technische Hochschule Ingolstadt

We certified that this is the **original and final version** of the work, which was judged appropriate to obtain the title of master of engineering and mechanical sciences.

ICP-Edu
Documento assinado digitalmente
Wagner Mauricio Pachekoski
Data: 10/04/2023 19:30:41-0300
CPF: ***.494.569-**
Verifique as assinaturas em https://v.ufsc.br

Wagner Maurício Pachekoski, Dr. Eng.
Program Coordinator

ICP-Edu
Documento assinado digitalmente
Thiago Antonio Fiorentin
Data: 10/04/2023 17:47:32-0300
CPF: ***.053.219-**
Verifique as assinaturas em https://v.ufsc.br

Thiago Antonio Fiorentin, Dr. Eng.
Advisor

Joinville, 2023.

I dedicate this work to Jesus,
author and finisher of my faith.

ACKNOWLEDGMENTS

To Professor Dr. Thiago Antonio Fiorentin for the knowledge, guidance and support given during my academic journey at the Federal University of Santa Catarina, since the graduation until the challenge of this master degree.

To Professor Dr. Harald Göllinger for the generous reception in the Mechatronics Laboratory of the Technische Hochschule Ingolstadt, for the valuable conversations during the realization of this work, as well as the lab leader Frau Ebenbeck for the support in German life.

I thank my family and friends who provided unconditional assistance for this work to be accomplished. In particular to Maikol Funk Drechsler and his wife Ana Carolina with the first steps in Ingolstadt as well as the valuable experience as a research assistant in HiL Labor at the CARISSMA Research Center. Another necessary mention needs to be made to colleague Lucas Gomes Camargo with whom I shared part of the stories and challenges of this journey.

All this work was only possible because of the financial support of AWARE and FAPESC, to which I give my final thanks.

RESUMO

Um progresso considerável foi alcançado no desenvolvimento de sistemas avançados de assistência ao motorista (ADAS) nos últimos anos. Esses dispositivos aumentam a segurança do veículo e dos demais usuários do trânsito, oferecendo avisos ao motorista ou mesmo assumindo o controle total do carro, a fim de evitar possíveis acidentes. A frenagem autônoma de emergência (AEB) realiza a frenagem de acordo com as possíveis situações de risco, capturadas do ambiente por sensores. Este trabalho propõe um algoritmo de AEB básico, desenvolvido com técnicas de aprendizado por reforço profundo, onde a máquina aprende qual decisão tomar com base em recompensas ou punições, recebidas após cada ação tomada e seus estados futuros. Neste caso particular, as funções de recompensa foram desenvolvidas com fatores baseados no tempo de colisão e na velocidade desejada, que combinados buscam acumular recompensas, evitando colisões e mantendo a velocidade da estrada. O agente final é uma rede treinada com duas camadas ocultas de 38 nós cada, capazes de agir de maneira semelhante a um controlador AEB. São usadas como entrada, a velocidade atual, a distância dos objetos, e as posições do pedal do acelerador e do freio a fim de calcular as posições ideais para esses pedais e evitar colisões à frente. Para treinar o algoritmo em situações de tráfego e realizar o aprendizado por reforço, algumas abordagens são feitas no simulador de direção IPG CarMaker™. Atualmente, esses métodos simulados são amplamente aplicados no desenvolvimento inicial de dispositivos ADAS antes da migração para ambientes reais de teste. Após treinado, o algoritmo realiza dentro do ambiente simulado o protocolo de testes para AEB da EuroNCAP e seus resultados são comparados com outra solução AEB já validada. Durante os cenários CCRs e CCRm, o código apresentou melhores distâncias relativas finais do que seu concorrente na faixa entre 10 e 50 km/h. Apesar de passar e também não colidir nas velocidades mais altas, o sistema parou completamente em distâncias mais curtas, o que gerou discussões e possíveis caminhos de melhoria para suas próximas versões. Durante o cenário CCRb, entretanto, o sistema provou ser capaz de aplicar desacelerações maiores do que as encontradas no ambiente para garantir que não ocorresse colisão. A função de recompensa demonstrou eficácia uma vez que procuramos o agente e os resultados que ele realizou em cada cenário. Para testar a abordagem em novos cenários, não utilizados durante o treinamento, para avaliar a resposta dinâmica, o sistema teve um desempenho significativamente bom. Em uma análise qualitativa simples, ao não colidir o sistema seria definitivamente aprovado se ele realizasse em testes reais os mesmos resultados obtidos nas simulações. Entretanto, em uma análise quantitativa mais profunda, pode-se ver que ainda há oportunidades para melhorar a função de recompensa e o método de treinamento para velocidades interurbanas.

Palavras-chave: Frenagem autônoma de emergência; Aprendizado por reforço; Simulador de direção.

RESUMO EXPANDIDO

INTRODUÇÃO

Os acidentes de trânsito são uma das principais causas de morte no mundo, com aproximadamente 1,35 milhões de usuários das estradas morrendo a cada ano (World Health Organization, 2019). A segurança rodoviária é medida utilizando o número anual de mortes acima de 100.000 pessoas, no qual o Brasil não tem relatado melhorias significativas desde 2009 (World Health Organization, 2019). Organizações internacionais desenvolveram iniciativas para encorajar as autoridades regionais a tomar medidas para melhorar as tecnologias de segurança veicular, tais como a "Década de Ação para Segurança Viária 2011-2020" (World Health Organization, 2011) e o programa Visão Zero da Comissão Européia (European Commission, 2011). Os fabricantes de automóveis foram encorajados a acelerar a implantação de dispositivos para evitar colisões, os chamados sistemas avançados de assistência ao motorista (ADAS) (Yoffie, 2014). Dentro dessas tecnologias estão os sistemas autônomos de frenagem de emergência (AEB) que são capazes de reduzir a probabilidade de impacto (Spicer, 2018), por meio das Unidades de Controle Eletrônico (ECUs) que assumem o controle total do veículo em situações perigosas (Sini & Violante, 2020). Os recentes avanços na inteligência artificial (IA) permitiram o uso de novas e mais rápidas técnicas para desenvolver software para o gerenciamento de ADAS, como o aprendizado de máquinas (Kim, 2017), que pode treinar modelos para prever os resultados futuros. O aprendizado de reforço é usado para treinar agentes para tomar ações a fim de maximizar recompensas cumulativas (François-Lavet, 2018), e o aprendizado profundo é usado para implementar arquiteturas de redes neurais para executar essa tarefa de aprendizado (The MathWorks Inc., 2018). As redes neurais são combinadas com métodos de aprendizado por reforço para modelar sistemas mais complexos como já aplicado em um sistema de AEB para evitar colisão com pedestres (Li et al., 2020). Simuladores de condução podem ser usados para validar o software, que é a abordagem adotada no presente trabalho para desenvolver um algoritmo para a AEB usando o Aprendizado de Reforço Profundo.

OBJETIVOS

O objetivo central deste trabalho de pesquisa é desenvolver um algoritmo AEB com técnicas de aprendizado por reforço profundo, capaz de agir de forma semelhante a um controlador, a fim de evitar colisões traseiras em veículos à frente do veículo sob teste. Para isso serão necessários os objetivos específicos a seguir: (a) Desenvolver uma rede neural profunda capaz de resolver o problema; (b) Criar e integrar cenários virtuais ao algoritmo dentro de uma plataforma de simulação de condução, a fim de gerar dados para treinamento e posterior validação; (c) Estabelecer uma função de recompensa eficaz para o agente, possibilitando um processo de treinamento eficaz. (d) Testar a abordagem em novos cenários, não utilizados durante o treinamento, para avaliar a resposta dinâmica, e finalmente (e) medir velocidades, acelerações e distâncias, para uma discussão quantitativa sobre o nível de segurança da solução final.

METODOLOGIA

O problema foi modelado seguindo o conceito de aprendizado por reforço entre agente e ambiente, onde o veículo considerado o agente, recebe do ambiente seus estados s_t e as recompensas geradas para o momento atual r_t, e sua tomada de decisão para a próxima ação a_{t+1} é guiada pelas recompensas que terá no estado futuro r_{t+1}. O processo de treinamento por reforço buscará otimizar a política π endereçando a ação mais recompensadora para cada estado encontrado. Dessa forma são estabelecidos os vetores de estado, que inclui a distância relativa entre o veículo sob teste e o veículo à frente, a velocidade relativa entre ambos, a velocidade atual do veículo sob teste e suas posições de pedal do acelerador e pedal de freio. Como vetor de ações, o sistema deve retornar uma variável chamada pedal, que direciona entre -1 e 1 a posição de pedal de freio ou acelerador que deve ser aplicada. Para definir as funções de recompensa, foram criados dois fatores, o fator de tempo ΔT e o fator de velocidade ΔV. O fator de tempo fará uma comparação entre o tempo para colisão baseado a partir do ponto atual e o tempo para parada total. Já o fator de velocidade irá comparar a velocidade do veículo com a velocidade desejada para a via, a fim de manter o veículo em movimento. Baseadas nesses dois fatores, as funções de recompensa são estabelecidas, podendo somar nos melhores casos uma recompensa igual a 1 e nos piores casos uma recompensa igual a zero, ou também chamada punição. Para estabelecer a rede neural profunda foram criados os nós de input, um para cada variável de estado e o nó de output, que nesse caso é apenas a variável pedal. Entre as duas camadas de input e output, foram incluídas outras duas camadas de nós, as quais caracterizam o sistema como rede neural profunda, ambas com 38 nós cada, estabelecidos após testes prévios de otimização de performance. Definidos os vetores, as funções de recompensa e a rede neural, aplica-se então o processo de treinamento da rede. O ciclo de treinamento uniu treinamento offline com treinamento online, onde o sistema recebe inicialmente os dados a partir de um cenário fixo gerado por meio do software IPG CarMaker™, usando lotes pequenos e atualizando os pesos de cada ligação entre nós, o sistema atualiza então a rede para iniciar o treinamento online, onde as ações reportadas pelo algoritmo são enviadas para o simulador em tempo real. Esse processo é gerenciado por um diagrama de blocos via Matlab/Simulink™, onde o algoritmo roda como um bloco de função, recebendo os sinais do simulador e reenviando as ações para ele. A fim de testar a eficiência da rede neural e capacidade de evitar colisões do sistema, optou-se por comparar o mesmo com a solução de AEB já implementada no simulador IPG CarMaker™ e para tal, usou-se o protocolo de testes do European New Car Assessment Programme (EuroNCAP). EuroNCAP também introduziu terminologias importantes, como AEB City e AEB Inter-Urban, que são categorizadas por velocidade, assim como diferentes tipos de colisões, como CCRs, CCRm, e CCRb. No cenário CCRs, o veículo sob teste (VuT) varia sua velocidade em diferentes tentativas, de 10 a 80 km/h enquanto se aproxima do veículo objetivo global (GVT), que está estacionário no meio da estrada. No cenário CCRb, ambos os carros começam à mesma velocidade de 50 km/h, e a distância entre eles é testada a 12 metros e 40 metros, enquanto o freio varia a 2 m/s² ou 6 m/s². No cenário CCRm, o VuT deve se aproximar de um GVT que mantém uma velocidade constante de 20 km/h. Diferentes velocidades iniciais são testadas, variando de 30 a 80 km/h. Estes cenários e manobras fornecem uma avaliação abrangente dos sistemas AEB e sua capacidade de evitar colisões em diferentes situações. Ao longo do capítulo de resultados, serão feitas algumas discussões sobre a resposta dinâmica do VuT durante os cenários de teste, bem como a validação do algoritmo proposto em comparação com a outra solução conhecida, o AEB embutido em CarMarker™.

RESULTADOS

O capítulo de resultados apresenta o comportamento do sistema para os cenários CCRm, CCRb e CCRs em termos de distâncias finais alcançadas e acelerações máximas. Neste último, CCRs, são explorados os dados da resposta dinâmica para o pior caso, quando o VuT se aproxima de GVT com velocidade de 80 km/h. Partindo do cenário CCRm, nota-se que o algoritmo aprendeu como frear o VuT e evitar o choque com o carro à frente, mas as distâncias finais são significativamente diferentes da solução do concorrente. Quando parte de 80 km/h, o VuT pára a 3,33 metros usando o algoritmo proposto, ao mesmo tempo em que manteria cerca de 10 metros de distância usando a solução AEB do CarMaker™. Aqui, as estratégias por trás da distância de segurança são diferentes. O CarMaker™ segue uma distância relativa fixa de qualquer objeto principal, que, neste caso, tinha sido ajustada para 10 metros. Por outro lado, o algoritmo proposto segue apenas o que havia aprendido por suas recompensas e punições. Estas regras permitem ao controlador reduzir a velocidade do carro enquanto se aproxima, respeitando o fator de tempo estabelecido para cálculo das recompensas. A diferença entre as distâncias finais acontece porque a curva de desaceleração aplicada pelos códigos é diferente. Apesar de ter uma distância final relativa pequena, o algoritmo proposto realiza a manobra de frenagem com uma desaceleração menor em velocidades mais altas, como 70 e 80 km/h. Já no cenário CCRb, simulando as manobras inesperadas de frenagem, o algoritmo proposto mostrou distâncias finais mais seguras para todas as situações. Tanto partindo de uma distância de 40 metros quanto de uma distância de 12 metros, o VuT é capaz de frear com desaceleração suficiente para não se aproximar nem colidir com o veículo adiante. Este comportamento demonstra robustez no método de recompensa estabelecido, já que em nenhum momento nos cenários de treinamento o VuT enfrentou algo do gênero. Isto prova que o algoritmo proposto é capaz de aplicar desacelerações maiores do que as encontradas no ambiente, a fim de manter seus compromissos com a busca pela maior recompensa acumulada. Neste caso, deve-se concordar que o peso da não-colisão foi peça chave na tomada de decisão do algoritmo. Durante a execução do cenário CCRs, os algoritmos se comportaram de forma diferente entre altas e baixas velocidades. Ao operar dentro da faixa considerada pela EuroNCAP (2017) como AEB City, de 10 a 50 km/h, os resultados da AEB proposta são muito interessantes, com distâncias finais iguais ou até mais conservadoras do que seu concorrente. Nos casos de velocidade inicial igual a 70 e 80 km/h, há uma grande diferença entre os métodos. Nestes casos sabemos que, embora não haja colisão, as distâncias finais são muito pequenas, o que se assemelha muito ao caso CCRm do início. Apesar da pequena distância, ainda se pode considerar que, mesmo que haja colisões, a velocidade de impacto é significativamente reduzida. Dentro do conceito da AEB, também podemos entender sua função como atenuador de impacto e não apenas como um mecanismo capaz de evitar todas as situações perigosas com zero colisões. No caso mais crítico de CCRs, notou-se que o comportamento dos pedais de acelerador e freio é diferente para cada solução. O sistema proposto não aplica curvas contínuas de pedal de acelerador e freio, mas picos de frenagem e aceleração, potencialmente explicados pelo processo de treinamento, quantidade de dados para o treinamento inicial e pelo método de recompensa. Tais resultados podem ser melhorados pela geração de conjuntos maiores de dados para o treinamento inicial, com mais cenários e uma função de recompensa contínua. Outra influência neste comportamento é a função de recompensa descontínua, que dá valores escalonados e assim, limita a modulação da resposta final durante o tempo. Na análise dinâmica o sistema concorrente começa sua desaceleração tarde, porém mais acentuada, atingindo -3,99 m/s² e isto promove a convergência para a parada mais cedo. Enquanto isso, o sistema proposto inicia sua desaceleração instantaneamente a -1,83 m/s², o que é mais confortável, mas consequentemente leva mais tempo para parar completamente. O pedal do freio, que no caso proposto, embora oscilando inicialmente, entra em regime contínuo inferior, em torno de 25%. Após uma longa e mais suave manobra de frenagem, o algoritmo proposto pára completamente a 0,86 m antes do carro da frente. O competidor aplica uma desaceleração mais severa e converge antes para a distância final, que é de 4,98 metros. Mesmo sem colisão, a situação de 0,86 m para distância relativa final não é uma condição totalmente segura e isso é uma motivação sólida para continuar desenvolvendo o algoritmo AEB proposto. Para todos os cenários EuroNCAP, o sistema proposto estaria aprovado, entretanto numa análise quantitativa nota-se a necessidade de melhorias para situações em velocidades acima de 50 km/h. Algumas novas funções de recompensa poderiam ser propostas para estes casos, talvez mais centradas na distância de segurança e não tão preocupadas com o conforto dos ocupantes.

CONCLUSÃO

O trabalho atual propôs um algoritmo AEB básico desenvolvido com técnicas de aprendizado por reforço profundo. A estrutura do sistema tem duas camadas ocultas de 38 nós cada, escolhidas após verificação inicial onde mais nós e camadas não foram fatores significativos para o processo de aprendizagem. O algoritmo é capaz de executar o AEB de forma semelhante a um controlador, a fim de evitar colisões traseiras diretamente à frente de um veículo em teste. Cenários virtuais foram usados para treinar e melhorar a resposta do algoritmo, usando uma plataforma de simulador de direção. Os dados iniciais foram produzidos para fins de treinamento e, posteriormente, novos cenários foram criados para validar o sistema. O ciclo de desenvolvimento e treinamento da rede neural para o aprendizado do reforço foi validado dentro dos três principais protocolos de teste para AEB do EuroNCAP, todos simulados virtualmente e comparados com outra solução já validada. A função de recompensa demonstrou eficácia. Ao testar a abordagem em novos cenários, não utilizados durante o treinamento e avaliar a resposta dinâmica, o sistema teve um desempenho significativamente bom. É importante notar que o sistema proposto foi melhor que seu concorrente em cenários de até 50 km/h, onde alcançou distâncias finais mais longas, desacelerações mais seguras e confortáveis para os ocupantes. A solução final também permitiu medir velocidades, acelerações e distâncias, resultando na discussão quantitativa, que concluiu que o sistema está dentro de um nível de segurança adequado. No caso do teste CCRm, o sistema mostrou uma curva de desaceleração muito melhor do que seu concorrente para todas as velocidades realizadas, porém a distância final foi reduzida significativamente a partir de 50 km/h. Em todo caso, não foram registradas colisões. Durante o teste CCRb, onde a frenagem brusca é realizada a 40 metros e 12 metros, o sistema teve um desempenho melhor que o de seu concorrente, com distâncias finais mais seguras.

Neste caso, o conforto dos ocupantes foi negligenciado, alcançando acelerações de até -6,83 m/s² para evitar colisões. No último cenário de CCRs, o sistema contou com um tempo de aproximação mais longo contra um GVT estacionário à frente. Neste caso, novamente nota-se que há uma fraqueza em manter desacelerações mais baixas em altas velocidades, resultando em distâncias finais mais curtas quando acima de 50 km/h. No pior caso, a partir de 80 km/h, o sistema parou 0,86m atrás do GVT. Em uma análise qualitativa simples, ao não colidir o sistema, o sistema está aprovado. Entretanto, em uma análise quantitativa mais profunda, pode-se ver que ainda há oportunidades para melhorar a função de recompensa e o método de treinamento para velocidades interurbanas. O código em seu estágio atual, já é capaz de receber dados do ambiente e treinar suas redes neurais profundas off-line, a fim de encontrar as melhores ações para as posições de gás e pedal de freio. Potenciais próximos passos seriam: (A) fechar o loop entre hardware e software e realizar algumas aquisições on-line, treinando-o para melhorar a percepção da rede, (B) combiná-lo também com aplicações com simuladores físicos como DYNA4™ ou CARLA™, utilizando softwares integradores como ROS, por exemplo. (C) Implementar a rede final em DYNA4™, ou qualquer outro simulador físico para uso experimental com interação humana antes de qualquer implementação real; e por último (D) executar outras soluções AEB baseadas em controladores, diferentes dos plug-ins dos simuladores, no mesmo ambiente e gerar um benchmark para avaliar mais uma vez o algoritmo proposto.

Palavras-chave: Frenagem autônoma de emergência; Aprendizado por reforço; Simulador de direção.

ABSTRACT

Considerable progress has been reached in the development of advanced driver assistance systems (ADAS) in recent years. These devices enhance the vehicle and traffic participants' safety, while offering warnings to the driver or even taking full control of the car in order to avoid possible crashes. The autonomous emergency braking (AEB) performs braking according to potential risk situations, exposed by data captured from the environment by sensors. This study proposes a basic AEB algorithm, developed with deep reinforcement learning techniques, where the machine learns which decision to make based on rewards or punishments, received after each action taken and its future states. In this particular case, the reward functions were developed with factors based on the collision time and the desired velocity, which combined seek to accumulated rewards by avoiding collisions and maintaining the road velocity. The final agent is a network trained with two hidden layers of 38 nodes each, capable of acting similarly to an AEB controller. The current velocity, the distance to objects, and the positions of the gas and brake pedals are used as input in order to calculate the optimal positions for these pedals and avoid collisions ahead. To train the algorithm in traffic situations and perform the reinforcement learning, some approaches are taken in the IPG CarMaker™ driving simulator. Currently, these simulated methods are extensively applied in the initial development of ADAS devices before moving to real test environments. Once trained, the algorithm performs within the simulated environment the EuroNCAP test protocol for AEB and its results are compared to another AEB solution already validated. During the scenarios CCRs and CCRm, the code presented better final relative distances than its competitor in the range between 10 and 50 km/h. Despite passing and also not crashing at the higher speeds, the system completely stopped at shorter distances which generated discussions and potential improvement paths for its next versions. During the CCRb scenario, however, the system has proven capable of applying decelerations greater than those found in the environment to ensure no collision. The reward function demonstrated effectiveness once we looked for the agent and to the results it performed in each scenario. To test the approach in new scenarios, not used during training, to evaluate the dynamic response, the system performed significantly well. In a simple qualitative analysis, by not crashing the system would definitely be approved if it performed in real tests the same results obtained in the simulations. However, in a deeper quantitative analysis, it can be seen that there are still opportunities to improve the reward function and the training method for interurban velocities.

Key-words: Autonomous Emergency Braking; Reinforcement Learning; Driving Simulator.

LIST OF FIGURES

Figure 1. Scope of technologies to be addressed by this work. 23
Figure 2. Basic FCW system diagram. 26
Figure 3. Basic AEB system diagram. 27
Figure 4. Functional block diagram of an AEB system. 28
Figure 5. Machine learning basic workflow. 29
Figure 6. Machine learning techniques. 30
Figure 7. Agent-environment interaction in an RL process. 33
Figure 8. Average performance of the ϵ-greedy method. 35
Figure 9. Artificial neural node with three inputs, a bias and output. 39
Figure 10. Layered structure of nodes. 40
Figure 11. Neural Network learning workflow. 41
Figure 12. The environment of the RL agent. 45
Figure 13. RL process. 47
Figure 14. Deep Neural Network proposed. 53
Figure 15. Previous simulations to define the number of nodes per hidden layer. 54
Figure 16. The RL training workflow. 55
Figure 17. Animation extracted from simulator during tests. 56
Figure 18. CCRm scenario. 59
Figure 19. CCRs Scenario 59
Figure 20. CCRb Scenario. 60
Figure 21. Final Relative Distance between VuT and the leading car, comparing proposed algorithm to CarMaker™ solution in Scenario CCRm. 62
Figure 22. Maximum deceleration during CCRm scenario. 63
Figure 23. Final Relative Distance between VuT and the leading car, comparing proposed algorithm to CarMaker solution in the CCRb Scenario. 64
Figure 24. Final Relative Distance between VuT and the leading car, comparing proposed algorithm to CarMaker™ solution in the CCRs Scenario. 66
Figure 25. Gas Pedal position - Comparison between CarMaker solution and proposed algorithm. 68
Figure 26. Relative Velocity - Comparison between CarMaker™ solution and proposed algorithm. 69

Figure 27. Brake Pedal position - Comparison between CarMaker solution and proposed algorithm 70
Figure 28. Relative Distance - Comparison between CarMaker™ solution and proposed algorithm. 71
Figure 29. The function block diagram without AEB (A) and with AEB (B). 72

LIST OF TABLES

Table 1. Setup of Algorithm Parameters. 56

Table 2. Training Scenario. 57

Table 3. Testing Scenarios. 60

Table 4. Maximum acceleration during CCRb scenario. 65

LIST OF ABBREVIATIONS

ADAS	Advanced Driver Assistance Systems
AEB	Autonomous Emergency Braking
AEBSS	Autonomous Emergency Braking and Steering Systems
AI	Artificial Intelligence
CCRb	Car-to-Car Rear braking
CCRm	Car-to-Car Rear moving
CCRs	Car-to-Car Rear stationary
DL	Deep Learning
DQN	Deep Q Network
ECU	Electronic Control Units
EuroNCAP	European New Car Assessment Programme
FCW	Forward Collision Warning
GVT	Global Vehicle Target
IRL	Inverse Reinforcement Learning
LIDAR	Light Detecting and Ranging
MDP	Markov Decision Process
ML	Machine Learning
RL	Reinforcement Learning
SAE	Society of Automotive Engineering
SGD	Stochastic Gradient Descent
TTC	Time to Collision
TTS	Time to Stop
VuT	Vehicle under Test

LIST OF SYMBOLS

a	Action	
a_{brake}	Deceleration	
[m/s²]		
b	Bias	
BP	Brake pedal	
$d_{safezone}$	Safezone distance	[m]
d_r	Relative distance to leading vehicle	[m]
e	Error	
FB	Full brake	
GP	Gas pedal	
P	Probability	
$PB1$	Partial brake 1	
$PB2$	Partial brake 2	
Q	State-action value function	
R	Reward function	
r	Reward	
s	State	
t	Timestep	
T	Probability of state transition	
T_{PB1}	Time to partial brake 1	[s]
T_{PB2}	Time to partial brake 2	[s]
T_{FB}	Time to full brake	[s]
V	State value Function	
v_{VuT}	Vehicle under test velocity	[m/s]
v_x	Leading vehicle velocity	[m/s]
v_r	Relative velocity to leading vehicle	
[m/s]		
w	Weight	

x	Node output	
β	Learning rate	
γ	Discount factor	
ϵ	Probability of exploration	
π	Deterministic policy	
φ	Activation function	
τ_{FCW}	Travel time (time of Forward Collision Warning)	[s]
τ_{stop}	Time to stop	[s]
τ_{react}	Human reaction time	[s]

SUMMARY

1. INTRODUCTION **21**

1.1. Objectives 24

1.1.1. Specific objectives **24**

2. LITERATURE REVIEW **25**

2.1. Autonomous Emergency Braking (AEB) 25

2.2. Machine Learning 29

2.3. Reinforcement Learning 31

2.3.1. Elements of Reinforcement Learning **31**

2.3.2. Agent-environment interface **32**

2.3.3. Online versus Offline learning **33**

2.3.4. Exploration and Exploitation strategies **34**

2.3.5. Markov Decision Process **36**

2.3.6. Value Functions **36**

2.3.7. Solution methods for model-free applications **38**

2.3.8. Neural Network **39**

2.4. State of the Art 42

3. METHODOLOGY **45**

3.1. The problem as a RL process 46

3.2. State space and actions 47

3.3. TTC and TTS Calculation 49

3.4. Proposed Reward Function 50

3.5. Deep Neural Network and Offline Learning Process 52

3.6. Proposed scenarios 56

3.7. EuroNCAP test protocol for AEB systems 58

4. RESULTS **61**

4.1. Response to CCRm Scenario: Slower Leading Car. 61

4.2. Response to CCRb Scenario: Sudden Breaking Leading Car 64

4.3. Response to CCRs Scenario: Stopped Car Ahead 66

4.4. Dynamic Response to CCRs Scenario 67

4.5. Final block diagram for the code execution 72

5. CONCLUSION **73**

5.1 Potential Future Development 74

REFERENCES 76

ANNEX 1 - Testing scenarios data 81

ANNEX 2 – Structure of the Matlab code. 84

1. INTRODUCTION

According to the latest global reports, approximately 1.35 million road users die each year, which is almost 3,700 deaths per day. Traffic accidents are also the leading cause of death for people between the ages of 5 and 29. (World Health Organization, 2019).

One important indicator to measure road safety is the annual number of fatalities over 100,000 people. Since 2009, Brazil has not reported significant improvements (World Health Organization, 2019). In 2018, there were 32,655 fatalities and a rate of 15.58 according to its population of 209.6M (Ministério da Saúde, 2020). In the same year, the United States of America registered a rate of 11.17, an outcome 3.1% better than its own result of 2017 (National Health and Transportation Safety Administration, 2020). The United Kingdom and Germany reported 3.1 (UK Department for Transport, 2019) and 4.1 (UN Road Safety Collaboration, 2020), respectively, in the same period.

This indicator absorbs many aspects of the road safety at a given location, such as national transport management, mobility design, quality of fleet, users behavior and post-crash response. However, once administration and legislation actions are taken, further results are reached by vehicle safety technologies improvement. Since last decade, international organizations have required effective actions of national authorities addressed to this theme.

In 2010 the United Nations Road Safety Collaboration developed a Global Plan entitled "Decade of Action for Road Safety 2011-2020" (World Health Organization, 2011) which some of its pillars was to develop safer vehicles, encouraging universal deployment of crash avoidance technologies with proven effectiveness. The European Commission also launched in 2011 the Vision Zero program with the challenging objective to reduce road fatalities to zero within 2050, by promoting the development of vehicles capable of assuring an increasing level of safety (European Commission, 2011).

Some approaches such as The Bloomberg Initiative for Global Road Safety (BIGRS) 2015-2019 made efforts to reduce fatalities and injuries from road traffic crashes in low-and middle-income countries by implementing proven road safety interventions at city level (Bloomberg Group, 2020) .

Although bold, such initiatives have boosted the scientific community in the development of new technologies for vehicle safety. In the same vein, car manufacturers were encouraged to accelerate their vehicle automation by the deployment of such devices.

The mainstream of methods for collision avoidance is to apply and combine advanced driver assistance systems (ADAS). Primarily, these systems can use environmental sensors to detect an imminent crash and then to inform the driver by sound or light alerts (Yoffie, 2014), as forward collision warning (FCW) (Honda Worldwide, 2003) and pedestrian protection systems (PPS) (Kovaceva et al, 2020).

Devices such as the autonomous emergency braking system (AEB) are one step ahead, with the probability of impact being significantly reduced (Spicer, 2018). These systems are placed in the second level of vehicles automation accordingly to the Society of Automotive Engineers (SAE International, 2020), and uses not only sensors and warning methods, but also mechatronic actuators operated by Electronic Control Units (ECUs), capable of running large software packages, making decisions and taking full control of the vehicle's longitudinal acceleration in dangerous situations (Sini & Violante, 2020).

Recent advances in fields of artificial intelligence (AI) enabled the use of new and faster techniques to develop software for ADAS management. Machine Learning (ML) is a tool of AI that figures out some desired “model” out of “data” (Kim, 2017), in this case, from data-sets of previous recorded situations on the road.

Machine learning originally had two types of techniques: supervised learning, which trains a model on known input and output data so that it can predict future outputs, and unsupervised learning, which finds hidden patterns or intrinsic structures in input data (The MathWorks Inc., 2016). As a third and mixed type, Reinforcement learning (RL) is the task of training an agent to take sequences of actions in an environment in order to maximize cumulative rewards, based on predetermined rules (François-Lavet, 2018).

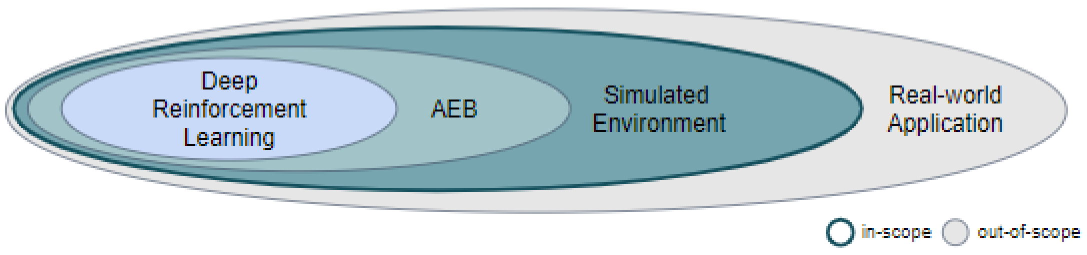
Deep learning (DL) is usually implemented by inserting neural networks architectures to perform the task of learning under some ML technique. The term "deep" refers to the number of layers in the network, and the more layers, the deeper the network. When passing 2 layers, it is no longer a traditional network and this term can be appropriately applied to describe the method, reaching up to hundreds of layers if necessary (The MathWorks Inc., 2018).

Combinations of DL networks into RL methods are usually used to manage larger quantities of data and modeling more complex systems. To increase the speed of training, greater computational processing capacity is also required (The MathWorks Inc., 2018).

Finally, before any real application, the software regardless of the method used to build it, needs to be validated. For this purpose, driving simulators can be satisfactorily applied. Their use in industry has been highly employed in recent years, as they allow safety-critical scenarios to be reproduced under controlled laboratory conditions. (Winner, 2016).

In the context of such technologies mentioned so far, the present work aims to develop an algorithm for AEB using Deep Reinforcement Learning. To be tested and validated, a simulated environment will be used, which avoids the high costs of testing in real conditions. Figure 1 presents an illustrative scheme of the technologies and their approach within this work.

Figure 1. Scope of technologies to be addressed by this work.



Source: Elaborated by the author.

Although already present in the market, the proposed AEB stands out for the novelty of including a solution based on artificial intelligence, which differentiates it from existing systems, as innovative within the ADAS research field.

1.1. Objectives

Develop a basic AEB algorithm with deep RL techniques, able to act similarly to a controller in order to avoid rear-end collisions straight ahead of a vehicle under test.

1.1.1. Specific objectives

A. To develop a deep neural network capable of solving the problem;

B. To create and integrate virtual scenarios to the algorithm within a driving simulator platform, in order to generate data for training and later validation;

C. To establish an effective reward function for the agent, enabling an effective training process.

D. To test the approach in new scenarios, not used during training, to evaluate the dynamic response.

E. To measure velocities, accelerations and distances, for a quantitative discussion about the safety level of the final solution.

2. LITERATURE REVIEW

This chapter presents an overview of the important and necessary topics to be addressed during this work, as the consolidated knowledge about the autonomous emergency braking function, the machine learning structure, reinforcement learning tools and it ends with the state of the art on previous studies that combined similar ideas to the purpose of this work.

The main task to be performed by the final algorithm is an AEB, theme of the first section.

2.1. Autonomous Emergency Braking (AEB)

According to the Society of Automotive Engineering (SAE), autonomous emergency braking is an active safety technology that can avoid or mitigate collisions by integrating software and hardware for automatic braking in hazard situations. It can be classified as an Advanced Driver Assistance System (ADAS), a category that can also include other safety solutions such as active lane keeping, adaptive cruise control, speed limit, lane changing and parking assistance (He and Zhang, 2020).

Honda has been the first manufacturer in the automotive industry to announce the development of this technology. The system was implemented in the 2003 Honda Inspire, still with the commercial name of Collision Mitigation Brake System (CMS). A radar sensor of 100 meters range calculated forward objects, relative distances and velocities before activating the Forward Collision Warning (FCW). Beyond the warning, braking was automatically activated in case of emergency, equalizing the pressure in the circuit if the driver brake pedal position was not enough for a safe stop (Honda Worldwide, 2003).

Since then, the development of the AEB systems has been massively increased and from 2014 onwards, its effectiveness is already part of some costumer safety testing programs as the European New Car Assessment Programme

(EuroNCAP), a catalyst initiative for encouraging significant safety improvements to new car design (EuroNCAP, 2020).

The basis of the system starts from the observation of the driver's behavior in a situation of sudden braking. In a simple approach, the stopping time is counted from the moment when the vehicle under test (VuT) first applies its brakes with a known deceleration, a_{brake}, until the moment when it reaches steady state, as shown in (1).

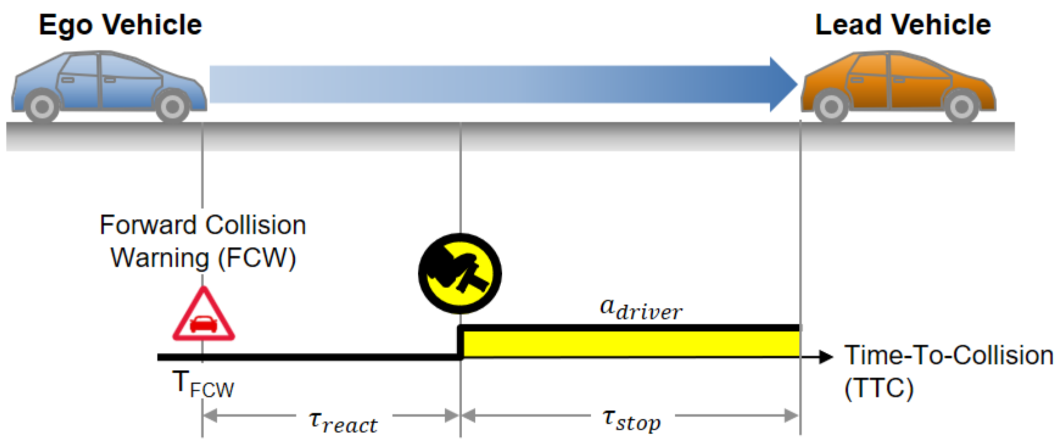
$$\tau_{stop} = v_{VuT}/a_{brake} \qquad (1)$$

As an inherent human factor, the driver has a reaction time to the stimuli and reflexes he receives. This delay time, τ_{react}, must be considered and added to the stopping time if the system is designed to deliver for the driver some assistance, as the FCW system for example. In this case, the total travel time, which can be taken as τ_{FCW}, will be the sum of the car and driver processes, expressed in (2).

$$\tau_{FCW} = \tau_{react} + \tau_{stop} = \tau_{react} + v_{VuT}/a_{brake} \qquad (2)$$

Considering only the FCW system and the human driver intervention, the first concept basic idea of the system is presented in Figure 2. It is considered that only a FCW system is not yet a complete AEB solution, as there is no automatic brake application at this stage.

Figure 2. Basic FCW system diagram.



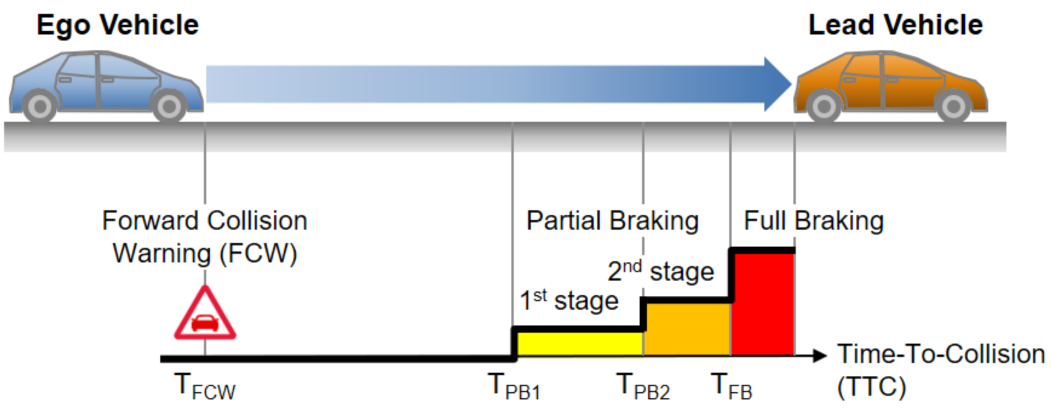
Source: The MathWorks, Inc. (2020).

It is important to explain that Ego Vehicle or Vehicle under Test (VuT) means the same subject and for easy understanding, VuT will be selected as the standard for this work. In Figure 2, another important term to explain is the Time-To-Collision (TTC). Based on current distance and velocities of both cars, TTC will be the necessary time for both cars to collide if they stay in the same path and no intervention happens. Basically, when the TTC becomes less than τ_{FCW}, the alert of an imminent collision must be activated. A detailed explanation of the TTC expression deployed in this work will be presented in the Methodology chapter.

If the driver fails to interfere into the brakes during the expected time after the alerts, perhaps due to distractions, or if the minimum deceleration a_{brake} is not achieved, the system is allowed to take control of the vehicle and perform the necessary brake maneuver. In this way, the system can therefore be called an AEB system, because it is not only a warning emitter, but an autonomous decision maker.

Basic AEB systems typically apply cascaded braking, which consists of the multi-stage partial braking followed by full braking (Hulshof et al.,2013). This type of approach is shown in Figure 3.

Figure 3. Basic AEB system diagram.

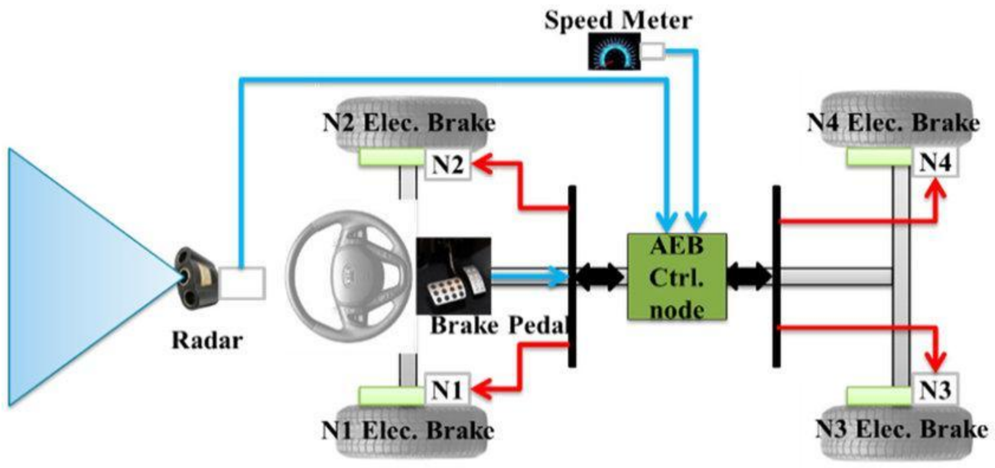


Source: The MathWorks, Inc. (2020).

The Figure 3 shows three potential stages of deceleration, two of them, $PB1$ and $PB2$ on partial braking and FB using the full available braking power provided by the mechanical system. The system intervention in this case must follow the data updated in real time from the sensor, which means that the three stages will not always be necessary. It occurs, for example when the driver manually brakes, the leading vehicle accelerates or leaves the same lane from the VuT. For this reason the system constantly updates the times and makes new decisions (The MathWorks, Inc. 2020).

In real applications, the system can be mounted as the functional diagram in Figure 4. The study by Lu and Chen (2019), followed the standard composition of the solutions currently on the market, where AEB is embedded in a central control unit, also called AEB control node. The data from the front sensor, which in this case was a radar type, the current speed and the position of the brake pedal are defined as inputs for the node (blue lines). The outputs (red lines) are the final brake signals to be applied to each wheel. In this case, the car has been equipped with individual electrical brakes.

Figure 4. Functional block diagram of an AEB system.



Source: Lu & Chen (2019).

The physic assemble may follow different strategies, changing input and output technologies, or even adopting more complex diagrams to ensure the necessary redundancies for the occupants safety. The actual study will be centered in the main algorithm to be executed inside the control node, the ECU responsible for the system. The inputs and outputs will be discussed in the methodology chapter.

The programming structure adopted for the main algorithm of ECU and controllers may follow different strategies. In the field of stability control, Amaral (2018) has implemented machine learning for stability control of an electric sport car, in her case the resulting controller was able to perform torque vectoring, reducing the vehicle sideslip angle. Drechsler (2019), applied an actor-critic reinforcement learning strategy to control the slip ratio of an electric kart in different ground conditions, only with the input of the longitudinal velocity, motor current, vehicle acceleration, gas pedal position and wheel velocity.

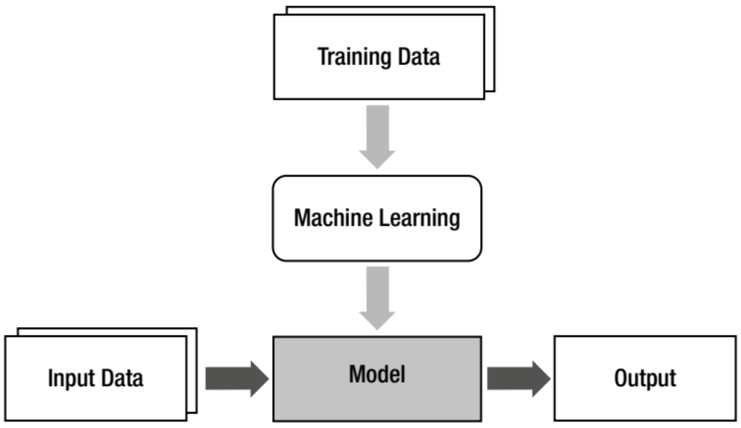
As the cited works have already confirmed the solution capacity of controllers generated from machine learning techniques, this theme will be expanded in the next sections.

2.2. Machine Learning

In the field of Artificial Intelligence (AI), Machine Learning (ML) is a promising tool and it has received emphasis in the last years by its high capability of solving complex systems. Different applications had success, such as applications in autonomous driving, big data management, financial market solutions and medical diagnosis systems (Paluszek & Stephanie, 2017).

For Kim (2017), ML can be defined as a method that extracts some desired model from some data, in this case, from data-sets of previous tried scenarios, called training data. In Figure 5, ML is part of the vertical flow, also called the learning process. Once delivered by the learning process, the model must return outputs based on the input data it will experience in new scenarios.

Figure 5. Machine learning basic workflow.

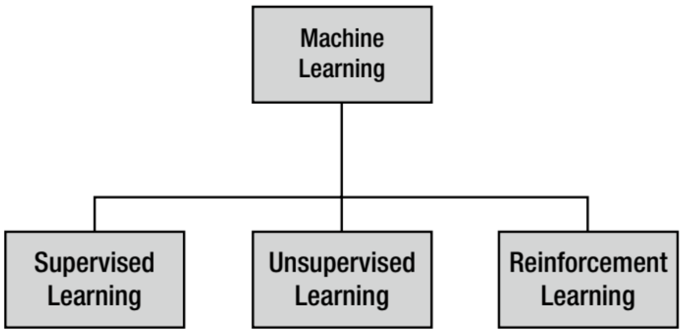


Source: Kim (2017).

Although they originate from similar scenarios, it is not recommended that the training data fully cover all input data to which the model can be submitted in later application A validation set, hidden from the training data, is used for checking the accuracy of the model responses. Through this process, called generalization, the creation of a good model is guaranteed, where its performance is not locked only into the known situations. The more generalized the model, the more prepared it will be for unexpected scenarios (Kim, 2017).

Machine learning initially had two types of techniques: supervised and unsupervised learning. Later, the development of Reinforcement Learning was integrated, achieving the three known machine learning techniques, summarized in Figure 6.

Figure 6. Machine learning techniques.



Source: Kim (2017).

In supervised learning, a set of training data is applied to the system whose answers are already known. The model, therefore, learns from the combinations that are already established. The learning process is successful when the differences between the model outputs and the ideal outputs are minimized (Paluszek & Stephanie, 2017).

Unsupervised learning is a process in which only the inputs are known. Thus, learning happens in the search for recognizing hidden patterns and classifying future responses. Since there are no expected answers, the system can find patterns in data, which previously were not identified (Paluszek & Stephanie, 2017).

Reinforcement learning, according to Kim (2017) is the most used technique. It can be considered a combination of the previous techniques, where inputs can be known but not all outputs can be established. Thus, the model is built by a learning process where the best responses are obtained by mapping states and actions that return a maximum sign of reward. The necessary factors for learning are the training data, some outputs and a rule that guides the rewards or penalties during the process (Sutton & Barto, 2017; François-Lavet, 2018).

The three techniques, despite differences, start from a set of training data. For this reason, it is important to reinforce that the composition of the initial data is fundamental to achieve a successful model, prepared for a substantial range where multiple inputs can be experienced (Kim, 2017).

The main features that are useful for creating an RL method will be presented in the following sections

2.3. Reinforcement Learning

Although there are previous records of the application of this technique, the current concepts of RL, for the most part, were developed by Richard Sutton in the eighties, and have continued to be improved since then, as more problems are solved by it, becoming the main technique of machine learning. (Sutton & Barto, 2017; Sutton, 1999; Wiering & Otterlo, 2012).

The great differential of the technique is that it does not depend on the complete knowledge of the outputs that the model must deliver. But through pre-established feedback, as rewards or punishments, the system learns how to execute the best decisions. This can be very useful to model non-linear systems or implement controllers where previously decisions depended on human intervention (Sutton & Barto, 2017).

The essential features of RL and its elements are listed in the following subsections.

2.3.1. Elements of Reinforcement Learning

The RL approach starts with the definition of two main elements: agent and environment. The agent comprises the unit that makes decisions and executes actions based on the states in which the environment is located. The environment is all the external characteristics of the agent, known or not, which receives the actions and modifies itself, returning to the agent new states (Sutton & Barto, 2017; Nandy & Biswas, 2018).

The system's composition may have sub-elements, listed by Sutton and Barto (2017) as: policy, reward function, value function and environment model.

The policy, henceforth identified by π, is the rule developed by the agent who guides it in generating the set of actions for each set of states received from the environment. There are several methods of generating the policies, being them simple, complex, fixed, stochastic generated or updated as the system is re-trained (Sutton & Barto, 2017; Wiering & Otterlo, 2012).

The reward function is important to define the purpose of the agent within the environment. It is defined during the system's conception phase and remains constant throughout the learning process. It is the mechanism that assigns to each set of states generated by a previous set of actions, a certain feedback, which can be positive (a reward) or negative (a punishment) (Sutton & Barto, 2017).

While the reward function quantifies how good are the actions taken for the current states, the value function cares about the long term. The value function aims to estimate the amount of rewards that the agent can expect starting from the actual state and taken action. The best sequence of actions will be the one that accumulates the highest number of rewards (Sutton & Barto, 2017).

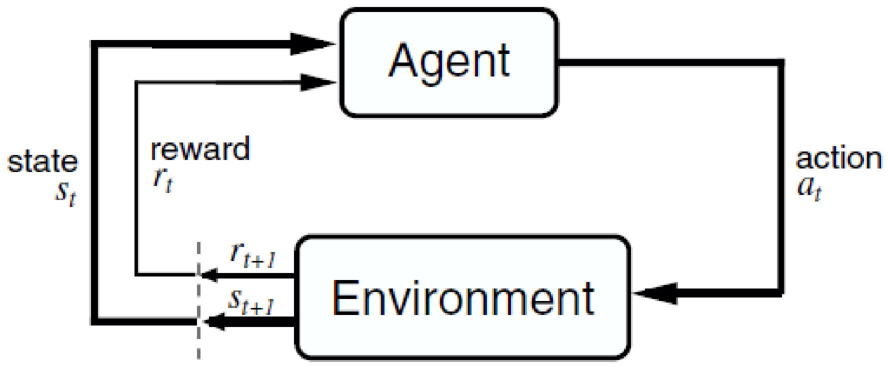
The environment model is basically an attempt to reproduce the behavior of the environment. Modeling the environment can be a good strategy for predicting future states and rewards. This way, better actions can be performed without the costly trial and error process. (Sutton & Barto, 2017).

The expressions set for each RL element introduced above will be presented partially in the next subsection and in the methodology chapter, while these parameters need to be fitted to the study case.

2.3.2. Agent-environment interface

The basic operation of an RL process occurs in the cyclically interaction between its two main elements previously defined as agent and environment, presented in the diagram of Figure 7. The agent's continuous decision-making process creates new actions based on a current state. Such actions, applied to the environment, cause new states and with them comes new respective rewards. A new cycle is started from then on (Sutton & Barto, 2017).

Figure 7. Agent-environment interaction in an RL process.



Source: Sutton and Barto (2017, p. 52).

It is necessary to point out that the two elements interact within a discrete model, where at each step t, the agent receives some representation of the state. From this state, a possible action is chosen, leading to the next state s_{t+1}. The reward r_{t+1} is then calculated based on the state and actions taken. The state and the reward are sent to the agent that chooses the next action. In this way, the process will always try to maximize the reward by means of actions that return the best values in the long term (Sutton & Barto, 2017).

The learning cycle may be implemented by an online mode or an offline one. Both are themes for the next section.

2.3.3. Online versus Offline learning

The RL method can be applied during a problem execution in online or offline mode. This is differentiated by the way the state data is being updated. This data can be originated directly from the execution of the actions in the real environment or from a simulated environment. The real environment approach, called online learning, can present some challenges such as the time consuming and safety in hazard situations when the policies are not trained enough (Wiering & Otterlo, 2012).

Simulated environment, or also called virtual environment, is applied for offline learning and performs the learning using a simulator to generate data, or in some cases, using a dataset previously captured and saved. Offline learning is a safe and fast way to train the policy in the previous stages of development, while permitting the agent to avoid dangerous errors and high costs. The combination of both processes, offline followed by online learning, can be a great solution, fitting the policy for the dataset and then, fine-tuning it in a real task (Wiering & Otterlo, 2012).

The way in which the process advances searching actions to obtain the highest possible reward value can follow two strategies: exploration or exploitation. Both will be discussed in the next item.

2.3.4. Exploration and Exploitation strategies

During the process of choosing the action for each state received, the agent performs a search that returns the best or most recommended action for that situation. The RL, different from other ML methods, guides the policy construction by rewards given to previous states and actions. A trial and error approach can be used as a method to define this policy. In this case, it follows an exploration process (Sutton & Barto, 2017).

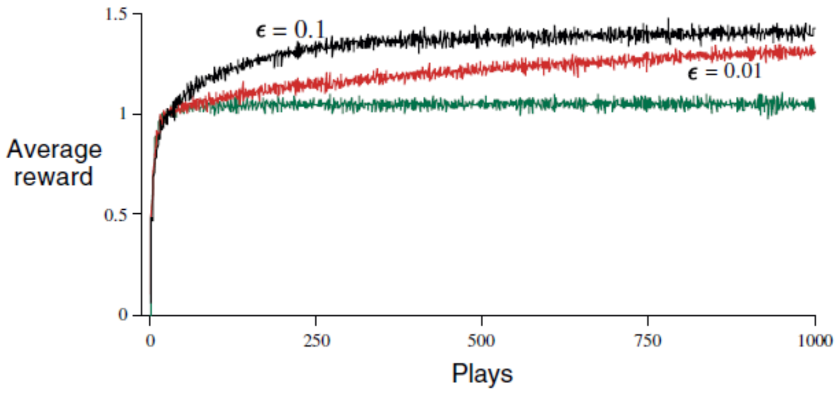
An exploration strategy aims to maximize the cumulative amount of rewards in the long term. It takes into account not only the current action, but also the next potential actions in order to reach, in the end, a sequence of well rewarded choices. It returns the most recommended action, which perhaps generates intermediate rewards at the moment, but leads to better actions and rewards in the next stages. If the method does not look for the future and performs only the most rewarded action for the current state, it is following an exploitation approach (Sutton & Barto, 2017).

Exploiting strategy looks for maximizing the expected reward in the present cycle. In a pure exploitation approach, the agent makes choices without paying attention to the behavior of the system in the next stages, failing to consider better actions. Well-designed learning processes must balance between these two approaches during the policy construction, performing what is called by Sutton and Barto (2017) as an exploration-exploitation problem.

The combination of both strategies can lead to a very complex management between values estimation, uncertainties and remaining number of tries. However, there are alternatives that simplify this impasse. One of them is the ε-greedy method, which exploits most of the time, and based on the small probability ϵ, chooses a random action from a set of possible actions (Sutton & Barto, 2017).

Applications using an ε-greedy methods often permit the convergence of all rewards to the maximum value. It occurs because as the number of plays increases, all possibilities are tried many times. However, the convergence behavior is attached to the defined probability ϵ. A better illustration of this influence is shown in Figure 8.

Figure 8. Average performance of the ϵ-greedy method.



Source: Sutton and Barto (2017, p. 29).

In the Figure 8, three average reward curves are plotted, following probabilities of $\epsilon = 0$, $\epsilon = 0.01$ and $\epsilon = 0.1$. The convergence to the highest value is faster reached by the method that sets $\epsilon = 0.1$. It means that the agent used 10% of its choices to explore among possible actions and 90% of the time it passed exploiting. Even receiving bad rewards at the very beginning, it quickly understood the system and adapted its policy.

An intermediate response is presented by the method of $\epsilon = 0.01$. In this case, it has a slower convergence, and that is expected because now the agent is only 1% of the time performing exploration. If more plays would be performed, probably the average reward would pass the $\epsilon = 0.1$ method results in the long term.

Of course, the worst situation comes from the $\epsilon = 0$ method. This strategy is purely exploitation, which stops the bad rewards generation very fast in the beginning, but the converged value is a lower offset of the total possible reward it could extract from the system. This occurs due to the high possibility of getting the policy stuck in suboptimal performance, no longer exploring better rewards (Sutton & Barto, 2017).

Regardless of the strategy adopted, it is necessary to consider that the problem also adopts a Makrovian approach. The benefits of this property are discussed below.

2.3.5. Markov Decision Process

In order to solve all interaction problems between environment and agent in an efficient way, it is necessary to make predictions about states and expected rewards. Normally, the environment in time $t + 1$ is the outcome not only of an action taken at time t but all past states and actions. If the system is kind of a Markov Decision Process (MDP), the current state and action in time $t + 1$ depend only on the state and action in time t. In other words, it is possible to predict values and make an optimal future decision based only on the knowledge of the current state, which carries all information about the past (Wiering & Otterlo, 2012).

It is possible to describe the concept in terms of a probability, stated in the relation of (3).

$$P\left(s_{t+1}\middle|s_t,\ a_t,\ s_{t+1}\middle|a_{t-1},\ ...,\ s_0, a_0\right) = P(s_{t+1}|s_t,\ a_t) \tag{3}$$

The Equation (3) shows that the probability of an event where a state s_{t+1} is resultant of all previous states and actions, is the same of the one where s_{t+1} results only from the last state and action s_t and a_t. Therefore, if an action is taken in some state, the probability distribution to activate the next state is the same every time the system makes the same decision. Thus, the last state and last action will have all decisive information about the next state's development (Wiering & Otterlo, 2012).

The rewards behavior also follows a function called $R(s_t)$, and the relation of past, actual and future values can be very similar. To model-free applications, normally the actual reward is also dependent on next the resultant state, $R(s_t,\ a_t,\ s_{t+1})$, including a term that works as an evaluation for the transition (Sutton & Barto, 2017).

2.3.6. Value Functions

Value functions exist inside of RL algorithms to quantify how favorable to the agent is the current situation it faces. The concept is based on the potential amount of reward it will receive in the future, starting from condition of interest. Sutton and

Barto (2017) explain the state-value function as how good for the agent is to be at some state s regarding the total reward expected if a policy π is followed. Similarly, state-action-value function expresses the reward influence not only by the state s, but also by the taken action a if the same policy π is respected.

Adding the MDP concept, it is possible to define the state-value V^{π} and the action-state-value function Q^{π} as the expressions of (4) and (5), respectively. Both are defined in terms of what Wiering and Otterlo (2012) call Bellman Equation.

$$V^{\pi}(s) = \sum_{s'} T(s, \pi(s), s')\left(r\left(s, a, s'\right) + \gamma V^{\pi}\left(s'\right)\right) \tag{4}$$

$$Q^{\pi}(s, a) = \sum_{s'} T(s, \pi(s), s')\left(r\left(s, a, s'\right) + \gamma Q^{\pi}\left(s', a'\right)\right) \tag{5}$$

Two components of (4) and (5) are important: the term $T\left(s, \pi(s), s'\right)$ and the discount factor γ. The first one means the probability of transition between the state s to the state s_{t+1} (simplified by s'). The discount factor, a value limited from 0 to 1, quantifies the influence of past rewards on the agent's actual and future decisions. If $\gamma = 0$ older values have no importance and the agent is called myopic, just taking recent rewards into account. As the γ increases, the rewards records become more relevant (Wiering & Otterlo, 2012).

If the policy π is improved, the entire system returns more rewards and the final value is increased. Then, the goal of an RL algorithm stands on the maximization of the value functions by an optimal policy π^{*}. This optimal policy selects the best actions using the state-value function of (4), resulting in the greedy policy (6) (Wiering & Otterlo, 2012).

$$\pi^{*}(s) = \arg\max_{a} \sum_{s'} T(s, \pi(s), s')\left(r\left(s, a, s'\right) + \gamma V^{*}\left(s'\right)\right) \tag{6}$$

The optimal state-action-value function, also called Q-function, can be described by (7), where the best actions delivered by the optimal policy $\pi *$ are used to return the greater possible rewards, resulting in the greatest value (Wiering & Otterlo, 2012).

$$Q^{*}(s,a) = \sum_{s'} T(s,\pi^{*}(s),s')\left(r\left(s,a,s'\right) + \gamma\, Q^{*}\left(s',a'\right)\right) \tag{7}$$

For model-free applications, the Q-functions can assume the probability of transition function equal to 1 and the learning process is significantly simplified (Wiering & Otterlo, 2012). The

2.3.7. Solution methods for model-free applications

Most RL algorithms have a model-free approach, which uses sampling and exploration to obtain the optimal policy for unknown environments, applying actions and observing the generated states to estimate the state-action-value functions (Wiering & Otterlo, 2012).

The main mathematical problem consists always in an approximation of the optimal policy π^{*} that maximizes the value function Q^{*} for each state. There are three general methodologies available to solve this problem (Wiering & Otterlo, 2012):

- Model-based solution: Makes a MDP approximation to calculate the policy. Using s, a and γ, it learns the probability transition function T and the rewards function r before achieving π^{*} and Q^{*}. The dependency of so many stages results in a very difficult implementation.
- Policy-based solution: These algorithms work as continuous developers of some policy, making constant updates on it until finding the optimal condition, called also as direct policy-search or actor-algorithms. It learns a parametrized policy to choose the actions.
- Value-based solution: These algorithms use the actions, states and rewards to update a value function, and based on it, next actions are selected. It is widely applied, in on-line, off-line, on-policy and off-policy algorithms as Q-Learning and Deep Q Network (DQN).

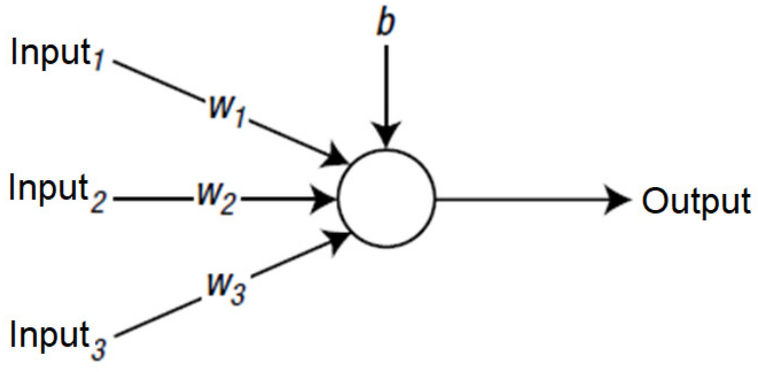
The study of Chen (2018) reinforces that value-based methods are recommended for discrete actions spaces, while policy-based methods are recommended for continuous actions spaces.

2.3.8. Neural Network

Neural networks seek to artificially reproduce the same functioning of the human brain through mathematical models. The basic unit of the network is the node, which is combined in complex interconnections ruled by different weights in order to act in a similar way to human neurons (Wiering & van Otterlo, 2012).

A node by itself is not capable of storing information, but the association of nodes with their weighted connections becomes capable of acquiring the ability to understand how to perform complex functions, just as the human brain does through its extensive network of neurons (Kim, 2017). Figure 9 illustrates an artificial neural node.

Figure 9. Artificial neural node with three inputs, a bias and output.



Source: Kim (2017, p. 20).

A node, like the one shown in Figure 9, is an input-output relationship as a function. It receives its inputs pondered by specific and different weights. In the example, the 3 inputs have weights w_1, w_2 and w_3. The node also carries a bias b that must be added to the sum of the inputs, as stated in (8).

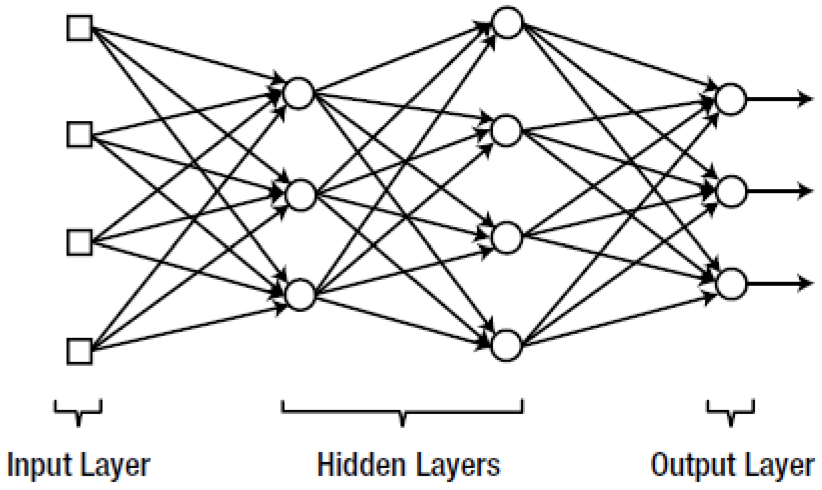
$$Output = Input_1 w_1 + Input_2 w_2 + Input_3 w_3 + b \quad (8)$$

Before proceeding outward from the node, the signal goes through an activation function φ. This function acts as the processing rule that governs the node and gives the output signal a peculiar format. (Kim, 2017). One commonly used activation function is the sigmoid, presented in (9). It allows the node to always deliver an output value contained between zero and one.

$$\varphi(Output) = \frac{1}{1+e^{-Output}} \tag{9}$$

The assembly of many interconnected nodes as a mesh forms what is known as a network, and similar to the brain, the execution of a single complex function can be distributed and executed by many simple individual processes (Kim, 2017). Its representation can be divided into layers as in Figure 10.

Figure 10. Layered structure of nodes.



Source: Kim (2017, p. 22).

In Figure 10, there is an entry layer, the square shapes, two intermediate layers of nodes, and the output layer with the nodes that deliver the results. The inputs, closer to the left, are only signal receivers and do not act as operating nodes. The middle layers can also be called hidden, because they are not accessible from the extremities (Kim,2017).

The number of layers classifies the type of network. According to Kim (2017), the absence of intermediate layers results in the pioneering format of the method, the single-layer network, which has only one layer of inputs connected to one layer of outputs. Multi-layer networks are specifically called those that have hidden layers, having the nomenclature of shallow neural network when there is only one hidden layer and deep neural network when there are two or more hidden layers.

There is a direct relationship between a network's data processing capacity and the number of nodes and layers, with the most complex cases being demanded by more elaborate networks. In this context extensive deep neural networks are applied to the most unpredictable nonlinear behavior systems (Kim, 2017).

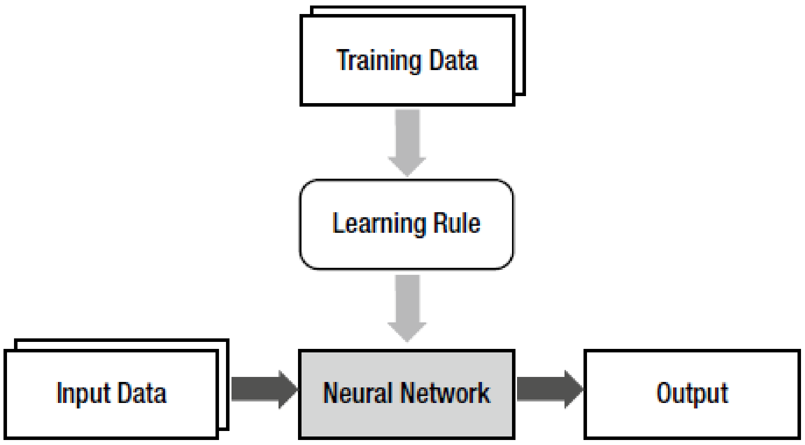
The learning within the structure occurs by updating weights and biases while the achieved outputs are compared to expected outputs. Following the updating equation (10), weights and biases use the remaining error to converge into values that will deliver the correct output-input relation for the system.

$$w_{ij} = w_{ij} + \beta\varphi_i^{'}(ή_i)e_i x_j \qquad (10)$$

The indexes i and j are used to represent the traveling signal between two nodes, respectively. Thus, w_{ij} is the weight of node i output to node j input; β means the learning rate that establishes the update velocity in a range of 0 to 1; The term $\varphi_i^{'}(ή_i)$ is the derivative of activation function φ evaluated at the weighted sum of node i output; e_i is the error of node i output and x_j is the output from node j (Kim, 2017).

The learning process assumes the workflow of Figure 11, where the neural network replaces the central model and a learning rule becomes the ML method to develop the policy able to correlate input and output data properly.

Figure 11. Neural Network learning workflow.



Source: Kim (2017, p. 12).

The learning rule may follow three different schemes, the Stochastic Gradient Descent (SGD), the batch or the minibatch (Kim, 2017):

1) SGD calculates the error for each training data and immediately updates the weights.
2) Batch method updates the weights with the average error obtained after an entire cycle of training data, also called as *epoch*. It delivers more stability for the process but increases processing time if the datasets have larger sizes.
3) Minibatch makes a mix of SGD and Batch methods. The algorithm uses part of the total training data to calculate the errors and the updating of weights is made at the same pace, allowing faster performance for larger datasets (Kim, 2017).

The normal traveling direction of the data is from input to the output, what is called forward propagation. Unfortunately, the error in multi-layered networks cannot be calculated because the expected results are not known. The solution in this case is the backpropagation, which consists of reversing the traveling direction of the error, sending it back from output to the input. The updating equations are very similar, but in this case after an entire cycle, the weights are calculated firstly for the latest nodes and based on their outcomes, the previous connections are updated (Kim, 2017).

2.4. State of the Art

There is a substantial amount of ADAS currently available, which allow to cover a wide variety of parameters and maneuvers, as lateral control, longitudinal control, parking aid, reversing aid, vision enhancement, driver monitoring-crash systems, road surface or low-friction warning (Lundgren & Tapani, 2006).

Low Speed AEB technology is cited to have led to a 38% reduction in real-world rear-end crashes (Fildes et al., 2015). This is possible because the device acts when it is still possible to avoid the accident, or significantly reduce its severity while maintaining the high level of attention of the driver by its warnings and interventions (Bella & Russo, 2011).

Autonomous emergency braking and steering systems (AEBSS) have proven to be even better for perpendicular collisions, as autonomously perform braking or evasive maneuver by steering. (Kovacevaa et al., 2020).

Variations and combinations with AEB are also subject of research. The study of Sandera & Lubbea (2018) showed that 80 to 90% of collisions in junctions or intersections could be avoided by called Intersection-AEB, where the vehicle is equipped with 180º coverage front sensors. Clearly, such a percentage of reduction is only achievable with the hypothesis of 100% market penetration of these devices for the long-term.

Remaining in the idea of fully equipped cars, not only direct interventions in the driving but also forward collision warning (FCW) in addition to AEB could avoid almost 1 million U.S. police-reported rear-end crashes and more than 400.000 injuries in 2014 (Cicchino, 2017).

For chaotic traffic situations, the concern about cyclists safety led (Duan et al, 2017) to release a method to design an adaptive Bicyclist-AEB. This particular study is based on drivers' behavior captured from driving simulators, during different scenarios of conflicts vehicle-bicycle remounted from real events.

In parallel, with the advent of increasingly efficient RL methods, successful control applications were achieved, for instance, learning of robot behaviors (Peters & Schaal, 2006; El-Fakdi & Carreras, 2008), car driving (Riedmiller et al., 2007), engine control (Liu et al., 2008) and even helicopter guidance (Abbeel et al., 2010).

RL techniques applied to AEB or any maneuver assistant system may act in two levels of approach: A decision maker or a maneuver executor. As decision makers, RL agents can be used in interactive highway environments to learn how to drive as close as possible to a desired velocity by giving to the vehicle standard orders to change the lane, to accelerator to brake (Mirchevska et al, 2018). As maneuver executors, such methods actually can learn how to control the available inputs, to understand the environment behavior and to perform the entire transition (Wang et al., 2018).

Specifically in the field of deep learning, this technique was recently used to learn an effective driving policy for pedestrian collision avoidance with a fast convergence rate, acting as an AEB algorithm (Li et al., 2020).

Some advanced approaches as You et al. (2020) started from a stochastic traffic modeling, and achieved the desired driving behaviors using both RL and inverse reinforcement learning (IRL). In this particular case, the road geometry was took into consideration and the algorithm was prepared to perform over n possible lanes, returning actions that avoid collisions.

Providing faster results and saving high hardware implementation costs in the development of these algorithms, driving simulators are disseminated among these lines of research.

Advanced-interactive solutions provide a high degree of realism, allow experiments to be conducted in controlled conditions and assure the highest degree of safety for test drivers (Bella & Russo, 2011). Various studies also have shown that observations derived from driving simulators are a reliable source of data to examine drivers' behavior and the assessment of rear-end collision risk (You et al., 2020; Bella, 2009; Farah et al., 2009; Jenkins & Rilett, 2004; Broughton, 2007).

In Jirgl et al. (2019), a basic car driving simulator was designed and implemented with MATLAB/Simulink tools to assess the human response to different scenarios and conditions. Further studies with the aim of analyzing the efficiency of driver assistance systems have also been developed (Cheng et al., 2002; Genya & Richardson, 2005; Jamson et al., 2008).

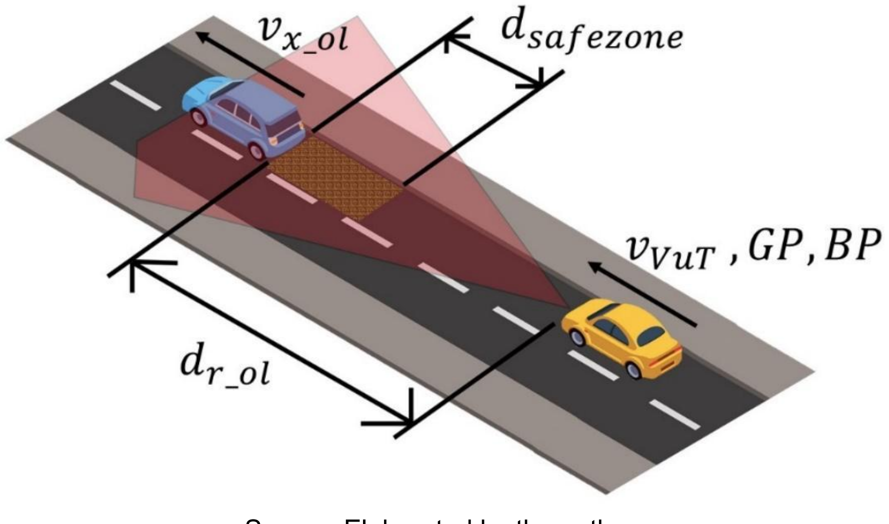
This work aims to unite some of the techniques presented so far to propose a basic AEB algorithm developed with deep RL techniques. To generate enough data for training and to enable faster, safer and more affordable results, the tests are virtually performed in a driving simulator. The methodology developed by the author is shown in the next chapter.

3. METHODOLOGY

The problem consists of a vehicle under test (VuT) that travels on a double-lane way, where surrounding cars can suddenly brake or cross onto its lane at lower speeds, as shown in Figure 12. All participating vehicles are randomly generated and managed by a driving simulator while the virtual VuT actions are performed by the developed algorithm, called RL agent. The agent does not have any other information on the intentions of the other vehicles besides the data provided by its frontal sensor and basic telemetry as gas pedal, brake pedal and current speed.

To train and deliver a final RL algorithm capable of longitudinally controlling the car, this work assumes a Light Detecting and Ranging (LIDAR) sensor located on the front end of the vehicle and a pre-processing system that compiles reliable information on the distance and speed of the identified objects, transforming them into the sensor's reference coordinates. A deeper research on sensing technology is not yet the focus of this initial work, but the development of the algorithm for AEB.

Figure 12. The environment of the RL agent.



Source: Elaborated by the author.

In addition to the variables read by telemetry and the frontal-sensor, Figure 12 also presents a safety distance called safe zone. This parameter will be used further in subsection 3.4 for the formulation of the reward function.

To solve the problem through a ML technique as a Reinforcement Learning method, some assumptions need to be made and the mathematical definition of the problem is stated in the following subsections.

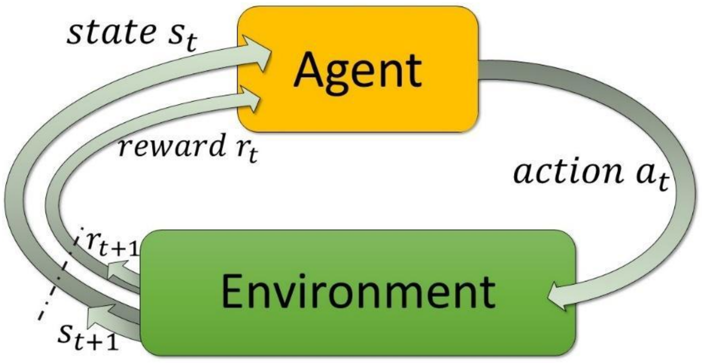
3.1. The problem as a RL process

The intention of a typical RL problem, when considered as a Markov Decision Process (MDP) (Howard, 1966), is to find an optimal policy π * which addresses the states ($s \in S$) of the environment into actions ($a \in A$) that the agent takes at the corresponding time step-in a way of maximizing a total expected return R . This last set is a cumulative sum of immediate rewards received over the completion of a task (Sutton & Barto, 1998), respecting a proper reward function $r: S\ x\ A \rightarrow R$.

The sets S and A are state space and action space, respectively, and can be discrete or continuous depending on the problem. In the case of this work, the states are considered continuous and the actions are assumed as finite, configuring a model-free approach.

In the Figure 13, the RL agent and the environment interact during a sequence of discrete time steps $t = \{0, 1, 2, ...\}$. At each time step the RL agent receives information about the environment's state $s_t \in S$ and based on that, selects an action $a_t \in A\left(s_t\right)$. One time step later, as a result of the action performed in s_t, the RL agent receives a reward $r_t \in R$, and finds itself in a new state s_{t+1}. A policy π is a mapping $S \rightarrow A$ that maps each state s to an action that should be executed when in s.

Figure 13. RL process.



Source: Adapted from Kim (2017).

The goal of the RL agent is finding a policy to achieve the highest reward in the long run. The state-value function which yields the optimal policy π^* is called optimal state-action value function Q^*, defined by (11) (Sutton & Barto, 1998).

$$Q^* = Q^{\pi}(s, a) \qquad \forall \ s \in S, \ a \in A \tag{11}$$

3.2. State space and actions

The inputs chosen from the environment to represent it are all the information available for the RL agent to learn the optimal value function in the training stage. In such an approach, even variables from the vehicle itself are considered part of the environment and the agent can be literally reduced to the ECU, or the software inside it. All of these data are included in the state vector S, updated in discrete time step s.

Aiming further expansions, the current state vector was prepared to allocate objects in more than just the actual VuT's lane. Then, the indexes used for this purpose would be ol-central lane where the VuT is, ll-left lane and rl-right lane, which are the most relevant lanes for the VuT in case of any maneuver and also achievable by the sensor range.

The state vector S, at this moment, is composed of 9 values:

$$S = \left[d_{r_{ol}}, v_{r_{ol}}, d_{r_{ll}}, v_{r_{ll}}, d_{r_{rl}}, v_{r_{rl}}, v_{VuT}, GP, BP\right] \tag{12}$$

$$v_r = v_x - v_{VuT} \tag{13}$$

In (12), all variables referring to surrounding cars are already converted to relative states, as the relative velocity v_r , follows (13). The values are always defined between the VuT and the nearest cars, or objects, of each lane. When there is no adjacent lane available, relative distance is defined as $dr = 0$, and the relative velocity to $vr \rightarrow \infty$. In this case, it’s assumed the VuT keeps running in the central lane, because there is no chance to take a different lane where the relative distance to any object will be zero, or its relative velocity of approximation is substantially hazardous.

In subsection 3.5, a simplification for the last two values of the vector S will be presented to facilitate implementation and training.

In terms of actions, the agent acts directly in the positions of accelerator and brake pedals, shown in (14). The control of such variables can act directly in the longitudinal response of the vehicle and to avoid collisions or to mitigate them.

$$A = \left[GP_{Desired}, BP_{Desired}\right] \quad (14)$$

The easy reading of gas pedal position (GP) and brake pedal position (BP), which are already normalized values between 0 and 1, and the possibility to make interventions in them, enable the agent to test reward responses and quickly reach a good knowledge of the environment. In subsection 3.5, on the Training Method, a simplification of the vectors S and A will be presented to facilitate implementation.

These actions are the minimal set for achieving the goal of avoiding collisions. No much information must be provided for a basic AEB system. The only information about pedal position is enough to be returned into the driving simulator and the same in a real application with no further processing, once modularized systems already have been reported to perform better in autonomous driving than end-to-end systems(Shalev-Shwartz & Shashua, 2020).

Before assuming some action as the best for a given situation, the RL method uses a check step through a reward function. To understand the proposed reward function to be shown in the subsection 3.4, the concepts of Time To Collision (TTC) and Time To Total Stop (TTS) need to be presented.

3.3. TTC and TTS Calculation

The idea of computing a TTC was first suggested by Hayward (Hayward, 1972). It is defined as “the time required for two vehicles to collide if they continue at their present speed and on the same path”. The time-to-collision of a vehicle driver combination i instant t with respect to a leading vehicle $i - 1$ on the same path is calculated with

$$TTC_i = \frac{X_i(t) - X_{i-1}(t) - l_i}{\dot{X}_i(t) - \dot{X}_{i-1}(t)} \qquad \forall\ X_i > X_{i-1}(t) \tag{15}$$

where $\dot{X}$ denotes the speed, X the position, and l the vehicle length (Minderhoud & Bovy, 2001). For the problem stated in the Figure 12, the term l can be canceled, since the sensor is positioned directly on the front bumper of the VuT and the first point of interest in the car ahead is its rear bumper.

From (15) it also can be seen that the TTC is usually computed for a specific path. Such a condition is assumed in this work, as the pre-processing algorithm of the frontal sensor already must consider that the objects are straight ahead of the VuT in the same path. It is also assumed that this basic algorithm is fitted for highway environments or low speed situations where the curvature of the road is not relevant.

As presented in (12), the current algorithm already receives as input the relative velocity and the relative distance between the VuT and the leading car. Then, the calculation of TTC_{VuT_r} can be replaced by (16)

$$TTC_{VuT_r} = \frac{d_r - d_{safezone}}{v_r} \tag{16}$$

where the relative velocity v_r is positive in situations when the leading vehicle is slower than the VuT (approaching) and negative when it is faster (moving away).

To create an additional mechanism for safety, the distance $d_{safezone}$ is included in (16). Such dimensions must respect what is determined by Bussgelkatalog (2020), as at least, “5/10 of half the vehicle speed”, leading to (17).

$$d_{safezone} = \frac{v_{V\,uT}}{4} \tag{17}$$

Another important concept to the hereafter reward function is the time to total stop. This is stated as the time between the driver starts to brake from a cruise velocity with total pressure ($BP = 1$) and the moment when the vehicle reaches a resting state.

This particular number is extracted from the driving simulator which considers all vehicle dynamics modeled and also where the environment is running. It intrinsically considers special characteristics from the specific car model used as its weight, center of gravity, suspension response, ground adherence and even air dynamics. The calculation also includes a 0.63 seconds delay of response due to the human reflects and changing of pedals, presented by the last term of equation (18).

$$TTS_{V\,uT} = 0,0001v_{VuT}^{2} + 0,0733v_{VuT} + 0,6324 \tag{18}$$

The measurement was performed empirically, by varying cruising speeds and measuring $TTS_{V\,uT}$. For instance, at a cruising speed of 100km/h, the expected to perform an emergency stop is 8.9 seconds. In a situations of very low speeds as 10 km/h, $TTS_{V\,uT}$ must be next to the delay of response.

For applications with different car models, gauging needs to be replicated in order to establish the $TTS_{V\,uT}$ and to fit the problem to the specific vehicle.

3.4. Proposed Reward Function

The RL agent must be rewarded for maintaining the safety distance and always being in a position to completely brake the vehicle before collisions.

The final reward function is a sum of two terms, a reward $REW_{\Delta T}$ based on the comparison between TTC and TTS, and a reward $REW_{\Delta V}$ based on an error to a desired velocity. This second reward is design to prevent the agent from concluding that the best action is to stay still, it is based in a desired velocity $v_{Desired}$, considered the limit velocity of the road. The calculation of ΔT and ΔV is expressed, respectively, in (19) and (20).

$$\Delta T = \frac{TTC_{VuT_r}}{TTS_{VuT}} \tag{19}$$

$$\Delta V = \left| \frac{v_{VuT} - v_{Desired}}{v_{Desired}} \right| \tag{20}$$

In situations where $0 < \Delta T \leq 1$, the calculation is a warning for a potential collision, as the time to completely stop the vehicle is less than the expected time to collide. In this case it is not recommended to give rewards to the agent, as it must avoid such situations by reducing its velocity steps earlier. Considering that $TTC_{VuT_r} < 0$ represents a leading car moving away, in this case $\Delta T < 0$ means a safe condition and the agent can receive rewards.

To measure how near of the desired velocity and to reward the agent, when ΔV is near to zero, the reward must be bigger. It means that the vehicle is running near to the road specified velocity. This term of reward also allows the agent to learn of accelerating again the vehicle after some braking procedure, when the leading car has already returned into movement.

Based on the performance of the agent and consecutive iterations the equivalent rewards $REW_{\Delta T}$ and $REW_{\Delta V}$ are listed, respectively, in (21) and (22).

$$REW_{\Delta T} = \begin{cases} 0.5 & if \quad \Delta T > 1 \; or \; \Delta T < 0 \\ 0 & if \quad 0 < \Delta T \leq 1 \end{cases} \tag{21}$$

$$REW_{\Delta T} = \begin{cases} 0.5 & if \quad \Delta V = 0 \\ 0.4 & if \quad 0 < \Delta T \leq 0.1 \\ 0.25 & if \quad 0.1 < \Delta V \leq 0.2 \\ 0 & if \quad \Delta V > 0.2 \end{cases} \tag{22}$$

As stated in the beginning of this subsection, the final reward REW_{final} is a sum of the two terms of (21) and (22). It means that the maximum reward $REW_{final} = 1$ will be achieved when the safety distance is maintained at the same time that VuT travels at desired velocity. Obviously, that will only happen if there is no car ahead or if the leading car is in at the same velocity or faster.

But the combination of these two factors of reward forces the agent to recover its velocity when the traffic turns to move again. As a further expansion to the current algorithm, such a reward term can be used for deciding whether or not to change lanes if the braking maneuver is not enough to prevent collisions. It may also take such action if slower vehicles in his lane may reduce his cruising speed. These topics will be discussed in the chapter of conclusions.

3.5. Deep Neural Network and Offline Learning Process

As stated before, the current algorithm seeks to learn how to perform actions in an environment as an AEB system to avoid or mitigate collisions. For this purpose, a deep neural network is implemented. The Figure 14 presents a simplified flowchart of the multi layered network.

The neural network imitates the human brain replying its neurons by nodes. The neurons are connected, receiving inputs, applying weights to them, following some bias, and delivering output. The final output of each node relies on an activation function which is responsible for giving to this method the understanding of non-linear behaviors (Kim, 2017).

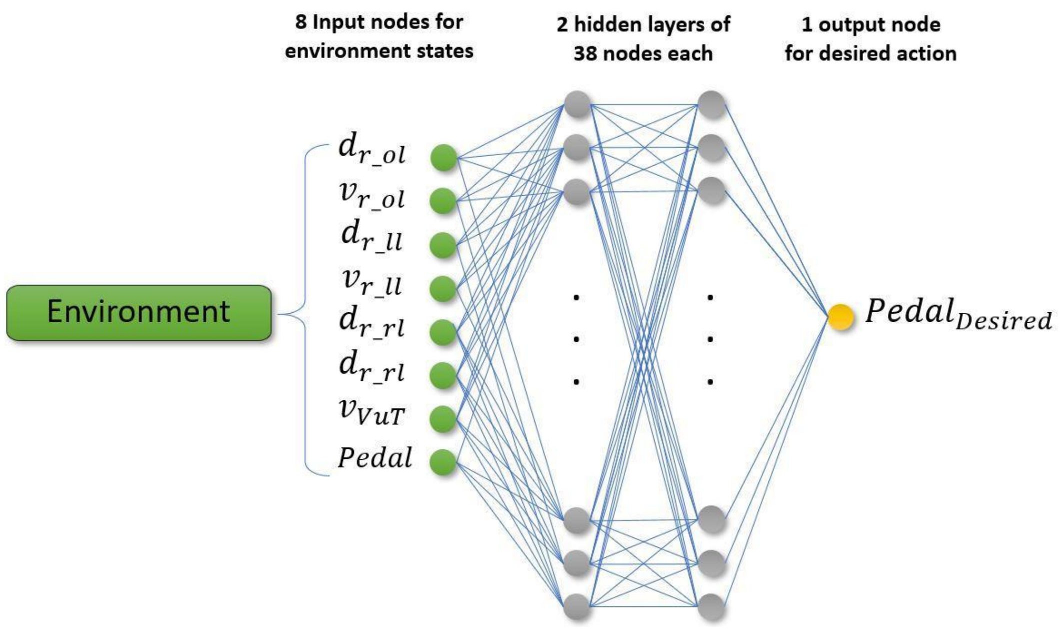
The layers at the borders make contact with the external ambient and are responsible for data input and output. When two or more layers are placed between these two borders, the mechanism is called a deep network, because the new layers are not accessible from the outside. Networks with more hidden layers and hidden neurons are allowed to solve more complex classifications and are recommended for big sets of data. (Kim, 2017). As a first approach, the purpose of this work is creating a basic deep neural network algorithm, so it will be considered only two hidden layers to create the hidden network.

In terms of increasing the computational performance, two important steps are taken to increase the convergence speed of the algorithm. First, all data are normalized between 0 and 1. It is known as a technique to facilitate the calculation inside the neurons without using so much computational memory (Ioffe & Szegedy, 2015).

The second coding trick applied is to unite the information of GP and BP, which vary between 0 and 1, in a single variable called $Pedal$, with interval [-1,1]. In the new format, the negative values represent the brake pedal, where -1 is maximum braking and the positive set means the gas pedal, where 1 is maximum acceleration. For this simplification, it is assumed that the driver does not press both pedals at the same time.

Thus, the 9 values of the state vector presented in (12) are reduced to 8 and the output action vector of actions (14) is turned into a single variable, as shown in Figure 14.

Figure 14. Deep Neural Network proposed.



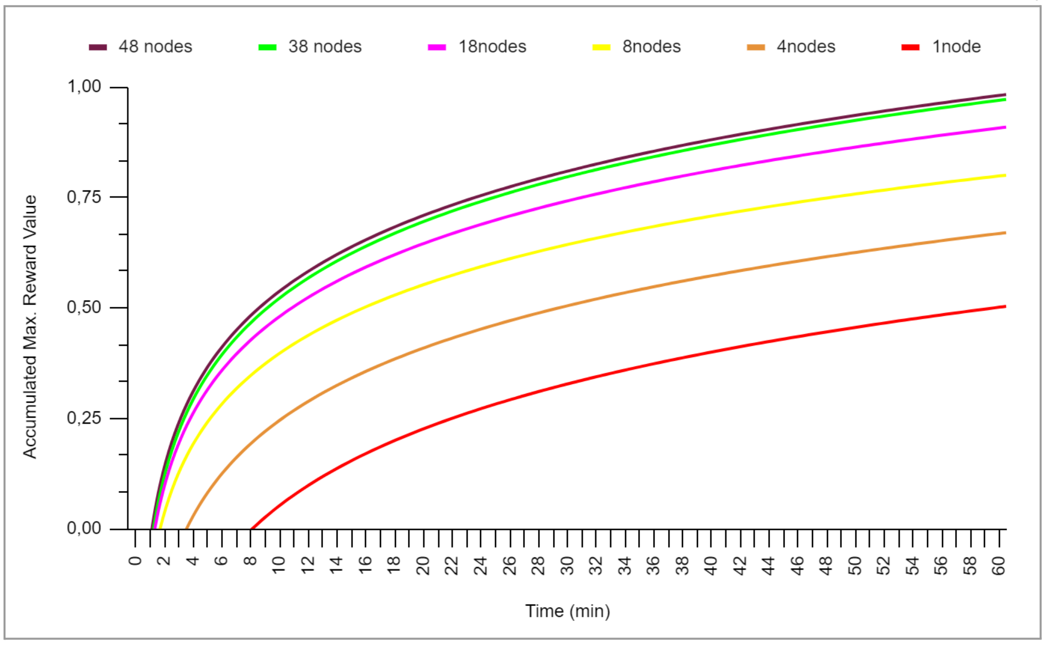
Source: Elaborated by the author.

The RL agent is composed by a Deep Network with four layers: the first has 8 nodes and it is where the inputs from the environment are received; second and third layers are hidden networks, defined with 38 nodes each one; the last and fourth layer is a single node where used as output for the desired pedal position which has a range between -1 and 1.

The proper number of nodes in each hidden layer was defined after previous simulations, described in Figure 15, considering the hardware available for the tasks, an Intel Core™ processor in parallel pool with 16 GB of RAM and its software, Windows 11 Pro running CarMaker™ combined with MATLAB/Simulink 2019 edition.

It is considered that all necessary settings to potentialize the hardware were performed. The final results performed between 25 and 32 minutes of simulations each.

Figure 15. Previous simulations to define the number of nodes per hidden layer.



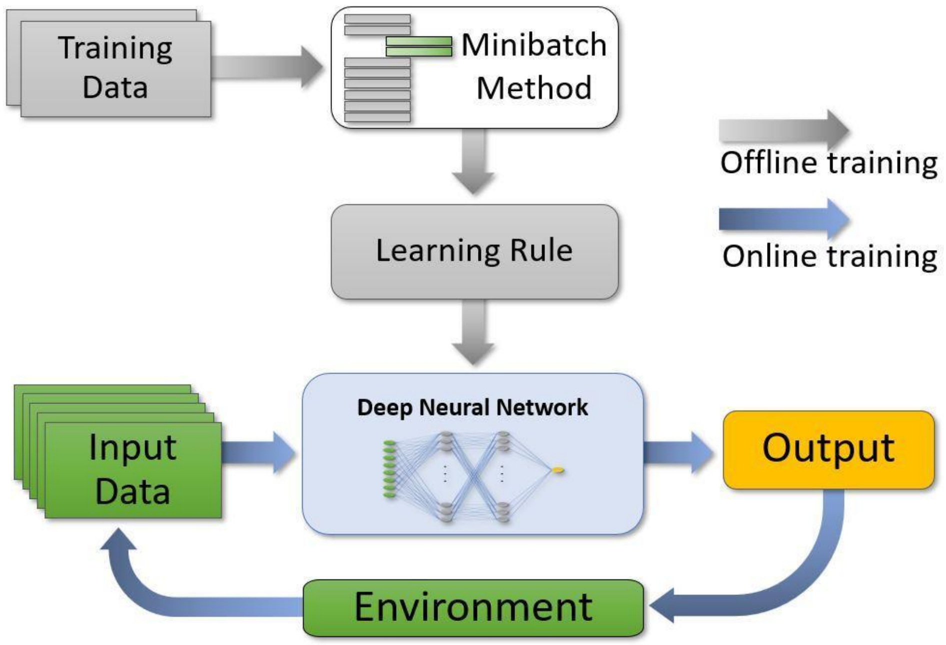
Source: Elaborated by the author.

In Figure 15, an initial approach of the code was performed in 6 different combinations of hidden layers. The results were interpolated and are best represented by logarithmic functions, where the maximum reward value converged over time. As the number of nodes increases, it is clear that the time to train and accumulate the maximum reward value reduces until the increase in nodes no longer represents a significant difference for that particular case. For this reason, without better behavior above, the mesh was set in 38 nodes.

To generate the first set of data, a simple scenario is developed in the driving simulator platform CarMaker™. The VuT runs in a freeway with no car ahead. This initial step allows to assign previous values for weights and rewards, preparing the system to operate in online flow.

The process of training and development of the agent response accuracy follows the flowchart presented in Figure 16. The first training procedure comprises all steps in the gray arrows which starts by extracting data from the initial set using a Minibatch technique.

Figure 16. The RL training workflow.



Source: Elaborated by the author.

The minibatch method selects a part of the training data set and uses them for training in the batch method. Each weight update is calculated for all errors of the selected data, and the average of the weight updates is used for adjusting the weights. After this step, it trains the neural network with the averaged weight update.

As Kim, (2017) explains, if 20 arbitrary data points are selected out of 100 training data points, the batch method is applied to these 20 data points. In this case, a total of five weight adjustments are performed to complete the training process for all the data points (5 = 100/20). It combines speed and stability, a fitted solution for Deep Learning, which manipulates a significant amount of data.

The way the algorithm uses these portions of points to update the Deep Neural Network of the agent is illustrated in Figure 16 by the Learning Rule block. It includes the activation function applied, the learning rate, discount factor and other parameters for convergence purposes (François-Lavet, 2018), which will not be deeply discussed in this paper.

The final parameters used during the learning process are listed in Table 1.

Table 1. Setup of Algorithm Parameters.

Algorithm Parameter	Setup
Number of input neurons	4
Number of hidden layers	2
Number of neurons in each hidden layer	38
Connection between layers	Fully Connected
Number of output neurons	1
Activation function	TanH
Mini-batch size	0.5
Gamma	0.9
Number of training iterations	20

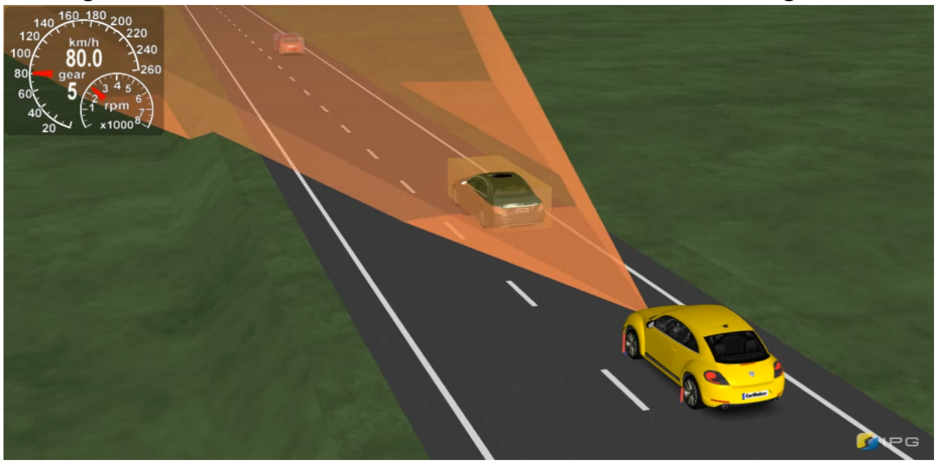
Source: Elaborated by the Author.

Once written the main functions of the algorithm, its effectiveness can be already tested. For learning and post-testing it is necessary to provide useful scenarios, subject of the next section.

3.6. Proposed scenarios

All the learning and testing procedures executed with the proposed algorithm were performed inside of a driving simulator software, the chosen solution was CarMaker™, from IPG Automotive GmbH. Figure 17 shows the animation interface of the simulator, where the sensor range is represented by the orange area, where identified leading car receive a box and telemetry results can be displayed in the background.

Figure 17. Animation extracted from simulator during tests.



Source: Elaborated by the Author.

The simulator runs with pre-set scenarios, where each maneuver of the car at each time step can be configured. Its dynamic responses are already verified and validated by works as the one presented by Weber & Kanarachos (2019).

To execute the learning process, the VuT equipped with the proposed algorithm was submitted to the training scenario described by Table 2. All maneuvers described were performed in a straight road and all cars appeared in front of the VuT while its desired velocity was 80 km/h. During the learning process the AEB were not activated and the VuT was able to pass through the leading cars, this condition was needed to deliver bad rewards to the system and to teach it what should be the right actions before the collisions.

Table 2. Training Scenario.

Instant [s]	**Maneuver**	**Leading Car 1 [km/h]**	**Leading Car 2 [km/h]**	**Leading Car 3 [km/h]**
0	Constant velocity	60	-	-
23	Constant velocity	-	60	-
35	Const. Acceleration	-	120	-
48	Constant Braking	-	0	-
56	Stopped Car	-	-	0

Source: Elaborated by the Author.

For the training process, the system followed Figure 16. Once the AEB is unable and the system only captures impressions from the environment, it is called an offline training. After reaching the trained deep neural network, the routine enables the AEB algorithm and submits it to the environment, starting the online training. While the car runs again through the training scenario, its output changes the environment which improves the algorithm even more.

The combination of different numbers of offline and online training cycles will build the algorithm faster and/or smarter and this is one part of optimization the system can have. But the final response can only be validated in exclusively new scenarios where the code has not been proven. For this purpose, this work will combine the CarMaker™ solution with the existing test protocol used by EuroNCAP to rank AEB systems.

3.7. EuroNCAP test protocol for AEB systems

Over the last decade EuroNCAP has played an important role in vehicle safety by setting the test standards and safety ratings, such as the well-known 5-star ranking, which raises the bar for manufacturers while making the safety level of their cars public and accessible to consumers. With the entry of ADAS, EuroNCAP has reinvented itself and established the protocols for testing this type of technology.

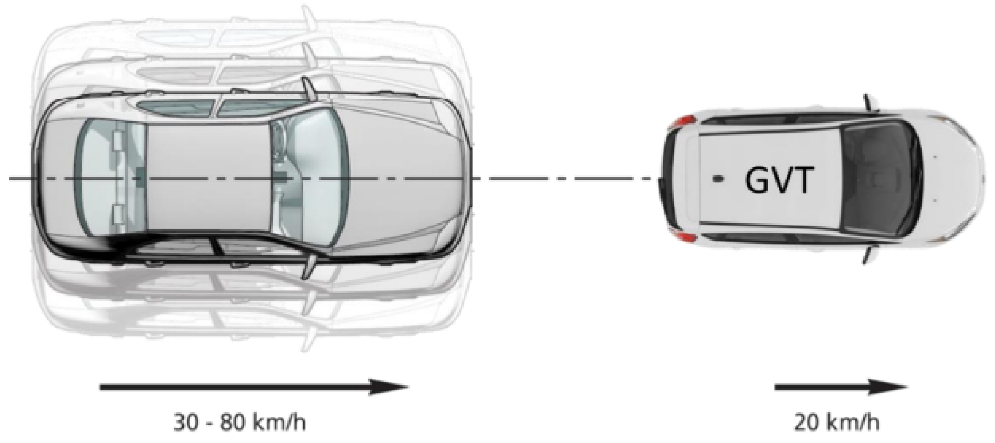
For AEB, the specific protocol has some important terminologies:

1) AEB City: categorized between velocities of 10 and 50 km/h.
2) AEB Inter-Urban: categorized between velocities of 30 and 80 km/h.
3) CCRs - Car-to-Car Rear Stationary: a collision in which a vehicle travels forward towards another stationary vehicle and the frontal structure of the vehicle strikes the rear structure of the other.
4) CCRm- Car-to-Car Rear Moving: a collision in which a vehicle travels forwards towards another vehicle that is traveling at constant speed and the frontal structure of the vehicle strikes the rear structure of the other.
5) CCRb- Car-to-Car Rear Braking: a collision in which a vehicle travels forwards towards another vehicle that is traveling at constant speed and then decelerates, and the frontal structure of the vehicle strikes the rear structure of the other.
6) GVT: Global Vehicle Target. Also called the leading car during this work.

For AEB City, only the CCRs scenario is applicable where the AEB functionality at lower speed is tested. In this case, the lower peak acceleration for the passengers (called Whiplash) guarantees the better score. For AEB Inter-Urban, the system is tested in three scenarios (CCRs, CCRm and CCRb).

For the CCRm scenario (Figure 18), the VuT must approach to a GVT that keeps its constant velocity of 20 km/h. Different initial velocities are tried, from 30 to 80 km/h.

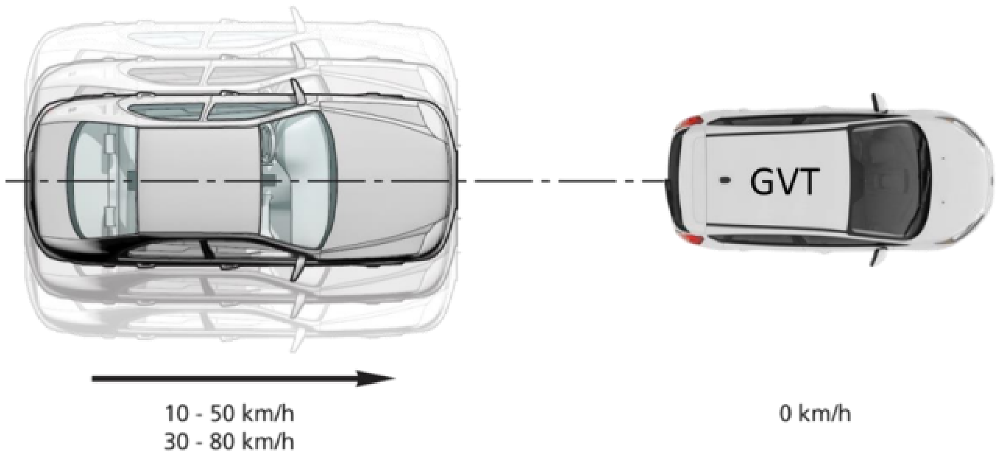
Figure 18. CCRm scenario.



Source: EuroNCAP (2017).

In the CCRs scenario (Figure 19), the VuT will vary its velocity in different attempts from 10 to 80 km/h, while approaching the GVT which is stationary in the middle of the road.

Figure 19. CCRs Scenario.

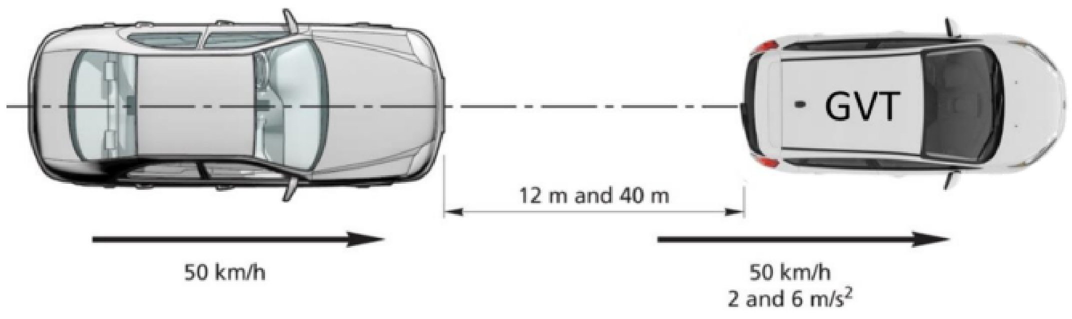


Source: EuroNCAP (2017).

In the CCRb scenario (Figure 20), both cars start at the same velocity, 50km/h. The distance between each other is test with 12 meters and 40 meters, at the same time the a_{brake} is also varied as 2 m/s² or 6 m/s².

All the maneuvers to be executed are compiled in Table 3.

Figure 20. CCRb Scenario.



Source: EuroNCAP (2017).

Table 3. Testing Scenarios.

Scenario	**Maneuver**	**VuT [km/h]**	**Leading Car [km/h]**
CCRm	Lead car at constant lower velocity, starting 150m ahead	30	20
		40	
		50	
		60	
		70	
		80	
CCRb	Sudden braking of the leading car at -2m/s², starting 40m ahead.	50	50
	Sudden braking of the leading car at -6m/s², starting 40m ahead.	50	
	Sudden braking of the leading car at -2m/s², starting 12m ahead.	50	
	Sudden braking of the leading car at -6m/s², starting 12m ahead.	50	
CCRs	Stopped leading car, starting 150m ahead.	10	0
		20	
		30	
		40	
		50	
		60	
		70	
		80	

Source: Elaborated by the Author.

Across the chapter of results, some discussions will be made about the dynamic response of the VuT during the testing scenarios, as well as the validation of the proposed algorithm compared to the other known solution, the AEB embedded in CarMarker™.

4. RESULTS

During the code development, tests were performed, as shown in Figure 15, in order to choose the number of hidden nodes, to reduce the simulation time, and defining the parameters presented in Table 1 of the last chapter. The final code was capable of training the network for the later three testing scenarios described in Table 3.

The proposed AEB system was evaluated comparing its behavior to other similar software already implemented and released for learning and research purposes. The AEB solution from CarMaker™ in the current work, performed the same testing scenarios of the proposed AEB algorithm and the dynamic response was recorded for comparison.

The aim of the whole process was to generate an AEB system capable, in the simulated environment, to avoid rear-end. All the scenarios were performed and optimized the code through new learning cycles. Such result, points that the reward function and the training method was structured in a satisfactory manner, however some considerations will be made to improve the code in the following sections.

In addition to the occurrence of collisions and the maximum acceleration peak, some variables can be compared as gas pedal position, relative velocity, brake pedal position and relative distance to the target car. After discussing the response to each one of the three scenarios, there will be one example of dynamic response for the critical case, CCRs scenario. Data extracted during the simulation are available in the Annex 1.

Thus, to evaluate and compare the algorithm, let's start with the CCRm scenario, where there is a slower car ahead, differently from the road velocity.

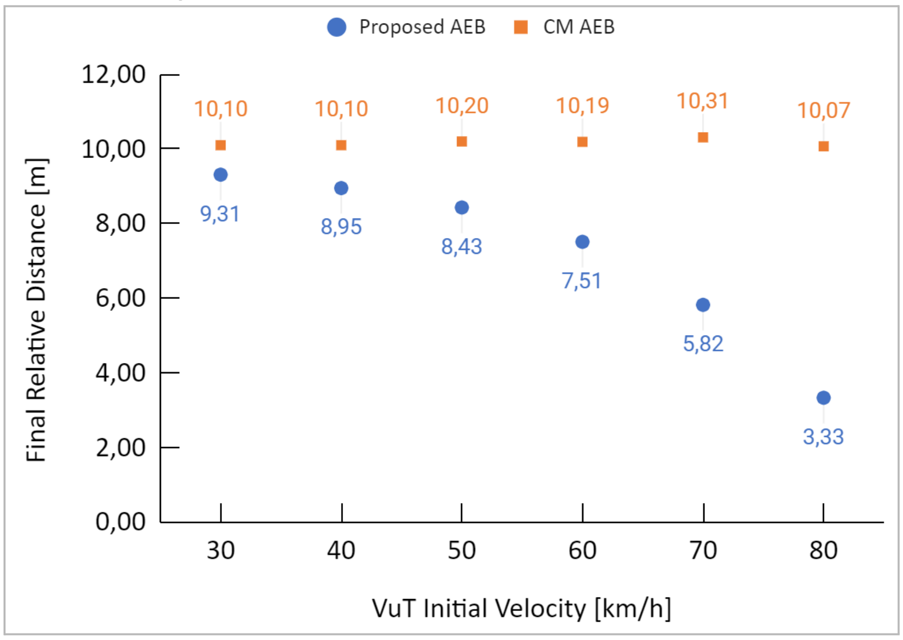
4.1. Response to CCRm Scenario: Slower Leading Car

To validate the algorithm for CCRm Scenario, once trained in all scenarios and saved, the code has been enabled during a similar situation of a slower car ahead, but in this case, it is important to highlight that the values were new, differently from the training data, as shown in Tables 2 and 3. Now, the VuT faces, from training to testing scenarios, relative velocities up to 60 km/h.

These values are the key to prove if the training process was capable of developing the AEB characteristics only by the rewarding methods. At the same time, new environments must be challenging as it would be in real situations of traffic.

In Figure 21, it is possible to compare the final distance reached between the cars, VuT and GVT, using the proposed AEB and the CarMaker™ AEB.

Figure 21. Final Relative Distance between VuT and the leading car, comparing proposed algorithm to CarMaker™ solution in Scenario CCRm.



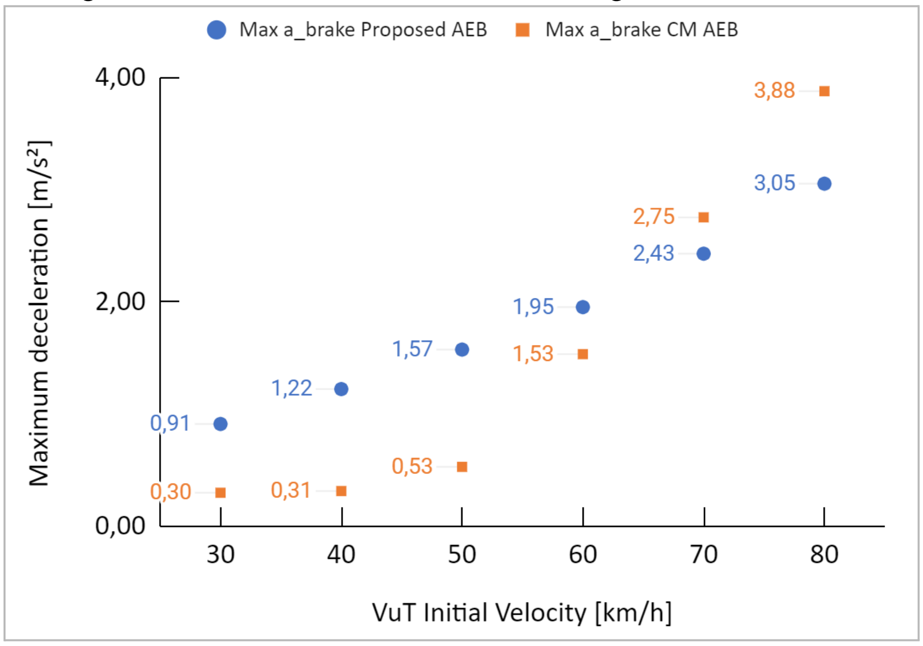
Source: Elaborated by the Author.

In Figure 21, it is possible to see that the algorithm has learned how to brake the VuT and avoided crashing into the leading car, but the final distances are significantly different to the competitor solution. For example, when starting from 80 km/h, the VuT reaches 3,33 meters from the leading car using the proposed algorithm, at the same time it would maintain around 10 meters of distance using the CarMaker™ AEB solution.

Here, the strategies behind the safety distance are different. The CarMaker™ AEB follows a fixed relative distance from any leading object, which in this case, had been set to 10 meters. On the other hand, the proposed algorithm had learned by its rewards and punishments using the rules of (16), (18) and (19). These rules allow the controller to reduce the velocity of the car while approaching and calculating the new safety distance, in some cases it may appear shorter but still respects the TTS of the VuT.

In the case of initial velocity of 80 km/h, the VuT reduces until reaching the same velocity of the leading car (20 km/h) only when their relative distance is 3,33 meters. That happens because the deceleration curve applied by the codes are different, as shown by Figure 22.

Figure 22. Maximum deceleration during CCRm scenario.



Source: Elaborated by the Author.

Despite having a small relative final distance, the proposed algorithm in Figure 22 performs the braking maneuver with a lower deceleration for passengers at higher velocities, as 70 and 80 km/h. In the last section of the results, the dynamic response will show that this does not respectively mean better comfort for the passengers because the brake pedal actuation is not fluid all the time, as normally pressed by the driver in real situations.

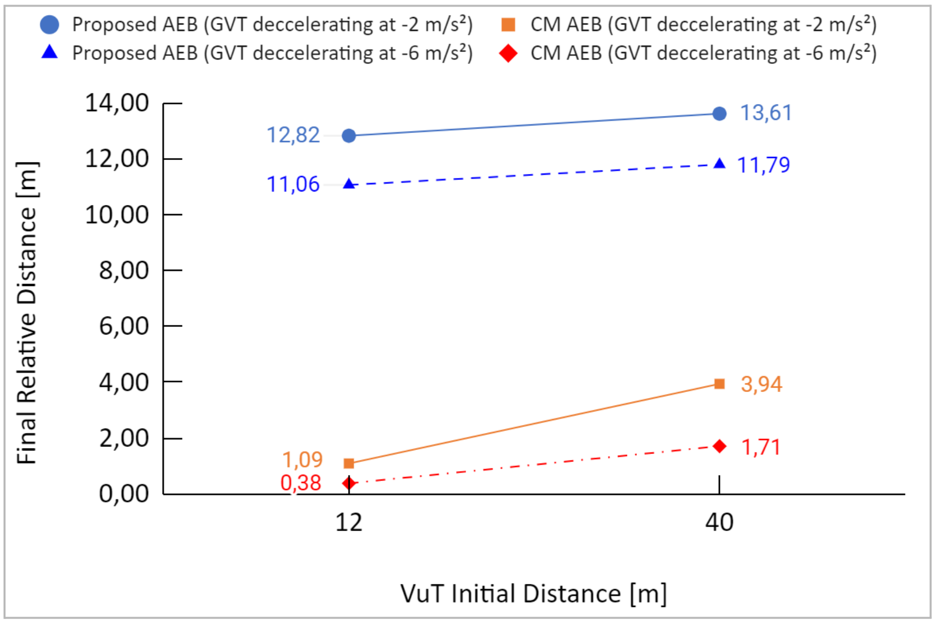
During the development of the rewards functions, no value was directly imputed to carry the influence of the braking deceleration into the actions of the algorithm. It could be one of the future improvements to make the deceleration even softer when in slower approximations.

It can be considered that in cases of approach against slower vehicles, although it has not generated any collisions, still AEB is not as decisive for the safety of the occupants as in cases of sudden braking of the vehicle ahead. Its performance in this type of situation was evaluated in CCRb scenario, described below.

4.2. Response to CCRb Scenario: Sudden Breaking Leading Car

To ensure the surprise effect between training situations and online AEB use, during CCRb Scenario, the VuT was evaluated in two situations, when the lead car decelerates at -2m/s² and when it does so at -6m/s². The latter seeks to reproduce situations where unexpected situations occur in front of the lead car and the risk is transmitted to the following vehicles. Figure 23 summarizes the behavior of both methods in the hazard situation.

Figure 23. Final Relative Distance between VuT and the leading car, comparing proposed algorithm to CarMaker solution in the CCRb Scenario.



Source: Elaborated by the Author.

When compared to the validated CarMaker model, the new proposed algorithm showed safer final distances for all situations. Both when starting from a distance of 40 meters and from a distance of 12 meters, the VuT is able to brake strongly enough to neither approach nor collide with the leading vehicle. This behavior demonstrates robustness in the established reward method, since at no point in the training scenarios did the VuT face anything of the sort.

Something interesting is the final distance in the case of GVT -2 m/s² starting at 12 meters. Here the VuT finishes at an even greater distance than before. This proves that the proposed algorithm is capable of applying higher braking accelerations than those found in the environment in order to keep its commitments to the search for the highest accumulated reward. The table 4 shows these accelerations in the second column, while the CarMaker solution is limited to brake lower than the leading car, resulting in the shorter final distance already presented.

Table 4. Maximum acceleration during CCRb scenario.

Initial Distance [m]	**Proposed AEB Max Acc [m/s²]**		**CarMaker AEB Max Acc [m/s²]**	
	while GVT braking at -2 m/s²	**while GVT braking at -6 m/s²**	**while GVT braking at -2 m/s²**	**while GVT braking at -6 m/s²**
12	-3,73	-6,83	-1,76	-3,68
40	-2,46	-6,81	-1,84	-2,40

Source: Elaborated by the Author

Through the results, it is evident that the proposed algorithm complies strictly with the equations established in (16), (18), and (19). However, it is necessary to agree that for such cases, considering the short distance between the vehicles, the deceleration efforts are greater and the consequent comfort for the user is compromised.

In the CCRm scenario, decelerations were better managed because in all cases the initial distance between vehicles was 140 meters. Thus the system was able to seek the best compromise between its reward functions to reach the final distance more smoothly.

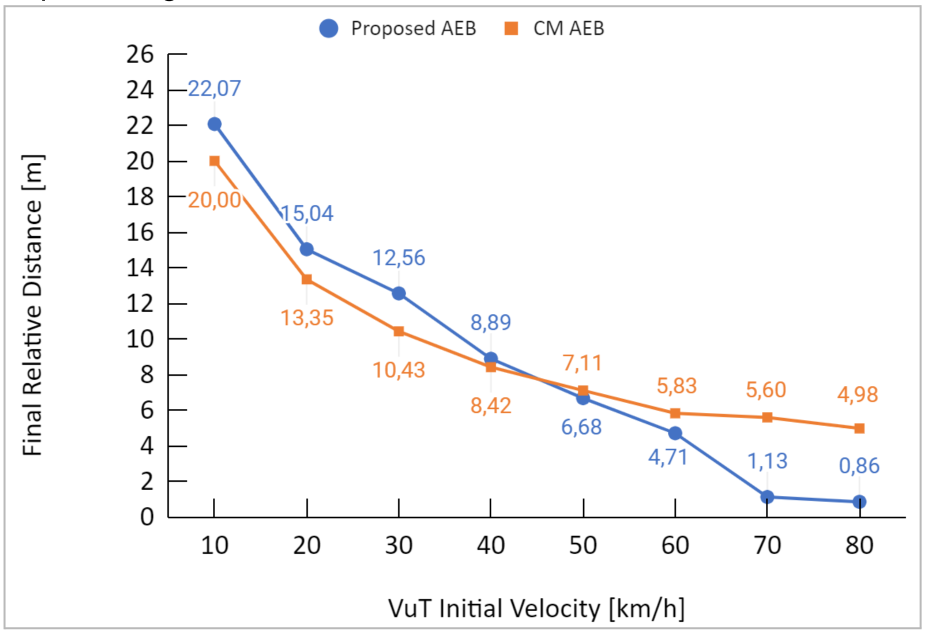
When exposed to the short distances of 40 meters and 12 meters in the CCRb case, the proposed solution required maximum braking of the vehicle to avoid the collision as learned in training. In these cases it must be agreed that the weight of non-collision was greater and dismissed the concern for occupant comfort.

4.3. Response to CCRs Scenario: Stopped Car Ahead

During the CCRs scenario run, the algorithms behaved differently between high and low speeds. In Figure 24 it is possible to see some similarity between the curves performed by both methods. However, in the case of the proposed AEB, there is no direct correlation between distances and decelerations. It seems not to be an exact interpolation between a well-established physical behavior.

.This is because during the training process, the hidden neural net assumes its own weights between nodes and generates the correlations according to the rewards it receives. And since we assume a Markov Decision Process, last state and last action will have all decisive information about the next state's development. So if the VuT needed more braking as it increased its initial speed, it doesn't mean that this increase will be propositional.

Figure 24. Final Relative Distance between VuT and the leading car, comparing proposed algorithm to CarMaker™ solution in the CCRs Scenario.



Source: Elaborated by the Author.

When operating within the range considered by EuroNCAP (2017) as an AEB City, from 10 to 50 km/h the results of the proposed AEB are very interesting, with final distances equal or even more conservative than the concurrent.

In the cases of initial speed equal to 70 and 80 km/h there is a severe difference to the validated AEB. In these cases we know that although there is no collision, the final distances are very small, close to the dimension of passenger cars.

Something can be pondered, despite the small distance, one can still consider that even if there are collisions, the impact speed is significantly reduced. Within the concept of AEB, we can also understand its function as an impact attenuator and not only as a mechanism capable of avoiding all dangerous situations with zero collisions.

To better understand how the mechanism is different from a standard controller, it is possible to analyze in the next section the behavior of the main output variables of the code within the latter scenario, the CCRs.

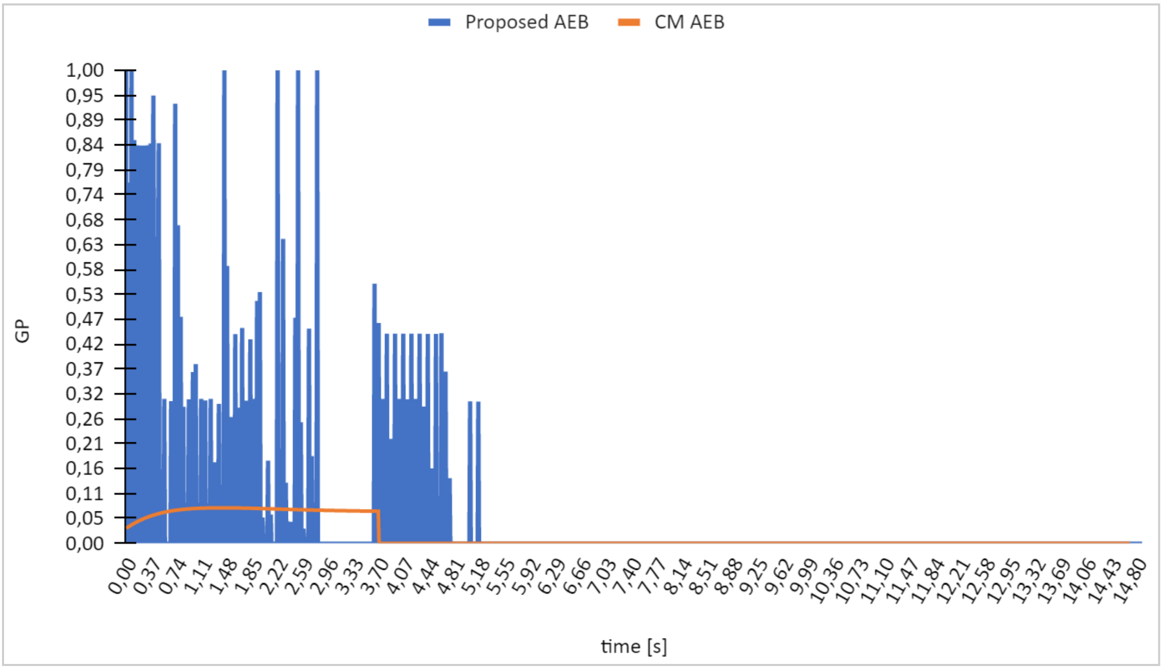
4.4. Dynamic Response to CCRs Scenario

The VuT telemetry is capable of starting interesting discussions. In the last case of the CCRs scenario, where initial velocity is 80 km/h against a steady GVT, four main variables can be discussed, GP, v_r, BP and d_r.

The Gas Pedal position (GP) was directly linked to the capability of following the target velocity of the road. In this aspect, both software actuated similarly to an ACC system, which could be an interesting feature for future development from the current code status.

In Figure 25, special attention must be given to the gain of the proposed AEB Gas Pedal signal, which is higher and very oscillating in the first seconds. It occurs because of the sampling rate, compulsorily applied to calculate next actions and states using the neural network. The same sampling limitation is also the cause of the sudden lacks in the signal, present in all four graphics shown in Annex 1.

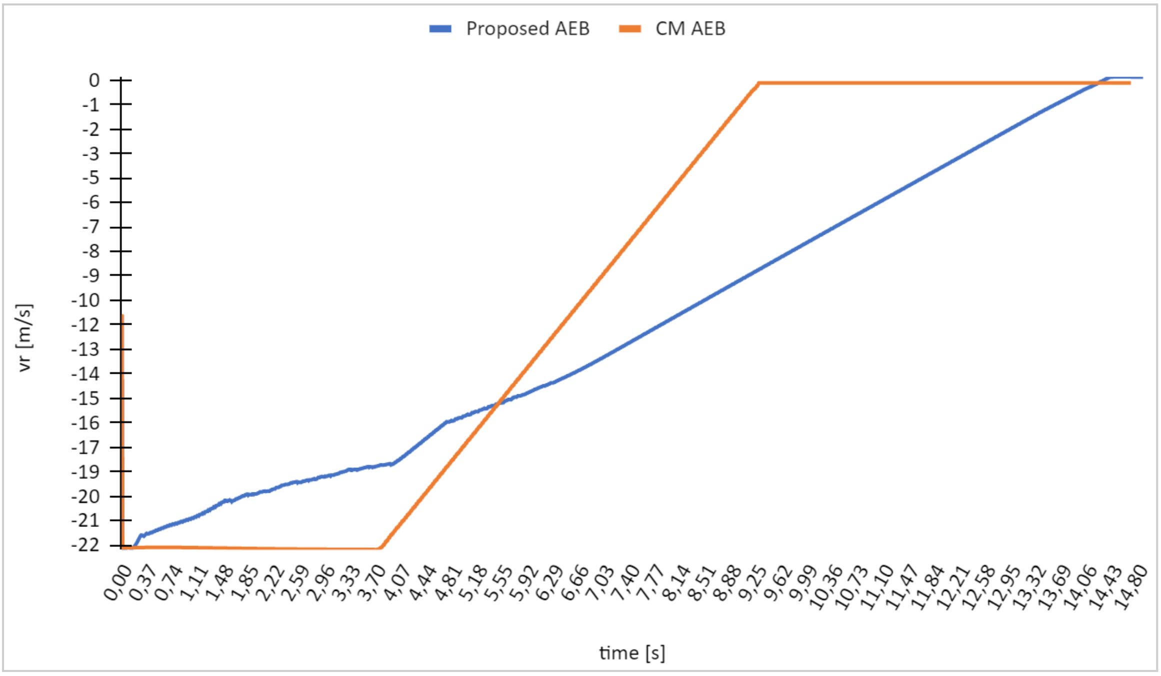
Figure 25. Gas Pedal position - Comparison between CarMaker solution and proposed algorithm.



Source: Elaborated by the Author.

The Gas Pedal value, near or equals 1.00 (100%), is an effect of the training process, amount of data for initial training and the reward method. Such results can be improved by generating larger sets of data for initial training, with more scenarios and a continuous reward function. Another influence in this behavior is the discontinuous reward function, which gives stepped values and so, It limits the modulation of the final answer during the time. Despite the presence of Gas Pedal in the first seconds of the test for both systems, the proposed system is able to initiate the deceleration earlier, since the identification of the GVT. The competitor system, otherwise, stays accelerating longer before initiating its braking, as can be seen in Figure 26.

Figure 26. Relative Velocity - Comparison between CarMaker™ solution and proposed algorithm.



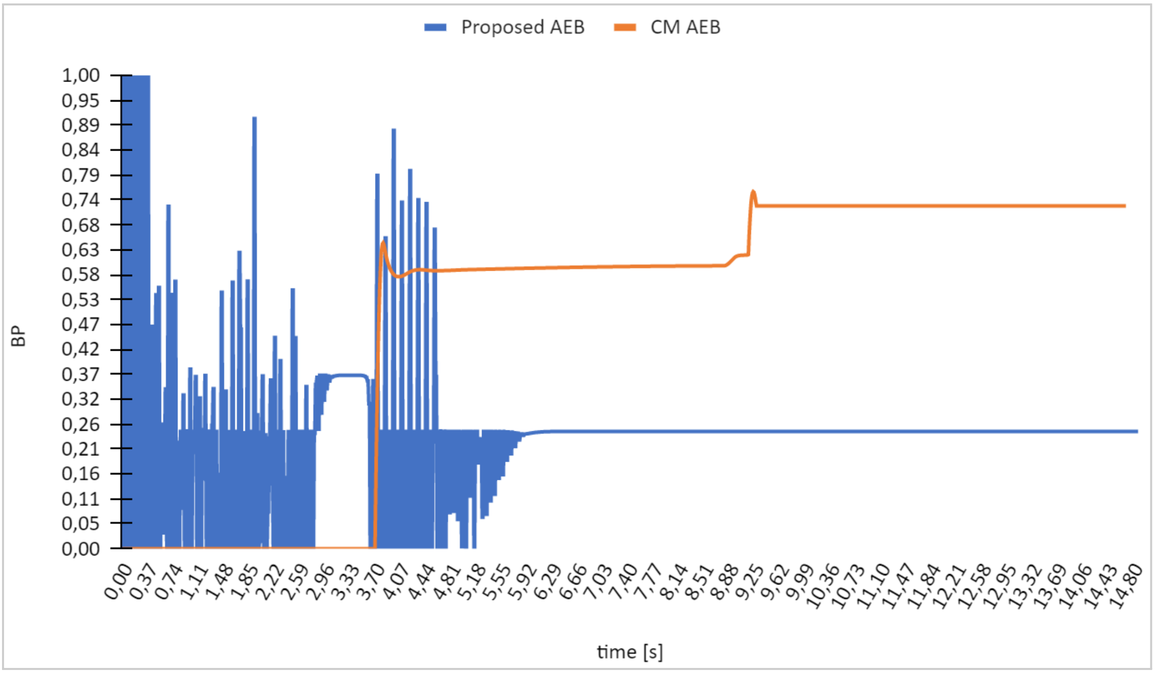
Source: Elaborated by the Author.

In Figure 26, both systems were able to locate the object at t=0s and immediately calculate the relative velocity which started in -22m/s (-80km/h) as expected. From the adopted coordinate system, it is assumed that negative v_r is an object approaching the vehicle. This way, both systems assume a hazard situation and start monitoring the object to execute the AEB maneuver.

It can be seen that the competitor system starts its deceleration late, but more sharply, reaching -3.99m/s² and this promotes convergence to the stopping earlier. Meanwhile, the proposed system starts its deceleration instantaneously at -1.83m/s², which is more comfortable, but consequently takes longer to come to a full stop.

This is due to the position of the brake pedal, which in the proposed case, although oscillating initially, enters in a lower continuous regime, around 25%, as presented in Figure 27.

Figure 27. Brake Pedal position - Comparison between CarMaker solution and proposed algorithm.



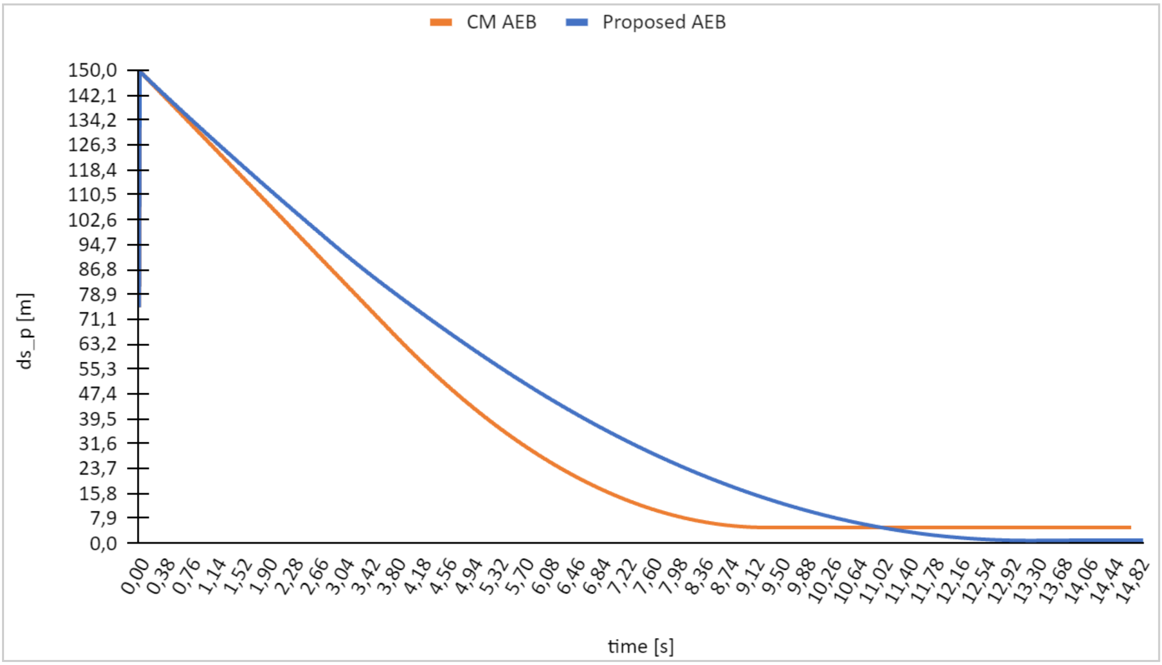
Source: Elaborated by the Author.

Even without crashing in both cases, this action is better than the one taken by the proposed algorithm because it offers to the occupants more safety and comfort.

This phenomenon is a trade-off, which needs to be better developed for the high speeds like this exposed case, CCRs 80 to 0 km/h. Despite being more abrupt, the CM AEB action is safer, as can be seen from the final distance obtained, shown in the next figure.

Figure 28 shows the relative distance between the two cars. After a long and softer braking maneuver, the proposed algorithm completely stops at 0.86 m before the forward car. The solution used for comparison makes differently, braking harder and converging for the safety distance earlier, which is 4.98 meters.

Figure 28. Relative Distance - Comparison between CarMaker™ solution and proposed algorithm.



Source: Elaborated by the Author.

Even without collision, the situation of 0.86 m for final relative distance is not a totally safe condition and that is a solid motivation to keep developing the proposed AEB algorithm, as it has a solid understanding of the risks and has taken its decisions already in the right way. The final relative distance in this case is lower than the competitor’s, which is 4.98 meters could be acceptable once there is almost the length of one car ahead.

For both cases, the EuroNCAP tests would be approved in a real testing protocol, but in the quantitative analysis it is clear the gap of the proposed algorithm in velocities above 50 km/h. Some new reward function could be proposed for these cases, maybe more centered into the safety distance and not so concerned about the occupants comfort.

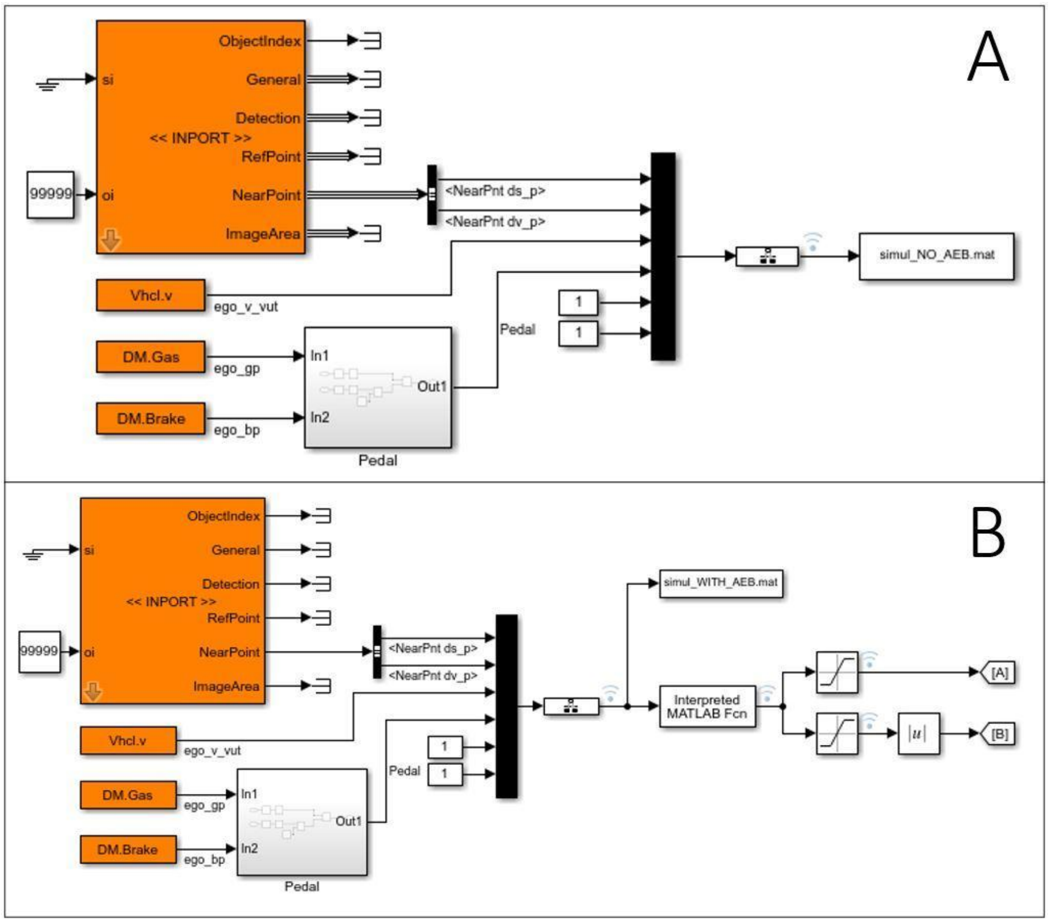
In any case, great comments can be extracted from the dynamic tests: the Simulink™ workflows were properly designed to interpret the environment and they have proven to work correctly in this matter. As already stated, the limitations are on computational resources for increasing sampling rates and the definition of a different reward function, which is already avoiding collisions but not making it using a safe final distance at the same time it makes a soft maneuver.

More about the block diagram is in the next section.

4.5. Final block diagram for the code execution

In order for the code to run synchronously with the simulated one, a MATLAB framework with Simulink was required. The final functional block diagram is presented in the part (B) of Figure 29. Inside of the block "Interpreted MATLAB Function" is the main code, executing the neural network decision in every time step to deliver the next gas and brake pedal positions. Part A of Figure 29 is the diagram used during the first training cycle, when the AEB system is disabled.

Figure 29. The function block diagram without AEB (A) and with AEB (B).



Source: Elaborated by the Author.

The algorithm was also prepared to receive more variables as states from the environment. It allows further implementations using data from two or even three lanes, combining front with rear sensors. For this purpose it can be used for example to predict lane changes. Ideas for future studies in this field are also listed in section 5.1.

5. CONCLUSION

The current work has proposed a basic AEB algorithm developed with deep RL techniques. The system's structure has two hidden layers of 38 nodes each, chosen after initial verification where more nodes and layers were not significant factors for the learning process. The algorithm is capable of performing the AEB similarly to a controller in order to avoid rear-end collisions straight ahead of a vehicle under test.

Virtual scenarios were used to train and improve the algorithm response, using a driving simulator platform. Initial data were produced for training purposes and later, new scenarios were created to validate the system. The neural network development and training cycle for reinforcement learning has been validated within the three main EuroNCAP AEB system test protocols, all simulated virtually and compared to another already validated solution.

The reward function demonstrated effectiveness once we looked for the agent and to the results it performed in each scenario. To test the approach in new scenarios, not used during training, to evaluate the dynamic response, the system performed significantly well. Importantly, the proposed system was better than its competitor in scenarios up to 50 km/h, achieving longer final distances, safer and more comfortable decelerations for the occupants.

The final solution was possible to measure velocities, accelerations and distances, resulting in the quantitative discussion in the Results chapter, which concluded the system is inside an adequate level of safety for a specific range.

For the CCRm test case, the system showed a much better deceleration curve than its competitor for all speeds performed, however the final distance was reduced significantly from 50 km/h onwards. At all events, no collisions were recorded.

During the CCRb test, where hard braking is performed at 40 meters and 12 meters, the system performed better than its competitor with very good results. In this case the comfort of the occupants was neglected, achieving accelerations of up to $-6.83 m/s^2$ to avoid collisions as trained.

In the last CCRs scenario, the system relied on a longer approach time against a stationary GVT ahead. In this case, again there is a weakness in maintaining lower decelerations at high speeds, resulting in shorter final distances when above 50 km/h. In the worst case, starting from 80 km/h, it came to a stop 0.86m behind the GVT.

In a simple qualitative analysis, by not crashing the system would definitely be approved if it performed in real tests the same results obtained in the simulations. However, in a deeper quantitative analysis, it can be seen that there are still opportunities to improve the reward function and the training method for interurban velocities.

5.1. Potential Future Development

As the results show, especially in CCRm and CCRs scenarios, since the code provides better results in urban situations, between 10 and 50 km/h, and worse results for the inter-city range, between 50 and 80 km/h, the most important next step should be to improve the reward functions and propose specific scenarios to train the code for higher velocities.

Changing the quality function, also known as the reward function, is one way to modify the Deep Reinforcement Learning algorithm. However, it is not the only way to do so, and sometimes changing the reward function alone may not be enough to achieve the desired outcome.

Deep RL algorithms consist of several components as discussed, including the neural network architecture, the learning algorithm, the exploration strategy, and the reward function. Changing any of these components can alter the behavior of the algorithm.

For example, changing the neural network architecture can affect the agent's ability to learn complex tasks, while modifying the learning algorithm can impact how the agent updates its policy over time. Additionally, modifying the exploration strategy can change how the agent explores its environment, which can be crucial in certain scenarios.

In this sense it is important to understand within neural networks how to put together pre-acquired knowledge for a certain operating range up to 50km/h with a new code for responses above that. The composite of different codes for different speed ranges should lead to an even more robust system, resulting from the optimization of neural networks for each different type of situation.

The code in its current stage, is already capable of receiving data from the environment and to train its deep neural networks offline, in order to find the best actions for gas and brake pedal positions. Another potential next step would be to close the loop in the framework and perform some online acquisitions, training to improve the network perception.

Here there are some interesting paths to follow in order to add this algorithm some discussion, including some applications with simulators as DYNA4™ or CARLA™, using integrating softwares as ROS for example.

A. To implement the final network in DYNA4™, or any other physical simulator for experimental use with human control before any real implementation;

B. To run controller-based AEB solutions, different from the simulators plug-ins, in the same environment and generate a benchmark to evaluate once more the proposed algorithm;

C. To close the loop using a real car, plugged into the virtual scenario to evaluate the performance of the system in a real implementation;

REFERENCES

Abbeel, P. Coates, A. and Ng, A., 2010 *Autonomous Helicopter Aerobatics through Apprenticeship Learning*, The International Journal of Robotics Research, vol. 29, no. 13, pp. 1608-1639, 2010. Available: 10.1177/0278364910371999.

Amaral, J. Göllinger, H. and Fiorentin, T. *Improvement of Vehicle Stability Using Reinforcement Learning*, XV Encontro Nacional de Inteligência Artificial e Computacional (ENIAC 2018), 2018. Available: 10.5753/eniac.2018.4420 [Accessed 31 August 2020].

Bella F., 2009. *Can the driving simulators contribute to solving the critical issues in geometric design?*. In Transportation Research Record: Journal of the Transportation Research Board, No 2138, TRB, Washington, D.C., 120-126.

Bella, F. and Russo, R, 2011. *A Collision Warning System for rear-end collision: a driving simulator study*, Procedia - Social and Behavioral Sciences, vol. 20, pp. 676-686, 2011. Available: 10.1016/j.sbspro.2011.08.075.

Bloomberg Group. Bloomberg *Philanthropies Selects Ten Cities and Five Countries to Participate in New Phase of the Global Road Safety Initiative*.[Online] Available: https://www.bloomberg.org/press/releases/bloomberg-philanthropies-selects-ten-cities-five-countries-participate-new-phase-global-road-safety-initiative/ [Accessed on: Apr. 27, 2020]

Broughton K. Switzer, F. Scott, D., 2007. *Car following decisions under three visibility conditions and two speeds tested with a driving simulator*. Accident Analysis and Prevention 39, 106–116.

Bussgeldkatalog.org, 2020. *Abstand und Abstandsvergehen - Bußgeldkatalog 2020,* [Online]. Available: https://www.bussgeldkatalog.org/abstand/. [Accessed: 24- May-2020].

Chen, S. 2018. *Comparing Deep Reinforcement Learning Methods for Engineering Applications*. 2018. 140 p. Thesis (Master) - Faculty of Computer Science, Otto-von-Guericke-Universität Magdeburg, 2018.

Cheng B., Masahiro H., Takamasa S., 2002. *Analysis of driver response to collision warning during car following*. Society of Automotive Engineers of Japan (JSAE) Review 23, 231–237.

Cicchino, J., 2017. *Effectiveness of forward collision warning and autonomous emergency braking systems in reducing front-to-rear crash rates*, Accident Analysis & Prevention, vol. 99, pp. 142-152, 2017. Available: 10.1016/j.aap.2016.11.009.

Drechsler, M. F., 2019. *Anti-Slip Control with an Actor-Critic Reinforcement Learning Algorithm.* Master Thesis, Technische Hochschule Ingolstadt.

Duan J. et al., 2017. *Driver braking behavior analysis to improve autonomous emergency braking systems in typical Chinese vehicle-bicycle conflicts*, Accident

Analysis & Prevention, vol. 108, pp. 74-82, 2017. Available: 10.1016/j.aap.2017.08.022.

El-Fakdi, A., & Carreras, M., 2008. *Policy gradient based reinforcement learning for real autonomous underwater cable tracking.* In International conference on intelligent robots and systems, 2008. IROS 2008. IEEE/RSJ (pp. 3635–3640).

EuroNCAP. 2017. *TEST PROTOCOL – AEB systems.* Available at: <https://cdn.euroncap.com/media/26996/euro-ncap-aeb-c2c-test-protocol-v20.pdf> [Accessed 23 February 2022]

EuroNCAP. 2020. *Euro NCAP Presents Latest Overhaul Of Its Safety Rating*. [online] Available at: <https://www.euroncap.com/en/press-media/press-releases/euro-ncap-presents-latest-overhaul-of-its-safety-rating/#> [Accessed 6 September 2020].

European Commission, 2011. *Roadmap to a single European transport area - towards a competitive and resource efficient transport system.* Brussels: White Paper, March 28, 2011.

Farah H., Bekhor S., Polus A., 2009. *Risk evaluation by modeling of passing behavior on two-lane rural highways*. In proceedings 88th Annual Meeting Transportation Research Board. Washington D.C.

Fildes B. et al., 2015. *Effectiveness of low speed autonomous emergency braking in real-world rear-end crashes*, Accident Analysis & Prevention, vol. 81, pp. 24-29, 2015. Available: 10.1016/j.aap.2015.03.029.

François-Lavet, V. Henderson, P. Islam, R. Bellemare M. and Pineau, J. *An Introduction to Deep Reinforcement Learning*, Foundations and Trends in Machine Learning, vol. 11, no. 3-4, pp. 219-354, 2018. Available: 10.1561/2200000071.

Genya A., Richardson A., 2005. *The influence of alarm timing on braking response and driver trust in low speed driving*. Safety Science 43, 639–654.

Hayward, J. C., 1972. *Near miss determination through use of a scale of danger*, Highway Research Record, vol. 384, pp. 24–34, 1972. - Open Access Library", Oalib.com, 2020. [Online]. Available: http://www.oalib.com/references/13765983. [Accessed: 24- May- 2020].

He, R. and Zhang, D. *Research on AEB Collision Avoidance Strategy Based on Characteristics of Driver-Vehicle-Road*, SAE Technical Paper Series, 2020. Available: 10.4271/2020-01-1213 [Accessed 4 September 2020].

Honda Worldwide. 2020. | *World News* | *News Releases* | *May 20, 2003*. [online] Available at: <https://web.archive.org/web/20141229235818/http://world.honda.com/news/2003/4030520.html> [Accessed 5 September 2020].

Howard R. A. *Dynamic programming and Markov processes*. Wiley, 1966.

Hulshof, W. et al., *Autonomous Emergency Braking Test Results,* 23rd International Technical Conference on the Enhanced Safety of Vehicles (ESV), Paper Number 13-0168, 2013

Ioffe, S. and Szegedy, C., 2015. Batch normalization: *Accelerating deep network training by reducing internal covariate shift*. Lille, Proceedings of 32nd International Conference on Machine Learning.

Jamson A. Lai, F. Carsten O., 2008. *Potential benefit of an adaptive forward collision warning system*. Transportation research. Part C, Emerging Technologies, Vol.16c, No.4, 471-484.

Jenkins J. M., Rilett L. R., 2004. *Application of distributed traffic simulation for passing behavior study*. In Transportation Research Record: Journal of the Transportation Research Board, 1899, TRB, Washington, 11-18.

Jirgl, M. Fiedler, P. Bradac. Z., 2019. *Using Matlab-based Driving Simulator for Human Factor Assessment*. IFAC PapersOnLine 52-27 (2019) 27–32.

Kim, P., 2017. *MATLAB Deep Learning*. 1st ed. Seoul: Apress

Kovaceva, J. Bálint, A. Schindler, R. and Schneider, A., 2020. *Safety benefit assessment of autonomous emergency braking and steering systems for the protection of cyclists and pedestrians based on a combination of computer simulation and real-world test results*, Accident Analysis & Prevention, vol. 136, p. 105352, 2020. Available: 10.1016/j.aap.2019.105352.

Li, J. Yao, L. Xu, X. Cheng, B. Ren, J., 2020. *Deep Reinforcement Learning for Pedestrian Collision Avoidance and Human-Machine Cooperative Driving*. Information Sciences 00 (2020) 1–17.

Liu, D. Javaherian, H. Kovalenko, O. and Huang, T., 2008 *Adaptive Critic Learning Techniques for Engine Torque and Air–Fuel Ratio Control*, IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), vol. 38, no. 4, pp. 988-993, 2008. Available: 10.1109/tsmcb.2008.922019.

Lu, K. and Chen, Y., 2019. *ISO 26262 ASIL-Oriented Hardware Design Framework for Safety-Critical Automotive Systems*. 2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE),.

Lundgren J. and Tapani A., 2006. *Evaluation of Safety Effects of Driver Assistance Systems through Traffic Simulation*, Transportation Research Record: Journal of the Transportation Research Board, vol. 1953, no. 1, pp. 81-88, 2006. Available: 10.1177/0361198106195300110.

Minderhoud, M. and Bovy, P., 2001. *Extended time-to-collision measures for road traffic safety assessment*, Accident Analysis & Prevention, vol. 33, no. 1, pp. 89-97, 2001. Available: 10.1016/s0001-4575(00)00019-1.

Ministério da Saúde. *SIM: Sistema de Informação sobre Mortalidade: Dados de 2018.* [Online]. Available: http://sim.saude.gov.br/ [Accessed on: Apr. 24, 2020]

Mirchevska, B. Pek, C. Werling, M. Althoff, M. Boedecker, J*, 2018. High-level Decision Making for Safe and Reasonable Autonomous Lane Changing using Reinforcement Learning*. 2018 21st International Conference on Intelligent Transportation Systems (ITSC) Maui, Hawaii, USA, November 4-7, 2018.

Nandy, A. & Biswas, M., 2018. *Reinforcement Learning.* 1st ed. New York: Apress.

National Health and Transportation Safety Administration. *FARS data tables*. [Online] Available: https://www-fars.nhtsa.dot.gov/Main/index.aspx [Accessed on: Apr. 24, 2020].

Paluszek, M. & Stephanie, T., 2017. MATLAB Machine Learning. 1st ed. New Jersey: Apress.

Peters, J., & Schaal, S., 2006. *Policy gradient methods for robotics.* In Proceedings of the IEEE international conference on intelligent robotics systems, 2006.

Riedmiller, M., Montemerlo, M., & Dahlkamp, H., 2007*. Learning to drive in 20 minutes*. In Proceedings of the FBIT 2007 conference, Jeju, Korea. Berlin: Springer. Best Paper Award.

SAE International, 2020*. J3016B: Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles - SAE International*, Sae.org, 2020. [Online]. Available: https://www.sae.org/standards/content/j3016_201806/. [Accessed: 24- May- 2020].

Sander, U. and Lubbe, N., 2018. *Market penetration of intersection AEB: Characterizing avoided and residual straight crossing path accidents*, Accident Analysis & Prevention, vol. 115, pp. 178-188, 2018. Available: 10.1016/j.aap.2018.03.025.

Shalev-Shwartz, S. and Shashua, A., 2020. *On the Sample Complexity of End-to-end Training vs. Semantic Abstraction Training*, arXiv.org, 2020. [Online]. Available: https://arxiv.org/abs/1604.06915. [Accessed: 24- May- 2020].

Sini, J. and Violante, M. *A simulation-based methodology for aiding advanced driver assistance systems hazard analysis and risk assessment*, Microelectronics Reliability, vol. 109, p. 113661, 2020. Available: 10.1016/j.microrel.2020.113661.

Spicer, R. Vahabaghaie, A. Bahouth, G. Drees, L. Martinez von Bülow, R and Baur, P. *Field effectiveness evaluation of advanced driver assistance systems*, Traffic Injury Prevention, vol. 19, no. 2, pp. S91-S95, 2018. Available: 10.1080/15389588.2018.1527030.

Sutton, R. S. & Barto, A. G., 2017. *Reinforcement learning: an introduction.* 1st ed. Cambridge: The MIT Press.

Sutton, R. S., McAllester, D., Singh, S. & Mansour , Y., 1999. Policy gradient methods for Reinforcement Learning with Function Approximation. *Advances in Neural Information Processing Systems,* Volume 12, pp. 1057-1063.

The MathWorks, Inc., 2016*. Introducing Machine Learning*. Mathworks Inc.: n.92991v00, 2016.

The MathWorks, Inc., 2018. *Introducing Deep Learning with Matlab*. Mathworks Inc.: n.80789v00, 2018.

The MathWorks, Inc., 2020. *Autonomous Emergency Braking With Sensor Fusion-MATLAB & Simulink*. [online] Mathworks.com. Available at: <https://www.mathworks.com/help/driving/examples/autonomous-emergency-braking-with-sensor-fusion.html> [Accessed 6 September 2020].

UK Department for Transport, 2019. *Reported road casualties in Great Britain: 2018 annual report.* UK: Department for Transport, Sep. 26, 2019.

UN Road Safety Collaboration. *Decade of Action for Road Safety 2011-2020 seeks to save millions of lives*. [Online] Available: https://www.who.int/roadsafety/decade_of_action/en/ [Accessed on: Apr. 27, 2020].

Wang, P. Chan, C. La Fortell, A. de, 2018. *A Reinforcement Learning Based Approach for Automated Lane*. 2018 IEEE Intelligent Vehicles Symposium (IV) Changshu, Suzhou, China, June 26-30, 2018.

Weber, Y. & Kanarachos, S. 2019. *The Correlation between Vehicle Vertical Dynamics and Deep Learning-Based Visual Target State Estimation*: A Sensitivity Study. Sensors. 19. 4870. 10.3390/s19224870.

Wiering, M. & van Otterlo, M., 2012. *Reinforcement Learning State-of-the-Art.* 1st ed. Berlin: Springer.

Winner, H. Hakuli, S. Lotz, F and Singer C., 2020. *Handbook of Driver Assistance Systems - Basic Information, Components and Systems for Active Safety and Comfort*, Springer.com, 2020. [Online]. Available: https://www.springer.com/gp/book/9783319123516. [Accessed: 24- May- 2020].

World Health Organization, 2011, 25p. *Global Plan for the Decade of Action for Road Safety 2011-2020*. Geneva: World Health Organization.

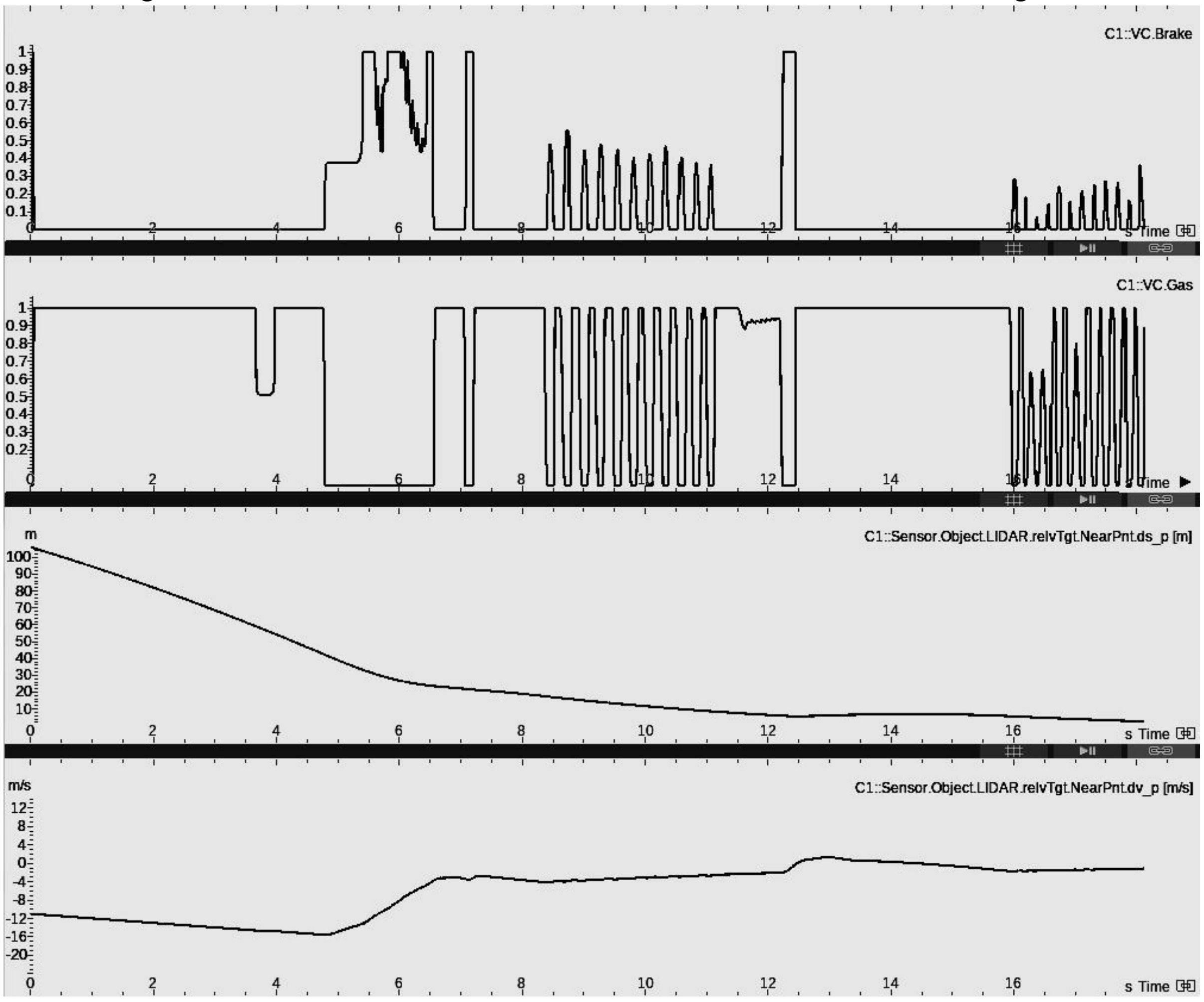
World Health Organization, 2019. *Global Status Report On Road Safety 2018*. Geneva: World Health Organization.

Yoffie, D. B, 2014. *Mobileye: The Future of Driverless Cars*. Harvard Business School Case 715-421, October 2014.

You, C. Lu, J. Filev, D. Tsiotras, P., 2019. *Advanced planning for autonomous vehicles using reinforcement learning and deep inverse reinforcement learning*. Robotics and Autonomous Systems 114 (2019) 1–18

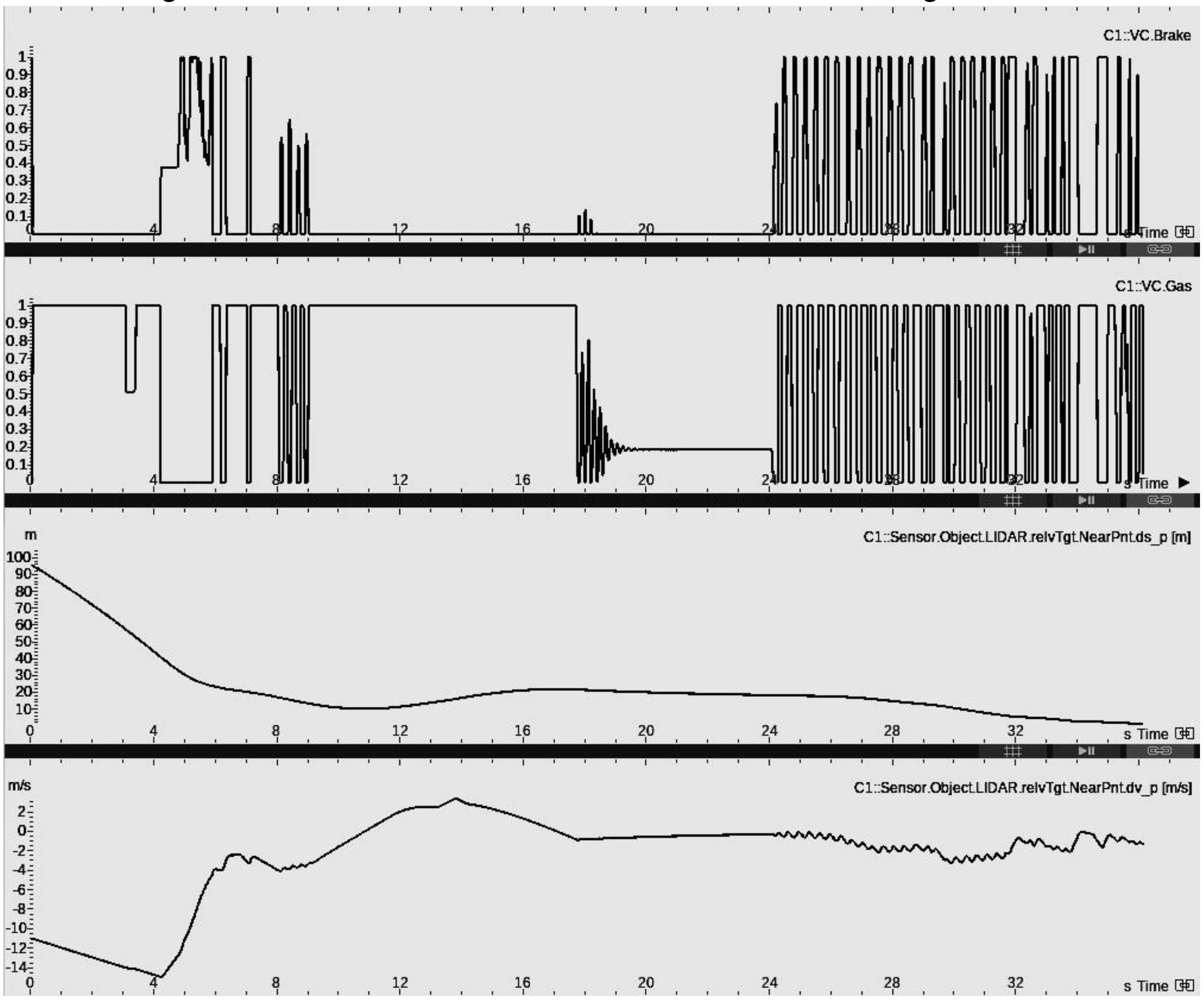
ANNEX 1 – Testing scenarios data.

Figure 30. Data extracted from Scenario CCRm – Slower leading car.



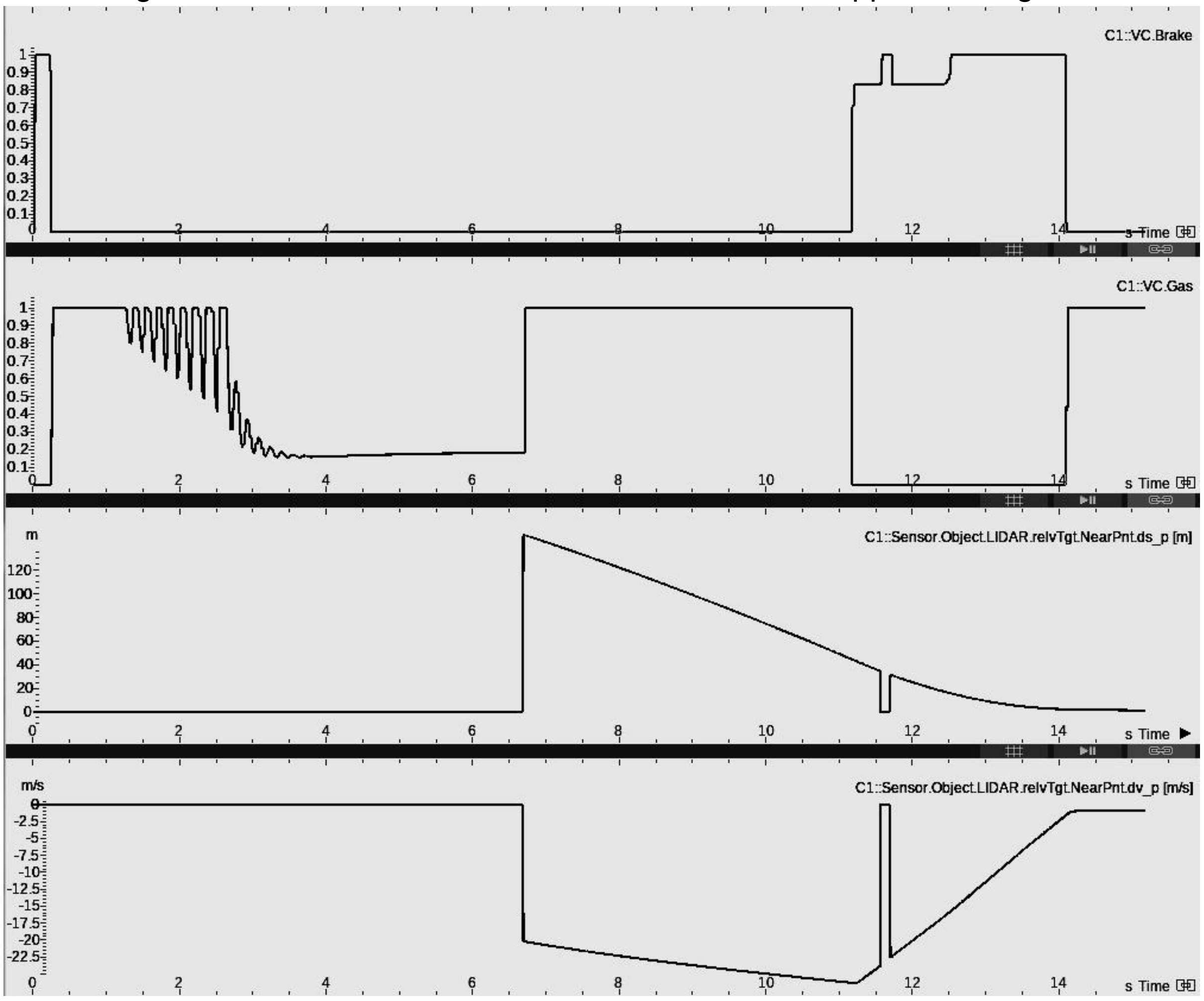
Source: Elaborated by the Author.

Figure 31. Data extracted from Scenario CCRb – Braking lead car.



Source: Elaborated by the Author.

Figure 32. Data extracted from Scenario CCRs – Stopped leading car.



Source: Elaborated by the Author.

ANNEX 2 – Structure of the Matlab code.

Below, there is the main structure of the code, which was written in Matlab. The connection with IPG CarMaker™ was made by Simulink™ block diagram as shown in Figure 29 inside Results.

```
clear; close all; clc;

TVL=22;               % Desired Velocity (m/s)
safezone=2;
stnu = 13;            % number of states
tau = 1;              % Target update
gamma = 0.9;          % discount factor
minipcent= 0.5;       % Minibatch size

% Convengence parameter
hmin = 20; %number of training cycles
acte = 0.001; % Max squared error between to consecutives Q evaluations
acterror = acte;
hmax = 50;
hcount=0;

%% Data treatment
[DATA,leng]=Simulation_treatment(safezone,TVL);

%% Main Algorithm
crtb=zero_critic(stnu+1);
actb=zero_actor(stnu);

fprintf('--->->Minibatch done!\n')
fprintf('...Calculating Actor\n...Calculating Critic\n...Finding Best actions\n')

Minibatch = Select_data(DATA,stnu,actb,crtb,gamma,minipcent);
crta = create_critic(Minibatch,stnu+1);
fprintf('--->->Critic Network Creation done!\n')
acta = create_actor(Minibatch,crta);
fprintf('--->->Actor Network Creation done!\n')

save('act.mat','acta');

actb = target_update(actb,acta,tau);
crtb = target_update(crtb,crta,tau);

if stnu == 13
   actbb =
actb([DATA(:,2)';DATA(:,3)';DATA(:,4)';DATA(:,5)';DATA(:,6)';DATA(:,7)';DATA(:,
8)';DATA(:,9)';DATA(:,10)';DATA(:,11)';DATA(:,12)';DATA(:,13)';DATA(:,14)']);
   Qb =
crtb([actbb;DATA(:,2)';DATA(:,3)';DATA(:,4)';DATA(:,5)';DATA(:,6)';DATA(:,7)';D
ATA(:,8)';DATA(:,9)';DATA(:,10)';DATA(:,11)';DATA(:,12)';DATA(:,13)';DATA(:,14
)']);
   actaa =
acta([DATA(:,2)';DATA(:,3)';DATA(:,4)';DATA(:,5)';DATA(:,6)';DATA(:,7)';DATA(:,
8)';DATA(:,9)';DATA(:,10)';DATA(:,11)';DATA(:,12)';DATA(:,13)';DATA(:,14)']);
   Qa =
crta([actaa;DATA(:,2)';DATA(:,3)';DATA(:,4)';DATA(:,5)';DATA(:,6)';DATA(:,7)';D
ATA(:,8)';DATA(:,9)';DATA(:,10)';DATA(:,11)';DATA(:,12)';DATA(:,13)';DATA(:,14
)']);
else
   fprintf('some value has been inputed wrongly');
end

error = gsubtract(actaa,actbb); % Calculate difference between Q
[s,n] = sumsqr(error);
acterror(hcount+1)=s/n;
hv(hcount+1)=hcount;  % Create a vector of the h times that the cicle are tried
hcount=hcount+1;  %Increase the counter
plot_convergence (Qa,Qb,actaa,actbb,hv,acterror);

while and(or(acterror(hcount)>=acte, hcount<=hmin),hcount<=hmax)

   Minibatch = Select_data(DATA,stnu,actb,crtb,gamma,minipcent); %%
Select the data and calculates the Q value
   crta = train_critic(Minibatch,crta);
   acta = train_actor(Minibatch,crta,acta);
   actb = target_update(actb,acta,tau);
   crtb = target_update(crtb,crta,tau);

   if stnu == 13
      actbb =
actb([DATA(:,2)';DATA(:,3)';DATA(:,4)';DATA(:,5)';DATA(:,6)';DATA(:,7)';DATA(:,
8)';DATA(:,9)';DATA(:,10)';DATA(:,11)';DATA(:,12)';DATA(:,13)';DATA(:,14)']);
      Qb =
crtb([actbb;DATA(:,2)';DATA(:,3)';DATA(:,4)';DATA(:,5)';DATA(:,6)';DATA(:,7)';D
ATA(:,8)';DATA(:,9)';DATA(:,10)';DATA(:,11)';DATA(:,12)';DATA(:,13)';DATA(:,14
)']);
      actaa =
acta([DATA(:,2)';DATA(:,3)';DATA(:,4)';DATA(:,5)';DATA(:,6)';DATA(:,7)';DATA(:,
8)';DATA(:,9)';DATA(:,10)';DATA(:,11)';DATA(:,12)';DATA(:,13)';DATA(:,14)']);
      Qa =
crta([actaa;DATA(:,2)';DATA(:,3)';DATA(:,4)';DATA(:,5)';DATA(:,6)';DATA(:,7)';D
ATA(:,8)';DATA(:,9)';DATA(:,10)';DATA(:,11)';DATA(:,12)';DATA(:,13)';DATA(:,14
)']);
   else
      fprintf('some value has been inputed wrongly');
   end

   error = gsubtract(actaa,actbb); % Calculate difference between Q
   [s,n] = sumsqr(error);
   acterror(hcount+1)=s/n;
   hv(hcount+1)=hcount;  % Create a vector of the h times that the cicle are
tried
   hcount=hcount+1;  %Increase the counter
   plot_convergence (Qa,Qb,actaa,actbb,hv,acterror);

end

fprintf('\nAction converged \n')

save ('act.mat','actb');
%count_reward = sum((DATA(end-(2*leng-1):end,7))<=0.1);
save('act_count.mat','actb');
%mean_reward=mean(DATA(end-(2*leng-1):end,7));
save ('act_mean.mat','actb');
%reward_error = 0;

function net = zero_actor(n)
net=feedforwardnet([38 38]);  % Create the network with 2 hidden layers
net.numInputs = n;   % Define the number of the inputs
{dsx_ol...dvx_ol,GP,brake} 14
for i=2:n
   net.inputConnect(1,i) = true;
net.trainParam.showWindow = 0;
net.trainParam.epochs = 1;
for i=1:n
   in{i,1}=randn(1,3);
end
out = randn(1,3);
net.inputs{1}.processFcns = {};
net.outputs{2}.processFcns = {};
net.outputs{3}.processFcns = {};
fprintf('...Actor weights are reseted\n')
net = train(net,in,out);
m = length(getwb(net));
net = setwb(net,zeros(m,1));
end

function net = create_actor(Minibatch,crt)

actst = cell2mat(Minibatch{1,1});
[~,n] = size(actst);
in = (crt.numInputs - 1);
Q=zeros(27,n);
action=zeros(1,n);
max1=0;
max2=42;
zeroone = linspace(max1,max2,27);
acttest = ones(27,n);
for k = 1:27
   acttest(k,:) = zeroone(k).*ones(1,n);
end
for j=1:2
   for k = 1:27
      if in == 13
```

```
Q(k,:)=crt([acttest(k,:);actst(2,:);actst(3,:);actst(4,:);actst(5,:);actst(6,:);actst(7,:)
;actst(8,:);actst(9,:);actst(10,:);actst(11,:);actst(12,:);actst(13,:);actst(14,:)]);
        end
    end
    for k = 1:n
        [~, ind1(k)] = max(Q(:,k));
        Q(ind1,k) = -Inf;
        [~, ind2(k)] = max(Q(:,k));
        max1(k) = acttest(ind1(k),k);
        max2(k) = acttest(ind2(k),k);
        acttest(:,k)= linspace(max1(k),max2(k),27)';
    end
end

net=feedforwardnet([38 38]);
net.numInputs = in; % Define the number of the inputs {ax;i;GP;vx;vw}
for i=2:in
    net.inputConnect(1,i) = true;
end
net.inputs{1}.processFcns = {};
net.outputs{2}.processFcns = {};
net.outputs{3}.processFcns = {};
net.trainParam.max_fail = 20;         %Maximum validation failures
net.trainParam.min_grad = 1e-7;      %Minimum performance gradient
net.trainParam.mu_max = 1e10;       %Maximum mu
net.trainParam.showWindow = 0;
fprintf('...(Create_Actor) Actor training\n')
net = train(net,Minibatch{3,1},max1);
end

function act = train_actor(Minibatch,crt,act)

actst = cell2mat(Minibatch{1,1});
n = length(actst);
in = (crt.numInputs - 1);
Q=zeros(27,n);
action=zeros(1,n);
max1=0;
max2=42;
zeroone = linspace(max1,max2,27);
acttest = ones(27,n);
for k = 1:27
    acttest(k,:) = zeroone(k).*ones(1,n);
end

for j=1:2
    for k = 1:27 % find the Q value for all of the input actions
        if in == 13 %(6 distaces, 6 velocities, vv, gp and break)

Q(k,:)=crt([acttest(k,:);actst(2,:);actst(3,:);actst(4,:);actst(5,:);actst(6,:);actst(7,:)
;actst(8,:);actst(9,:);actst(10,:);actst(11,:);actst(12,:);actst(13,:);actst(14,:)]);
        end
    end
    for k = 1:n
        [~, ind1(k)] = max(Q(:,k));
        Q(ind1,k) = -Inf;
        [~, ind2(k)] = max(Q(:,k));
        max1(k) = acttest(ind1(k),k);
        max2(k) = acttest(ind2(k),k);
        acttest(:,k)= linspace(max1(k),max2(k),27)';
    end
end

fprintf('...(Train_Actor) Actor training\n')
act= train(act,Minibatch{3,1},max1);
end

function net = zero_critic(n)

net=feedforwardnet([38 38]);
net.numInputs = n;
for i=2:n
    net.inputConnect(1,i) = true;
end
net.trainParam.showWindow = 0;
net.trainParam.epochs = 1;
for i=1:n
    in{i,1}=randn(1,3);
end
out = randn(1,3);
net.inputs{1}.processFcns = {};
net.outputs{2}.processFcns = {};
net.outputs{3}.processFcns = {};
fprintf('...Critic weights are reseted\n')
net = train(net,in,out);
positions of the weights
m = length(getwb(net));
net = setwb(net,zeros(m,1));
End

function net = create_critic(Minibatch,in)

net=feedforwardnet([38 38]);
net.numInputs = in;
for i=2:in
    net.inputConnect(1,i) = true;
end
net.inputs{1}.processFcns = {};
net.outputs{2}.processFcns = {};
net.outputs{3}.processFcns = {};
net.trainFcn='trainlm';

net.trainParam.max_fail = 20;          %Maximum validation failures
net.trainParam.min_grad = 1e-7;       %Minimum performance gradient
net.trainParam.mu_max = 1e10;        %Maximum mu
net.trainParam.showWindow = 0;

%        Minibatch Array
%|   {1,1}   | {2,1} | {3,1} |
%|actionstate| Qvalue| state |

net = train(net,Minibatch{1,1},Minibatch{2,1});
end
function crt = train_critic(Minibatch,crt)
crt = train(crt,Minibatch{1,1},Minibatch{2,1});
end

%% Root Functions

function Minibatch = Select_data(DATA,netin,act,crt,gamma,minicent)

[m,~]=size(DATA);          % take the DATA size
minisize=round(m*minicent); % Minibatch size = minipcent % of the total
DATA size
idx = randperm(m,minisize); % create a random position vector
rawbatch = DATA(idx,:);     % Take the desired data from DATA to Minibatch
rawbatch = rawbatch';

% Create training data structure with vehicle speed
if netin==13 %States cosidered (6 distaces, 6 velocities, vv, gp and break)

state={rawbatch(2,:);rawbatch(3,:);rawbatch(4,:);rawbatch(5,:);rawbatch(6,:);ra
wbatch(7,:);rawbatch(8,:);rawbatch(9,:);rawbatch(10,:);rawbatch(11,:);rawbatc
h(12,:);rawbatch(13,:);rawbatch(14,:)};

nextstate={rawbatch(17,:);rawbatch(18,:);rawbatch(19,:);rawbatch(20,:);rawbat
ch(21,:);rawbatch(22,:);rawbatch(23,:);rawbatch(24,:);rawbatch(25,:);rawbatch(
26,:);rawbatch(27,:);rawbatch(28,:);rawbatch(29,:)};

actionstate={rawbatch(1,:);rawbatch(2,:);rawbatch(3,:);rawbatch(4,:);rawbatch(
5,:);rawbatch(6,:);rawbatch(7,:);rawbatch(8,:);rawbatch(9,:);rawbatch(10,:);raw
batch(11,:);rawbatch(12,:);rawbatch(13,:);rawbatch(14,:)};
    Qvalue= rawbatch(15,:);
else
    fprintf('\n### Wrong input value ###');
    brake
end
Minibatch={actionstate;Qvalue;state};
end

function [DATA,leng]=Simulation_treatment(safezone,TVL)

load('simul_inicial_treated.mat');
rawdata=stateaction.Data;

MaxDsx   = 0;  %Radar range (m)  ##150
MaxDvx   = 0; %Max. velocity (mph) ##50

[sizei,sizej]=size(rawdata);

%Normalizes the values between 0 and 1 to improve the learning performance
for j=1:sizej
    if and(j>=2,j<=7)%Columns of distance values
        MaxDsx=max(abs(rawdata(:,j))); %Save maximum range(m)
        if (MaxDsx==0)
            rawdata(:,j) = 0;
        else
            rawdata(:,j) = rawdata(:,j)./MaxDsx;
        end
    elseif and(j>=8,j<=14)%Columns of velocity values
        MaxDvx=max(abs(rawdata(:,j))); %Save maximum velocity(mph)
        if (MaxDvx==0)
            rawdata(:,j) = 0;
        else
            rawdata(:,j) = rawdata(:,j)./MaxDvx;
        end
    end
end

leng = length(rawdata(:,1))-1;

rawdata=calculate_reward(rawdata,safezone,TVL);

% Create the export data structure
DATA=rawdata(1:leng,[1:14 28 29 15:27]);
save ('DATA.mat','DATA');
fprintf('DATA Updated. First Rewards Assigned!\n')
end

function rawdata   = calculate_reward(rawdata,safezone,TVL)

leng = length(rawdata(:,1))-1;

%Copy the next states to the same line that the last state
for i=1:leng
    for j=1:13
        if (i==leng)
            rawdata(i,(j+14))=rawdata(i,(j+1));
        else
            rawdata(i,(j+14))=rawdata(i+1,(j+1));
        end
    end
end
rawdata(:,28)=ones(); %future rewards (28)-->>(15)
rawdata(:,29)=ones(); %future TTC     (29)-->>(16)
[sizei,sizej]=size(rawdata);

%% TTC-based Reward ('Time To Collision' against leading car) and
TTS-Based Reward ('Time To Stop' VUT)
RewardTTC=0;
RewardDelT=0;
for i=1:leng % Calculate TTC at the next step and save in rawdata(:,33)
    TTC=time_to_collision(rawdata(i,15),rawdata(i,21),safezone); %Generate
TTC for lane O
    TTS=(0.323.*rawdata(i,27)+1.999); %Time for total stop %Fitted for
currently car
    DelT=TTC./TTS; %Time to collision compared to time for total stop
    if (TTC>0) %Positive Time to Collision
        if and(rawdata(i,15)>safezone,rawdata(i,21)>0)      %No collision/Leading
vehicle moving away//Best situation
            RewardTTC=0.5;
        elseif and(rawdata(i,15)<safezone,rawdata(i,21)<0)  %Collision/Leading
vehicle approaching//Worst situation
            RewardTTC=0;
        end
    elseif (TTC<=0) %Negative Time to Collision
        if and(rawdata(i,15)>safezone,rawdata(i,21)<0)      %No collision/Leading
vehicle approaching
            RewardTTC=0.4;
        elseif and(rawdata(i,15)<safezone,rawdata(i,21)>0)  %Collision/Leading
vehicle moving away
            RewardTTC=0;
        end
    elseif (DelT>1)
        RewardDelT=1; %Best situation, when TTC is bigger than TTS;
    else
        RewardDetT=0; %For situations where TTC is shorter and there will be
collision
    rawdata(i,29)=TTC;
    rawdata(i,28)=RewardTTC+RewardDetT;
    end
end
%% VVL-Based Reward (Real and Desired Velocities)
RewardDelVVL=0;
for i= 1:leng
    DelVVL=((minus(rawdata(i,27),TVL))./TVL);
    if DelVVL>=1.1                      %Real velocity over 110% os desired one
        RewardDelVVL=0.6;
    elseif and(DelVVL<1.1,DelVVL>=1)    %Real velocity between 100-110%
        RewardDelVVL=1;
    elseif and(DelVVL<1,DelVVL>=0.8)    %Real velocity between 80-100%
        RewardDelVVL=0.6;
    elseif and(DelVVL<0.8,DelVVL>=0.6)  %Real velocity between 60-80%
        RewardDelVVL=0.4;
    elseif and(DelVVL<0.6,DelVVL>=0.4)  %Real velocity between 40-60%
        RewardDelVVL=0.2;
    elseif DelVVL<0.4                   %Real velocity under 40%
        RewardDelVVL=0;
    rawdata(i,28)=rawdata(i,28)+RewardDelVVL; %SUM with last reward
    end
end
%% TTS-Based Reward ('Time To Stop' VUT)
RewardDelT=0;
end
function netNew    = target_update(netNew,netOld,tau) %b  a
netNew = setwb(netNew,(tau.*getwb(netOld)+(1-tau).*getwb(netNew)));
end
function TTC       = time_to_collision(deltaX,deltaV,safe)
X=minus(deltaX,safe);
if deltaV==0
    TTC=0;
else
    TTC=X./deltaV;
end
end
```