COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

CREATION OF COMPLEX TEST SCENARIOS FOR AUTOMATED VEHICLES BY MEANS OF MACHINE LEARNING.

Autor: Eduardo Villalobos Requena

Director: Prof. Dr. Ing.- Markus Lienkamp

Co-Director: M. Sc. Thomas Ponn

Madrid

Agosto de 2019

AUTHORIZATION FOR DIGITALIZATION, STORAGE AND DISSEMINATION IN THE NETWORK OF END-OF-DEGREE PROJECTS, MASTER PROJECTS, DISSERTATIONS OR BACHILLERATO REPORTS

1. Declaration of authorship and accreditation thereof.

The author Mr. /Ms. Eduardo Villalobos Requena

HEREBY DECLARES that he/she owns the intellectual property rights regarding the piece of work: Creation of Complex Test Scenarios for Automated Vehicles by Means of Machine Learning
that this is an original piece of work, and that he/she holds the status of author, in the sense granted by the Intellectual Property Law.

2. Subject matter and purpose of this assignment.

With the aim of disseminating the aforementioned piece of work as widely as possible using the University's Institutional Repository the author hereby **GRANTS** Comillas Pontifical University, on a royalty-free and non-exclusive basis, for the maximum legal term and with universal scope, the digitization, archiving, reproduction, distribution and public communication rights, including the right to make it electronically available, as described in the Intellectual Property Law. Transformation rights are assigned solely for the purposes described in a) of the following section.

3. Transfer and access terms

Without prejudice to the ownership of the work, which remains with its author, the transfer of rights covered by this license enables:

a) Transform it in order to adapt it to any technology suitable for sharing it online, as well as including metadata to register the piece of work and include "watermarks" or any other security or protection system.

b) Reproduce it in any digital medium in order to be included on an electronic database, including the right to reproduce and store the work on servers for the purposes of guaranteeing its security, maintaining it and preserving its format.

c) Communicate it, by default, by means of an institutional open archive, which has open and cost-free online access.

d) Any other way of access (restricted, embargoed, closed) shall be explicitly requested and requires that good cause be demonstrated.

e) Assign these pieces of work a Creative Commons license by default.

f) Assign these pieces of work a HANDLE (*persistent* URL). by default.

4. Copyright.

The author, as the owner of a piece of work, has the right to:

a) Have his/her name clearly identified by the University as the author

b) Communicate and publish the work in the version assigned and in other subsequent versions using any medium.

c) Request that the work be withdrawn from the repository for just cause.

d) Receive reliable communication of any claims third parties may make in relation to the work and, in particular, any claims relating to its intellectual property rights.

5. Duties of the author.

The author agrees to:

a) Guarantee that the commitment undertaken by means of this official document does not infringe any third party rights, regardless of whether they relate to industrial or intellectual property or any other type.

b) Guarantee that the content of the work does not infringe any third party honor, privacy or image rights.
c) Take responsibility for all claims and liability, including compensation for any damages, which may be brought against the University by third parties who believe that their rights and interests have been infringed by the assignment.
d) Take responsibility in the event that the institutions are found guilty of a rights infringement regarding the work subject to assignment.

6. Institutional Repository purposes and functioning.

The work shall be made available to the users so that they may use it in a fair and respectful way with regards to the copyright, according to the allowances given in the relevant legislation, and for study or research purposes, or any other legal use. With this aim in mind, the University undertakes the following duties and reserves the following powers:

a) The University shall inform the archive users of the permitted uses; however, it shall not guarantee or take any responsibility for any other subsequent ways the work may be used by users, which are non-compliant with the legislation in force. Any subsequent use, beyond private copying, shall require the source to be cited and authorship to be recognized, as well as the guarantee not to use it to gain commercial profit or carry out any derivative works.
b) The University shall not review the content of the works, which shall at all times fall under the exclusive responsibility of the author and it shall not be obligated to take part in lawsuits on behalf of the author in the event of any infringement of intellectual property rights deriving from storing and archiving the works. The author hereby waives any claim against the University due to any way the users may use the works that is not in keeping with the legislation in force.
c) The University shall adopt the necessary measures to safeguard the work in the future.
d) The University reserves the right to withdraw the work, after notifying the author, in sufficiently justified cases, or in the event of third party claims.

Madrid, on 25th….. of ……August……..., 2019….

HEREBY ACCEPTS

Signed……Eduardo Villalobos Requena……

Reasons for requesting the restricted, closed or embargoed access to the work in the Institution's Repository

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título Creation of Complex Test Scenarios for Automated Vehicles by Means of Machine Learning en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2018/2019 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Eduardo Villalobos Requena Fecha: 21 / 05 / 2019

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Prof. Dr. -Ing. Markus Lienkamp Fecha: 21 / 05 / 2019

MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

CREATION OF COMPLEX TEST SCENARIOS FOR AUTOMATED VEHICLES BY MEANS OF MACHINE LEARNING.

Autor: Eduardo Villalobos Requena

Director: Prof. Dr. Ing.- Markus Lienkamp

Co-Director: M. Sc. Thomas Ponn

Madrid

Agosto de 2019

Creation of Complex Test Scenarios for Automated Vehicles by Means of Machine Learning

Author: Villalobos Requena, Eduardo.

Director: Prof. Dr. Ing.- Lienkamp, Markus.

Collaborating Entity: Technische Universität München.

PROJECT SUMMARY

***Abstract*- The purpose of this Thesis is the creation of complex situations for automated vehicles by means of Machine Learning. To launch these vehicles on to the market they need to be tested in real-life traffic situations, which is not only expensive but also dangerous. The current tests are far too simple, and with the objective of avoiding them and creating better ones, traffic situations must be filtered according to their complexity. Complexity is an ambiguous term that has many interpretations and definitions. Finding one definition that suits traffic scenarios is one of the objectives of the present work. In addition, this measure for complexity can be used to generate complex situations in different scenarios. The scenarios of choice occur in a three-way highway following the German regulations. There, some traffic participants are included and using machine learning in MATLAB, specifically a double deep q-learning algorithm for Reinforcement Learning. First the parameters of the model are inferred, and then the model or agent is trained to obtain the most difficult succession of events according to the previous measure of complexity. The trained agent is simulated in new scenarios for it and the complexity measure that it obtains in them is not as good as the maximum value possible. In parallel, another model is trained in Python, showing that the training in this programming language is more efficient in terms of time of training, even with lower computational power. In conclusion, the MATLAB agents show a poor performance when simulated in scenarios with added vehicles and the Python approach seems more adequate due to the reduced time to train.**

Keywords: automated vehicles, complexity, Machine Learning, Reinforcement Learning, MATLAB, Python.

i. Introduction

Nowadays, the automotive industry is going through a period of continuous changes, which are mainly related to the need for electrification of the traditional means of transport and to the search for an automobile that is capable of driving alone, without the supervision or the interaction from the passengers with it. These are the so-called autonomous vehicles. Especially in the area of public transport, be it with buses or taxis, autonomous vehicles are bringing a change in the traffic system.

In order to launch onto the market automated driving vehicles, it is necessary that they behave safely in a constant way. Real time traffic situations are always unpredictable, and they are rarely repeated, since the conditions of the different scenarios is not the same. For this reason, it is impossible to test all feasible situations that may arise in an everyday drive, being some of them of a really complex nature.

In addition to this, the usual real-life tests that are performed to check the safety of some of the systems that already exist, like braking when a pedestrian crossing in front of the vehicle is detected, are way too simple and well proven.

Aiming to avoid testing the easy scenarios, it is necessary to filter the traffic situations according to its complexity, which is related to the behavior of the surrounding traffic. Therefore, a traffic complexity measure is obliged. With this measure, it is possible to find the most challenging conditions that can occur in everyday driving and simulate in them a self-driving vehicle and control its performance.

ii. State of the art

Autonomous driving vehicles can be classified according to the level of automatization that they are making use of. This classification is divided in six levels, from level zero to level five, both included [1]. The lowest level corresponds to the driving as it is known today, where the driver is in charge of handling all the functionalities and systems of the vehicle. In contrast, the last level would be a fully automated vehicle with no need for intervention of the users. In the most recent developments presented by Audi [2], the automatization stands on the level 3 of the autonomy scale. Another famous brand in terms of autonomous driving is Tesla, whose autonomous driving system stays in the level 2 at the moment.

The difficulties that automated vehicles (AV) involve, not only in the area of technology, but also in the fields of philosophy, morality and social responsibility, fully autonomous vehicles (level 5) will only be able to operate from the year 2040 onwards according to Altenburg [3].

Conducting the test in real life scenarios is costly when not dangerous, so an evaluation in a virtual environment is of great use. One of the most powerful methods to generate the most complex scenarios that can be used is Machine Learning, specifically Reinforcement Learning.

Therefore, the subject of this Thesis is the development of a measure for complexity in traffic scenarios, with the aim to generate difficult situations to test the performance of the automated vehicles in them. The model of the vehicles and the development of the scenarios will be carried out using Reinforcement Learning in Matlab. A comparison with Python will be also carried out, in order to check for differences in means of training performance.

First of all, the complexity term needs to be studied and bounded taking into account the problem that is tried to be solved. Complexity is an ambiguous term, which can potentially have many different interpretations. Finding a definition of complexity that satisfies everyone and every imaginable application is practically impossible.

Starting with the complexity of physical systems in general, it can be said that the complexity of a system is given by its disorder [4]. The entropy, defined as a probability of actions that end in a certain state, is used to measure this disorder.

Human behaviour is challenging to model based on probabilities. For this reason, other approaches that are related to traffic situations are based on the composition of the road, named Road Semantic Complexity (RSC); and the traffic participants, named the Traffic Element Complexity (TEC). RSC groups the environment complexity and includes the road type, the scene type (intersection, tunnel, etc), and the challenging conditions (overtaking, pedestrian, etc). Traffic participants conform the TEC, which depends on the distance from car to car and the angles between them. Equation (1) shows the assessment of the overall complexity C, based on the RSC C_R and the TEC C_E. [5]

$$C = \lambda_1 C_R + \lambda_2 C_E \quad (1)$$

An example of a complex city scene, where many vehicles difficult the vision of

the automated vehicle (AV) is shown in Figure 1 [6].

Figure 0.Complex City scene. The recognition of the lanes is difficulted by the close distances to other traffic participants and the great number of them [6].

Besides, the criticality of the traffic situations can also have an impact on the complexity of those scenarios, especially when the driver is an automated vehicle. The criticality can be assessed making use of the time to collision (TTC) [7], which evaluates the threat levels for a self-driving vehicle based on the calculation of the remaining time to collide with the preceding vehicle in the case that no action was performed. In this Thesis, no action means driving with constant velocity and no lane changes. The time integrated to collision (TIT) is obtained integrating over time the TTC.

Given some of the limitations of the approach used and the factors of influence that can be modified, the complexity measure is based on the number of actions of all vehicles that are present in the environment, the number of reactions of the autonomous vehicle, and the TIT. Due to the fact that the complexity to assess is the complexity of the overall simulation and not of every timestep, and the TIT makes sense only once all the TTCs are collected, the complexity measure will be evaluated at the end of every episode in training following equation (2).The absolute value of the weights is equal to one. In the beginning, w_1 and w_2 are equal to 0.4, and w_3 is equal to 0.2. The weight of the TIT is positive due to the fact that an increasing value of TIT would mean a more critical scenario for the AV, hence more complex. Besides, it is divided by 2 in order to reduce the importance of the TIT in comparison with the importance of the number actions and reactions for the complexity.

$$C = [w_1 \quad w_2 \quad w_3] \begin{bmatrix} Number\ of\ actions \\ Number\ of\ reactions \\ TIT_{AV} \end{bmatrix} \qquad (2)$$

The measure of complexity can be applied to find the most difficult situation for different traffic situations. Starting from a base scenario, Machine Learning is implemented in order to find which actions complicate the situation the most, when are they performed, and which traffic participant does them. Machine Learning can be divided in three main groups, depending on the knowledge that is available of the inputs and outputs.

In Supervised Learning, the inputs and desired outputs are known. The data is divided in three sets: the training data, the validation data, and the test data [8]. The training data is used to infer a function to relate inputs and outputs. Next, the validation data tune the hyperparameters (parameters whose values are established before the training begins and have an impact on the speed and performance of the process), choosing the best model that has been trained. This chosen model is evaluated using the test dataset. The key objective of Supervised Learning is to develop an optimal algorithm that is capable of classifying new data that it has never seen [9].

Depending on the problem faced, different algorithms from Supervised Learning can be utilized. Nonetheless, there is no one superior algorithm, and every one of them has its own advantages and disadvantages. One of the most known algorithms are Neural Networks, mainly utilized for image recognition.

Neural Networks are an imitation of what the network of an animal brain looks like. A Neural Network is composed of nodes called neuros that are connected between them, being a group of networks a layer. [10]

For this Thesis, the variation of Neural Networks that play a major role are the Convolutional Neural Networks (CNN) [11]. The name is given as the core of CNNs is the convolutional layer. A

convolution is an operation consisting of a dot product between the array of inputs and a two-dimensional set of weights called filter or kernel. As a result, a feature map is obtained, which represents a filtering ofthe input. The main problem of this approach is solved using pooling layers that treat each feature map independently. Besides, non-linearity functions are added to the network, so there are no negative values, as for instance the Rectified Linear Units (ReLU). Figure 2 [12] shows a full CNN used for image recognition.

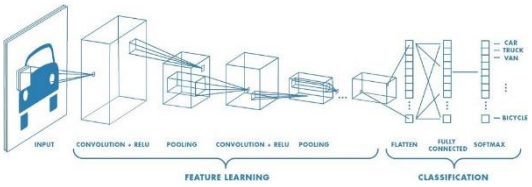


Figure 2. Example of a CNN applied to object recognition [12]. There are convolutional and ReLU layers combined before every pooling.

The next approach of Machine Learning is Unsupervised Learning. This approach is used when the desired outputs are not known beforehand. In this case, it is tried to discover the underlying patterns that are present in the input data. The most common uses are clustering, anomalies detection, association mining, and latent variable models. The algorithms are Hierarchical Clustering, K-means, Mixture Models, DBSCAN and OPTICS, Principal Component Analysis and the Method of Moments.

Reinforcement Learning is the approach of choice from the three possible ones. Reinforcement Learning (RL) is used when the outputs are not known, but it is possible to assess the quality of the outputs. The principle of this model is based on the interaction of a learning agent with its environment to reach a specific goal [13]. The agent does not receive the actions that it has to perform and their order, instead it tries to find out which actions receive the greater reward. To achieve this, the agent receives an observation of the state of the environment and then executes and action according to it. Figure 3 represents the interaction between an agent and the environment.

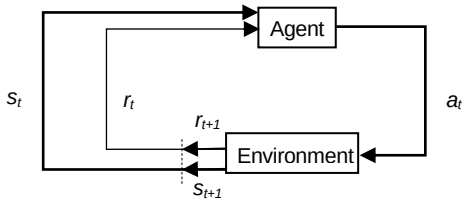


Figure 3. The agent-environment interaction.

The elements that compose a Reinforcement Learning problem are the environment, the policy (strategy that the agent uses to choose the actions), the reward function or return, and the value function (estimates how good the actual state is). [13]

There are different instances to describe a RL problem. One of them is the Markov Decision Process (MDP) [14], in which is required that the environment has a finite set of states and actions for each possible state. The algorithm for MDPs works in two steps. In the first step the values are updated and in the second one, the policy is updated. These steps are repeated in order for all the states until there are no more changes. Based on this functioning, there are different methods that can be applied to solve a RL problem: Dynamic Programming, Monte-Carlo Learning, and Temporal Difference Learning. Considering the nature of the problem, the method chosen in this Thesis is Temporal Difference Learning.

Temporal Difference (TD) [15] methods learn from the interaction with the environment or experience. One advantage of this method is the utilization of bootstrapping (estimating the value function from estimates), so it is not necessary that the episodes end before learning.

TD(λ) is a generalization of the TD or TD(0) method. The main difference relies in the way that the return is calculated. *SARSA* is an on-policy method (the algorithm is concerned about the policy which yielded past decisions) while Q-Learning is an off-policy method. As there is no policy in the environments proposed, the algorithm of choice is a variation of Q-Learning, which also has

some performance advantages when compared to *SARSA*. [16]

The variant of Q-Learning is Deep Q-Learning (DQN) [17], which improves the performance of simple Q-Learning when the state space is of a considerable size introducing a CNN. As explained, CNNs are used mainly in image recognition, but as the input of the CNN is just an array of values, the input could be anything. In this case, the inputs are the observation of the state of the environment and the action chosen by the agent. The problem with DQN is that it tends to overestimate actions once it is chosen, hindering the convergence of the algorithm. To solve this issue, a double network is used, creating a Double DQN (DDQN). One network selects the actions and the other one updates the value function. Copying weights from the action network to the target network keeps both networks synchronized.

iii. Methodology

The environment that is going to be used is a German highway of three lanes, following German regulations [18]. Apart from the road, the traffic participants are modelled. There are three possible vehicles: cars, trucks, and motorbikes. The cars are 4.9 *m* long and 2 *m* wide, the trucks have a length of 16.5 *m* and a width of 2.5 *m*, and the motorbikes are 2 *m* long and 1 *m* wide. Besides, the movement of the vehicles needs to be designed. Apart from the automated vehicles (that are programmed to drive on their own and cannot be controlled by the agent), every vehicle has the possibility of performing four actions. Accelerating and braking are made at constant acceleration. For the lane changes, apart from the direction of the lane change (left and right), the lateral displacement and acceleration are needed.

The automated vehicle moves in a similar way but alone and has some added rules in order to perform correctly the movements and to have a realistic behavior. First of all, the AV tries to reach a desired velocity given by the user. Next, if there is an obstacle or other vehicle that goes slower in front of him, the AV checks if overtaking is possible. If it is, the AV changes its lane to the left and performs the overtake. If the overtake is not possible, the AV brakes preserving a certain security distance. If the AV finds itself in any lane that is not the right one, and is not overtaking, it tries to go to the lane to its right (if its free of vehicles).

The three elements together (road, vehicles, and the movement of the vehicles) create the environment for the training. The environment is simulated in episodes, which at the same time are divided into steps (timesteps of 0.1 *s*). In every step, the agent chooses an action to be performed. The actions of braking and accelerating can be selected for only one step, but the lane changes need to be selected for around 24 steps, so the movement is complete. Otherwise, the vehicles that change their lanes could end up in the middle of two lanes. Besides to these actions for very vehicle that is not the AV, an action 0 is created, which implies do nothing (all vehicles just drive straightforward).

The next step is to tune the hyperparameters of the DDQN. To do it, a base scenario is selected. In the base scenario the AV drives in the left lane, there are two cars in the middle lane, and three trucks are driving in the right lane.

The hyperparameters to tune are the learning rate, the number of neurons per hidden layer, the number of hidden layers of the network, the number of episodes to average the reward, and the batch size. After running multiple trainings with different combinations of parameters, the final values are:

- Number of episodes to average the reward: 5.
- Number of neurons per layer: 200.
- Learning rate α: 1e-4.
- Batch size: 1024 samples.

- The number of hidden layers is three for the state, two for the actions, and one for the common path.

The final layout of the CNN of the algorithm is the one showed in Figure 4. In Figure 5, an example of the training output is shown. The blue dots are the reward of every episode run, the red ones are the average reward of the last 5 episodes, and the green line represents the expected reward for the agent when every episode begins.

In addition to the MATLAB approach, the problem is also addressed in Python. The environment is the same as the one in MATLAB, but the CNN that the agent makes use of is different. This Python approach is compared to the MATLAB one in order to assess the performance of both.

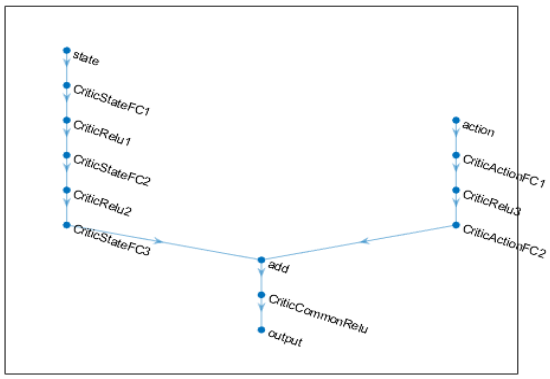


Figure 4. CNN used for training the agent in MATLAB.

iv. Results and Conclusions

With the algorithms ready, the agents are trained in an episode. Then, the trained agent is simulated in the same scenario to test if it is a valid one or not (sometimes the training might be stopped but the agent has not learned). If the agent has learned, it can be used in other scenarios to try to obtain the most complex succession of events.

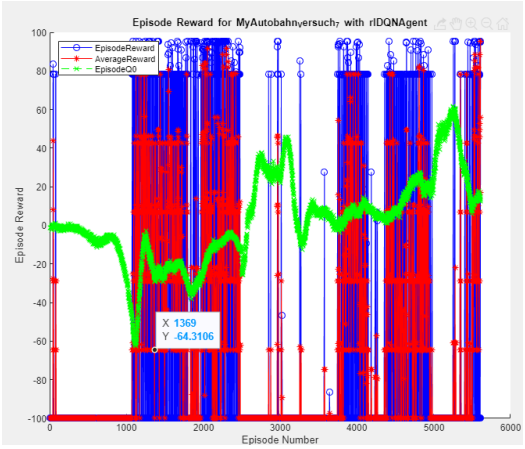


Figure 5. Training progress with a batch size of 1024 samples, 200 neurons and a learning rate of 1e-4. The training is successful after 37 hours for a total of 5611 episodes.

In the test scenario, the AV drives on the right lane with a truck in front of him, and there are two cars in the middle lane. In this scenario, the most complex situation is composed of many movements from the two normal cars. The AV stays behind the truck, keeping a safety distance.

Next, the trained agent it is tested in three different scenarios. The first one adds to the training scenario a third car driving fast on the left lane. The reward in this scenario is smaller than in the training one, but the agent shows an acceptable behavior. Next, a truck is added in front of the first truck for the second scenario. Here the agent performs worse and makes the truck and one of the normal cars collide, so the agent becomes a negative reward. In the third scenario, the truck and one of the normal cars exchange their initial positions. The simulation of the agent makes the other car crash on the back of the truck, so again it receives a negative reward.

In the comparison between MATLAB and Python, it is only assessed the required amount of time and the number of episodes for a successful training. It can be clearly seen that the Python approach learns faster in terms of number of episodes, but also in terms of total time. This is caused partly because the network is smaller and the training is

faster because of the smaller batch size, Nonetheless, there is no precision lost. Therefore, it can be said that Python performs better.

In conclusion, the trained agent performs good enough when the initial traffic situation is not much different from the one in which it was trained. Adding more traffic participants or changing the initial conditions has a negative impact on the performance of the agent in the new scenarios.

v. Bibliography

[1] SAE International. “Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems.” On-Road Automated Driving (ORAD) committee.

[2] “Der neue Audi A8 – hochautomatisiertes Fahren auf Level 3“, *Audi MediaCenter*. (11.09.2017). Retrieved from: https://www.audi-mediacenter.com:443/de/per-autopilot-richtung-zukunftdie-audi-vision-vom-autonomen-fahren-9305/der-neue-audi-a8-ochautomatisiertes-fahrenauf-level-3-9307. Accessed on: 13/08/2019

[3] S. Altenburg, “Einführung von Automatisierungsfunktionen in der Pkw-Flotte“, Prognos AG (2018): p. 58. Retrieved from: https://www.adac.de/-/media/pdf/motorwelt/prognos_automatisierungsfunktionen.pdf?redirectId=quer.mwe.prognos-studie. Accessed on: 13/08/2019

[4] S. Lloyd and H. Pagels. “Complexity as thermodynamic depth”. *Annals of physics* 188.1 (1988): pp. 186-213.

[5] J. Wang, et al. “Traffic Sensory Data Classification by Quantifying Scenario Complexity”. *2018 IEEE Intelligent Vehicles Symposium (IV).* IEEE, 2018.

[6] R. Danescu, and S. Nedevschi. "Probabilistic lane tracking in difficult road scenarios using stereovision." *IEEE Transactions on Intelligent Transportation Systems* 10.2 (2009): pp. 272-282.

[7] J. C. Hayward. "Near miss determination through use of a scale of danger." (1972): pp. 24-34.

[8] P. Lison. "An introduction to machine learning." *Language Technology Group (LTG), 1* 35 (2015).

[9] Skbkekas. Training validation and test sets. URL: https://en.wikipedia.org/wiki/Training,_validation,_and_test_sets. Accessed on: 14/08/2019.

[10] S. S. Haykin. *Neural networks and learning machines*. New York: Prentice Hall, 2009: pp. 579-626.

[11] Y. LeCun, K. Kavukcuoglu, and C. Farabet. "Convolutional networks and applications in vision." *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. IEEE, 2010.

[12] S. Saha. A Comprehensive Guide to Convolutional Neural Networks.

URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. Accessed on: 21/08/2019

[13] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, Second edition. Cambridge, Massachusetts: The MIT Press, 2018.

[14] R. Bellman. "A Markovian decision process." *Journal of mathematics and mechanics* (1957): pp. 679-684.

[15] R. S. Sutton. "Learning to predict by the methods of temporal differences." *Machine learning* 3.1 (1988): pp. 9-44.

[16] G. A. Rummery, and M. Niranjan. *On-line Q-learning using connectionist systems*. Vol. 37. Cambridge, England: University of Cambridge, Department of Engineering, 1994: p. 21.

[17] H. Van Hasselt, A. Guez, and D. Silver. "Deep reinforcement learning with double q-learning." *Thirtieth AAAI conference on artificial intelligence*. 2016.

[18] Forschungsgesellschaft für Straßen- und Verkehrswesen e. V: Richtlinien für die Anlage von Autobahnen – RAA08. Köln: FGSV-Verlag, 2008.

CREACIÓN DE ESCENARIOS COMPLEJOS DE PRUEBA PARA VEHÍCULOS AUTOMATIZADOS MEDIANTE MACHINE LEARNING.

Autor: Villalobos Requena, Eduardo.

Director: Prof. Dr. Ing.- Lienkamp, Markus.

Entidad Colaboradora: Technische Universität München.

RESUMEN DEL PROYECTO

Resumen- **El objetivo de este trabajo es la creación de situaciones complejas para vehículos automatizados mediante Machine Learning. Para lanzar estos vehículos al mercado es necesario que sean probados en situaciones de tráfico real, lo cual es caro y peligroso. Las pruebas actuales son demasiado simples y con el objetivo de evitarlas, las situaciones del tráfico han de ser filtradas de acuerdo a su complejidad. Complejidad es un término con muchos significados y definiciones. Encontrar una definición que se aplique al tráfico es uno de los objetivos de este trabajo. Además, la medida de la complejidad puede ser usada para generar escenarios difíciles. Los escenarios elegidos transcurren en una autopista que sigue las normas alemanas para su construcción. En ella se incluyen algunos participantes del tráfico y, usando Machine Learning en MATLAB (concretamente un algortimo double deep q-learning para Reinforcement Learning). En primer lugar, los parámetros del modelo son obtenidos, y luego el modelo o agente es entrenado para obtener la sucesión de eventos y acciones que producen la situación del tráfico más compleja de acuerdo con la medida creada. El agente entrenado es simulado en nuevas situaciones y la medida de la complejidad que obtiene no resulta tan buena como el máximo valor posible. Paralelamente, otro modelo es entrenado en Python, demostrando que este lenguaje de programación es más eficiente en términos de tiempo necesario y episodios totales de entrenamiento incluso usándolo en un ordenador de peores características. En conclusión, los agentes de MATLAB presentan un desempeño peor cuando son simulados en escenarios con más actores y el uso de Python parece más adecuado.**

Palabras clave: vehículos automatizados, complejidad, Machine Learning, Reinforcement Learning, MATLAB, Python.

i. Introducción

La industria automovilística está bajo un proceso de continuos cambios que están principalmente relacionados con la necesidad de electrificación de los medios de transporte tradicionales y con la búsqueda de un vehículo capaz de conducir por su cuenta sin la supervisión o la interacción de los pasajeros. Son los llamados coches autónomos. Especialmente en el área del transporte público, ya sea con autobuses o taxis, los vehículos autónomos traen un cambio en el sistema de tráfico.

Para poder lanzar al mercado los vehículos automatizados, es necesario que éstos se comporten de manera segura constantemente. Las situaciones reales de circulación son siempre impredecibles, raramente repetidas debido a que las condiciones de las mismas no son iguales. Por este motivo, es imposible probar todos los posibles

escenarios que pueden suceder en la conducción cotidiana, siendo algunas de ellas realmente complejas.

Además, las pruebas realizadas para comprobar la seguridad de algunos de los sistemas que ya existen, como frenar cuando se detecta un peatón que cruza delante del vehículo, son demasiado simples.

Con el objetivo de evitar probar los escenarios sencillos, es necesario filtrar las situaciones del tráfico de acuerdo a su complejidad, que está relacionada con el comportamiento del tráfico alrededor. Con esta medida, es posible encontrar las condiciones más desafiantes que pueden suceder en la conducción del día a día y simularlas en un vehículo automatizado controlando su desempeño.

ii. Estado del arte

Los vehículos automatizados pueden ser clasificados de acuerdo a su nivel de automatización. Esta clasificación está divida en seis niveles, del cero al 5, ambos incluidos [1]. El nivel más bajo se corresponde con la conducción tal y como es conocida hoy en día, donde es el conductor el que se encarga de realizar todas las funciones. Por el contrario, el último nivel sería un vehículo completamente autónomo que no tiene la necesidad de intervención de los usuarios. Los últimos desarrollos presentados por Audi [2], la automatización se encuentra en el nivel tres. Otra marca famosa por su conducción automatizada es Tesla, cuyos sistemas se encuentran actualmente en el nivel 2.

Las dificultades que los vehículos automatizados presentan (AV) no sólo en el apartado tecnológico, sino también en el filosófico, moral y de responsabilidad social, provoca que los vehículos completamente autónomos (nivel 5) podrán operar a partir de 2040 según Altenburg [3].

Realizar las pruebas en situaciones de la vida real es costoso cuando no peligroso, por lo que la evaluación en escenarios virtuales es de gran utilidad. Uno de los métodos más potentes para generar los escenarios más difíciles que puede ser usado es Machine Learning, en concreto el Aprendizaje por Refuerzo (Reinforcement Learning).

Por ello, el tema de esta Tesis es el desarrollo de una medida de complejidad en escenarios de tráfico, con el objetivo de generar situaciones difíciles para probar el desempeño de los vehículos automatizados en ellos. El modelo de los vehículos y el desarrollo de los escenarios será realizado usando Aprendizaje por Refuerzo en MATLAB. Además, se realiza una comparación con Python para comprobar las diferencias en términos de rendimiento del entrenamiento.

En primer lugar, el término complejidad tiene que ser estudiado y limitado teniendo en cuenta el problema que se está intentado solucionar. Complejidad es un término ambiguo que puede potencialmente tener muchas interpretaciones. Encontrar una definición que satisfaga a todo el mundo y sea útil para toda aplicación es prácticamente imposible.

Empezando por la complejidad de los sistemas físicos en general, se puede decir que la complejidad viene dada por su desorden [4]. La entropía, definida como la probabilidad de que las acciones realizadas acaben en un cierto estado, se utiliza para medir el desorden.

El comportamiento humano es difícilmente modelable basado en probabilidades. Por ello, otros enfoques relacionados con escenarios de tráfico están basados en la composición de la carretera, llamada *Road Semantic Complexity* (RSC), y los participantes del tráfico, denominados *Traffic Element Complexity* (TEC). RSC agrupa la complejidad del ambiente e incluye el tipo de carretera, el tipo de escena (intersección, etc.), y las condiciones desafiantes (adelantamientos, etc.). TEC depende de la distancia y los ángulos

entre los vehículos. La ecuación (1) muestra la valoración de la complejidad en general *C*, basada en el TEC C_E y en el RSC C_R. [5]

$$C = \lambda_1 C_R + \lambda_2 C_E \quad (1)$$

Un ejemplo de una escena de ciudad compleja se muestra en la Ilustración 1 [6], donde muchos vehículos dificultan la visión del coche.

Ilustración 1. Escena de ciudad compleja. El reconocimiento de los carriles se ve dificultado por las cortas distancias entre vehículos y el gran número de ellos [6].

Asimismo, la criticidad de las situaciones de trafico puede tener un impacto en la complejidad de las situaciones, especialmente cuando el conductor es un AV. La criticidad puede ser evaluada haciendo uso del tiempo para colisión (TTC) [7], que mide los niveles de amenaza para un vehículo automatizado basado en el cálculo en el tiempo restante para colisionar con el vehículo de delante en el caso de que ninguna acción fuera realizada. En este trabajo, no realizar acciones significa conducción a velocidad constante sin cambios de carril. El tiempo integrado para colisión (TIT) se obtiene integrando sobre el tiempo el TTC.

Dadas algunas de las limitaciones de los factores que es posible modificar, la medida de complejidad viene definida por el número de acciones de todos los vehículos presentes, el número de reacciones del AV, y el TIT. Debido a que la complejidad a valorar es la de la situación en general, y el TIT tiene sentido una vez se han recogido todos los TTC, la complejidad es evaluada solo al final de cada episodio durante el entrenamiento siguiendo la ecuación (2). El valor absoluto de los pesos suma 1. El peso de las acciones y reacciones es 0.4 y el del TIT 0.2. El peso del TIT es positivo por el hecho de que un incremento del valor del TIT significa un escenario más crítico, y por lo tanto puede ser más complejo.

$$C = [w_1 \quad w_2 \quad w_3] \begin{bmatrix} Number\ of\ actions \\ Number\ of\ reactions \\ TIT_{AV} \end{bmatrix} \quad (2)$$

Esta medida de complejidad es aplicada para encontrar la situación de tráfico más complicada posible. Empezando con un escenario base, Machine Learning es implementado para encontrar las acciones que complican la situación al máximo y que actores de esta las realizan. Machine Learning generalmente es dividido en tres grupos dependiendo del conocimiento que se tiene sobre las entradas y las salidas.

En Supervised Learning (Aprendizaje supervisado), las entradas y salidas deseadas son conocidas. Los datos están divididos en tres grupos: el grupo de entrenamiento, el de validación y el de prueba [8]. Los datos de entrenamiento se usan para encontrar una función que relacione entradas con salidas. Luego, los datos de validación ajustan los hiper parámetros (parámetros cuyos valores son establecidos antes de empezar el entrenamiento y tienen influencia en la velocidad y rendimiento del proceso), eligiendo el mejor modelo entrenado. El modelo elegido se evalúa con los datos de prueba. El objetivo principal del Aprendizaje supervisado is el desarrollo un algoritmo óptimo capaz de clasificar nuevos datos que nunca ha visto [9].

Dependiendo del problema enfrentado, existen diferentes algoritmos que pueden ser utilizados, cada uno de ellos con sus ventajas y desventajas. Uno de los más conocidos son las Redes Neuronales.

Las Redes Neuronales son una imitación del cerebro de un animal. Está compuesta por nodos llamados neuronas conectadas entre sí, formando grupos denominados capas. [10]

La variante utilizada de una Red Neuronal es una Red Neuronal Convolucional (CNN) [11]. El nombre viene por la función principal desarrollada por algunas de las capas de esta red: la convolución. La operación consiste en el producto escalar del vector

de entrada a la capa y un conjunto de pesos llamado filtro. El principal problema de esta red se soluciona utilizando capas de agrupación (pooling layers en inglés). Además, funciones de no linealidad se añaden a la red para no obtener valores negativos. Un ejemplo de estas funciones son las ReLU (Rectified Linear Units). La Ilustración 2 [12] muestra una CNN completa usada para reconocimiento de imágenes.

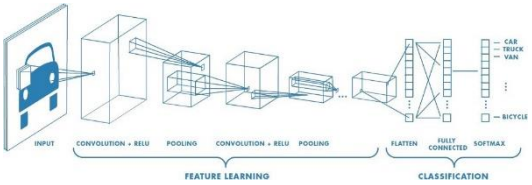


Ilustración 2. Ejemplo de CNN aplicada al reconocimiento de objetos [12]. Delante de cada capa de agrupación hay una capa de convolución y una ReLU.

El segundo método de Machine Learning es el Aprendizaje sin supervisión. A diferencia del supervisado, los valores deseados de las salidas son desconocidos. Se intentan descubrir los patrones implícitos en los datos. Los usos más comunes son de agrupación (clustering), detección de anomalías, minado de asociaciones, y modelos variables latentes. Los algoritmos para llevar a cabo estos usos son Hierarchical Clustering, K-Means, Mixture Models, Análisis de Componentes Principales (PCA) y el método de los momentos.

El Aprendizaje por refuerzo (RL) es el enfoque elegido de los tres posibles. Éste es utilizado cuando las salidas deseadas no son conocidas, pero se pueden evaluar. El principio de este modelo se basa en la interacción de un agente que aprende con su entorno para alcanzar un objetivo específico [13]. El agente no recibe las acciones a realizar ni el orden de éstas, sino que trata de encontrar qué acciones reciben la mayor recompensa. Para conseguirlo, el agente recibe una observación del estado del entorno y después ejecuta una acción de acuerdo al mismo. La Ilustración 3 representa esta interacción.

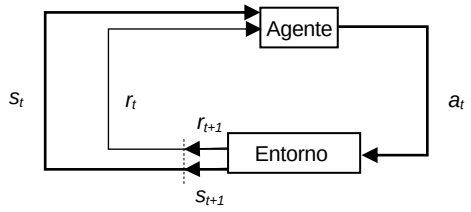


Ilustración 3. Interacción agente-entorno en RL.

Los elementos que componen un problema de RL son el entorno, la política (estrategia que utiliza el agente para elegir nuevas acciones), la función de recompensa o retorno y la función de valor (estima como de bueno es el estado actual). [13]

Existen diferentes casos para describir un problema de RL. Uno de ellos son los Procesos de Decisión de Markov (MDP) [14], en los que se requiere que el entorno tenga un número limitado de estados y acciones. El algoritmo para los MDPs funciona en dos pasos. En el primero, los valores son actualizados; en el segundo, la política se actualiza. Estos pasos son repetidos en orden para todos los estados hasta que no hay más cambios. Hay diferentes métodos que pueden ser usados para resolver este tipo de problemas: Programación Dinámica, Aprendizaje Monte-Carlo y Aprendizaje por Diferencia Temporal.

Los métodos de Diferencia Temporal (TD) [15] aprenden de la interacción con el entorno o experiencia. Una de las ventajas de estos métodos es la utilización de *bootstrapping* (estimar la función de valor de otras estimaciones), por lo que no es necesario que cada episodio termine antes de aprender.

TD(λ) es la generalización de TD o TD(0). La principal diferencia reside en la forma en la que el retorno es calculado. *SARSA* es un método con política (el algoritmo tiene en cuenta la política que produjo pasadas decisiones) mientras que el aprendizaje Q (Q-Learning) es un método sin política. Como en el entorno de esta Tesis no hay una política predefinida, el método elegido es Q-Learning, que además tiene algunas ventajas de rendimiento comparado con *SARSA*. [16]

La variante de Q-Learning elegida es Q-Learning profundo (Deep Q-Learning, DQN) [17], que mejora el rendimiento del Q-Learning cuando hay muchos posibles estados, introduciendo una CNN. En este caso, las entradas a la CNN son la observación del estado y la acción realizada. El problema de DQN es que tiene a sobreestimar una acción que ya ha elegido, por lo que la exploración se ve comprometida y por tanto la convergencia. Para solucionarlo se introduce una segunda red, formando un doble DQN (DDQN). Una red selecciona las acciones y la otra se encarga de actualizar la función de valor. Copiando los pesos de la red de acción a la red de objetivos se mantienen ambas redes sincronizadas.

iii. Methodology

El entorno que va a ser usado es una autopista alemana de tres carriles siguiendo la normativa alemana [18]. La Ilustración 5 muestra las dimensiones de ésta. Aparte de la carretera, los actores o participantes son modelados. Hay tres posibles vehículos: camiones, coches y motocicletas. Los coches miden 4.9 *m* de largo y 2 *m* de ancho, los camiones 16.5 *m* de largo y 2.5 *m* de ancho, y las motocicletas tienen una longitud de 2 *m* y una anchura de 1 *m*. Además, el movimiento de los vehículos tiene que ser diseñado. Sin tener en cuenta los vehículos automatizados, cada vehículo puede realizar cuatro acciones. Frenar y acelerar se realizan con aceleración constante. Para cambiar de carril a la derecha o a la izquierda, son necesarios el desplazamiento y la aceleración laterales.

El vehículo automatizado se mueve de manera similar a los otros vehículos, pero no puede ser controlado por el agente. Además, tiene ciertas reglas impuestas para realizar los movimientos de manera realista. En primer lugar, el AV trata de alcanzar una velocidad deseada impuesta por el usuario. Si delante de él encuentra otro vehículo que circula más lento, lo intentará adelantar. Si realizar el adelantamiento es posible, el AV cambia de carril a la izquierda y realiza el adelantamiento. Si el carril a la izquierda está ocupado y el adelantamiento no es posible, el AV frena manteniendo una distancia de seguridad predefinida. Si el AV se encuentra en cualquier carril que no sea el de la derecha y no está adelantando, tratará de volver al carril de su derecha hasta acabar en el de más a la derecha posible del todo.

Los tres elementos conjuntos (vehículos, carretera y el movimiento de los vehículos) forman el entorno para el entrenamiento. El entorno es simulado en episodios, que a su vez se dividen en pasos (pasos de tiempo de 0.1 *s*). En cada paso, el agente elige qué acción realizar. Las acciones de frenado y aceleración pueden ser seleccionadas solo por un paso, pero los cambios de carril son seleccionados de forma consecutiva durante 24 pasos para poder completar el movimiento. De otra forma, los vehículos podrían acabar en el medio de dos carriles. También hay una acción 0 que implica no hacer nada (conducción recta con velocidad constante para todos los vehículos).

El siguiente paso es ajustar los hiper parámetros del DDQN. Para hacerlo, se crea un escenario de partida en el que el AV conduce por el carril izquierdo, dos coches normales van por el carril de en medio, y tres camiones circulan por el derecho.

Los hiper parámetros a ajustar son la ratio de aprendizaje, el número de neuronas de cada capa oculta, el número de capas ocultas de la red, el número de episodios para hacer la media de la recompensa, y el tamaño del lote de muestras. Después de varias pruebas con diferentes combinaciones de valores de los anteriores hiper parámetros, los valores seleccionados son:

- Número de episodios para hacer la media de la recompensa: 5.
- Número de neuronas por capa: 200.

- Ratio de aprendizaje α: 1e-4.
- Tamaño del lote: 1024 muestras.
- El número de capas de la red son tres para el estado, dos para las acciones y una para el camino común.

La Ilustración 4 muestra la disposición final de la CNN del algoritmo y la Ilustración 5, un ejemplo de la salida del training. Los puntos azules son la recompensa de cada episodio, los rojos la recompensa media de los últimos 5 episodios, y los verdes representan la recompensa esperada por el agente cuando empieza cada episodio.

Paralelamente a MATLAB, también se llevan a cabo entrenamientos en Python. El entorno es el mismo que en MATLAB, pero la CNN que usa el agente cambia. Esta propuesta de Python se compara con MATLAB para evaluar el desempeño de ambos lenguajes.

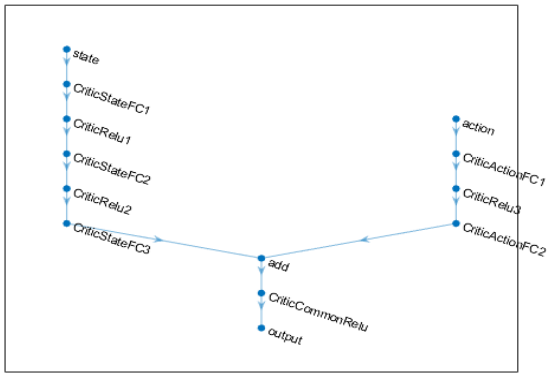


Ilustración 4. CNN utilizada para el entrenamiento en MATLAB.

iv. Resultados y Conclusiones

Con los algoritmos preparados, los agentes son entrenados en un escenario. Luego, el agente entrenado es simulado en el mismo escenario para comprobar si es un agente válido (en ocasiones, el entrenamiento acaba, pero el agente no ha aprendido en realidad). Si el agente ha aprendido, se puede utilizar en otros escenarios para obtener la sucesión de eventos más compleja.

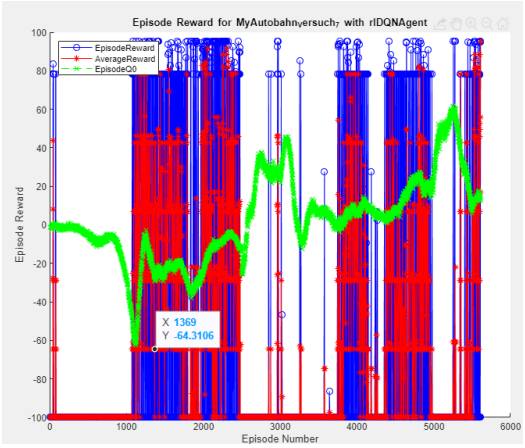


Ilustración 5. Progreso del entrenamiento con un tamaño de muestra de 1024, 200 neuronas, y una ratio de aprendizaje de 1e-4. El entrenamiento termina con 5611 episodios tras 37 horas.

En el escenario de prueba, el AV conduce en el carril derecho con un camión delante y hay dos coches en el carril central. La situación más compleja se da cuando los dos coches del carril central cambian varias veces sus posiciones de derecha a izquierda. El AV por el contrario se mantiene siempre detrás del camión, manteniendo una distancia de seguridad dada.

Luego, el agente entrenado es simulado en tres situaciones distintas. La primera añade al escenario de entrenamiento un tercer coche que viaja por el carril izquierdo. En esta situación, el agente obtiene una recompensa ligeramente inferior a la del entrenamiento, pero aceptable. En el segundo escenario de simulación, además se añade a lo anterior un segundo camión delante del primero en el carril derecho. En este caso el agente hace colisionar uno de los coches normales con el segundo camión, por lo que recibe una recompensa negativa. El tercer escenario cambia la posición del camión y el segundo coche del escenario de entrenamiento. En este caso también hace colisionar uno de los coches con el camión, por lo que de nuevo recibe una recompensa negativa.

En la comparación entren MATLAB y Python, solo se tendrá en cuenta el tiempo total y el número de episodios requeridos para realizar el entrenamiento. Se puede observar

claramente que Python aprende más rápido en el número total de episodios, pero también en tiempo total. En parte esto es causado por el menor tamaño de la red y el número de muestras. Sin embargo, no se pierde precisión por lo que se puede decir que Python es una mejor implementación.

En conclusión, el agente entrenado funciona de manera correcta cuando la situación inicial de los nuevos escenarios es parecida a la del escenario en el que se le ha entrenado. Añadir más participantes o cambiarlos de posición afectan negativamente al desempeño del agente.

v. Bibliografía

[1] SAE International. “Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems.” On-Road Automated Driving (ORAD) committee.

[2] “Der neue Audi A8 – hochautomatisiertes Fahren auf Level 3“, *Audi MediaCenter*. (11.09.2017). Retrieved from: https://www.audi-mediacenter.com:443/de/per-autopilot-richtung-zukunftdie-audi-vision-vom-autonomen-fahren-9305/der-neue-audi-a8-ochautomatisiertes-fahrenauf-level-3-9307. Accessed on: 13/08/2019

[3] S. Altenburg, “Einführung von Automatisierungsfunktionen in der Pkw-Flotte“, Prognos AG (2018): p. 58. Retrieved from: https://www.adac.de/-/media/pdf/motorwelt/prognos_automatisierungsfunktionen.pdf?redirectId=quer.mwe.prognos-studie. Accessed on: 13/08/2019

[4] S. Lloyd and H. Pagels. “Complexity as thermodynamic depth”. *Annals of physics* 188.1 (1988): pp. 186-213.

[5] J. Wang, et al. “Traffic Sensory Data Classification by Quantifying Scenario Complexity”. *2018 IEEE Intelligent Vehicles Symposium (IV).* IEEE, 2018.

[6] R. Danescu, and S. Nedevschi. "Probabilistic lane tracking in difficult road scenarios using stereovision." *IEEE Transactions on Intelligent Transportation Systems* 10.2 (2009): pp. 272-282.

[7] J. C. Hayward. "Near miss determination through use of a scale of danger." (1972): pp. 24-34.

[8] P. Lison. "An introduction to machine learning." *Language Technology Group (LTG), 1* 35 (2015).

[9] Skbkekas. Training validation and test sets. URL: https://en.wikipedia.org/wiki/Training,_validation,_and_test_sets. Accessed on: 14/08/2019.

[10] S. S. Haykin. *Neural networks and learning machines*. New York: Prentice Hall, 2009: pp. 579-626.

[11] Y. LeCun, K. Kavukcuoglu, and C. Farabet. "Convolutional networks and applications in vision." *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. IEEE, 2010.

[12]] S. Saha. A Comprehensive Guide to Convolutional Neural Networks.

URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. Accessed on: 21/08/2019

[13] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, Second edition. Cambridge, Massachusetts: The MIT Press, 2018.

[14] R. Bellman. "A Markovian decision process." *Journal of mathematics and mechanics* (1957): pp. 679-684.

[15] R. S. Sutton. "Learning to predict by the methods of temporal differences." *Machine learning* 3.1 (1988): pp. 9-44.

[16] G. A. Rummery, and M. Niranjan. *On-line Q-learning using connectionist systems*. Vol. 37. Cambridge, England: University of Cambridge, Department of Engineering, 1994: p. 21.

[17] H. Van Hasselt, A. Guez, and D. Silver. "Deep reinforcement learning with double q-learning." *Thirtieth AAAI conference on artificial intelligence*. 2016.

[18] Forschungsgesellschaft für Straßen- und Verkehrswesen e. V: Richtlinien für die Anlage von Autobahnen – RAA08. Köln: FGSV-Verlag, 2008.

Index

List of Figures

List of Tables

1. Introduction

Nowadays, the automotive industry is going through a period of continuous changes, which are mainly related to the need for electrification of the traditional means of transport and to the search for an automobile that is capable of driving alone, without the supervision or the interaction from the passengers with it. These are the so-called autonomous vehicles. Specially in the area of public transport, be it with buses or taxis, autonomous vehicles are bringing a change in the traffic system.

In order to launch onto the market automated driving vehicles, it is necessary that they behave safely in a constant way. Real time traffic situations are always unpredictable, and they are rarely repeated, since the conditions of the different scenarios is not the same. For this reason, it is impossible to test all feasible situations that may arise in an everyday drive, being some of them of a really complex nature.

In addition to this, the usual real-life tests that are performed to check the safety of some of the systems that already exist, like braking when a pedestrian crossing in front of the vehicle is detected, are way too simple and well proven.

Aiming to avoid testing the easy scenarios, it is necessary to filter the traffic situations according to its complexity, which is related to the behavior of the surrounding traffic. Therefore, a traffic complexity measure is obliged. With this measure, it is possible to find the most challenging conditions that can occur in everyday driving and simulate in them a self-driving vehicle and control its performance.

Conducting the test in real life scenarios is costly when not dangerous, so an evaluation in a virtual environment is of great use. One of the most powerful methods to generate the most complex scenarios that can be used is Machine Learning, specifically Reinforcement Learning.

Therefore, the subject of this Thesis is the development of a measure for complexity in traffic scenarios, with the aim to generate difficult situations to test the performance of the automated vehicles in them. The model of the vehicles and the development of the scenarios will be carried out using Reinforcement Learning in Matlab. A comparison with Python will be also carried out, in order to check for differences in means of training performance.

1.1. Automated Vehicles

Autonomous driving vehicles can be classified according to the level of automatization that they are making use of. This classification is divided in six levels, from level zero to level five, both included [1]. The lowest level corresponds to the driving as it is known today, where the driver is in charge of handling all the functionalities and systems of the vehicle. In contrast, the last level would be a fully automated vehicle with no need for intervention of the users. In the most recent developments presented by Audi [2], the automatization stands on the level 3 of the autonomy scale. Another famous brand in terms

of autonomous driving is Tesla, whose autonomous driving system stays in the level 2 at the moment. In Table 1.1, the different stages and their characteristics are presented:

Table 1.1. Stages of Automatization in Vehicles according to SAE [1]

Increasing level of automatization according to SAE [1]

	Level 0	**Level 1**	**Level 2**	**Level 3**	**Level 4**	**Level 5**
Driver	Lateral **and** longitudinal guidance	Lateral **or** longitudinal guidance	**Permanently** monitor system	**Must not permanent ly** monitor system	No driver is required for the specific case of application	No driver needed **from start to finish**
System	Does not intervene	Takes over the function that driver is not performing	Lateral **and** longitudinal guidance	Driver must be potentially able to take over	**Can cope** with all situations in specific applications	Driving task in every situation and road

The difficulties that automated vehicles (AV) involve, not only in the area of technology, but also in the fields of philosophy, morality and social responsibility, fully autonomous vehicles (level 5) will only be able to operate from the year 2040 onwards according to Altenburg [3].

1.2. Objectives of own work

The generation of particularly challenging road traffic scenarios is to be implemented mainly using MATLAB by means of Reinforcement Learning. The first part of the work consists in developing a measure for the complexity of such traffic situations. The formulation of the complexity evaluation can utilize some of the estimates or a mixture of the estimates proposed in the section of state of the art. Since these estimates are limited to an extent by the definition of the environment and the physical system that this Thesis makes use of and taking into account the fact that the measure of complexity has to apply for the whole scenario, some changes and additions are needed. It is important to remark that the environment (such as the highway) and traffic elements and situations that are going to be simulated should be as realistic as possible. Otherwise the real-life application of the results would make no sense.

Once the complexity measure is formulated, the second task of this job begins. The development of complex test scenarios is done in MATLAB. This program has its own Reinforcement Learning packages and toolkits (version 2019a and later versions). The predefined functions and environments will be modified to make be able to train agents under the conditions of the traffic environments. Besides, the action space must be defined in the environment. Finally, the agent will be trained and will obtain the most complex scenarios and their associated complexity.

In order to compare the performance of the MATLAB implementation, Python will be used too to carry out the training in the same traffic situations, using the same algorithms that the ones utilized in MATLAB.

1.3. Work structure

The Thesis begins with the Introduction, which includes the presentation of the automated vehicles and the objectives of this Thesis. Next, a chapter including the state of the art for both the complexity measure and Machine Learning. Machine Learning is divided into its three main methods of learning and for each method it is explained the most relevant parts for this work. The methodology section first analyses the models used for the highway and for the cars and their movements, following it with the implementation of those models in MATLAB and Python. To end with, a discussion of the results obtained in both programming languages and a comparison between the two is carried out. In Figure 1.1, the structure is presented as a scheme.

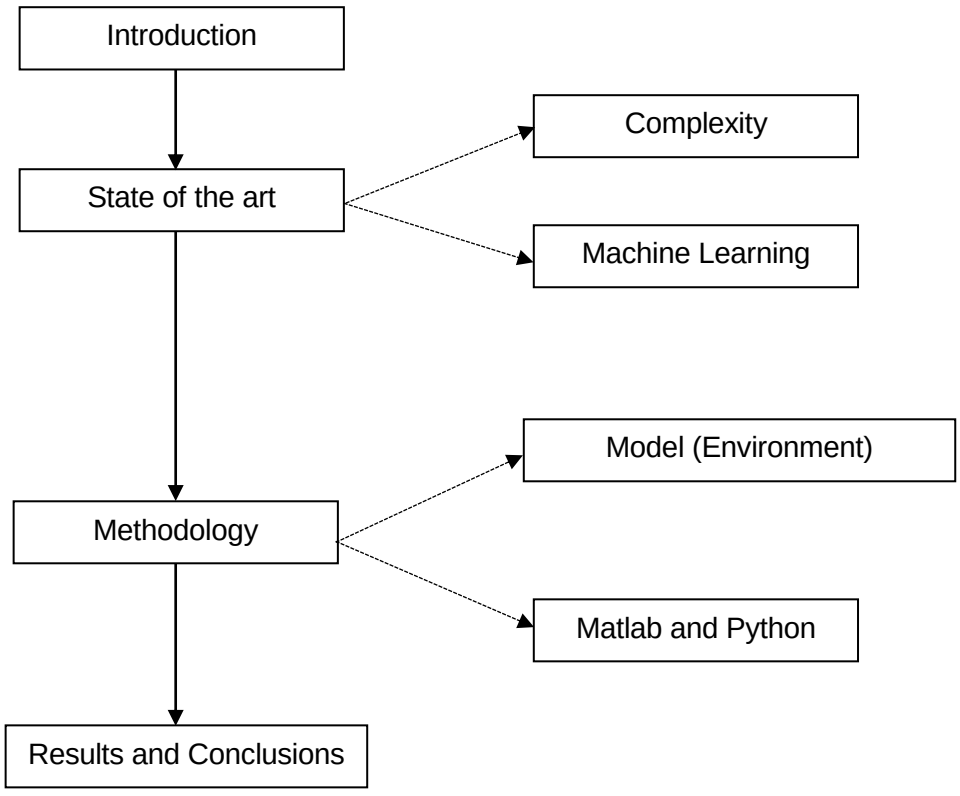


Figure 1.1. Work structure.

2. State of the Art

In this section, the different parts that play an important role in this Thesis are presented. First of all, the complexity term will be studied and bounded taking into account the problem that it is tried to be solved. Next, a short introduction to Machine Learning, its different methods and algorithms will be presented.

2.1. Traffic Scenario Complexity

Complexity is an ambiguous term, which can potentially have many different interpretations. Finding a definition of complexity that satisfies everyone and every imaginable application is practically impossible.

2.1.1. Complexity in general

Trying to find a definition of complex physical systems is a good starting point, as roads, vehicles and other traffic participants (such as pedestrians) are physical entities which combined make up physical systems with many different configurations.

In physics and more concretely in thermodynamics, the most frequent measure used to evaluate the complexity of a defined system is usually characterized as disorder or the decrease of order. The determination of such lack of order or change in it is assessed using the system's entropy [4], [5]. The system entropy is defined as a probability of actions or trajectories that end in a certain state. The higher the number of possible states, the higher the disorder. Being Δ the disorder, Ω the order, k the Boltzmann's constant and p_i the probability of actions, complexity Γ can be measured taking into account the system entropy S and the maximum entropy that can be achieved in that system S_{max} as a monotonically increasing measure of disorder equation (2.4). The coefficients α and β are subjective weights given.

$$S = -k \sum p_i \ln p_i \tag{2.1}$$

$$\Delta = \frac{S}{S_{max}} \tag{2.2}$$

$$\Omega = 1 - \Delta \tag{2.3}$$

$$\Gamma = \Delta^{\alpha} \cdot \Omega^{\beta} \tag{2.4}$$

Taking into account that human behavior is challenging to model based on probabilities, the complexity of a traffic situation would be better defined by the actual disorder caused by actions, rather than be based on the probabilities of those actions.

Complexity is also linked with the number of parts of a system, but more importantly with the correlations between these components that shape a system [6]. This approach is related somehow to a traffic scenario, in the sense that the action of any actor present in the situation would presumably influence the other participants, provoking their (re)actions. This would show that both the action and the reaction are indeed correlated.

In addition, the correlation of the environment and the system is of great relevance for the complexity of a given situation.

2.1.2. Complexity in traffic

Focusing in the problem of road complexity, the adversity of the situation usually depends on the environment and the traffic participants of the scene. So, it is possible to divide the complexity in two parts: the Road Semantic Complexity (RSC) and the Traffic Element Complexity (TEC) [7]. RSC groups the environment complexity and includes the road type, the scene type (intersection, tunnel, etc.), and the challenging conditions (overtaking, pedestrian, etc.). The traffic participants conform the TEC, which depends on the distance from car to car and the angles between them.

The RSC is calculated using the Support Vector Regression (SVR) [8] based on the number of tasks in off-line testing *N*, the number of participants *M*, and the accuracy of the prediction and recall $F_{ij}^{(t)}$:

$$C_R^{(t)} = 1 - \sum_i^N \sum_j^M \frac{F_{ij}^{(t)}}{NMmax(F_i^{(t)})} \quad (2.5)$$

The TEC only depends on the eight possible actors surrounding the autonomous vehicle, the distance D_j to car *j* and the minimum angle A_j to car *j*:

$$C_E = \frac{1}{8} \sum_{i=1}^N \alpha e^{-(\lambda D_n \cos(A_n))} + \beta e^{-(\lambda D_n \sin(A_n))} \quad (2.6)$$

With these two measures and giving each one of them a relative weight λ_1 and λ_2, it is possible to evaluate the complexity (given that $\lambda_1 + \lambda_2 = 1$):

$$C = \lambda_1 C_R + \lambda_2 C_E \quad (2.7)$$

Other approaches only make use of the RSC [9], using a Support Vector Machine that makes use of Supervised Learning in order to calculate the complexity, maximizing the output score of the SVM classifier. In addition to this, lane tracking [10] is difficulted when the road is not continuous, i.e., where there happen to be sharp turns, lane changes, atypical road geometries, etc. As the road is described using the lane width, the horizontal and vertical curvatures, the lateral offset, and the own vehicle pitch, roll and yaw angles, these discontinuities only make it more difficult for the self-driving vehicle to identify the lanes. In Figure 2.1 [10], a complex traffic scene according to these definitions can be observed. In the image, there are many vehicles that block the view of the automated vehicle, making it more difficult to recognize the lanes.

On the other hand, data acquired from vehicles can be utilized to define the measure for complexity [11]. The driving situation complexity (DSC) is calculated finding the perplexity (how well a given sample is predicted by a probability distribution) and the standard deviation of the data. In this particular case, perplexity is the average number of contenders identified in the sensing area of the vehicle

$$DSC_t = \lambda \prod_{i=1}^M \left(\frac{Perplexity_{m,t}}{Perplexity\_mean}\right) + (1-\lambda) \sum_{i=1}^M \left(\frac{\sigma_{m,t}}{\sigma\_mean}\right) \quad (2.8)$$

where *M* is the number of variables and λ is a correlation factor independently selected.

Figure 2.1. Complex city scene [10]. The recognition of the lanes is difficulted by the close distances to other traffic participants and the great number of them.

In addition to these approaches, the criticality of the traffic situations can also have an impact on the complexity of those scenarios, especially when the driver is an automated vehicle. The criticality can be assessed making use of the time to collision (TTC) [12], which evaluates the threat levels for a self-driving vehicle based on the calculation of the remaining time to collide with the preceding vehicle in the case that no action was performed. In this Thesis, no action means driving with constant velocity and no lane changes.

In addition to the TEC, sometimes it is also possible that the vehicles ahead the leading vehicles (vehicles 3, 5, and 8 in Figure 2.2) have an impact on the behavior and security of the autonomous vehicle. For instance, if the difference in speed between the vehicle 5 and its preceding vehicle is large, vehicle 5 might need to change its lane in order to avoid collision. Then, it can occur that the autonomous vehicle must change its lane. So, the influence of the +1 vehicles in Figure 2.2 is explained.

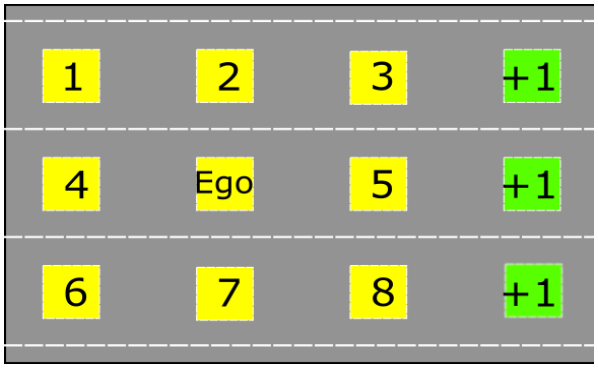


Figure 2.2. Surrounding positions of the AV(EGO) that potentially have an impact on [44]..

2.2. Machine Learning

Machine Learning is a subgroup of Artificial Intelligence, that is based on the implementation of algorithms and statistical methods with the aim of developing techniques that allow machines (computers) to *learn*. It is said, that a machine learns when its performance gets better with the experience it gathers. The general idea behind these algorithms is the collection of data for the problem and use them to solve the task. The main advantages against traditional optimization methods are the robustness in solving

complex tasks, the reliance on *real-world* data, and the capacity to adapt to new situations. [13]

The different algorithms that constitute Machine Learning can be divided in three main different groups: Supervised Learning, Unsupervised Learning, and Reinforcement Learning, depending on how the machine performs the task of learning.

The formulation for all three approaches can be defined as a (complex) relation between inputs and outputs. In short, it is tried to find the best output *o* to produce for each input *i*. Although this objective is common for the three approaches, the information that is available changes from one to another. If there is a set of data were the inputs and its outputs are known, the problem can be solved using Supervised Learning. In other cases, it is possible that only the inputs are available and therefore the most suitable tool would be Unsupervised Learning. Reinforcement Learning is used when there is no direct knowledge of the "best" output, but it is feasible to evaluate the quality of an output *o* when an input *i* is observed [14].

2.2.1. Supervised Learning

As stated before, Supervised Learning maps a known set of inputs *I* with a set of inputs *O*, which are known beforehand. This information is usually divided in three groups, that have different functions. [14]

The first group is the *training dataset*, which is used to construct a function (or various functions) or model inferring its parameters to relate inputs and outputs. The procedure goes as follows: the functions are run with this dataset and their results are compared with the output or *target*. The performance is evaluated and the parameters from the model are adjusted based on the results of this evaluation. Depending on the learning algorithm, the functions are conditional probability models or joint probability models. [14]

The second group of data is the *validation dataset*, which is used to tune the hyperparameters (parameters whose values are established before the training begins and have an impact on the speed and performance of that process) of the algorithm and avoid overfitting (the model fits too tight the training data and cannot be used to predict as the model cannot generalize from those data), choosing the best function/model trained by evaluating its performance. In order to choose the best functions, there are two basic approaches: empirical risk minimization (seeks the function that is better fitting the training data) and structural risk minimization (controls the variance/bias [15] trade-off introducing a penalty function).

Finally, the chosen function/algorithm is evaluated using the third dataset: the *test dataset.* This dataset should be independent of the other two datasets, but its probability distribution should be the same as the distribution in the training one. In this last step, the evaluation of the characteristics of the algorithm is obtained, such as the accuracy or the sensitivity. Accuracy is the ratio between correct predictions to the total number of inputs. Sensitivity is the probability of getting a correct test result when given a correct positive example. This process can be applied in early stopping, where the training is finished as the error of the validation set grows and the previous model with smaller error is chosen. [14]

While this procedure is common for all the algorithms and problems that are part of Supervised Learning, the problems can be classified according to the data type that is used as input/output. The problems are either categorical, meaning that the variables used are non-numerical, or numerical ones. As an example, an image recognition algorithm that identifies whether there is a dog or a cat in a picture is a categorical problem. To deal with this kind of problems, it is possible to convert the categorical variables into numerical ones using certain techniques of data pre-processing, such as one hot encoding [16].

One of the key objectives of Supervised Learning is to develop an optimal algorithm that is capable of classifying new data that it has never seen. This calls for the learning algorithm to be able to derive from the training data in an "acceptable" manner (Figure 2.3 [17]).

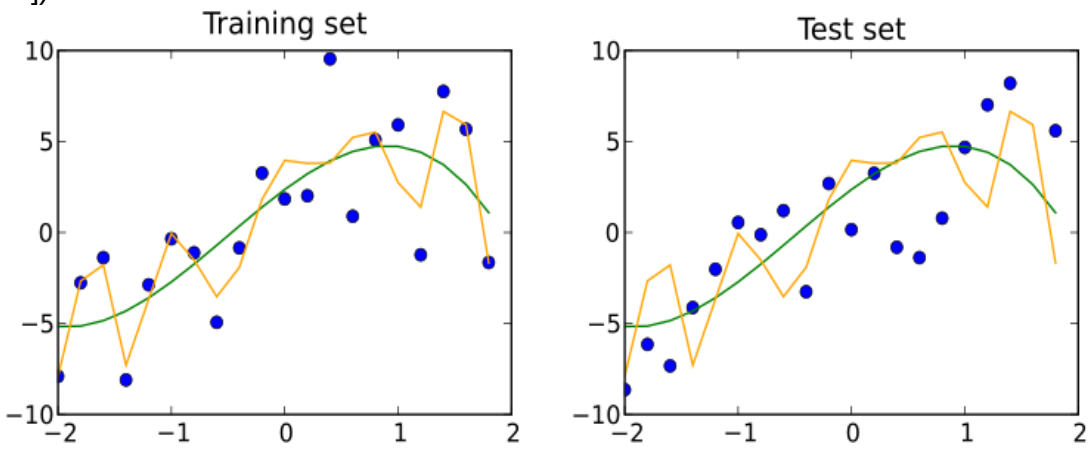


Figure 2.3. Comparison between training and data sets fitting [17]. Blue dots represent the data from the training and test sets, the orange and green lines represent models used to fit the data. X axis represent position and Y axis represents speed.

In Figure 2.3, there is a representation of a training set and a test set (blue dots), with two predictive models fitting each one of the sets (orange and green lines). While for the training set, the orange model results in an MSE of 4 and the green of 9, in the test set the MSE is 15 and 13 for the orange and green fits respectively. It can be concluded that the orange model is worse than the green one, since the orange overfits the data when compared to the green in terms of MSE.

A big range of Supervised Learning algorithms can be used, each with its own advantages and disadvantages and, depending on the problem it is faced, some may work better than others for that particular problem, but there is not one superior algorithm to the others for all kinds of supervised learning problems. Some of the questions that must be considered are the bias-variance dilemma [15], the complexity of the function and the amount of training data, the dimensionality of the input space, noise in the output values, and the heterogeneity, redundancy, and presence of interactions in the data.

Taking these factors into account, it is possible to determine for the problem at hand which algorithm works best. Some of the algorithms used are:

- Support Vector Machines
- Linear/Logistic Regression
- Naive Bayes

- Linear discriminant analysis
- Decision trees
- Neural Networks
- K-nearest neighbours

2.2.1.1. Neural Networks

Neural Networks are the most known tool for Supervised Learning when it comes to more complex and/or tasks that have high computational cost. The Neural Networks are an imitation of what the network of an animal brain looks like. The Neural Network is composed of nodes called neurons that are connected between them, and a group of neurons that are in the same level is called a layer (Figure 2.4 [18]).

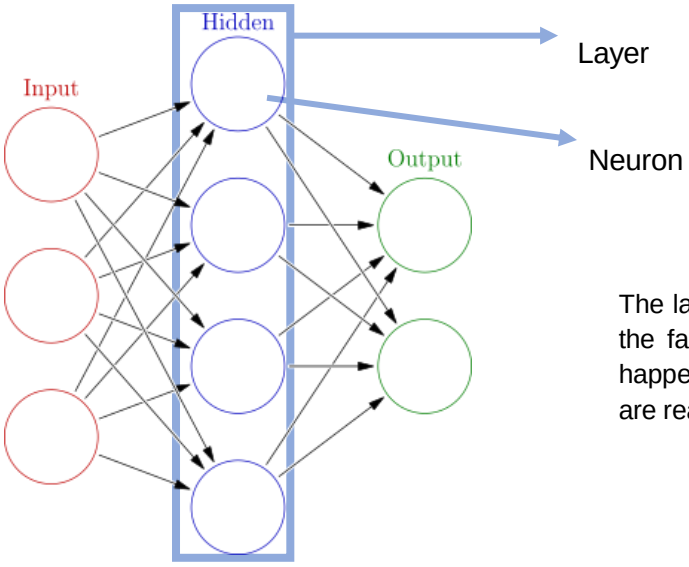


Figure 2.4. Representation of a Neural Network [18] in which are shown the neurons from the layers, connecting the inputs and the outputs

In the beginning, the functioning of the Neural Network was designed to solve problems in the same way that a brain of a human would do. Nonetheless, the power and applicability of this approach have increased and currently being used to solve problems like, for instance, computer vision, playing video games (in combination with Reinforcement Learning), or speech recognition. The way the algorithm works, can be divided in two phases: propagation and weights update. The outputs y are a function of the inputs *x* and the weights w that every node has: $y = f_N(w, x)$. There are certain formulas needed for the gradient of the function of weights, that update them in order to get the correct prediction; and for the propagation functions, which are functions used to transfer values through all neurons of a Neural Network layer.

To start with, making use of the propagation functions, the inputs go through the network generating the output value(s). Then, the error term (cost) is calculated and the calculated outputs are propagated back through the network in order to obtain the deltas (difference between the target and the achieved values) of all outputs and neurons of the network [19]. An example of a quadratic error term or quadratic cost term is shown in

equation (2.9). E_j^r is the desired output of neuron *j* of a training sample *r* and a_j^L is the activation value of that neuron *j* of the layer *L*.

$$C = 0.5 \sum_j (a_j^L - E_j^r)^2 \tag{2.9}$$

After, the weight's output delta and input activation (the internal state of the neurons) are multiplied to find the gradient of the weight. Finally, a percentage of the delta is subtracted to the weight, updating it. This percentage is called the learning rate and it determines how fast and precise the learning is. It is important to remark, that the sign of the delta indicates whether the error grows directly or inversely to the weights. That is the reason why the weight must be actualized in the contrary direction, descending the gradient (gradient descent: an optimization algorithm that is used to find the minimum of a function). Some variations can be made to this procedure in order to improve the precision and/or the learning speed, avoid oscillations, and avoid getting stuck: the adaptive learning rate [19] and introducing inertia, so the weights are updated taking into account the previous change.

Another component that needs to be selected to train a Neural Network is the learning mode. In stochastic learning [20] every input makes the weights to adjust. In contrast, batch learning [21] adjust the weights from a batch of inputs, aggregating the errors over the batch. The difference in terms of performance of both methods relies in that the batch method produces a faster and more balanced descent to a local minimum, while the stochastic method nearly avoids any chance of getting stuck in local minima. Since each approach has an advantage that would be convenient to use a compromise between the two can be achieved creating mini batches, which are batches of stochastically chosen inputs.

2.2.1.2. Convolutional Neural Networks

As expected, the basic algorithm of Neural Networks has variants such as group method of data handling, long short-term memory or auto-encoders. Convolutional Neural Networks (CNN) [22] are of unique importance in this Thesis, since a CNN is applied in combination with Reinforcement Learning.

As the name indicates, the core of CNNs is the convolutional layer, which at the same time receives this name from the operation it performs: a convolution. In the framework of CNNs, a convolution consists of a dot product (also called a scalar product), multiplying the array of inputs with a two-dimensional set of weights called a filter or kernel. The fact that the kernel is smaller than the input data enables the same kernel to be multiplied by the input array at different points on the input various times. More precisely, on each overlapping part of the input array the kernel is applied. The result of these multiple scalar products is a two-dimensional array called feature map made of the output values that represent a filtering of the input. In Figure 2.5, this process is represented:

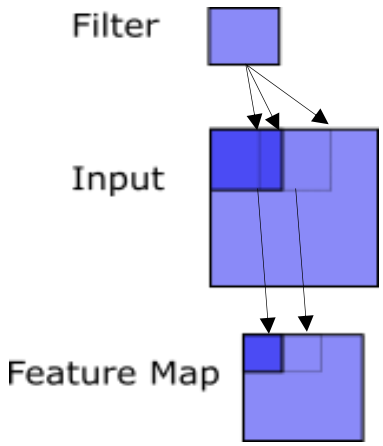


Figure 2.5. Example of how a filter (kernel) is applied to the inputs, ending in the feature map.

The main problem with this approach is that the feature maps are responsive to the location of the features in the input, meaning that moving the position of the feature in the inputs will result in a completely different feature map. To solve this issue, pooling layers provide a solution based on down sampling, treating each feature map independently. As a result, the layers within a CNN that are repeated are ordered and a new set of the same number of pooled featured maps is created. There are two common pooling methods: max pooling and average pooling. The first sums up the most activated presence and the second the average presence of a feature. The usual size of the operation is a 2x2 pixel applied with a stride of 2 pixels. Figure 2.6 [24] shows an example of both pooling methods.

The last element of CNNs are the non-linearity layers, that are applied to each element of the feature map and do not change its size. Frequent functions are the sigmoid and the hyperbolic tangent function. To avoid negative values, non-linearity functions are rectified using the Rectified Linear Units (ReLU). For this reason, the non-linearity layers are often referred to as ReLU layers. These layers are located after the feature maps and before the pooling (Figure 2.7).

One of the main applications of CNNs is object recognition. Images can easily be divided in pixels. The height and the width of an image measured in number of pixels in addition to the RGB colour are the inputs to the CNN. Many times, the colour is reduced to greys in order to reduce the dimensionality of the input. Another application of CNNs, again in combination with Reinforcement Learning, is learning to play videogames, such as Atari Breakout [23].

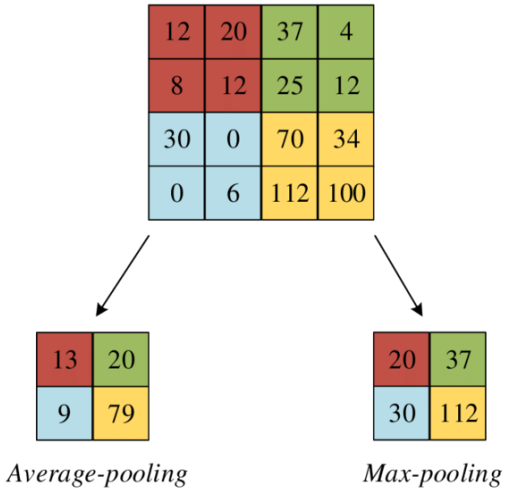


Figure 2.6. Example of max and average pooling [24].

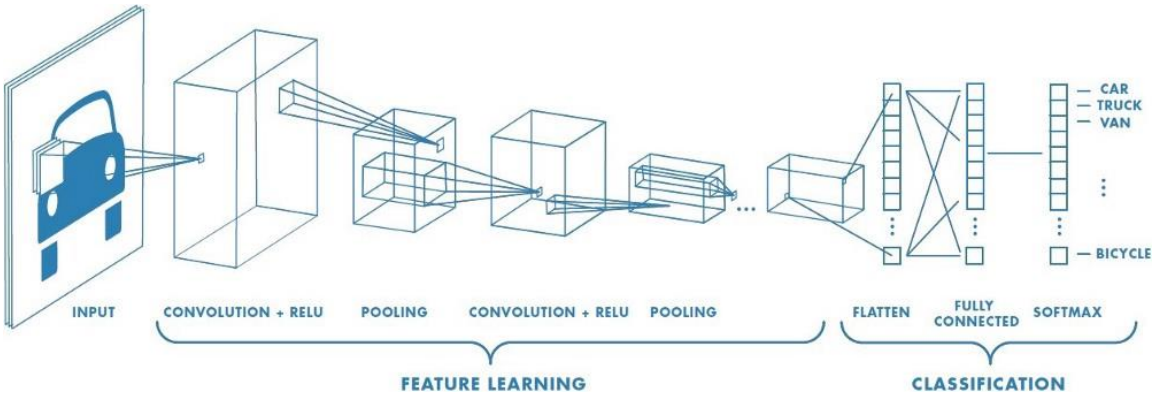


Figure 2.7. Example of a CNN applied to object recognition [25]. There are convolutional and ReLU layers combined before every pooling.

2.2.2. Unsupervised Learning

In contrast to Supervised Learning, in Unsupervised Learning the outputs are not known. In this case, it is tried to discover the underlaying patterns that the inputs have. The most common uses of this part of Machine Learning are clustering, anomalies detection, association mining and latent variable models.

Clustering divides the dataset into groups according to similarities found by the algorithm. This way, anomalies in the data can be detected. The main algorithms that can be used for this purpose are Hierarchical Clustering, K-means, Mixture Models, DBSCAN, and OPTICS algorithm, being Hierarchical Clustering and K-means the most utilized ones. As an illustration, Figure 2.8 [26] shows a possible result of these algorithms. As it can be observed, the result is very visual and can be easily used to categorize the data.

For anomaly detection the algorithm of Local Outlier Factor is also used and for learning latent variable models, the most known algorithms are the Principal Component Analysis (PCA) and the Method of Moments. The aim of PCA is to change a group of variables that are possibly correlated into the principal components, a set of values of linearly uncorrelated variables. Thus, the variability of the data is explained through the

principal components, being the first component the one that explains the most of it. It is important to note, that the succeeding components are orthogonal to the preceding components, so the different components are indeed linearly uncorrelated.

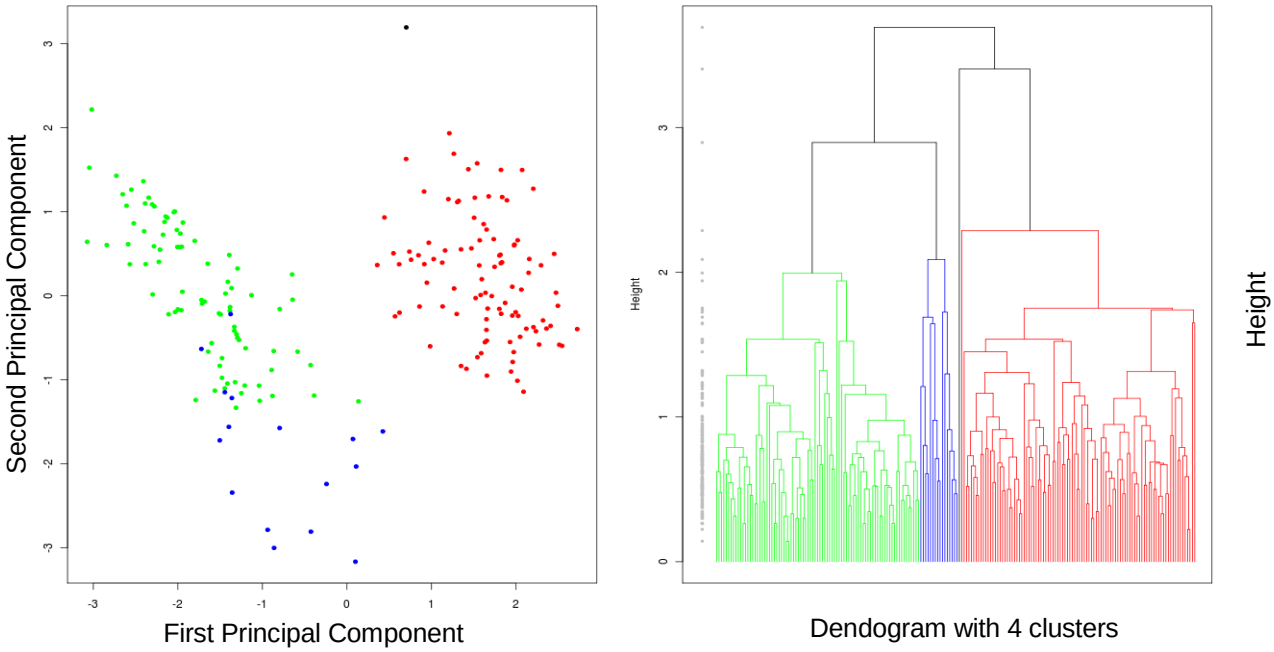


Figure 2.8. Hierarchical (right) and K-means (left) clustering [26]

The Method of Moments [27] tries to estimate the unknown parameters of the model based on the relation between the parameters and the moments of the random variables. The moments for each variable are its mean and its matrix of covariance. This method is of particular interest when learning the parameters of latent variable models [28].

One of the main applications of CNNs is object recognition. Images can easily be divided in pixels. The height and the width of an image measured in number of pixels in addition to the RGB colour are the inputs to the CNN. Many times, the colour is reduced to greys in order to reduce the dimensionality of the input. Another application of CNNs, again in combination with Reinforcement Learning, is learning to play videogames, such as Atari Breakout [23].

2.2.3. Reinforcement Learning

Reinforcement Learning (RL) is considered one of the best-known paradigms for exploring learning. Basically, the principle of this model is based on the interaction of a learning agent (the component that makes the decision of what action to take and learns) with its environment (space where the agent makes its decisions) to reach a specific goal [29]. The agent does not receive which actions, and in which order it should choose them, instead it should try to find out which actions receive the greater reward. In many occasions, the actions have not only an influence in an immediate reward but also in the next rewards. The delayed reward together with the exploration in form of trial and error are the most distinguishing features of Reinforcement Learning [29]. Related to them, the biggest challenge of Reinforcement Learning emerges, as the agent has to receive a great

number of rewards taking actions from past attempts that effectively generate rewards for it, which is called exploitation. But it is also needed to discover the rewarded actions, so the agent has to try actions that have not been tried before exploring all the possibilities. So, a compromise between the two has to be achieved, but there is not a definite solution. The overall challenge is called the Exploration-Exploitation Dilemma.

In comparison with Supervised and Unsupervised Learning, in Reinforcement Learning is not known the best output for a given input, but the quality of the output can be determined. In the situations where the agent cannot learn from some previous examples given to it, such as the training dataset in Supervised Learning, the agent must be able to learn from its own experience. Besides, a Reinforcement Learning agent does not recognize patterns in data like an agent from Unsupervised Learning. Instead, Reinforcement Learning tries to maximize a reward.

In Reinforcement Learning, the agent interacts with an environment. To do so, the agent receives an observation of the state (possible circumstances of the environment) of the environment and then executes an action according to it. This action results in a change in the environment and has a reward associated with it [30]. In Figure 2.9 [29], a representation of the interaction between an agent and the environment is presented.

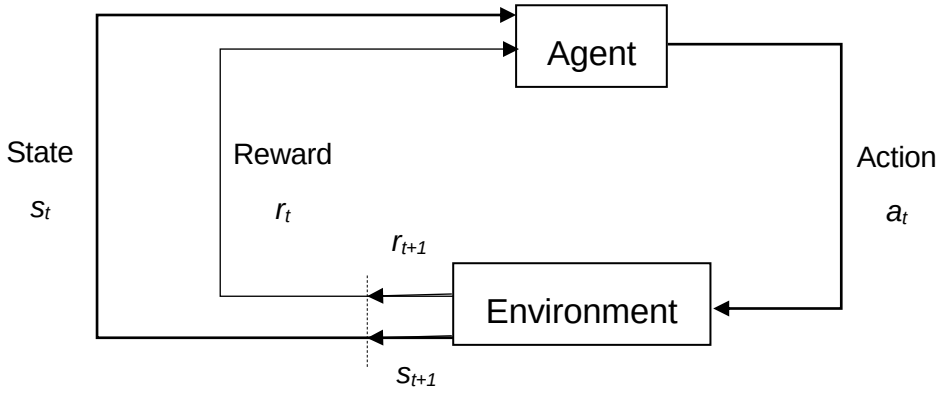


Figure 2.9. The agent-environment interaction [29]

The factors that compose a Reinforcement Learning problem are the following:

- The environment, in which the agent performs its actions making use of the trial and error principle. The agent must be able to observe the environment, so the better the observation, the greater the influence of the true state of the environment on the choice of actions [30]. The environment is represented by the state $s_t \in S$, the agent can do determined actions $a_t \in A(s_t)$ as a function of the actual state. After the action has been chosen in time t, the agent receives a reward $r_{t+1} \in R$ and the environment is in a new state s_{t+1}. This state s_{t+1} depends on the previous state s_t and the chosen action a_t [31].

- The policy π is the strategy that the agent uses to choose the actions. It associates every state with the probabilities of choosing any action that is possible in a certain state, so $\pi(s,a)$ is the correct form of a policy. Finding the optimal policy that maximizes the reward of the agent in the long run is the goal of RL. In general, the strategies to accomplish it are stochastic. It is also possible that there is more than one optimal policy π^*. A policy π' is better than other policy π'' when the expected return values for all states in π' is

greater or equal to the ones in π' [31]. A good policy is the one that makes the agent reach a state with better results and to evaluate this, the value function $V_\pi(s)$ (later explained) is utilized. Considering what has been explained, the optimal policy is (2.10):

$$\pi^* = argmax\, V_\pi(s)\ \forall s \in S \tag{2.10}$$

- The reward function or return R_t. After every action, the agent receives a reward. This reward acts as a signal for the agent, telling it which actions are good and which ones are bad actions. The reward is decisive for choosing the optimal policy, since a low or negative reward makes the agent change the policy [29]. To maximize the long-term reward after the actual time *t*, the reward function is determined as in equation (2.11) [29] for a finite case with ending time *T*.

$$\mathrm{R}_t = r_{t+1} + r_{t+2} + r_{t+3} + \cdots + r_T = \sum_{k=t+1}^{T} r_k \tag{2.11}$$

It is also possible that there is not an ending time, then a discount factor $\gamma \in [0,1]$ is used (2.12) [29]. This way, future rewards have less importance than the actual reward. The closer γ gets to 1, the greater the influence of the future rewards. The agent then becomes "far-sighted". The equation (2.12) can be further developed as in (2.13), showing that the returns in a time *t* depend on the returns of successive periods.

$$\mathrm{R}_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{2.12}$$

$$\mathrm{R}_t = r_{t+1} + \gamma(r_{t+2} + \gamma r_{t+3} + \cdots) = r_{t+1} + \gamma R_{t+1} \tag{2.13}$$

The equation (2.13) makes it easier to calculate returns, but it is not enough to maximize the long-term return just to look at the current state s_t. For this, the value function is needed.

- The already mentioned value function $V_\pi(s)$ is used to estimate how good the actual state is, and which actions should the agent choose. In short, the value function is the expected return starting from state $s_0 = s$ and following policy π (equation 2.14). Thus, the value function tells how good or bad it is to be in a certain state *s*.

$$V_\pi(s) = \mathbb{E}[R_t|s,\pi], \forall s \in S \tag{2.14}$$

A variation from the value function is the so-called q-value function $Q_\pi(s,a)$ shown in equation (2.15). The utility of this function is the same as the value function, with the added feature of taking actions into account too.

$$Q_\pi(s,a) = \mathbb{E}[R_t|s,\pi,a], \forall s \in S, \forall a \in A(s) \tag{2.15}$$

Similar to the relationship for the return R_t and the future returns, the value function can be described as a relationship between the value of the current state *s* and the next state *s'*, called the Bellman Equation (2.16) [32]. The equation affirms that the value of the starting state must be equal to the discounted value of the next state and the expected reward. Being π^* the optimal policy, the value and q-value functions are represented as in equations (2.16) and (2.17) respectively.

$$V_\pi(s) = \mathbb{E}[R_t|s,\pi] = \mathbb{E}[r_{t+1} + \gamma R_{t+1}|s,\pi]$$

$$= \sum_a \pi(a|s) \sum_{s'} \sum_r p(s',r|s,a)\left[r + \gamma \mathbb{E}[R_{t+1}|s']\right]$$

$$= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)[r + \gamma V_\pi(s')], \forall s \in S \quad (2.16)$$

$$V^*(s) = max\, V_\pi(s), \forall s \in S \quad (2.17)$$

$$Q^*(s,a) = max\, Q_\pi(s,a), \forall s \in S, \forall a \in A(s) \quad (2.18)$$

The optimal q-value function can be presented as a function of the optimal value function as in equation(2.19):

$$Q^*(s,a) = \mathrm{E}[r_{t+1} + \gamma V^*(s_{t+1})|s,a] \quad (2.19)$$

A good demonstration on how the value functions work can be produced with a 4x4 grid [30]. The agent can be in every square of the grid, each one representing one state. The aim of the agent is to get to the top left or bottom right squares. There are four possible movements: up, down, left, and right. For each movement that the agent makes, it receives a reward of -1. In Figure 2.10 (a), it is shown in each square the expected values of each state following a random policy. For instance, when the agent starts in the second square of the first row, it takes 14 actions to get to the objective squares. Figure 2.10 (b) shows the optimal policy values, which needs to be approximated using suitable methods and in Figure 2.10 (c) the optimal policy is represented.

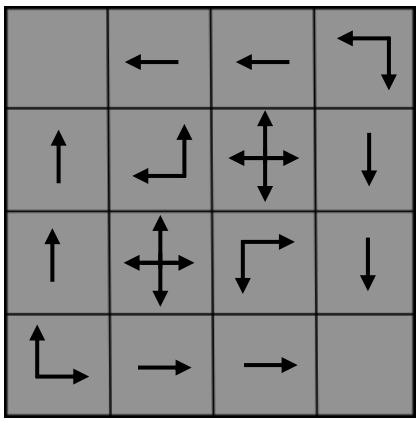
0	-14	-20	-22
-14	-18	-22	-20
-20	-22	-18	-14
-22	-20	-14	0

(a)

0	-1	-2	-3
-1	-2	-3	-2
-2	-3	-2	-1
-3	-2	-1	0

(b)

Figure 2.10. Random policy (a) and optimal policy (b) values [30]



(c)

Figure 2.10. Optimal policy [30]

2.2.3.1. Markov Decision Process

There are many possibilities available to solve a RL problem. One instance to describe those problems is the Markov Decision Process (MDP) [33, 34]. In an MDP it is required that the environment has a finite set of states and actions for each possible state. The MDPs need to satisfy the Markov Property (equation (2.21)): "the future is independent of the past given the present". That means, that a state s_t is Markov if and only if the previous states s_1 to s_{t-1} can be discarded:

$$P[s_{t+1}|s_t] = P[s_{t+1}|s_1, \dots, s_t] \tag{2.20}$$

The simplest MDP is a Markov Process [34]. As said, the number of states is finite, and all the probabilities of going from one state to another are grouped in rows forming a State Transition Matrix. In these processes, there is not a goal or objective nor a reward or value for being in a state. If they are added, the Markov process develops into a Markov Reward Process, that also counts with a discount factor γ and a reward function as in equation (2.13). The value function can be used in the same way that it was explained before, but without the policy π. In the moment that the policy is added, the agent also must make decisions, so now it is a Markov Decision Process.

The algorithm for MDPs works in two steps. In the first step the values are updated and in the second one, the policy is updated. These steps are repeated in order for all the states until there are no more changes. The most important variants to the algorithm of the MDPs are the value iteration, where the policy function is not used, and it is calculated within the value function combining the two steps into one; and the policy iteration, where the first step is performed and the second one is repeated until it converges. After, step one is repeated and so on. [34]

Based on this functioning, there are different methods that can be applied to solve a Reinforcement Learning problem.

2.2.3.2. Dynamic Programming

Dynamic Programming [35] breaks complex problems down into smaller ones to solve them. The two main requirements for Dynamic Programming are the optimal substructure

and the overlapping sub-problems. Optimal substructure refers to the fact that the optimal solution of the smaller problems can be utilized to solve the bigger problem. When breaking down the problem, many sub-problems can occur many times (overlapping). Therefore, solutions to sub-problems should be able to be saved and reused.

Besides, two kinds of problems can be distinguished. The prediction problem [36] starts with a given policy, for which it is needed to find the value function. That means, evaluate how good the given policy is. This process is called policy evaluation. In control problems, the goal is to find the optimal policy and value function, i.e. find what is the best thing to make (policy iteration).

- *Policy evaluation* bases on applying iteratively the Bellman Expectation backup. For the given policy, the problem is started with a value function V_1 with a value of 0. Next, using the Bellman Equation, a new value function V_2 is calculated. The process is repeated until the value function converges to V_π. The use of synchronous backups allows to consider all states at every step: if *s* is the current state and *s'* is a successor state, at each iteration *i+1* the policy $V_{i+1}(s)$ is updated from $V_i(s')$, which has been calculated using the Bellman Equation. The process is repeated until convergence to the value function of the policy V_π. Figure 2.10 (a) showed the result of starting with a random policy, which is far away from the optimal policy values in Figure 2.10 (b).

- The solution to this problem is the *policy iteration* from MDPs, which consists in applying the Bellman Equation to iterate the policy and *acting greedily*.

Using the before mentioned policy evaluation, the policy π is assessed. The policy can be improved *acting greedily*: look one step ahead to find the action that gives the maximum q-value function. Then, the policy is updated to a new policy π'. This process is repeated till the convergence to the optimal policy π^*.

- Another approach of MDPs to solve control problems is *value iteration*. It serves the same purpose as the policy iteration, but in this case the Bellman Optimality Equation [37] is used. The principle of optimality states that any optimal policy reaches the optimal value from a state *s*, $V^*(s)$, if and only if for any state *s'* that is a successor of *s*, the policy π achieves the optimal value from *s'*, $V^*(s')$. The approach is similar to the policy evaluation making use of synchronous backups. Value iteration uses the value function instead of the q-value function as in policy iteration. The difference between value iteration and policy evaluation relies in that the policy iteration has the advantage that it has a stopping condition (when the policy array does not change when applying step one to all states), but it is slower when the number of possible states is large.

One of the conditions that are needed to solve Reinforcement Learning problems using Dynamic Programming is that the environment is fully known. But this fact is not always possible. Therefore, in problems were a model of the environment is not available other methods must be used.

The principal methods for model free predictions are the Monte-Carlo Learning and Temporal-Difference Learning.

2.2.3.3. Monte-Carlo Learning.

Monte-Carlo Learning [38] learns directly from episodes of experience. These episodes must be complete before learning. Another drawback in comparison to Dynamic Programming is that Monte-Carlo does not use bootstrapping (use one or more estimated values to update the same kind of estimated value). Monte-Carlo policy evaluation uses the empirical mean return instead of expected return (equation (2.13)). There are two ways of evaluating the value function of a policy. The First-Visit policy evaluation asses the value of state *s* of a policy, which is given. The *first* time-step *t* that the state *s* is visited in an episode, the counter of number of times visited of that state *N(s)* is incremented by 1. Next, the total return of that state *S(s)* is incremented by R_t (the return in time-step *t*) and the value is estimated by the mean return, as in equation (2.21). As *N(s)* grows to infinity, *V(s)* is closer to $V_\pi(s)$.

$$V(s) = {S(s)}/{N(s)} \tag{2.21}$$

The second approach to policy evaluation is the Every-Visit. It works exactly the same as the First-Visit method except for the fact that the counter *N(s)* is updated in *every* visit to that state *s*, instead of only in the *first* visit. In both of these methods is necessary to keep record of the statistics of the algorithm, meaning that the value can only be obtained after all episodes have been completed. This problem can be dealt with the incremental mean equation (2.22) applied to the Monte-Carlo method in equation (2.23). μ_k represents the mean and x_k the input.

$$\mu_k = \mu_{k-1} + \frac{1}{k}(x_k - \mu_{k-1}) \tag{2.22}$$

$$V(s_t) \leftarrow V(s_t) + \frac{1}{N(s_t)}(R_t - V(s_t)) \tag{2.23}$$

Sometimes, it is not required to remember things that happened in the past. This occurs with non-stationary problems. Equation (2.24) shows what is recommended to use: a running average approach, being α the learning rate.

$$V(s_t) \leftarrow V(s_t) + \alpha(R_t - V(s_t)) \tag{2.24}$$

2.2.3.4. Temporal-Difference Learning

Temporal-Difference Learning methods [39] learn from the interaction with the environment or experience. One advantage from this method in comparison with Monte-Carlo Learning is the utilization from bootstrapping, so it is not necessary that the episodes end before learning. A good example to illustrate the difference between the two models is if we tried to estimate how long does it take to walk home from university. In Monte-Carlo, the states would receive its value once we reached home (actual result). In contrast, in Temporal-Difference the value would be updated along the way at each state based on the impact that the next state has on the current one (estimated result).

The simplest Temporal-Difference algorithm is TD(0). TD(0) updates the value function $V(s_t)$ to an estimated return in equation (2.25) instead of to the real return R_t.

$$V(s_t) \leftarrow V(s_t) + \alpha[R_{t+1} + \gamma V(s_{t+1}) - V(s_t)] \quad (2.25)$$

$R_{t+1} + \gamma V(s_{t+1})$ is the TD target and when subtracting the actual value $V(s_t)$ the TD error is obtained. While Temporal-Difference has low variance and some bias as the TD target is calculated with one random action, transition and reward; Monte-Carlo has high variance and no bias, since the target is the actual return R_t, which is calculated with a great number of random actions. In conclusion, Monte-Carlo is less sensitive to the initial value and good convergence characteristics while Temporal-Difference is generally more efficient but does not always converge and it is more sensitive to the initial value.

In addition to the way of calculating the value of each state as presented in equation (2.26), it is also feasible to do it taking a one-step look to all other possible states and compute the value of the current state as an expected value (2.26). Furthermore, an n-step Temporal-Difference learning can be used as a compromise between TD(0) and Monte-Carlo. The value will be determined looking n-steps ahead and implemented in Temporal-Difference algorithm (2.27).

$$V(s_t) \leftarrow \mathbb{E}[R_{t+1} + \gamma V(s_{t+1})] \quad (2.26)$$

$$V(s_t) \leftarrow V(s_t) + \alpha(R_t^{(n)} - V(s_t)) \quad (2.27)$$

The more advanced Temporal-Difference learning methods will now be explained, as they present some advantages and other uses as the basic TD(0).

1. *TD(λ)* is a more generic learning method than Temporal-Difference Learning or TD(0). TD(0) is a TD(λ) with λ = 0. The difference is in the way that the return is calculated. Instead of a n-step return $R_t^{(n)}$, a decaying weighted sum is utilized to combine all n-step returns (2.28) and to calculate the actual return (2.29).

$$R_t^{\lambda} = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} R_t^{(n)} \quad (2.28)$$

$$V(s_t) \leftarrow V(s_t) + \alpha(R_t^{\lambda} - V(s_t)) \quad (2.29)$$

The forward view of equation (2.30) needs of complete episodes to update values. To fix this issue, the backward view provides the mechanism to update the value from incomplete episodes. The value is updated for every state s proportional to the TD error in timestep t (δ_t) and the eligibility trace E_t (assigns more credit to most frequent and to most recent states).

$$V(s) \leftarrow V(s) + \alpha \delta_t E_t(s) \quad (2.30)$$

$$E_t(s) = \gamma \lambda E_{t-1}(s) + 1 \mid E_0(s) = 0 \quad (2.31)$$

2. An on-policy method (the algorithm is concerned about the policy which yielded past decisions) from Temporal-Difference is *SARSA* which receives its name from the acronym of the state s_t, the action a_t, the reward r_t, the successor state s_{t+1}, and the successor action a_{t+1} [40]. It's a control algorithm, so only the q-value function must be optimized, as in the policy iteration method. The objective is to determine for all pairs of states s and actions a the q-value function $Q_\pi(s,a)$ for the policy π. Even though the process is the same as in TD(0), instead of calculating the value taking into account

transitions from one state to another, the transitions that are considered are from state and action pair to another state and action pair. The update of the q-value function in equation (2.32) is analogous to the update of the value function in TD(0) in equation (2.26).

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] \quad (2.32)$$

The update rule in equation (2.32) is only used for non-terminal states. If the state s_{t+1} is a terminal state, then the q-value function for that state and actions is zero. The SARSA method acts according the ***ε-greedy*** policy as an adjustment for the Exploration-Exploitation Dilemma. With a probability of *1-**ε***, the agent chooses the best action possible in the state following the greedy policy. With a probability of ***ε*** a random action is chosen by the agent. With this practice it is tried to ensure that the agent explores new possibilities and accepts negative short-term rewards but achieving a better long-term return.

There is a variant of SARSA, called expected SARSA which uses the expected value from the q-value function (2.33) instead of the actual value as in equation (2.32).

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma \mathbb{E}[Q(s_{t+1}, a_{t+1})|s_{t+1}] - Q(s_t, a_t)] \quad (2.33)$$

3. *Q-Learning* in comparison with SARSA is an off-policy control method [41]. This means, that the agent chooses an action *a* without following a policy but just in a greedy way, taking the max of the q-value function over the action chosen. Comparing equations (2.34) and (2.32), the difference relies on the *max* for the q-value.

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R_{t+1} + \gamma max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \quad (2.34)$$

In general, Q-Learning performs better than SARSA and the expected SARSA. To illustrate the difference between the three algorithms in terms of performance, an example using the *Taxi-v2 gym* environment [42] is conducted. This problem consists of a taxi that has to pick up a passenger at one of the four available locations and drop him in another. The agent receives 20 points for a successful drop-off and loses 1 point for every step it takes. It also includes penalties for illegal pick-ups and drop-offs.

The following values for the hyperparameters were used [43]:

- $\alpha = 0.04$
- $\gamma = 0.999$
- $\varepsilon = 0.9$
- Number of episodes = 2000
- Maximum number of steps per episode = 2500

In Figure 2.11 it can be seen that Q-Learning (green line) converges faster as SARSA and the expected SARSA. Figure 2.12 shows the amount of time required for training the agent.

Figure 2.11. Convergence comparison of SARSA (orange), Q Learning (green), and Expected SARSA (blue) [43]

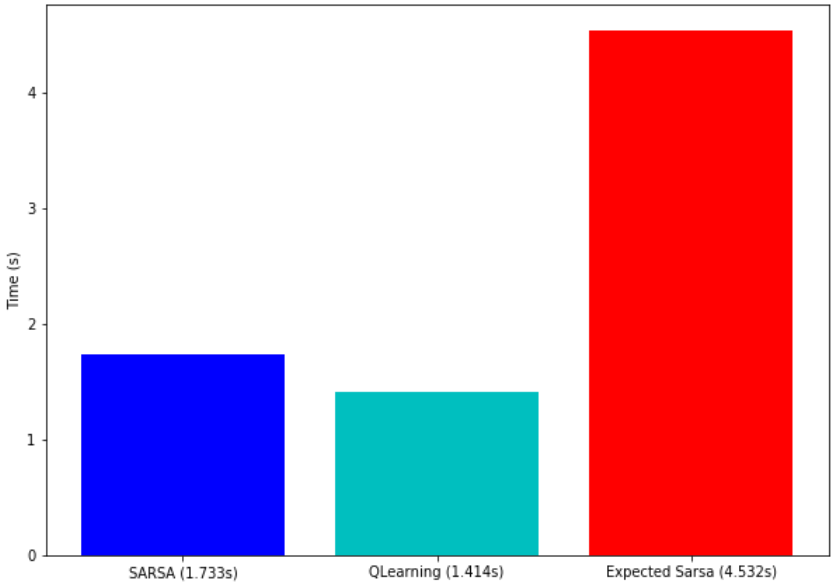


Figure 2.12. Comparison of time to complete the training between SARSA (dark blue), Q Learning (light blue), and Expected SARSA (red)

2.2.3.4. Deep Q-Learning

Q-Learning still can be improved, as it shows some limitations when the state space is of a considerable size, like in a videogame. In order to implement Q-Learning, it is necessary that all possible states are visited. While the right solution can be found even without visiting all states, as the state space grows it is more unlikely that this happens. To solve this problem, a Convolutional Neural Network is introduced, creating de Deep Q-Network (DQN) [44].

As CNN are used mainly with images, the explanation on how DQNs work will be given using the example of a videogame. In a videogame, our states will be each one of the frames of it while it is running.

The first thing to do is to pre-process all the data. This step is key in order to reduce the complexity of the state space, reducing the needed computation time for training the agent. So, the frames are greyscaled (getting rid of colours and presenting the images in black and white) as colours do not give important information in many cases. In the videogames that it does, other approaches to pre-process data should be used. Thus, a big step is made to reduce complexity, as the three channels of RGB colours are reduced to only one of greyscale. Another improvement to reduce the size of the states is cropping the images. For instance, in Atari Breakout [27] the upper, lower, and side pixels can be deleted (marked in red in Figure 2.13).

Figure 2.13. Atari Breakout game [27]

Next, some consecutive frames need to be stacked. The reason for it is that with just one frame as in the image above, it is not possible to deduce in which direction the ball is moving, so creating a sequence of frames stacked is of great use.

Then, the array containing the number of pixels and its colour in greyscale, is introduced in the CCN of the algorithm and the result is obtained. One characteristic of this algorithm is that the DQN tends to overestimate Q-values of the actions in a given state. The problem with this aspect of Deep Q-Learning is that not all the values are overestimated equally and once one action becomes overestimated, the exploration of the environment becomes harder, as it is more likely that the agent chooses the overestimated action in the next iterations, and the algorithm never converges.

To deal with the issue of unequally overestimated actions, a double network can be used. With the utilization of the double network, the action choice is decoupled from the target q-value generation. One network will select the action and the other one will compute the target q-value for that action. Copying weights from the action network to the target network will keep the networks synchronized. This weight-copy is usually done once every 10k. steps. The q-value function update for the double DQN (DDQN) [44] changes with

respect to the one for Q-Learning (2.34) in the TD target. Besides, as there are two networks, there are two different q-value functions, one for the action network (2.35) and another one for the target network (2.36).

$$Q^A(s_t, a_t) \leftarrow Q^A(s_t, a_t) + \alpha[R_{t+1} + \gamma Q^B(s_{t+1}, argmax_a Q^A(s_{t+1}, a)) - Q^A(s_t, a_t)] \quad (2.35)$$

$$Q^B(s_t, a_t) \leftarrow Q^B(s_t, a_t) + \alpha[R_{t+1} + \gamma Q^A(s_{t+1}, argmax_a Q^B(s_{t+1}, a)) - Q^B(s_t, a_t)] \quad (2.36)$$

3. Methodology

In this chapter is presented a procedure that allows to create complex traffic scenarios. The previous approaches from chapter 2.1 regarding complexity are utilized here and new ones are constructed aiming to develop a complexity measure that fits the problem that is faced in this Thesis.

Next, the environment for Reinforcement Learning is presented. In this part is not also included the model of the vehicles and the model of the road, but also the explanation of the chosen Reinforcement Learning algorithm and its parameters. In this part is also contained the previous complexity measure as part of the reward function needed.

The comparison of the implementation between MATLAB and Python is also commented at the end.

3.1. Complexity measure development

In order to evaluate the complexity of a given traffic situation it is needed a measure that can give some insight into the problem. As explained in chapter 2.1, creating a system that is valid to assess the difficulty of any problem in the real world is practically impossible. So, the measure proposed in this Thesis is only designed for the scope of traffic situations.

The planned approach is to define a vector of attributes for complexity. Each attribute is weighted depending on the influence assumed and the possibilities of the model. The formula to calculate the complexity *C* of a scenario is given by equation (3.1).

$$C = \sum_{i=0}^{t} \bar{w}^T \bar{a}_i\,, \qquad \bar{w}^T = [w_1, \dots, w_N], \qquad \bar{a}_i = \begin{bmatrix} a_1 \\ \dots \\ a_N \end{bmatrix} \tag{3.1}$$

In equation (3.1) $\bar{w}^T$ is the vector of attribute weights transposed, *i* are the different discrete timesteps, *t* is the total episode time (duration of scenario), and $\bar{a}_i$ is the complexity attribute vector for every timestep.). All the weights are normalized and sum up 1, and *N* is the number of attributes.

Regarding the complexity of physical systems, it is made clear that the disorder is a main driver of it. As in traffic scenarios it is difficult to estimate probabilities for the human behavior [4] [5], the disorder for a driving situation is more visible when the number of actions, such as changing lanes, is higher. For instance, a traffic jam does not seem like a very complex situation, but a situation when a car cuts in front of another car to overtake a third one, might be considered as more complex than the first one. So, even acknowledging that the number of vehicles has an impact on complexity, there is a certain number of vehicles in between no vehicles at all on the road and a traffic jam that makes the situation more complex. For this reason and given that the model of the environment always starts with the same number of vehicles for each training instance, there is no point in including the total number of traffic participants in the vector of attributes.

Following the example of the traffic jam, the number of actions performed during the episode is taken into account, counting actions from other vehicles as well as actions

performed by the autonomous car. Moreover, since the actions from the other vehicles have the potential of directly influencing the autonomous vehicle, the number of reactions of the autonomous vehicles as a response to other vehicles actions is also included as an attribute. For instance, if a vehicle cuts in front of the autonomous vehicle and the autonomous vehicle is forced to brake or change its lane to avoid a collision, the action of the AV would also be counted.

The actions that any vehicle can do are accelerating, braking, and changing the lane to the left or the right, and the autonomous vehicle drives on its own. In the next section is explained in detail how these actions are implemented into the model, but it is relevant to comment, that some boundaries are needed in order to count the actions of breaking and accelerating as actions themselves, since accelerating/braking for one timestep (between 0.1s and 0.04s) is not considered enough.

Schuldt [45] proposes a division in different layers for the representation of traffic scenarios (Figure 3.1). In the model used in this work, the road-level is fixed (a highway from Germany with three lanes) and there are no parameters related to the levels 2, 3 and 5 that can be modified. These factors are known as RSC [7] as presented in chapter 2.1. As these factors cannot be customized in the model, they are not included in the complexity measure.

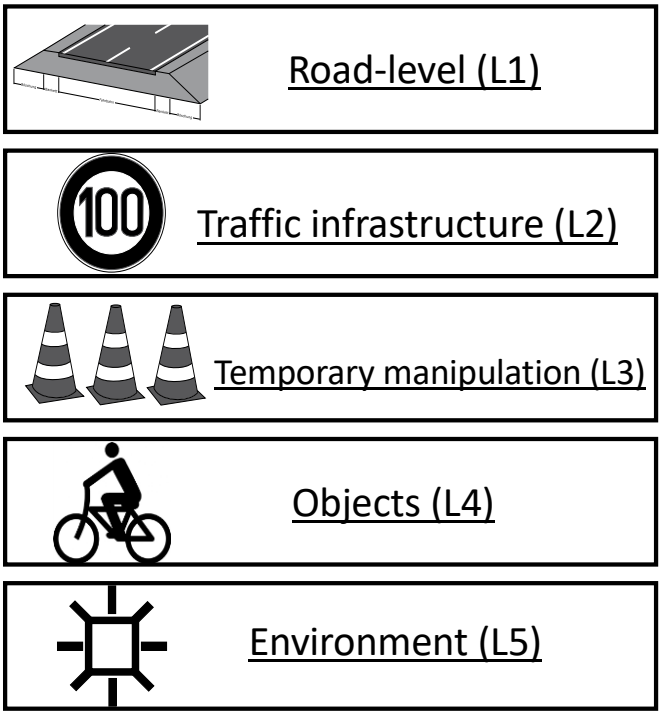


Figure 3.1. Road levels for traffic scenario representation [46]

The other factor influencing the complexity in traffic situations is the TEC [7]. The TEC pays attention to the relative positions and angles between the actors, taking only into account the surrounding actors in Figure 3.2. These actors can be cars, trucks, motorbikes, obstacles, etc. and would be included in the level 4 of Figure 3.1.

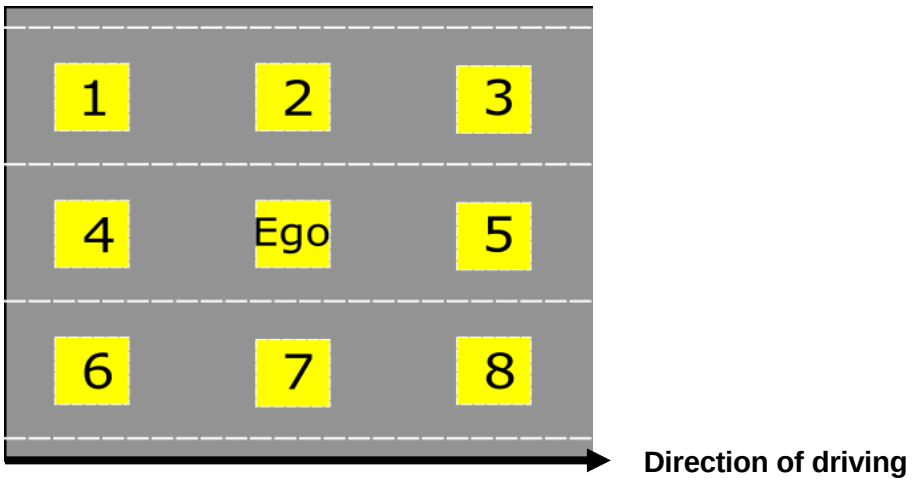


Figure 3.2. Visualization of surrounding vehicles.

Sometimes it is also possible that the vehicles ahead the leading vehicles (vehicles 3, 5, and 8 in Figure 3.2) have an impact on the behavior and safety of the automated vehicle [44]. For instance, if the difference in speed between the vehicle 5 and its preceding vehicle is large, vehicle 5 might need to change its lane in order to avoid collision. Then, it can occur that the autonomous vehicle must change its lane. So, the influence of the +1 vehicles in Figure 3.3 is explained.

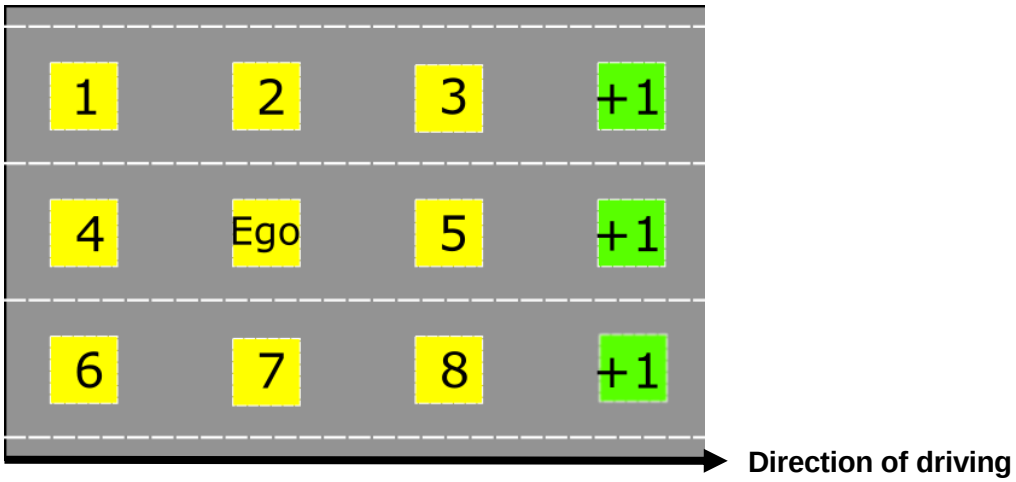


Figure 3.3. Surrounding vehicles with influence [44].

The positions of the other vehicles and angles is included in the calculations of the time to collision (TTC [12]). Equation (3.2) shows how TTC is measured. Integrated over time, the TTC transforms itself into the time integrated to collision (TIT, equation (3.3)) [47]. *TTC** is the boundary value of TTC.

$$TTC = \frac{\Delta x}{\Delta v} = \frac{x_2 - x_1 - (l_2 - l_1)}{v_2 - v_1} \tag{3.2}$$

$$TIT = \int_0^z [TTC^* - TTC(t)]dt \quad \forall\, 0 \leq TTC(t) \leq TTC^* \tag{3.3}$$

Besides, TIT and TTC include the differences in relative speed of the vehicles. Here, a difference between criticality and complexity needs to be made. Critical situations are

those in which an accident has a high potential of occurring. Some critical situations might be complex or not, and some complex situations can be critical or not. To account for this issue, the weighting of the TIT in the overall complexity measure will be smaller than the rest of the factors of influence. The TIT evaluated is the one for the autonomous vehicle, since it is the vehicle whose safety is crucial in this Thesis.

Another factor that is included into criticality and not complexity is the sudden occurrence of an accident. Even though it can be argued whether an accident can cause a complex scenario or accidents create just critical situations, accidents are maintained out of the scope of this Thesis. During the training, accidents can happen, but they will not be rewarded.

In summary, the complexity measure is based on the number of actions of all vehicles that are present in the environment, the number of reactions of the autonomous vehicle, and the TIT. Due to the fact that the complexity to assess is the complexity of the overall simulation and not of every timestep, and the TIT makes sense only once all the TTCs are collected, the complexity measure will be evaluated at the end of every episode in training following equation (3.4). The absolute value of the weights is equal to one. In the beginning, w_1 and w_2 are equal to 0.4, and w_3 is equal to 0.2. The weight of the TIT is positive due to the fact that an increasing value of TIT would mean a more critical scenario for the AV, hence more complex too in this approach. Besides, it is divided by 2 in order to reduce the importance of the TIT in comparison with the importance of the number actions and reactions for the complexity.

$$C = \begin{bmatrix} w_1 & w_2 & w_3 \end{bmatrix} \begin{bmatrix} Number\ of\ actions \\ Number\ of\ reactions \\ TIT_{AV} \end{bmatrix} \tag{3.4}$$

3.2. Development of the simulation environment

The simulation environment is divided in three parts. The first part is the definition of the road itself, following the regulations from Germany for the highways. The second element is the description of the vehicles and the actions they can make. In order to create each vehicle some inputs are necessary depending on the type of vehicle. The actions also depend on the vehicle type. The last element that unites the other two parts is Reinforcement Learning itself, which at the same time can be divided in two sections: the environment creation (as defined in RL) and the conception of the agent and the CNN for DDQN training.

3.2.1. Base road

The basic road which is used in all simulations and all possible scenarios is a German highway. This highway is depicted in Figure 3.4 and follows the German regulations [48] for a three-way highway with the traffic for every lane in the same direction.

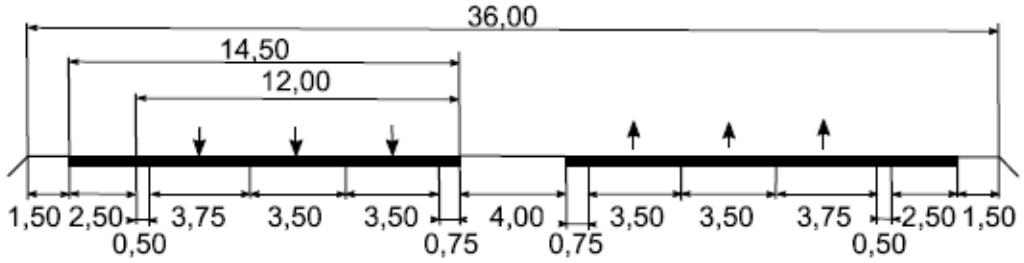


Figure 3.4. Road dimensions [47].

For the simulations, the road starts directly on the 2.50 *m* mark, without taking into account those extra 0.50 *m*. The width of the lanes from right to left is 3.75 *m*, 3.50 *m*, and 3.50 *m* respectively. The distance to the boundary of the road from the left lane is 0.75 *m* and can be neglected in the simulation. The width of the road marks is 0.15 *m*. The highway in the MATLAB implementation is plotted as in Figure 3.5 [49].

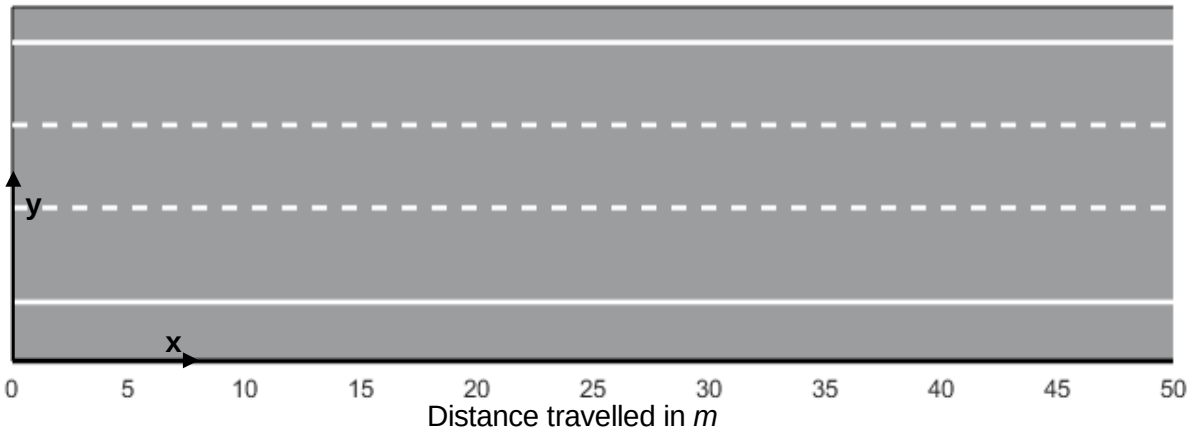


Figure 3.5. Highway representation in MATLAB [48]

3.2.2. Vehicle characteristics

The vehicles should be represented in a realistic way in order to obtain practical results that can be applied in real-life. The possible vehicles that can be included in the simulation are autonomous cars, normal cars, trucks and motorbikes. For this reason, the width and the length of every vehicle is included in its definition and the measures are taken from the mean values of the different kinds of vehicles in Germany. As the environment is simulated in two dimensions the height of the vehicles is irrelevant. For the cars (autonomous and normal) the length is 4.90 *m* and the width is 2.00 *m* [50]. The trucks are 16.50 *m* long and 2.50 *m* wide [51]. Motorbikes measures are based on the most sold motorbike in Germany for the year 2015 [52]. This motorbike has a length of approximately 2.20 *m* and a width of 0.92 *m*.

In addition to this, the initial positions of the vehicles in *x* and *y* directions (see Figure 3.5), i.e. in which position of the length of the road and in which lane, must be defined prior to the beginning of every episode. Also, the orientation plays a role in the definition of cars,

especially in case they are changing lanes to the left or right. The orientation of 0º is directed forward in *x* direction.

There are also other characteristics that define the vehicles that are not needed at the moment of the creation of each vehicle, but that are calculated through the simulation. Some of them are the name of the lane in which the vehicle is, the name of the car in front, TTC in general and for the lane change, the target lane, or the direction of the lane changes

3.2.3. Vehicle movements

Every vehicle can move in different ways and not only in the *x* direction with constant velocity. Except the autonomous vehicle that drives independently following some actions templates with rules, the other vehicles can move freely (what they do is operated by the agent, but this is later in this chapter explained). The actions are performed in timesteps, meaning that driving forward for two seconds is translated as going forward a certain number of steps. For instance, if the timestep is 0.1 seconds, then the action of going forward for two seconds takes 20 timesteps.

Regarding the actions of the normal vehicles, all of them can be performed no matter what the vehicle type is. The basic action is *drive forward* with constant velocity. Also, vehicles can brake and accelerate, with constant acceleration chosen by the user. The last action that can be done by the vehicles is to change lane. The lane change can be to the left or the right, and it needs to be provided with the absolute distance in *x* coordinate at which it starts, the lateral displacement, the direction (left or right), and the lateral acceleration. The pseudocode of this last function looks like:

```
CalculateChangeDistance=v*sqrt((LateralOffset*pi*pi)/(2*))
If direction = right
      Orientation is calculated as positive angle
If direction = left
      Orientation is calculated as negative angle
```

As there is no forward movement, the lateral displacement must be preceded with a driving forward instance every time.

In contrast, the movement of the autonomous vehicle is based on the Intelligent Driver Model (IDM) and lane changes with added rules [53]. First of all, the AV has to check if it has a vehicle in front of him, based on the safety parameters chosen by the user. These parameters are the desired velocity and the time gap (time distance) to the leading vehicle. If the leading vehicle is going slower than the desired velocity of the autonomous vehicle, it checks the overtaking possibility. In addition to this, the AV tries to drive always on the right lane of the road. If this is not possible, it will try to be on the middle lane, avoiding the third. Finally, it stays on the lane to the left of the road if the other two are not reachable. Checking if the lanes are free is made in order. If the AV is in the left lane, it will try to change to the middle lane, not to the right one. Only when the AV is in the middle lane it will check if the right lane is free in order to perform the lane change. The functioning is shown below:

```
Lane check #in which lane is the AV
OvertakeCheck(LeadingVehicle,DesiredV, DesiredGap)
If Overtake needed and possible
      AVLaneChange(desiredLane)
If Overtake needed but not possible
      AVBrake(BrakingAccel)
If lane exists to the right and it is free
      AVLaneChange(desiredLane)
```

All the values for accelerations/decelerations have to be in an order of magnitude that makes them realistic. Thus, some boundaries to select them are created [50] after studying the maximum values of some of the most successful in terms of sales cars of Germany. The boundary values are presented in Table 3.1:

Table 3.1. Boundary Values for accelerating and decelerating vehicles

Measure	Min value	Max value	Used value
Acceleration	1.0 m/s^2	8.0 m/s^2	3 m/s^2
Brake	-4.0 m/s^2	-1.0 m/s^2	-3 m/s^2

3.2.4. Automated Vehicle test

Before the training using Reinforcement Learning can begin, it is necessary to test the behavior of the AV. The behavior should be according to the rules and functions described in the previous section. To do so, two situations are created. The first one checks if the AV breaks in case it cannot overtake the vehicle in front of him because its left lane is blocked. There is an infinite truck which is blocking the middle lane, so the AV depicted in blue cannot change lanes to the left when it encounters a slower vehicle in the right lane. The situation is shown in Figure 3.6. In Figure 3.7 the AV velocity (after 5 seconds) is 23 *m/s* and it started at 29 *m/s.*

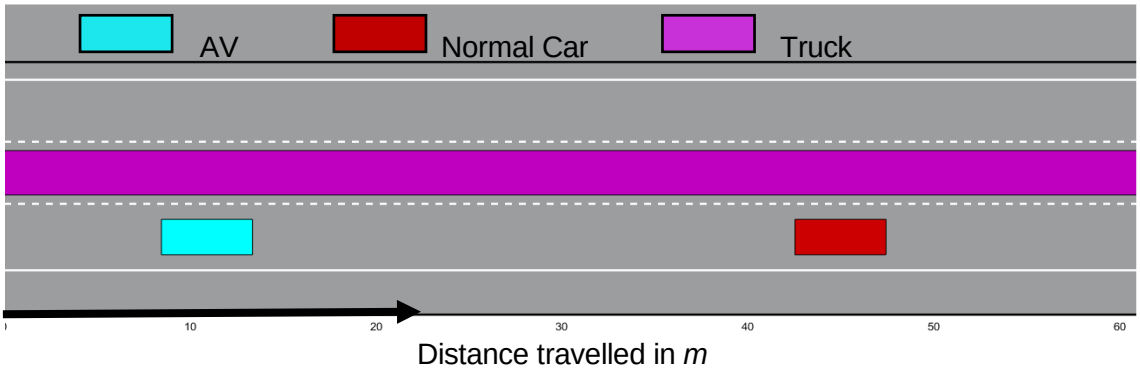


Figure 3.6.Test 1 AV starting position of the vehicles.

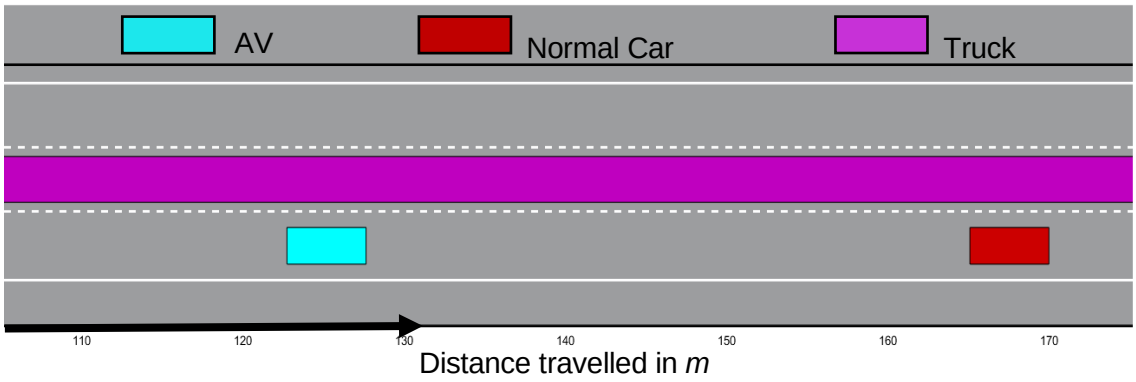


Figure 3.7. Test 1 AV end position of the vehicles.

The second test checks whether the AV is capable of overtaking vehicles and then return to the right lane. Figure 3.8 to Figure 3.10. show the process in which the AV first overtakes the car in the right lane after passing the truck (Figure 3.9), then overtakes the car in the second lane changing to the third one (Figure 3.10), and lastly returns to the right lane.

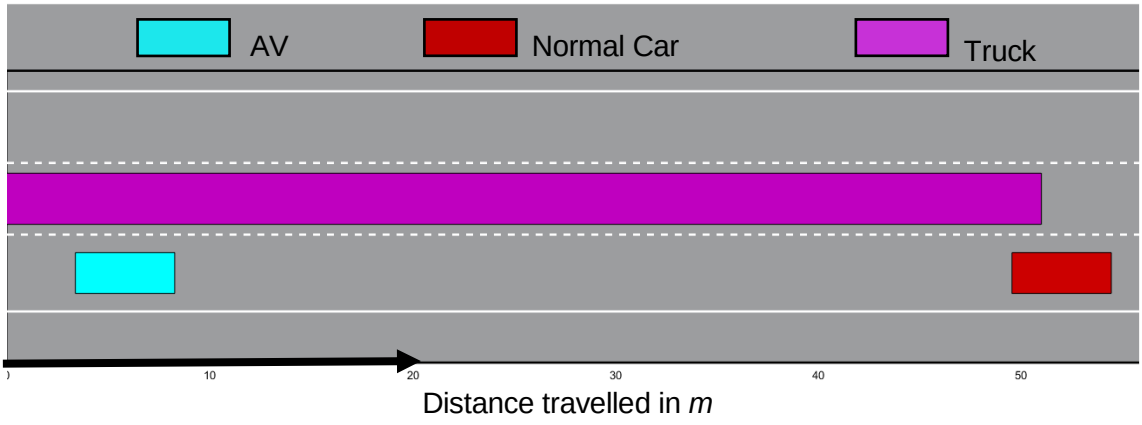


Figure 3.8. Test 2 AV starting position of the vehicles.

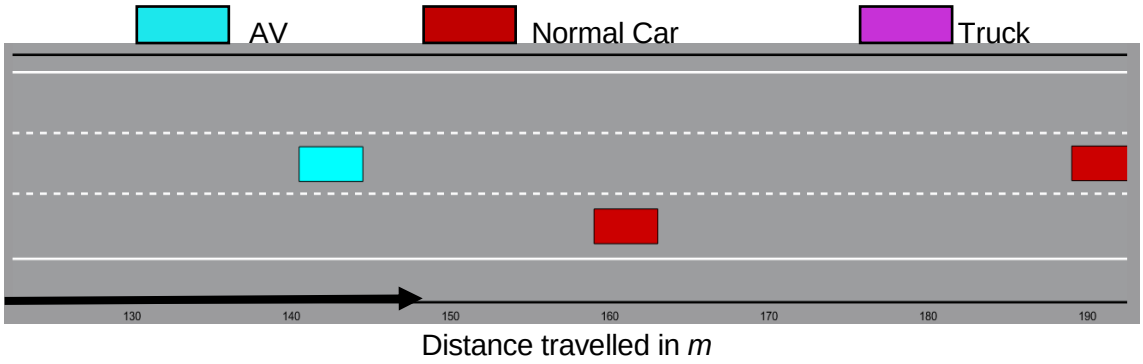


Figure 3.9. Test 2 AV Overtaking. First lane change of the AV completed.

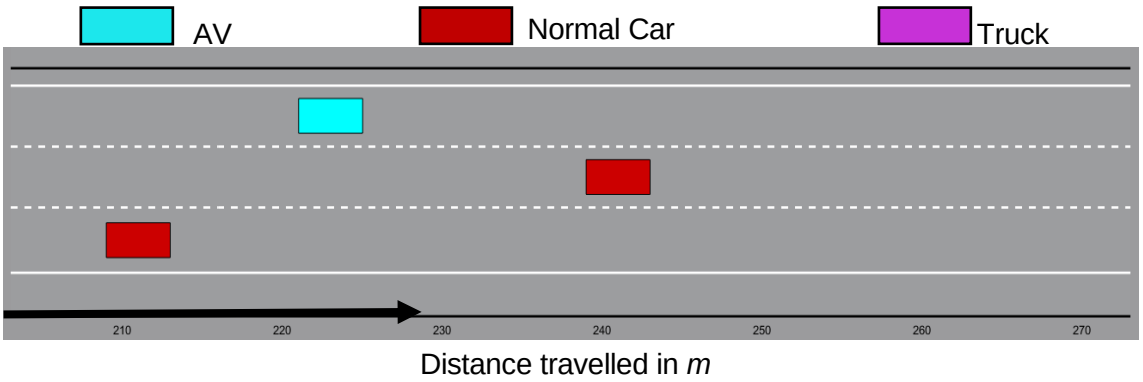


Figure 3.10. Test 2 AV Overtaking. Second lane change of the AV is performed

3.2.5. Reinforcement Learning environment

With the definitions of the road and the vehicles, when the algorithm for training is chosen the environment for Reinforcement Learning is ready to be set up. The code for this environment can be found in Appendix B.

To do so, the RL learning method is decided. As there is no policy predefined for the scenarios, the algorithm must be an off-policy method. Besides, the high number of possible movements and the fact that they can be performed in every timestep creates a huge state space. If during the exploration, many states are left, the algorithm may not work. Finally, the actions are discrete because they are performed using the movement templates illustrated in chapter 3.2.3. There is no decision to be made of acceleration or distance, only left or right, break or not with a fixed acceleration. It is true that actions with different values of, for instance, acceleration can be implemented, but that would make the action space unnecessarily big, at least for the first test and trainings. Taking all these factors into account, the algorithm chosen is a DDQN.

Every Reinforcement Learning environment has three fundamental parts or functions. *Properties* are the variables and objects that appear throughout the simulation of every scenario. The *step* function performs, as its name says, a step when the algorithm tells the environment to do it. And finally, the *reset* function, which puts all variables and objects in their starting positions from the *properties* (sometimes it can also randomize them). As the environment uses a discrete action space with a continuous state space, the environment is a modification of the known Cart pole environment, both in MATLAB and Python.

The *properties* of the environment include those variables and objects whose values need to be saved and they can be called from other functions inside the environment or call other functions from the outside. In this case, outside called functions are those created to move the vehicles. In the *properties* of the environment it is defined also the state vector as an array of zeros and the reward/penalty for certain actions.

Next, the action space and the state space are created. As the actions are deterministic, only one is chosen every step, and there are no physical properties of them that are modified, they are described as numbers (from 0 to K, where K is the total number of possible actions). The total number of actions K is calculated on equation (3.5) depending on the number of traffic participants N. Because the AV is not controlled by the

agent there is one actor less in the scenario. The +1 in equation (3.5) corresponds to action 0, that indicates the agent to do nothing and the vehicles just move forward.

$$K = 4(N - 1) + 1 \tag{3.5}$$

The state space needs to be related to the actions that are performed. The actions of the vehicles influence directly their velocity and positions in *x* and *y* directions. For this reason, the state space consists of three measures for every vehicle: *x* position, *y* position, and the absolute velocity. So, size of the state array *S* is determined as in equation (3.6). The number of different possible scenarios grows exponentially with the increase of the sizes of the action and state spaces.

$$S = 3(N - 1) \tag{3.6}$$

The *step* function advances the simulation one step every time it is called by the agent. The number of steps is selected by the user. The general functioning of any *step* function in Reinforcement Learning goes as follows. The first thing to do is to choose the action from the action space. Next, the action is made, the results of the actions are calculated, and the environment is observed, giving back to the agent the reward and the observed state. If the episode (one episode is one run of the environment from the beginning till the end) is not finished, another step is performed. Otherwise the agent resets the environment and starts a new episode.

However, in the highway environment some modifications had to be made, in order to simulate a realistic behavior. Every step that the agent takes is equivalent to 0.1*s* of simulation. Simulating for 25*s* is equivalent to do 250 steps in each episode but crashing two of the traffic participants makes the agent finish early the episode with a negative reward. Some of the actions take more than one timestep, for instance changing lanes. So, if the agent chooses to change the lane of one of the vehicles, the action remains the same for a fixed number of steps in order to complete the action. The other actions (do nothing, brake, and accelerate) can be done every step, they do not have a fixed number of steps to be completed. This difference is caused by the fact, that letting the agent change the action in every step made that the lane changes were not completed, ending with vehicles that were between two lanes.

Once the action is finished, the agent can choose any other action. When the actions are performed, the properties of the vehicles are updated. This part is of great significance because the actions highly depend on the properties of every vehicle. For instance, if the AV detects that the car that is in front of him brakes, it can also brake. But if the characteristic related to the AV that tells it which its leading vehicle is was not updated, the AV might crash.

If the action was a lane change, it is checked whether it has finished or not. If it is finished, a flag indicating the end of it is raised, and the agent in the next step can choose any other action. If the lane change is not finished when it is checked, then the action is forced to be the same in the next steps of the environment, until it is finished.

After the chosen action is made, the environment checks for collisions and the observation of the state of the environment is made. If there are no collisions and the total time is smaller than the maximum time, the agent performs another step and the process repeats. If there is a collision, the agent earns a negative reward and terminates the

episode. If the episode reaches the maximum number of steps, the agent also terminates the episode. As the complexity looked for is the one of the overall situations and not of every timestep, every step receives a reward of 0 unless it is the ending step of the episode, when it receives a reward following equation (3.4) if the episode was completed without any crashes.

Every time that an episode terminates, if the agent is not finished learning it simulates another episode. Before starting the episode, it is important to restart all variables and vehicles to their initial values and positions. This is done through the *reset* function. Besides, in this function the vehicles are created as an object and the initial conditions (the initial state) are given to the agent.

The process is repeated until the agent learns. The discussion on the learning of the agent is done in the next sections, but as a matter of fact, when the agent has learnt in MATLAB can be chosen by the user. When the conditions imposed by the user are met, the agent is saved and might be used to create other scenarios. The general pseudocode of the environment is presented, as a summary of what above is explained.

```
Environment Highway
      Properties:
            TimeStep
            Collision
            VehiclesDimensions
            Vehicles #as objects
            TotalTime
      Methods:
            Highway:
                  StateSpace
                  ActionSpace
            Function step:
                  SelectAction
                  if(action is not do nothing or brake or accelerate)
                        doAction for a predefined number of steps
                  else
                        doAction
                  GoForward(all vehicles)
                  UpdateProperties(all Vehicles)
                  CheckFinishedLaneChange
                  CheckCollisions
                  State #give back to the agent
                  If Collision or Time > TotalTime
                        getReward
                        end episode
            Function reset:
                  CreateVehicles(Positions, Velocities, Orientation)
                  InitialState
                  Reset properties
```

3.3. Reinforcement Learning algorithm

3.3.1. MATLAB

DDQN is the Reinforcement Learning algorithm of choice to solve the problem addressed in this Thesis. In chapter 2.2.3.4. the way of working of this algorithm is shown. DDQN uses a CNN between the inputs and the outputs. MATLAB has its own packages and toolkits, with predefined options to create the networks, the agent and to train it. On the other hand, all this is programmed by hand in Python, making use of some of the functions included in TensorFlow.

First, the MATLAB approach is explained. The first thing to do in MATLAB is to reset the environment (using the *reset* function explained in chapter 3.2.5). Then comes the CNN. The inputs to the CNN are the state and the action. In MATLAB these inputs go separated, so two paths are created for the CNN. The chosen parameters for the CNN are 200 neurons in each layer, and the chosen rectification layers are ReLU. As the state is an array with more than one value, there are three hidden layers in that path, which are named as CriticStateFCX in Figure 3.11. The action is only one, so there is also only two hidden layers, which are called CriticActionFCX (also in Figure 3.11). At the end of the CNN, both inputs merge together in the CommonPath and are rectified in the last ReLU layer. This Neural Network is based on the networks used for the Cartpole and the Walking Robot [54], with some more layers were added to improve the learning accuracy.

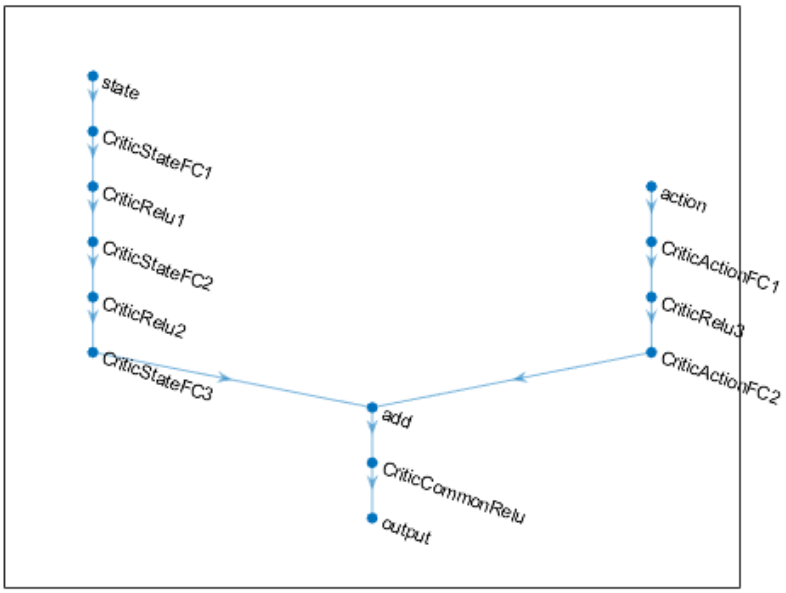


Figure 3.11. CNN used in MATLAB.

The agent is created with an Adam Optimizer [55]. The Adam Optimizer is an extension of the stochastic gradient descent used to update the network weights. Some of the benefits of using this optimizer are that it is straightforward to implement, it is efficient in terms of computation and has little memory requirements, it is well suited for large amount of data and parameters, and the hyper-parameters have intuitive interpretation. Adam is also a combination of Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). The advantages of combining those optimization

methods are realized in Adam. For example, it uses both moments (mean and variance) to adapt the learning rates instead of only the first moment as in RMSProp. The learn rate for Adam is set to 1e-4.

Other parameters from the agent that can be tuned are the Gradient Threshold (how much as a maximum can the weights vary in every step) and the regularization method (used to create less complex models when a large number of features is present adding a coefficient as penalty to the loss function) used is L2 (Ridge Regresion), with a factor of 1e-5.

In addition to this, a double DQN is used as explained in chapter 2.2.3.4. The discount factor γ is set to 0.99, the minibatch size to 1024, and the experience buffer to 10 thousand samples. These values are selected after trying many different combinations of parameters, as it is shown in the results chapter. Table 3.2 summarizes these data.

Table 3.2. Summarization of the values used in the CNN of the MATLAB approach.

Variable	Value
Learning Rate α	1e-4
L2 Regularization Factor	1e-5
Discount factor γ	0.99
Batch size	1024
Experience Buffer	10000

The training is stopped when the agent meets some requirements from the user. Here it is used the average reward from the last 5 episodes (as a running average). If the average goes over a certain threshold, the training is stopped. The agent is saved following the same criteria. Selecting the threshold is not trivial. Supposing the maximum reward was of 100, selecting 100 as a threshold for the training may not lead to the most complex scenario. Due to the definition of the TIT (equation 3.3) every negative value and every value over the boundary TTC* is reduced to 0 and thus, neglected. If the AV stays away from all other traffic participants, its TTC in every timestep will be large. This way, the TTC is 0 and by extension, so will be the TIT. As it is added in equation (3.4), this means that the large TIT does not affect the complexity of the scenario if it is 0, so the reward in the example could be 100. But, if in another situation that is very similar, but the AV gets closer enough to another so the TIT is not 0, the complexity will be greater than 100, being this scenario more complex than the first one. Apart from reducing the threshold to a reasonable level, the TTC* can also be increased so larger numbers of TTC are not discarded, and the criticality is further taken into account and affects in a better way the complexity measures.

3.3.2. Python

The Python approach uses the same algorithm as in MATLAB, but with some changes in the parameters and the way it is programmed.

To program in Python, the software used is ANACONDA and the Python version is Python 3. The code used for this part is available in the annex section. The code is divided in three parts. The first one is the creation of the class vehicle, which includes the same movements for the vehicles as the Matlab vehicles script. The second part is the creation of the CNN for Python and here are the main differences with the other approach. The CNN is created using only two hidden layers with 200 neurons each. While in Matlab there are many Critic layers (see Figure 3.11), in most applications in Python only two layers are enough. Besides, increasing the number of hidden layers affects negatively the speed of training with the resources available. Those are the reasons to select only two hidden layers, since the same situations are learned faster with no loss of precision when using smaller networks (see chapter 4). It is important to note that in this case, there is only one path, where the inputs are the actions and the output is the state. This difference comes from the way the MATLAB toolkits are programmed, which always require two paths(for states and actions) instead of only one like in Python. The rectifier layers are *tanh* functions instead of ReLU. The batch size is 64 and the learning rate stays the same at a value of 1e-4. These values are shown in Table 3.3.

Table 3.3. Summarization of the values used for the CNN in Python.

Variable	Value
Learning Rate α	1e-4
L2 Regularization Factor	1e-5
Discount factor γ	0.99
Batch size	64
Experience Buffer	10000

The last part of the Python script is the programming of the training. Here it is selected the total number of episodes for training and in which episodes the weights are copied from one network to the other. In this case, every 50 episodes the weights are transferred between the networks. The Python code for the DQN agents can be found in Appendix C.

3.4. Base Scenarios

3.4.1. Scenario for Hyperparameter Tuning.

With the developed simulation scenario, vehicle movements and characteristics, and the creation of the learning algorithm it is possible to generate complex traffic scenarios. To do so, a base scenario is created.

First of all, the number of vehicles has to be selected. If the number of vehicles is too low, the complexity is reduced. This also happens if the number of vehicles is too high and there are no possible movements for any vehicles, as if the situation was a traffic jam or

just a situation where there is too much traffic, but the vehicles are not stopped. With a highway of three lanes, the selected number of vehicles is six. From left to right, there are three trucks on the right lane, which drive close to each other at a constant speed. In the middle lane, two normal cars driving at some distance from one another, and in the left lane the AV, which is going faster than the other three vehicles. There are no special conditions in the road. The initial situation of the scenario can be seen in Figure 3.12.

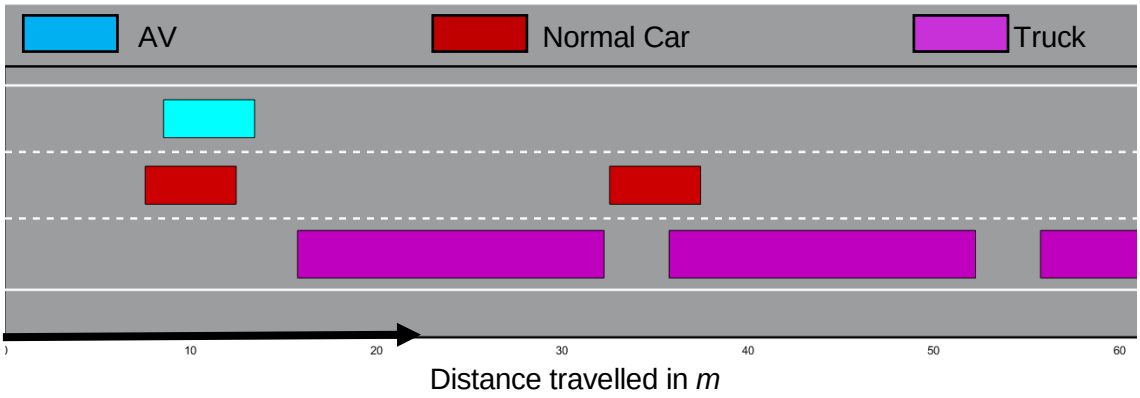


Figure 3.12. Starting situation of the environment for Hyperparameter tuning in MATLAB.

The initial velocities and positions of all vehicles are presented in Table 3.4. The velocity values are realistic values for the highways that are considered safe. It is true that in the German highways it is possible sometimes to go without speed limit, but in this case the velocity of the AV is selected to be in a reasonable level. The boundary value for the time to collision TTC* is set to 5 seconds.

Table 3.4. Starting positions and speed for the traffic participants of the base scenario.

Name	Starting X Position	Starting Lane	Starting Speed
Autonomous Vehicle	5 *m*	Left Lane	30 *m/s* (126 *km/h*)
Car1	5 *m*	Middle Lane	25 *m/s* (90 *km/h*)
Car2	40 *m*	Middle Lane	25 *m/s* (90 *km/h*)
Truck1	20 *m*	Right Lane	20 *m/s* (72 *km/h*)
Truck2	40 *m*	Right Lane	20 *m/s* (72 *km/h*)
Truck3	60 *m*	Right Lane	20 *m/s* (72 *km/h*)

If no movements were made by the cars in the middle lane and the trucks in the right lane, the automated vehicle goes to the right lane when it is free, making two movements. So, the value of the complexity for the scenario with no movements, according to equation (3.4) is 0.8.

3.4.2. Training Scenario

In order to train the agent in a different situation, a new scenario is created. In this scenario, the AV drives on the right lane with a truck that is going slower in front of him. In the middle lane there are two normal vehicles driving. The left lane is free. Figure 3.13 depicts the initial situation of this training scenario. This scenario is going to be tested in Python too.

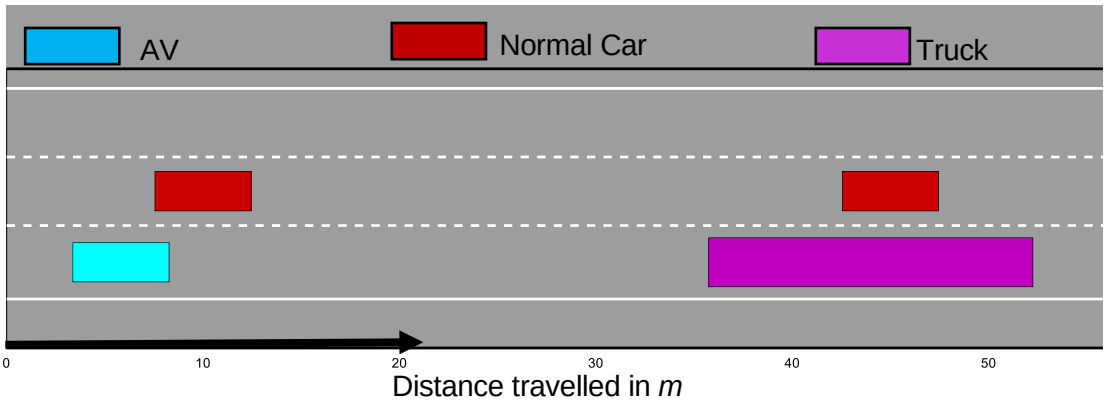


Figure 3.13. Starting situation of the training scenario. There are only four traffic participants in the whole scenario. The truck is not allowed to leave the right lane and does not perform any actions.

In Table 3.5., the values for the initial positions and velocities of the vehicles is presented.

Table 3.5. Starting values for positions and velocities of the different traffic participants in the training scenario.

Name	Starting X Position	Starting Lane	Starting Speed
Autonomous Vehicle	0 *m*	Right Lane	30 *m/s* (126 *km/h*)
Car1	5 *m*	Middle Lane	25 *m/s* (90 *km/h*)
Car2	40 *m*	Middle Lane	25 *m/s* (90 *km/h*)
Truck1	40 *m*	Right Lane	20 *m/s* (72 *km/h*)

3.4.3. Simulation Scenarios

The trained agent must be evaluated in other scenarios to check its performance and applicability to create complex situations in other scenarios that it has never seen.

There are two scenarios proposed that are based on the training scenario. The first one adds to the training initial situation a third car in the left lane driving faster (33 *m/s*) that cannot be controlled by the agent and only drives straightforward. In the second simulation scenario, the third car also appears and there is another truck in front of the first one. The agent cannot control the third car that is placed in the left lane.

Figure 3.14 shows the initial situation of the two scenarios (in the second one the second truck is way ahead and does not appear in the image) and Table 3.6 the position of the third vehicle and its velocity (for both simulations) and the positions and velocity of the truck that only appears in the second one.

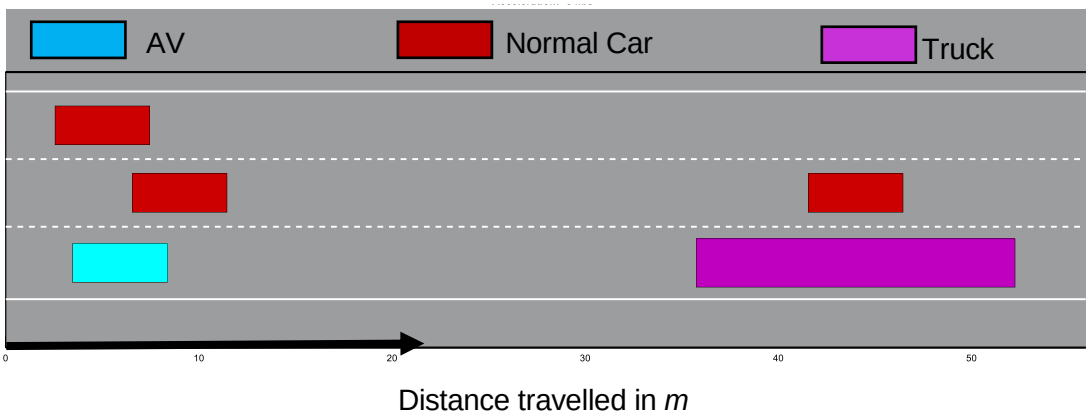


Figure 3.14. Starting situation of the traffic participants for the simulation scenarios.

Table 3.6. Starting positions and velocity of the car added to the simulation scenarios.

Name	Starting X Position	Starting Lane	Starting Speed
Car3	5 *m*	Left Lane	33 *m/s* (119 *km/h*)
Truck2	80 *m*	Right Lane	20 *m/s* (72 *km/h*)

A third scenario is created to avoid having the vehicles in the same positions than in the training scenario, as this might influence the performance of the trained agent. So, the truck is placed in the middle lane and the car2 is placed in the right lane. Their *x* starting positions and velocities remain the same as in Table 3.5. There are no more added traffic participants.

3.5. Hyperparameter tuning in MATLAB

Hyperparameter tuning are the activities conducted to find a combination of parameters that allow the RL algorithm to learn in the best way consuming the minimum time possible. Some hyperparameters allow for a more precise learning, but in exchange the time it takes to complete the learning is higher. Other parameters speed up the learning risking the probability of oscillating and never finding the minimum with the gradient descent. So, a compromise must be reached. Besides, it is important to differentiate from two ways of seeing the learning time. One is total time, and the other one is the total number of episodes needed. The total number of episodes is given more importance in this work but paying attention to the total time too.

In this Thesis, the hyperparameters that are going to be tuned of choice are the learning rate, the number of hidden layers, the number of neurons per layer, the number of episodes to average the reward (as a running average), and the batch size. Other parameters that can be tuned remain fixed. The discount factor γ is set to 0.99, the regularization factor for L2 is set to 1e-5, and the experience buffer length is fixed to 10000 samples. The experience buffer length can be understood as a table, in which the experiences from the agent in terms of states, rewards and actions [27]. It is used by the agent to take random samples from it in order to learn (apart from gaining more experience, so there still exists exploration).

The advantage of experience replay for this specific application is that learning with previous experience multiple times is a more efficient use of it. This is very useful when gaining real-world experience is costly. The updates of the algorithm are slowly converging, so performing multiple passes through the CNN with the data is beneficial. This effect is increased when given the same state and action pair, the variance of the outcome is low. [27]

The starting scenario is the one explained in chapter 3.4.1, but not all actions were allowed at first to check if the environment worked properly. Besides, the influence of the different hyperparameters is easier to monitor. The only actions allowed for the normal cars were changing lanes to the right and left, and action of doing nothing (every traffic participant just drives at constant velocity). Another difference is the reward function, which was only based on the TIT (equation (3.7)). The penalty for crashing two vehicles is -100. The threshold to finish learning is set to an average reward of 95.

$$C = 100 - TIT \tag{3.7}$$

The first parameter to be tuned is the number of episodes to end the training based on the running average of the reward of those episodes. Lower numbers of episodes (three episodes) made the learning faster (Figure 3.15), but the simulation of the agent showed that it did not learn. It was a matter of coincidence that the agent had performed those actions. In contrast, using five episodes to average the reward the learning was slower, but when it finished learning it performed correctly in the simulations. In the Figure 3.15, the training progress is showed.

The blue points are the reward for every episode, the red ones are the average reward of the last *n* episodes, and the green line is the expected reward for every episode given the initial state. In ideally designed networks, the expected reward converges to the final value of the reward, but this is not always possible.

Next, the number of hidden layers is adjusted. Starting with the network presented in Figure 3.11, one more layer is added to each path (action, state, and common paths). Figure 3.16 shows that increasing the number of hidden layers had a negative impact in the exploration, and at the same time made the learning slower in terms of time, in comparison with Figure 3.17, where the initial CNN was used.

As other parameters are not well adjusted when adjusting the first hyperparameters, the training often does not end, as the objective reward over 5 episodes is not reached. Nonetheless, the influence of the parameter that is tried to adjust can be seen. For example, it is clearly seen that increasing the number of hidden layers of the network affects negatively the training, not only in terms of required time per episode, but also in

the exploration. The lower exploration also impacts the training time negatively. The CNN with more layers makes 22000 episodes in 18 hours, while the original network can do 40000 episodes in 24 hours. So, there is an advantage when reducing the size of the CNN up to certain point (in this case, maintaining the original network), even when the training is neither finished (like in the Figure 3.15 and Figure 3.16) nor optimized.

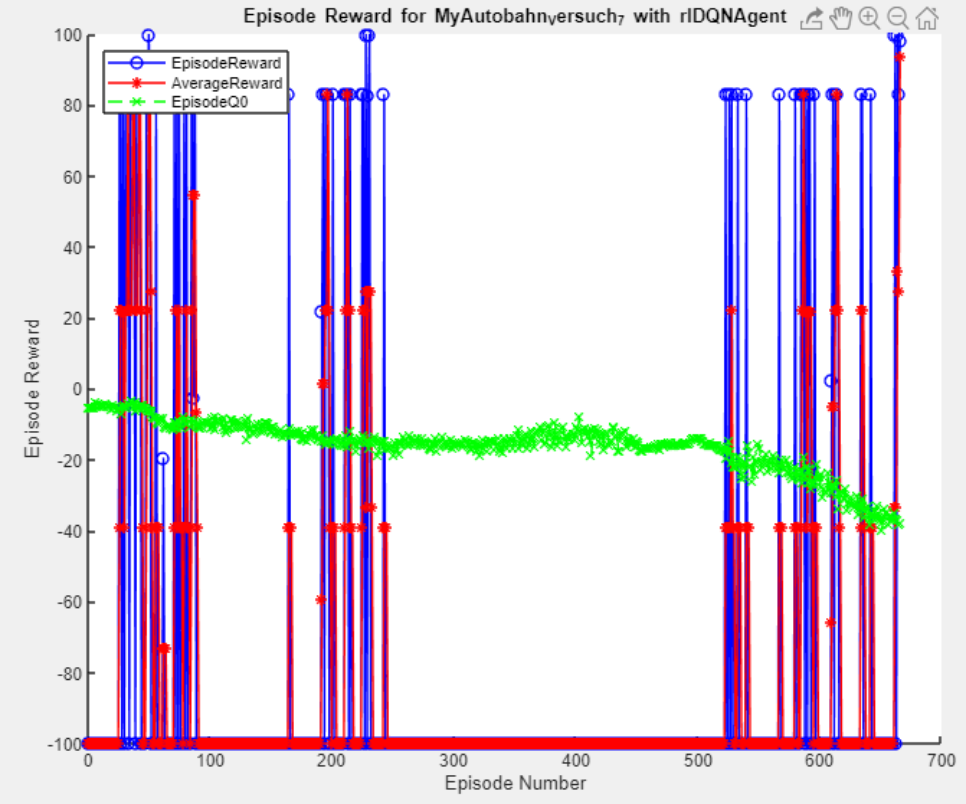


Figure 3.15. Training of the agent using 3 episodes to average the reward. The total time to train is 36 minutes, ending with 666 episodes.

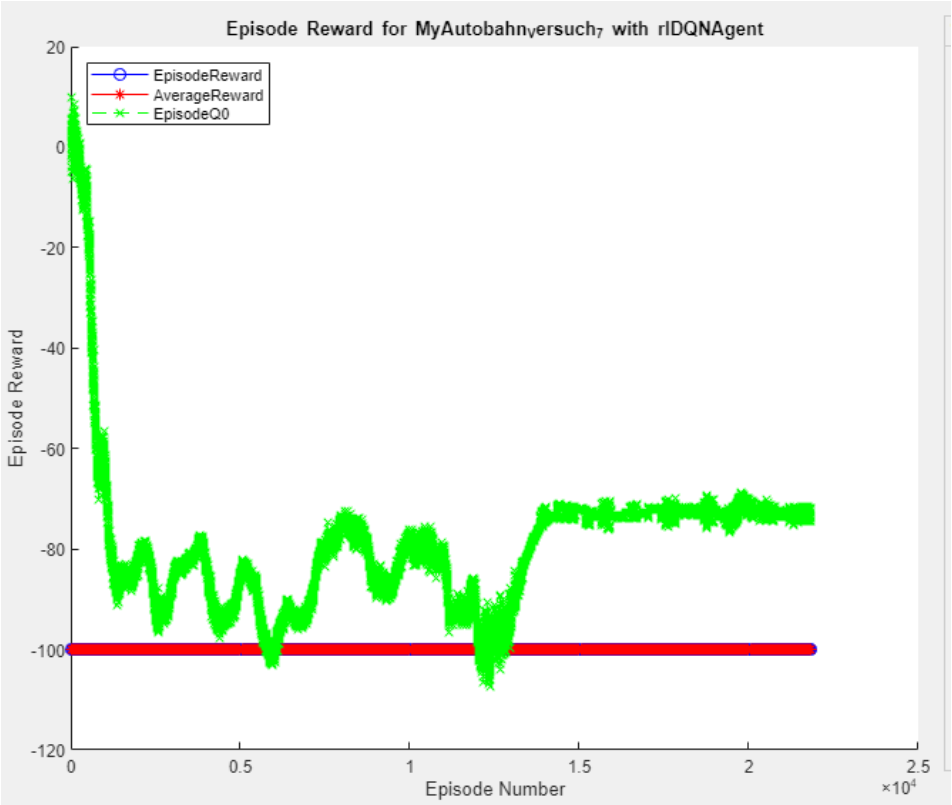


Figure 3.16. Training with more hidden layers than the starting CNN. The training is manually stopped without finishing after 22000 episodes (18 hours). Training is not completed.

To finish the creation a suitable CNN for the problem at hand, the number of neurons for each layer of the network is chosen. As in every iteration of learning the inputs go through all the neurons of the network (back and forth) to update the weights, the number of neurons has great significance in the learning time. Higher number of neurons per layer makes the training slower. In contrast, increasing the number of neurons has a positive effect on the precision of training. In Figure 3.18, 24 neurons per layer were used. In that example, 37565 episodes were tested in training for approximately 31.5 hours. That is 1189 episodes per hour. Besides, there are many oscillations in the expected reward (green line) with values that are not possible to reach: the precision of training is low. On the other hand, the training in Figure 3.19 shows no impossible values. The neurons used in this case were 200. The training stopped after 36.5 hours with 40000 episodes, what is around 1096 episodes per hour. As a result, 200 neurons are selected for each layer. The number of neurons is no further increased, as the training time became too large.

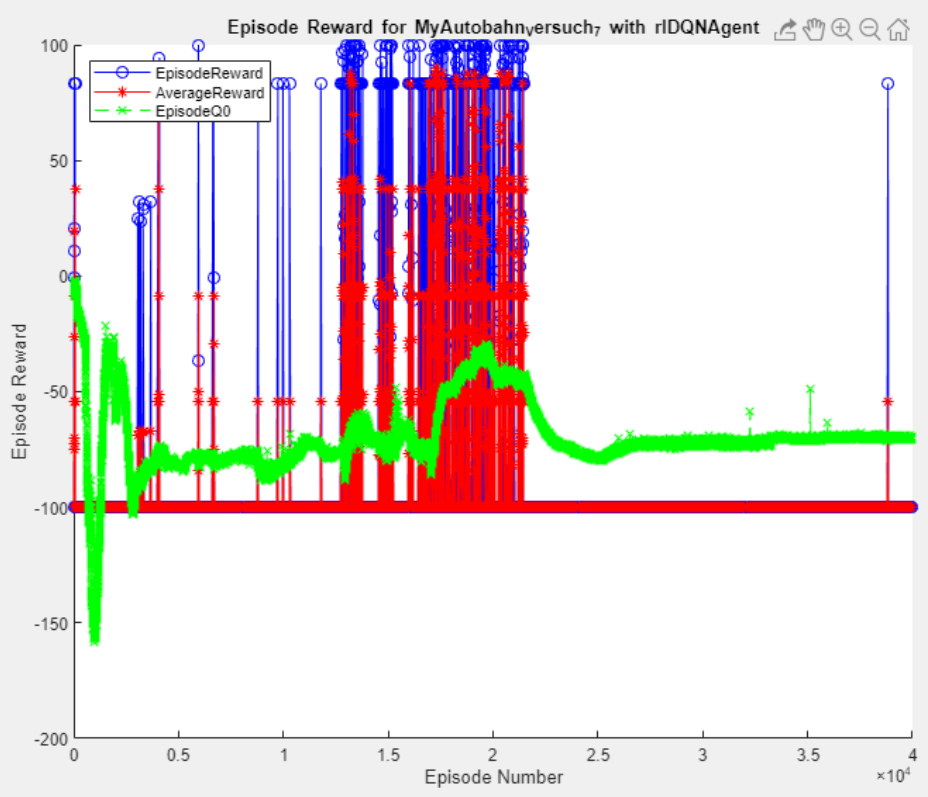


Figure 3.17. Training with the CNN presented in chapter 3.3.1. The training of 40000 episodes takes 24 hours. Training is not completed.

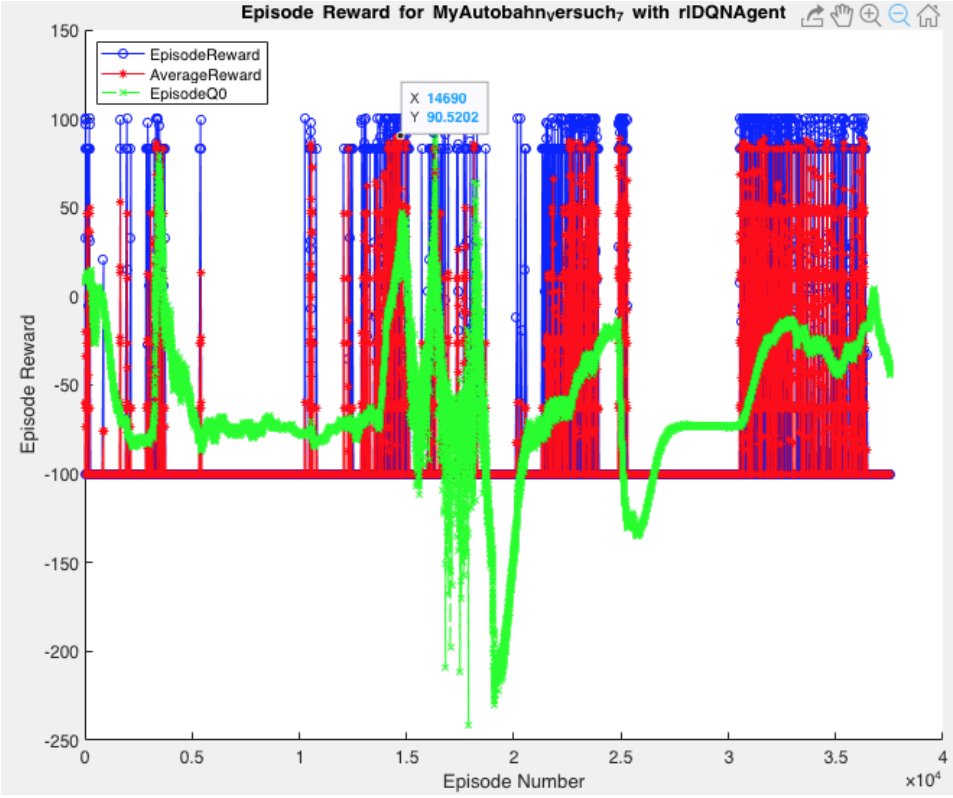


Figure 3.18. Training with 24 neurons per layer with the decided number of hidden layers. The training is manually stopped after 31.5 hours. Training is not completed.

The oscillations that are present can be further reduced increasing the precision of the algorithm. To do so, the batch size is increased. Figure 3.19 used a batch size of 32, and after trying many different batch sizes, the final selected batch size is 1024 samples. After 37 hours of training and only 5611 episodes, the training is finished (Figure 3.20) and the simulation proves that the training is effective. The training is very slow in terms of total time due to the big batch size. Increasing the number of actions (adding braking for both vehicles), it is necessary to keep increasing the batch size to 2048 samples.

In addition to this, the learning rate interacts closely with the batch size. Rising the batch size usually means that the learning rate can become larger. This way, a compensation of the total training time should happen. In this problem, increasing the learning rate only led to more oscillations, so it was fixed to a value of 1e-4.

In Figure 3.20, the expected reward is going upwards, in contrast to what happened in the previous examples. This indicates that the CNN is much better defined in accordance to the problem faced.

Besides, as said in chapter 3.3.1. the boundary TTC* has an influence in the rewards and because of that, in the training. In all these examples a TTC* of 10 *s* is used. This high number (well over the value of 1.5 *s* usually used as safety threshold [56]) means many possible scenarios. If this boundary is reduced, the number of possible scenarios (in terms of reward) reduces too, making the learning much easier.

Other meaningful realization while the hyperparameters were tuned, is the need of a relation between the observation of the state of the environment and the actions performed. If one action is added but has no effect on the observation given to the agent, the algorithm performs much worse. For instance, if the state is constituted only by the positions of the vehicles, adding the braking actions for the cars (that only affects their velocity) reduces the effectivity of the training. If the velocity of the vehicles in the state of the environment at the same time, the performance of the training is not affected too much, and an increase in the batch size would be enough to compensate for the increase number of possible states. This would be further commented and studied in the results chapter, as well as the effect of the boundary for the time to collision.

In conclusion the hyperparameter values selected for the training are:

- Number of episodes to average the reward: 5.
- Number of neurons per layer: 200.
- Learning rate α: 1e-4.
- Batch size: 1024 samples.
- The number of hidden layers is three for the state, two for the actions, and one for the common path (Figure 3.11).

The batch size and the learning rate can be further changed to improve the learning when more actions are included, or the number of traffic participants is increased. Table 3.7 summarizes the values for the training.

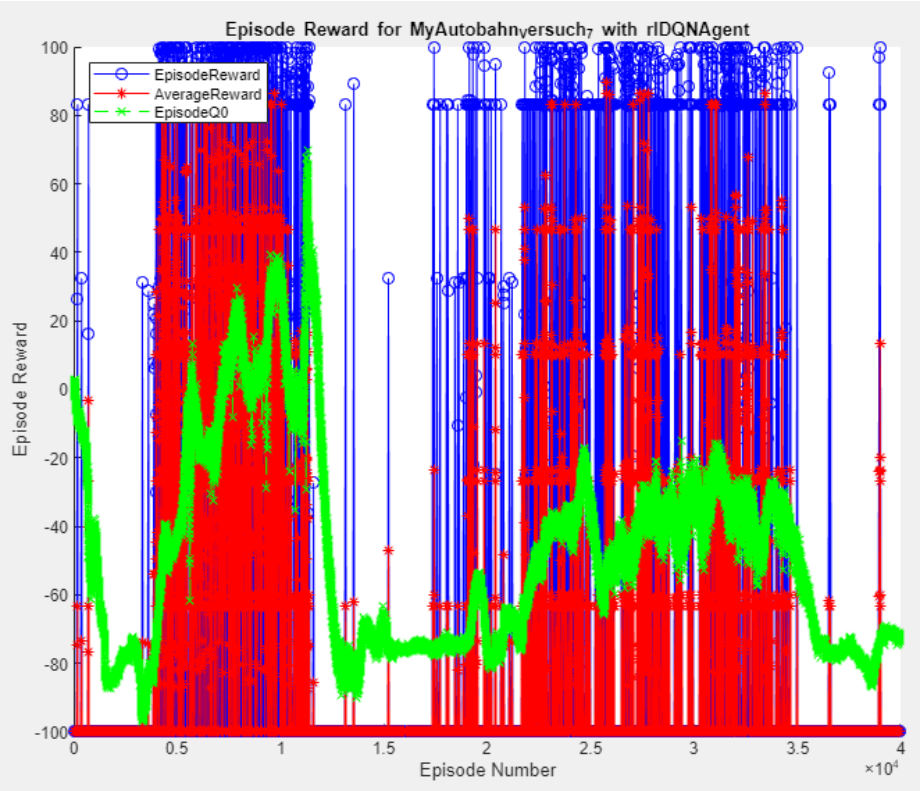


Figure 3.19. Training with 200 neurons per layer and a batch size of 32 samples. The total time is 36.5 hours for a total of 40000 episodes.

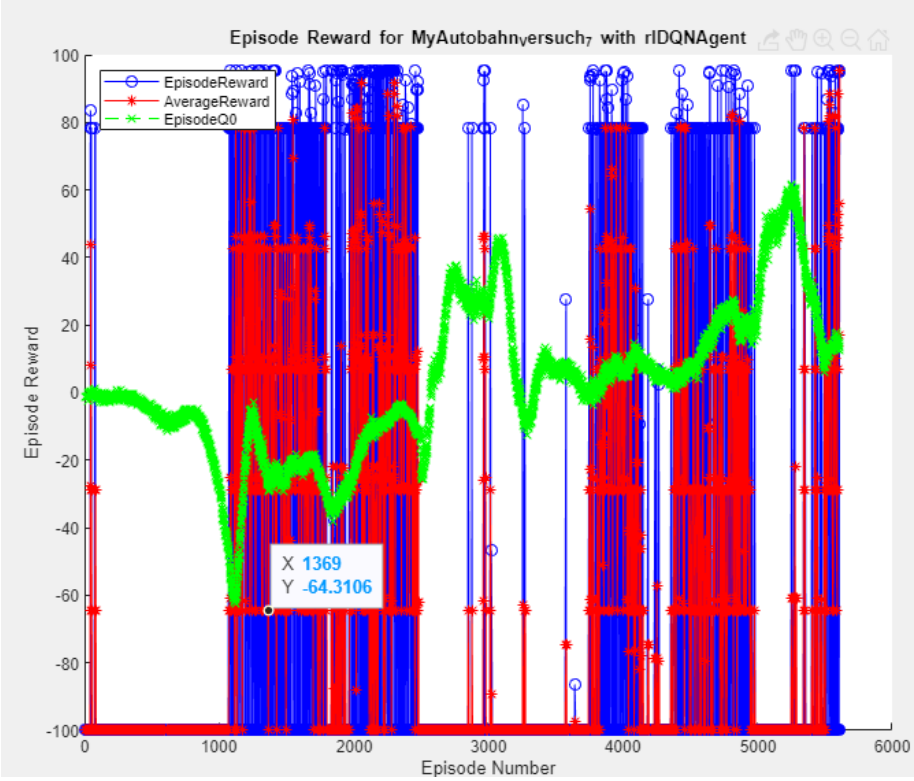


Figure 3.20. Training progress with a batch size of 1024 samples, 200 neurons and a learning rate of 1e-4. The training is successful after 37 hours for a total of 5611 episodes.

Table 3.7. Summarization of the values inferred for the training in the Hyperparameter tuning.

Variable	Value
Learning Rate α	1e-4
Number of Neurons per Layer	200
Number of episodes to average reward	5
Batch size	1024

4. Results

In this chapter, the results of training the agent in the scenario is assessed. The agent trained is tested in other situations that has never seen to check whether they are useful or not, and it is tried to extract some conclusions from them. In addition, a comparison between MATLAB and Python is conducted to test the performance.

4.1. Training Scenario in MATLAB

The scenario in which the agent is trained is the one presented in chapter 3.4.2. In this scenario, the AV drives on the right lane with a truck in front of it, driving slower. There are also two normal cars driving slower than the AV in the middle lane. The truck never moves from the right lane, and both cars have all possibilities of movement available. Figure 3.13 shows the initial situation of this scenario.

In this scenario, the most complex situation achieves a total reward of 4.82 points. First the two cars controlled are accelerated, and then the last car (car 1) changes of lane to its left. The car 2 is accelerated and then also changes its lane to the left. After this, car 1 goes back to the middle lane and the car 2 is accelerated up to 28.9 *m/s* (104 *km/h*). The car 1 goes to the right lane and then returns again to the middle one after driving on the right for a bit of time. Next the car 2 changes its lane to the middle lane and its accelerated up to 30.7 *m/s* (111 *km/h*). In Figure 4.1, Figure 4.2 and Figure 4.3 it is shown the behavior of the AV, which only follows the truck at 20 *m/s* after braking to avoid the collision.

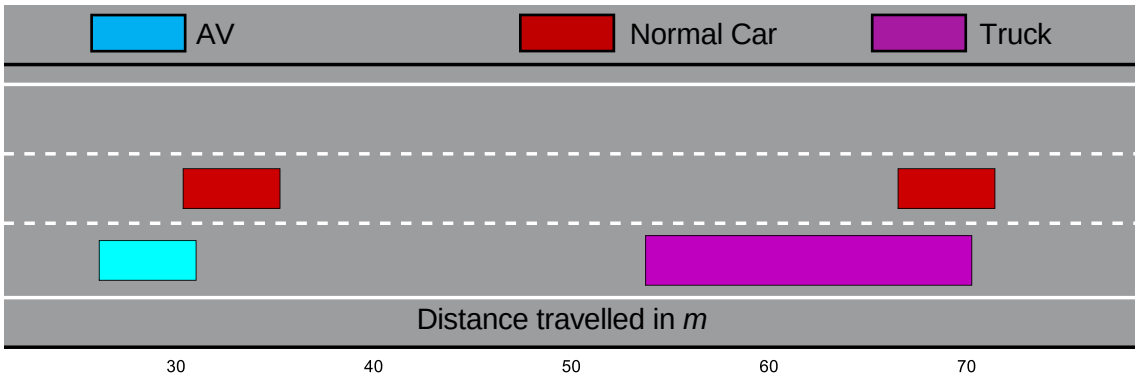


Figure 4.1. Position of the vehicles after 1 second in the most complex training scenario. Cars 1 and 2 are being accelerated, while the AV brakes to avoid colliding with the truck.

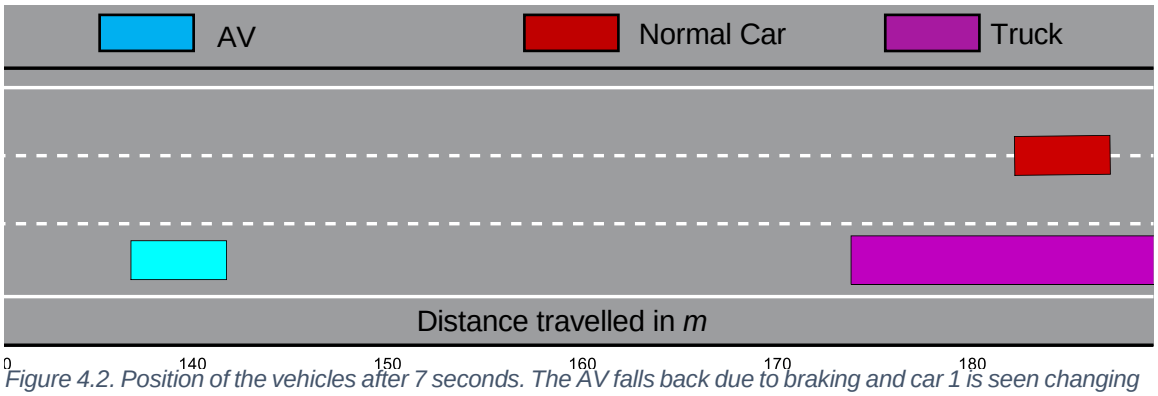


Figure 4.2. Position of the vehicles after 7 seconds. The AV falls back due to braking and car 1 is seen changing to the left lane.

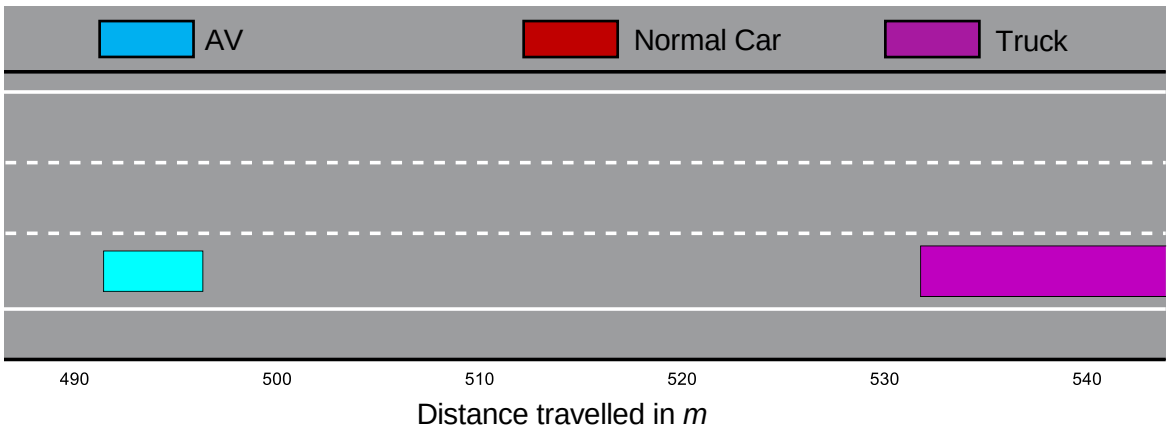


Figure 4.3. Final position of the AV after 25 seconds. It maintains the security distance given with the truck following it at 20 m/s.

4.2. Simulation Scenarios in MATLAB

First, the trained agent is validated in the training scenario. The trained agent receives a reward of 4.02, which is below of the maximum of 4.82. The reason for this is the threshold used during training to stop it. This issue is further discussed in chapter 5.

Next, the trained agent is simulated in the scenarios described in chapter 3.4.3. These scenarios are the same than the training one, except for the fact that there are some added traffic participants. In the first one, a fast travelling car is added to the third lane.

Simulating the agent in this scenario gives a reward of 4.02. The most complex situation according to the agent occurs accelerating the leading car (named car 2) up to 40 *m/s* (144 *km/h*). Then, the last car (car 1) changes its lane to the right and ends in the right lane. The car 2 changes its lane to the right and immediately after to the left again. To end the episode, the car 1 returns to the middle lane. The AV behaves exactly in the same way than in the most complex training scenario from section 4.1. This situation is the same as in the validation from the trained agent with and added third car.

In the second scenario, in addition to the third car driving in the left lane it is added a truck in front of the first truck present. In this scenario, a crash happens. Both vehicles behave in the same way as in the previous one, but when car 1 changes its lane to the right one, it collides with the new truck that was added. So, the reward for this simulation is -100.

As it is seen, the scenarios are highly influenced by the training scenario, as the agent chooses the same actions in both of them as the other vehicles added are not observable by the agent. In chapter 5 an analysis of this question is conducted.

The third situation of chapter 3.4.3 is used to test the behavior of the agent in an episode with no added vehicles but with some changes in the positions of the vehicles. In this third episode where only the truck and the car 2 are at the beginning in other lanes, the reward obtained by the agent is -100, i.e. makes the vehicles crash. The car 1 bumps into the rear part of the truck, as it does not break.

4.3. Scenarios in Python

The same scenarios are tested in Python in order to compare the performance of both systems. First, to check if the hyperparameters of the Python approach (Table 3.3) are good enough to train an agent in this environment, the same scenario used for tuning the hyperparameters in MATLAB (chapter 3.4.1) is used here. As shown in Figure 4.4. Python ends the training with the basic hyperparameters in around 150 episodes, which took around 4 hours.

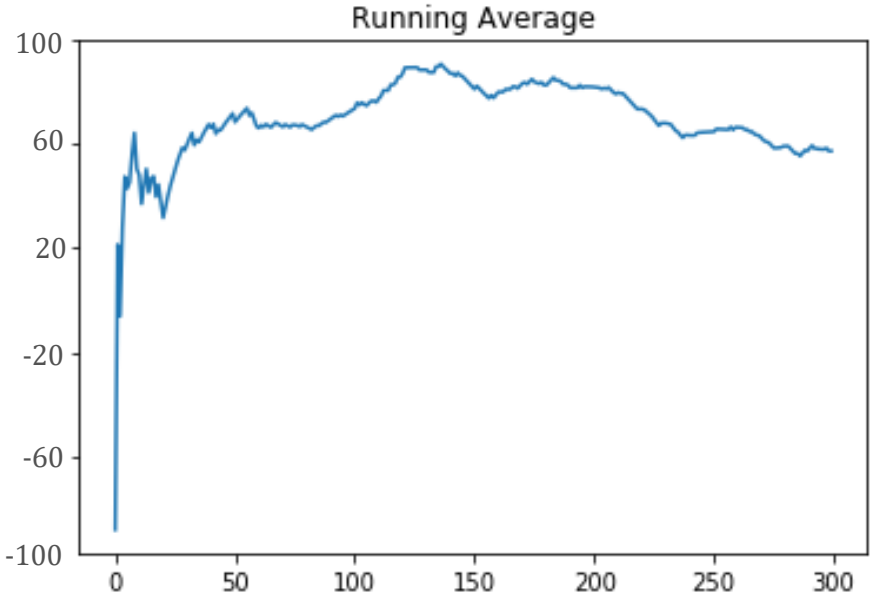


Figure 4.4. Running average of 5 episodes in Python for the scenario created for hyperparameter tuning. The training is successful in approximately 150 episodes.

The training of the agent is not stopped when reaching the objective value of the average reward, to check the exploration mechanism of the agent.

To try something that is more demanding for this approach, the same training is conducted using the last 100 episodes to average the reward. The threshold to consider that the training is successful is placed on a lower average reward of -54. This means that from those last 100 episodes, only three have a reward of -100 and the rest are 4.8 (which is the maximum reward for that scenario according to the training in MATLAB). The training is completed after around 1500 episodes as Figure 4.5 shows, for which it took 25 hours. Again, the training is not stopped when reaching the objective value (it took 34 hours to train 2000 episodes).

The CNN used in Python should be faster (as it happens) than the one used in MATLAB. In contrast, given the parameters used, it should be less precise and take more time to learn, or even not learn at all. But it happens the other way around. Trying the same CNN in MATLAB, also with the same hyperparameter tuning scenario than in Figure 4.4, does not allow the agent to learn (Figure 4.6, the training was stopped after 21 hours, because the agent did not learn in a comparable time as to the scenario for hyperparameter tuning).

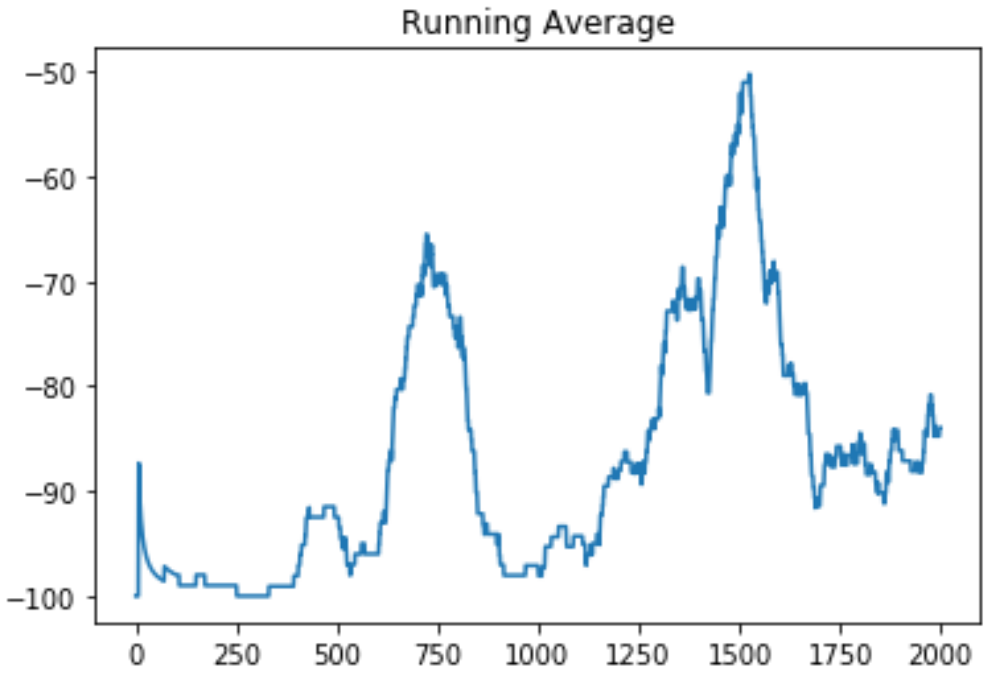


Figure 4.5. Running average in Python for the training scenario. It takes 25 hours and 1500 episodes to train.

The training in MATLAB and Python are conducted using the CPU of the computer. MATLAB is conducted in a better computer than Python. Despite this, the Python approach performs better, even with 100 episodes to average the reward. In chapter 5, some conclusions are drawn from this fact. Table 4.1 shows the different specifications of the computers used.

Table 4.1. Comparison of specifications of the computers used to train the agents in Python and MATLAB.

Specification	MATLAB Computer	Python Computer
RAM	16 GB	8 GB
Processor	i7-3770 CPU @3.40 GHz	i7-5500U CPU @2.40 GHz

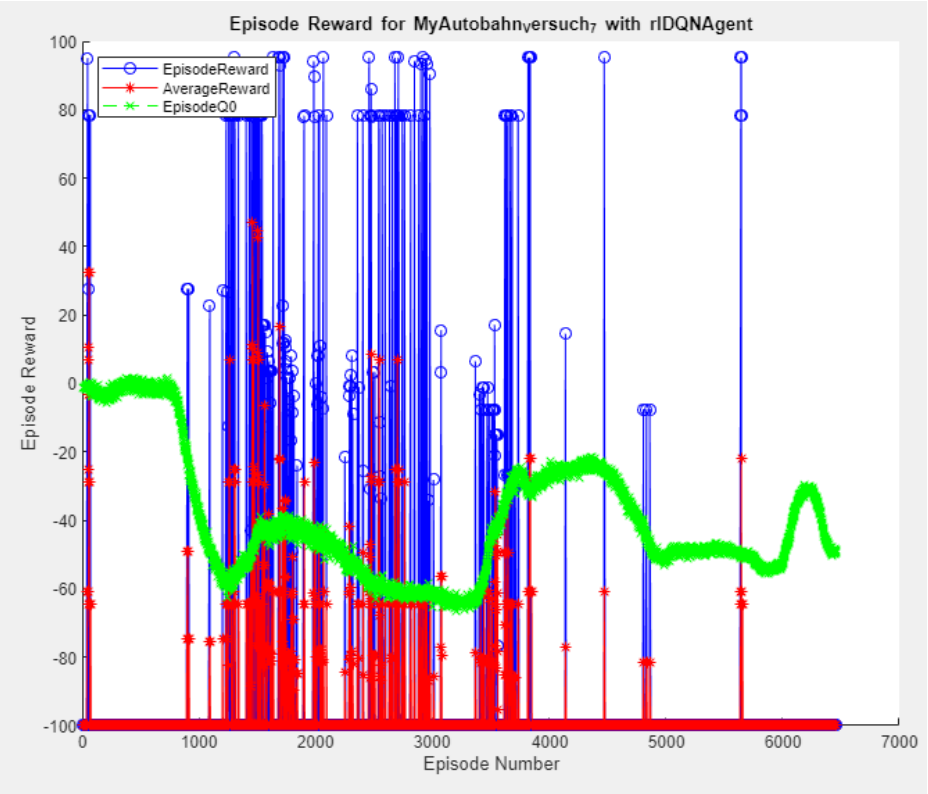


Figure 4.6. Training in MATLAB using the CNN parameters of the Python approach. The agent does not learn, and the exploration is not good enough. The training is stopped manually after 21 hours and 6500 episodes.

5. Conclusions

In this chapter is conducted an analysis of the results obtained in chapter 4: the results of the simulation of the training agent and the comparison between MATLAB and Python. In addition, some future lines of research and development of the work presented in this Thesis are proposed.

5.1. Discussion of Results

Starting from the hyperparameter tuning, , it was observed that having observations of the state that are not related somehow to the actions performed affects negatively the training. All the actions that are performed (braking, accelerating, and changing lanes) change the observation parameters (positions in *x* and in *y* direction as well as the velocity) of the vehicles controlled by the agent. Introducing in the state array the observation of other vehicles that are not controlled by the agent made the training slower, as the exploration mechanism suffered. Figure 5.1 and Figure 5.2 show the training of the same scenario with the same state array observed. The difference relies in that Figure 5.1 the agent does not have an action related to one of the observations and Figure 5.2 does have it, so the exploration is better.

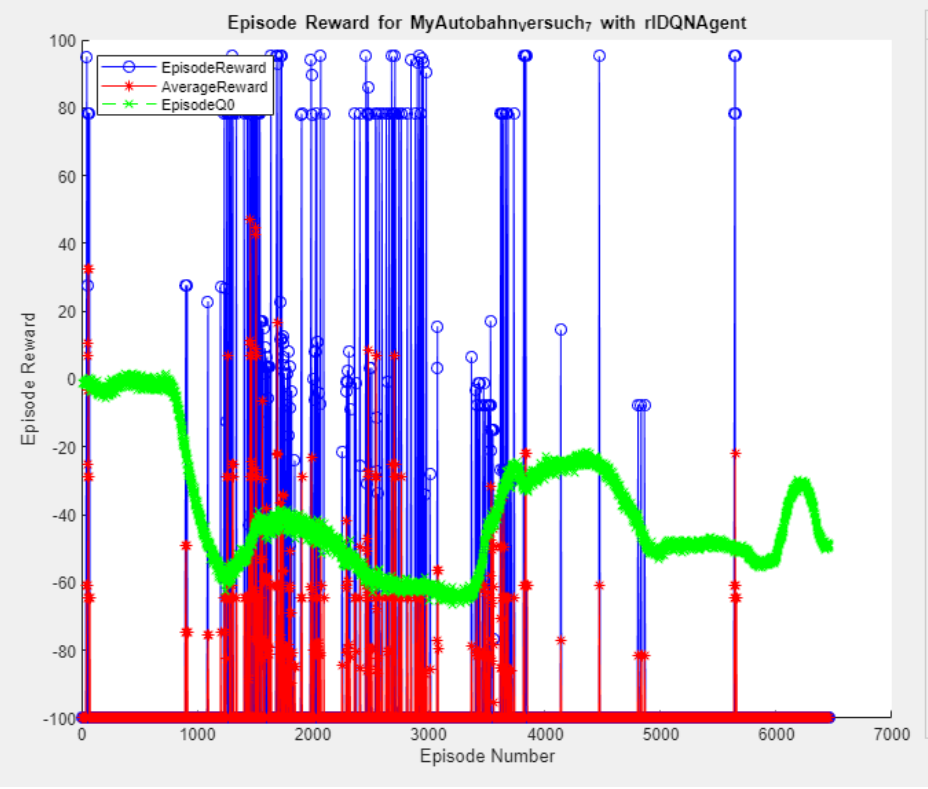


Figure 5.1. Training of the agent with observations in the state array that are not modified by the actions of the agent. The exploration mechanism does not perform as it is expected. The training never finished, and it was manually stopped after 20 hours.

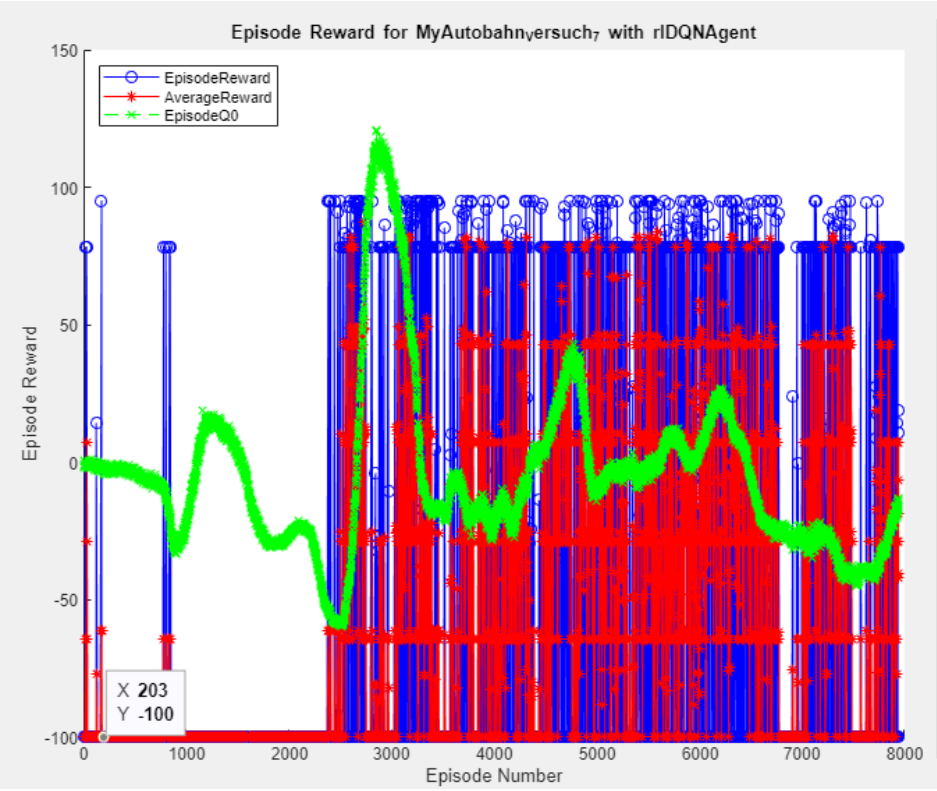


Figure 5.2. Training of the agent with all parameters of the state being able to be modified by the agent. In comparison with Figure 5.1, the exploration mechanism shows a better behavior.

Taking into account the results obtained in section 4.2 regarding the validation of the agent in its own training environment, the most complex scenario had a complexity of 4.82 and the validation earned 4.02 points. As said, this is caused by the threshold used to stop the training. In this Thesis, the agent identifies a training as successful when the average reward of the last 5 episodes is above the 95% of the maximum of 4.82. That is 4.58 points in complexity. The second highest value found by the agent is exactly that 4.02, and the mean of four episodes with a reward of 4.82 and one with a reward of 4.02 is 4.66, which is over the 95% threshold. This is the reason why the agent does not receive the maximum reward in the validation episode.

Besides, it can be said that the number of traffic participants has an impact on the creation and evaluation of complex traffic situations. Increasing the number of vehicles present does not make the agent perform in the best way possible in terms of complexity. There are two reasons for this to happen. One is that there exist a lower number of actions that the agent can do without crashing the vehicles, so it is more difficult to select the correct succession of actions to complete an episode. Next, there cannot be more observed vehicles when simulating than in the training due to limitations on MATLAB. The state array has to be of the same size in the training, so the third car and the second truck do not affect the observation of the environment that the agent has. Related to this, the vehicles in the simulating scenarios start on the same initial positions than in the training one, and being the only observable ones, they are the only influence of the choice of actions of the agent, since it does not know about the other vehicles that take part on the environment.

But in the third situation the agent also performs bad, crashing the vehicles. Looking at the results obtained in the previous simulation scenarios and in this one, it could be said that it would lead to better results to increase the size of the observation of the environment, even though the exploration mechanism is worse off if no actions are added. This results in a much slower training, but with the appropriate resources and time, it should not be considered as a problem.

Regarding the comparison between MATLAB and Python, it is clearly seen in section 4.3 that Python performs better in the trainings in terms of total time even with less computing power. In conclusion, Pythons is in general a better approach to Reinforcement Learning when paying attention to computational performance, but there exist the same limitations than in MATLAB regarding that the size of the state array is fixed by the agent trained.

5.2. Future Lines of Research

The results obtained in the present Thesis have permitted to identify new lines of research that might have some interest for future work. Due to the limitation on the scope of the Thesis, some variations and issues have not been studied. This section has the objective to summarize them and to spotlight their significance.

- A direct continuation of this Thesis would be the implementation of different road configurations such as city roads, intersections, etc. to increase the applicability of the complexity measure for the automated vehicles.

- The introduction of a stricter threshold value in training so the agent always performs in the best way possible in the validation but being careful of not overfitting the model.

- A better integration of the action templates of the vehicles in order to reduce the computation time needed. In addition, increasing the number of observations of the environment in order to improve the behavior of the trained agents in the simulated scenarios.

- Introduce new variables in the complexity measure such as the number of possible actions available for each vehicle to get more differentiated situations by the complexity measure in the same scenario.

- Use of completely new scenarios for the simulations, training the agent with variable initial situations. Varying the number of traffic participants in each episode for training can have a positive impact on the performance of the agent too.

- Use of Python instead of MATLAB, as Python is a more flexible programming language for Machine Learning and presents a better performance when compared with MATLAB.

6. Appendices

6.1. Appendix A

MATLAB code for the agent and the training

```
%create environment
env = MyAutobahn_Versuch_Sebastian_v2;

env.reset;

rng(0)
Ts = 0.1;
Tf = 25;

%creation of Agent
statePath = [
    imageInputLayer([6 1 1],'Normalization','none','Name','state')
    fullyConnectedLayer(200, 'Name', 'CriticStateFC1')
    reluLayer('Name','CriticRelu1')
    fullyConnectedLayer(200,'Name','CriticStateFC2')
    reluLayer('Name','CriticRelu2')
    fullyConnectedLayer(200,'Name','CriticStateFC3')
    ];

actionPath = [
    imageInputLayer([1 1 1],'Normalization','none','Name','action')
    fullyConnectedLayer(200,'Name','CriticActionFC1','BiasLearnRateFactor',0)
    reluLayer('Name','CriticRelu3')
    fullyConnectedLayer(200,'Name','CriticActionFC2')
    ];
commonPath = [
    additionLayer(2,'Name','add')
    reluLayer('Name','CriticCommonRelu')
    fullyConnectedLayer(1,'Name','output')];
criticNetwork = layerGraph();
criticNetwork = addLayers(criticNetwork,statePath);
criticNetwork = addLayers(criticNetwork,actionPath);
criticNetwork = addLayers(criticNetwork,commonPath);
criticNetwork = connectLayers(criticNetwork,'CriticStateFC3','add/in1');
criticNetwork = connectLayers(criticNetwork,'CriticActionFC2','add/in2');

figure
plot(criticNetwork)
```

```
%specify options for the critic
criticOptions       =       rlRepresentationOptions('Optimizer','adam','LearnRate',1e-4,
'GradientThreshold',1,'L2RegularizationFactor',1e-5, 'UseDevice',"cpu");

obsInfo = getObservationInfo(env);
actInfo = getActionInfo(env)
critic                                                                              =
rlRepresentation(criticNetwork,obsInfo,actInfo,'Observation',{'state'},'Action',{'actio
n'},criticOptions);

agentOptions = rlDQNAgentOptions(...
    'SampleTime',Ts,...
    'TargetSmoothFactor',1e-3,...
    'ExperienceBufferLength',100000,...
    'UseDoubleDQN',true,...
    'DiscountFactor',0.99, ...
    'MiniBatchSize',1024);

agent = rlDQNAgent(critic, agentOptions);

%training
trainingOptions = rlTrainingOptions(...
    'MaxEpisodes', 1, ...
    'MaxStepsPerEpisode',251, ...
    'ScoreAveragingWindowLength',5,...
    'Verbose',false, ...
    'UseParallel', false, ...
    'Plots','training-progress',...
    'StopTrainingCriteria','AverageReward',...
    'StopTrainingValue',95,...
    'SaveAgentCriteria','EpisodeReward',...
    'SaveAgentValue',95);

doTraining = true;

if doTraining
    % Train the agent.
    trainingStats = train(agent,env,trainingOptions);
else
    % Load pretrained agent for the example.
    load('SimulinkPendulumDQN.mat','agent');
end


%simulate
simOptions = rlSimulationOptions('MaxSteps',251);
experience = sim(env,agent,simOptions);
```

6.2. Appendix B

MATLAB code for the environment of training

Properties (set properties' attributes accordingly)

```
    properties
        t = 0;
        Coll = 0;

        deltaT=0.1;
        totalT = 25;

        LaneWidth = [];
        LaneCentre = [];

        LB_Truck = [16.5 2.5];
        LB_Car = [4.9 2];
        LB_Dummy = [1 1];

        % AV Data
        LB_AV = [4.9 2];          % [length width] [m]
        StartV = 30;             % Initial velocity [m/s]
        DesireV = 35;            % Desired speed [m/s]
        StartX = 0;              % Initial x-Coordinate [m]
        TTC_LaneChange = 1.5;    % From which TTC (to the target lane)
        aq = 3;                  % Lateral acceleration [m/s^2]
        t_Gap = 1.5;             % Minimum time distance to the front vehicle
        inTime = 0.1;

        TimeStep = 1;

        obj_Autonomous
        obj_Car1
        obj_Car2
        obj_Truck1
        obj_Truck2
        obj_Truck3
        obj_Dummy
        Vehicles = []

        number_steps = 0;
        starting_step = -24;     %to be able to repeat an actions
        starting_action = 20;    %to start in a non existing action
        finished = 0;            %flag raised when a lane change is finished
        action_count = 0;
        a = 0;                   %checks for invalid lane changes

        % Penalty for any crash
        PenaltyForCrashing = -100;
    end

    properties
        State = zeros(6,1)
    end

    properties(Access = protected)
```

```
        IsDone = false
    end
```

Necessary Methods

```
    methods
        % Contructor method creates an instance of the environment
        % Change class name and constructor name accordingly
        function this = MyAutobahn_Versuch_Sebastian_v3()
            % Initialize Observation settings
            ObservationInfo = rlNumericSpec([6 1]);
            ObservationInfo.Name = 'Car1 and Car2 Observations';
            ObservationInfo.Description = 'x,y positions and velocities for Car1
                                           and Car2';

            % Initialize Action settings
            ActionInfo = rlFiniteSetSpec([0 1 2 3 4 5 6 7 8]);
            ActionInfo.Name = 'Actions';

            % The following line implements built-in functions of RL env
            this = this@rl.env.MATLABEnvironment(ObservationInfo,ActionInfo);

            % Initialize property values and pre-compute necessary values
            updateActionInfo(this);
        end

        % Apply system dynamics and simulates the environment with the
        % given action for one step.
        function [Observation,Reward,IsDone,LoggedSignals] = step(this,Action)
            LoggedSignals = [];
            action = getAction(this,Action);
            this.number_steps = this.number_steps +1;

            if (action ~= 0 && action ~= 3 && action ~= 6 && action ~= 7 &&
            action ~= 8) && this.number_steps >= this.starting_step+24
                this.finished = 0;
                this.starting_step = this.number_steps;
                this.starting_action = action;
                disp(this.starting_step);
                disp(this.starting_action);
                this.action_count = this.action_count+1;
                this.obj_Car1.StartLaneChange = this.State(1);
                this.obj_Car2.StartLaneChange = this.State(3);
            elseif (this.number_steps <= this.starting_step+24) && this.finished
            == 0
                action = this.starting_action;
            elseif (this.number_steps <= this.starting_step+24) && this.finished
            == 1
                action = 0;
            end

            Autonomous(this.obj_Autonomous,this.DesireV,this.TTC_LaneChange,th
            is.aq,  this.t_Gap,  this.inTime,  this.LaneCentre,  this.TimeStep,
            this.t, this.Vehicles);

            %Actions for the Car1
            StraightCruise(this.obj_Car1, this.deltaT);
```

```
            if action == 1 %lane change to the right

                LaneChangeAtPlace_Normal(this.obj_Car1,this.obj_Car1.StartLaneC
                hange,'right',3.563,3);
                if(strcmp(this.obj_Car1.Lane, 'Lane2'))
                    this.obj_Car1.LaneChange21=1;
                elseif (strcmp(this.obj_Car1.Lane, 'Lane3'))
                    this.obj_Car1.LaneChange32=1;
                elseif (this.obj_Car1.PositionY <= 1.4)
                    this.a = 1;
                end
%                 disp(this.obj_Car1.PositionY)
            end

            if action == 2 %lane change to the left

                LaneChangeAtPlace_Normal(this.obj_Car1,this.obj_Car1.StartLaneC
                hange,'left',3.563,3);
                if (strcmp(this.obj_Car1.Lane, 'Lane2'))
                    this.obj_Car1.LaneChange23 = 1;
                elseif (strcmp(this.obj_Car1.Lane, 'Lane1'))
                    this.obj_Car1.LaneChange12 = 1;
                elseif (this.obj_Car1.PositionY >= 9.5)
                    this.a = 1;
                end
            end

            if action == 3
                Brake(this.obj_Car1,3,this.deltaT);
            end
            if action == 7
                Accelerate(this.obj_Car1,3,this.deltaT);
            end

            %Actions for Car2
            StraightCruise(this.obj_Car2, this.deltaT);

            if action == 4 %lane change to the right

                LaneChangeAtPlace_Normal(this.obj_Car2,this.obj_Car2.StartLaneC
                hange,'right',3.563,3);
                if(strcmp(this.obj_Car2.Lane, 'Lane2'))
                    this.obj_Car2.LaneChange21=1;
                elseif (strcmp(this.obj_Car2.Lane, 'Lane3'))
                    this.obj_Car2.LaneChange32=1;
                elseif (this.obj_Car2.PositionY <= 1.4)
                    this.a = 1;
                end
%                 disp(this.obj_Car1.PositionY)
            end

            if action == 5 %lane change to the left

                LaneChangeAtPlace_Normal(this.obj_Car2,this.obj_Car2.StartLaneC
                hange,'left',3.563,3);
                if (strcmp(this.obj_Car2.Lane, 'Lane2'))
                    this.obj_Car2.LaneChange23 = 1;
                elseif (strcmp(this.obj_Car2.Lane, 'Lane1'))
```

```
                this.obj_Car2.LaneChange12 = 1;
            elseif (this.obj_Car2.PositionY >= 9.5)
                this.a = 1;
            end
        end

        if action == 6
            Brake(this.obj_Car2,3,this.deltaT);
        end
         if action == 8
            Accelerate(this.obj_Car2,3,this.deltaT);
        end

        StraightCruise(this.obj_Truck1,this.deltaT);
        StraightCruise(this.obj_Dummy,this.deltaT);
        %action that makes all stay the same
        if action == 0
            %disp('Do nothing');
        end

        this.t = this.t+this.deltaT;

        for iCar = this.Vehicles
            UpdateProperties(iCar,this.Vehicles);
        end

        this.TimeStep=this.TimeStep+1;

        for iCar = this.Vehicles
            iCar.Values(this.TimeStep,this.t);
        end

        %check for collisions of autonomes auto
        this.Coll = Kollision(this.obj_Autonomous,this.obj_Car1) +...
            Kollision(this.obj_Autonomous,this.obj_Car2)+...
            Kollision(this.obj_Autonomous,this.obj_Truck1)+...
            Kollision(this.obj_Car1,this.obj_Car2)+...
            Kollision(this.obj_Car1,this.obj_Truck1)+...
            Kollision(this.obj_Car2,this.obj_Truck1);

        Observation  =  [this.obj_Car1.PositionX;  this.obj_Car1.PositionY;
        this.obj_Car1.Velocity;                    this.obj_Car2.PositionX;
        this.obj_Car2.PositionY; this.obj_Car2.Velocity];

        this.State = Observation;

        if this.Coll == 1 || this.a == 1
            c = 1;
            %disp('Crash!!');
        else
            c = 0;
        end

        %flags for finishing the movements
        for iCar = this.Vehicles
            if ~(strcmp(iCar.Name, 'AutonomousCar'))
                %change from lane 2 to lane 3
                if(iCar.PositionY     >=     this.LaneCentre(3)-0.05     &&
                iCar.LaneChange23 == 1)
```

```
                    iCar.LaneChange23 = 0;
                    this.finished = 1;
                    iCar.Orientation = 0;


                %change from lane 3 to lane 2
                elseif(iCar.PositionY    <=    this.LaneCentre(2)+0.05    &&
                iCar.LaneChange32 == 1)
                    iCar.LaneChange32 = 0;
                    this.finished = 1;
                    iCar.Orientation = 0;


                %change from lane 2 to lane 1
                elseif(iCar.PositionY    <=    this.LaneCentre(1)+0.05    &&
                iCar.LaneChange21 == 1)
                    iCar.LaneChange21 = 0;
                    this.finished = 1;
                    iCar.Orientation = 0;

                %change from lane 1 to lane 2
                elseif(iCar.PositionY    >=    this.LaneCentre(2)-0.05    &&
                iCar.LaneChange12 == 1)
                    iCar.LaneChange12 = 0;
                    this.finished = 1;
                    iCar.Orientation = 0;
                end
            end
        end

        Tit = TIT(this.obj_Autonomous);

        IsDone = this.t >= this.totalT || c == 1;
        this.IsDone = IsDone;

        % Get reward
        Reward = getReward(this,c,Tit);
        this.Coll = 0;
        notifyEnvUpdated(this);
    end

    % Reset environment to initial state and output initial observation
    function InitialObservation = reset(this)
        this.t = 0;
        this.Coll = 0;

        this.LB_Truck = [16.5 2.5];
        this.LB_Car = [4.9 2];
        this.LB_Dummy = [1 1];

        this.LaneWidth(10) = 2.5;       % Emergency lane [m]
        this.LaneWidth(1)  = 3.75;      % 1. Lane (Truck lane) [m]
        this.LaneWidth(2)  = 3.5;       % 2. Lane [m]
        this.LaneWidth(3)  = 3.5;       % 3. Lane [m]
        this.LaneWidth(11)  =  this.LaneWidth(1)  +  this.LaneWidth(2)  +
                               this.LaneWidth(3);     % Lane 1-3 added [m]

        this.LaneCentre(1) = this.LaneWidth(1) / 2;
        this.LaneCentre(2) = this.LaneWidth(2) / 2 + this.LaneWidth(1);
```

```
            this.LaneCentre(3) = this.LaneWidth(3) / 2 + this.LaneWidth(1) +
                                 this.LaneWidth(2);


            % AV Data
            this.LB_AV = [4.9 2];          % [length width] [m]
            this.StartV = 30;             % Initial velocity [m/s]
            this.DesireV = 35;            % Desired speed [m/s]
            this.StartX = 0;              % Initial x-Coordinate [m]
            this.TTC_LaneChange = 1.5;    % From which TTC (to the target lane)
            this.aq = 3;                  % Lateral acceleration [m/s^2]
            this.t_Gap = 1.5;             % Minimum time distance
            this.inTime = 0.1;

            this.TimeStep = 1;

            this.number_steps = 0;
            this.starting_step = -24;
            this.starting_action = 20;
            this.action_count=0;
            this.finished = 0;
            this.a = 0;

            this.obj_Autonomous                                                 =
            Output_Vehicles_v2('AutonomousCar',this.LB_AV(1),this.LB_AV(2),thi
            s.StartV,this.StartX,this.LaneCentre(1),0);

            this.obj_Car1                                                       =
            Output_Vehicles_v2('Car1',this.LB_Car(1),this.LB_Car(2),25,5,this.
            LaneCentre(2),0);

            this.obj_Car2                                                       =
            Output_Vehicles_v2('Car2',this.LB_Car(1),this.LB_Car(2),25,40,this
            .LaneCentre(2),0);

            this.obj_Truck1                                                     =
            Output_Vehicles_v2('Truck1',this.LB_Truck(1),this.LB_Truck(2),20,4
            0,this.LaneCentre(1),0);

            this.obj_Dummy                                                      =
            Output_Vehicles_v2('Dummy',this.LB_Dummy(1),this.LB_Dummy(2),100,5
            00,this.LaneCentre(1),0);
            % Fahrzeuge zusammenfassen

            this.Vehicles = [this.obj_Autonomous, this.obj_Car1, this.obj_Car2,
            this.obj_Truck1, this.obj_Dummy];

            for iCar = this.Vehicles
                iCar.ValueMatrix = cell((this.totalT/this.deltaT),12);
            end
            this.TimeStep = 1;
            for iCar = this.Vehicles
                UpdateProperties(iCar, this.Vehicles);
                iCar.Values(this.TimeStep,this.t);
            end

            InitialObservation    =    [5;    this.LaneCentre(2);    25;    10;
                                 this.LaneCentre(2); 25];
            this.State = InitialObservation;
```

```
            % (optional) use notifyEnvUpdated to signal that the
            % environment has been updated (e.g. to update visualization)
            notifyEnvUpdated(this);
        end
    end
%
```

Optional Methods (set methods' attributes accordingly)

```
    methods
        function Reward = getReward(this,c,TIT)
            if this.IsDone && c == 0
                %disp('The reward is');
                Reward           =           0.4*this.action_count           +
                          0.4*this.obj_Autonomous.NumActions -0.2*TIT;
            elseif c == 1
                Reward = this.PenaltyForCrashing;
            else
                Reward = 0;
            end
        end
    end
```

6.3. Appendix C

Python Code for the DQN Agent.

```
class HiddenLayer: #keeps track of params
     def __init__(self, M1, M2, f=tf.nn.tanh, use_bias=True):
         self.W = tf.Variable(tf.random_normal(shape=(M1,M2)))
         self.params = [self.W]
         self.use_bias = use_bias
         if use_bias:
             self.b = tf.Variable(np.zeros(M2).astype(np.float32))
             self.params.append(self.b)
         self.f = f

     def forward(self, X):
         if self.use_bias:
             a = tf.matmul(X, self.W) + self.b
         else:
             a = tf.matmul(X, self.W)
         return self.f(a)

class DQN:
    def __init__(self, D, K, hidden_layer_sizes, gamma, max_experience
s=10000, min_experiences=200,
                 batch_sz= 128):
        self.K = K #output actions

        #create the graph
        self.layers = []
        M1 = D
        for M2 in hidden_layer_sizes:
            layer = HiddenLayer(M1, M2)
            self.layers.append(layer)
            M1 = M2
        #final layer
        layer = HiddenLayer(M1, K, lambda x: x)
        self.layers.append(layer)

        #collect params for copy
        self.params = []
        for layer in self.layers:
            self.params += layer.params

        #inputs and targets
        self.X = tf.placeholder(tf.float32, shape=(None, D), name='X')
        self.G = tf.placeholder(tf.float32, shape=(None,), name='G')
        self.actions = tf.placeholder(tf.int32, shape=(None,), name='a
ctions')

        #calculation of output and cost
        Z = self.X
        for layer in self.layers:
            Z = layer.forward(Z)
        Y_hat = Z
        self.predict_op = Y_hat

        selected_action_values = tf.reduce_sum(
            Y_hat * tf.one_hot(self.actions, K),
            reduction_indices=[1]
        )
```

```
        cost = tf.reduce_sum(tf.square(self.G - selected_action_values
))
        self.train_op = tf.train.AdagradOptimizer(1e-4).minimize(cost)

        #create replay memory
        self.experience = {'s': [], 'a': [], 'r': [], 's2': [], 'done'
: []}
        self.max_experiences = max_experiences
        self.min_experiences = min_experiences
        self.batch_sz = batch_sz
        self.gamma = gamma

    def set_session(self, session):
        self.session = session

    def copy_from(self, other):
        #collect all the ops
        ops = []
        my_params = self.params
        other_params = other.params
        for p, q in zip(my_params, other_params):
            actual = self.session.run(q)
            op = p.assign(actual)
            ops.append(op)
        self.session.run(ops)

    def predict(self, X):
        X = np.atleast_2d(X)
        return self.session.run(self.predict_op, feed_dict={self.X: X}
)

    def train(self, target_network):
        #sample a random batch from buffer, do an iteration of GD
        if len(self.experience['s']) < self.min_experiences:
            #dont do anything with not enough experience
            return

        idx = np.random.choice(len(self.experience['s']), size = self.
batch_sz, replace=False)
        states = [self.experience['s'][i] for i in idx]
        actions = [self.experience['a'][i] for i in idx]
        rewards = [self.experience['r'][i] for i in idx]
        next_states = [self.experience['s2'][i] for i in idx]
        next_Q = np.max(target_network.predict(next_states), axis=1)
        targets = [r + self.gamma*next_q for r, next_q in zip(rewards,
next_Q)]

        #call optimizer
        self.session.run(
            self.train_op,
            feed_dict={
                self.X: states,
                self.G: targets,
                self.actions: actions
            }
        )

    def add_experience(self, s, a, r, s2):
        if len(self.experience['s']) >= self.max_experiences:
```

```
            self.experience['s'].pop(0)
            self.experience['a'].pop(0)
            self.experience['r'].pop(0)
            self.experience['s2'].pop(0)
        self.experience['s'].append(s)
        self.experience['a'].append(a)
        self.experience['r'].append(r)
        self.experience['s2'].append(s2)

    def sample_action(self, x, eps):
        if np.random.random() < eps:
            return np.random.choice(self.K)
        else:
            X = np.atleast_2d(x)
            return np.argmax(self.predict(X)[0])
```

7. Bibliography

[1] SAE International. “Taxonomy and Definitions for Terms Related to On-Road Motor Vehicle Automated Driving Systems.” On-Road Automated Driving (ORAD) committee.

[2] “Der neue Audi A8 – hochautomatisiertes Fahren auf Level 3“, *Audi MediaCenter*. (11.09.2017). Retrieved from: https://www.audi-mediacenter.com:443/de/per-autopilot-richtung-zukunftdie-audi-vision-vom-autonomen-fahren-9305/der-neue-audi-a8-ochautomatisiertes-fahrenauf-level-3-9307. Accessed on: 13/08/2019

[3] S. Altenburg, “Einführung von Automatisierungsfunktionen in der Pkw-Flotte“, Prognos AG (2018): p. 58. Retrieved from: https://www.adac.de/-/media/pdf/motorwelt/prognos_automatisierungsfunktionen.pdf?redirectId=quer.mwe.prognos-studie. Accessed on: 13/08/2019

[4] S. Lloyd and H. Pagels. “Complexity as thermodynamic depth”. *Annals of physics* 188.1 (1988): pp. 186-213.

[5] J.S. Shiner, M. Davison, and P. T. Landsberg. “Simple measure for complexity”. *Physical review E* 59.2 (1999): pp. 1459-1464.

[6] P. Grassberger. “Randomness, information, and complexity”. *arXiv preprint arXiv:1208.3459* (2012): pp. 6-7.

[7] J. Wang, et al. “Traffic Sensory Data Classification by Quantifying Scenario Complexity”. *2018 IEEE Intelligent Vehicles Symposium (IV).* IEEE, 2018.

[8] A. J. Smola, and B. Schölkopf. "A tutorial on support vector regression." *Statistics and computing* 14.3 (2004): pp. 199-222.

[9] C. Zhang, et al. “A Graded Offline Evaluation Framework for Intelligent Vehicle’s Cognitive Ability.” *2018 IEEE Intelligent Vehicles Symposium (IV).* IEEE, 2018.

[10] R. Danescu, and S. Nedevschi. "Probabilistic lane tracking in difficult road scenarios using stereovision." *IEEE Transactions on Intelligent Transportation Systems* 10.2 (2009): pp. 272-282.

[11] Y. Park, J. H. Yang, S. Lim. “Development of Complexity Index and Predictions of Accident Risks for Mixed Autonomous Driving Levels.” *2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC).* IEEE, 2018.

[12] J. C. Hayward. "Near miss determination through use of a scale of danger." (1972): pp. 24-34.

[13] S. J. Russell, and P. Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited, (2016): p. 226.

[14] P. Lison. "An introduction to machine learning." *Language Technology Group (LTG), 1* 35 (2015).

[15] S. Geman, E. Bienenstock, and R. Doursat. “Neural Networks and the bias/variance dilemma”. *Neural Computation 4* (1992): pp. 1-58.

[16] M. DelSole. *What is One Hot Encoding and How to Do It.* (25.04.2018). Retrieved from https://medium.com/@michaeldelsole/what-is-one-hot-encoding-and-how-to-do-it-f0ae272f1179. Accessed on: 14/08/2019

[17] Skbkekas. Training validation and test sets.
URL: https://en.wikipedia.org/wiki/Training,_validation,_and_test_sets. Accessed on: 14/08/2019.

[18] Glosser, Artificial Neural Network.
URL: https://en.wikipedia.org/wiki/Artificial_neural_network. Accessed on: 15/09/2018. Accessed on: 19/08/2019.

[19] Y. Li, et al. "The improved training algorithm of back propagation neural network with self-adaptive learning rate." *2009 International Conference on Computational Intelligence and Natural Computing*. Vol. 1. IEEE, 2009: pp. 73-76.

[20] S. S. Haykin. *Neural networks and learning machines*. New York: Prentice Hall, 2009: pp. 579-626.

[21] G. Hinton, N. Srivastava, and K. Swersky. "Neural networks for machine learning lecture 6a overview of mini-batch gradient descent." University of Toronto Computer Science (2012): p. 8.

[22] Y. LeCun, K. Kavukcuoglu, and C. Farabet. "Convolutional networks and applications in vision." *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. IEEE, 2010.

[23] Ö. Tuncer. *Entwicklung und Evaluation eines neuronalen Netzes für die bildbasierte Erkennung von Verkehrsteilnehmern mittels Reinforcement Learning*. Master Thesis, Technische Universität München, 2019.

[24] V. Mnih, et al. "Playing atari with deep reinforcement learning." *arXiv preprint arXiv:1312.5602* (2013).

[25] S. Saha. A Comprehensive Guide to Convolutional Neural Networks.
URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53. Accessed on: 21/08/2019

[26] Sigbert. Hierachische Clusteranalyse.
URL: https://de.wikipedia.org/wiki/Hierarchische_Clusteranalyse. Accessed on: 14/08/2019.

[27] A. R. Hall. *Generalized method of moments*. Oxford university press, 2005.

[28] J.C. Loehlin. *Latent variable models: An introduction to factor, path, and structural analysis*. Lawrence Erlbaum Associates, Inc, 1987.

[29] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*, Second edition. Cambridge, Massachusetts: The MIT Press, 2018.

[30] M. E. Harmon, and S. S. Harmon. *Reinforcement Learning: A Tutorial*. No. WL-TR-97-1028. WRIGHT LAB WRIGHT-PATTERSON AFB OH, 1997.

[31] F. Agostinelli, et al. "From Reinforcement Learning to Deep Reinforcement Learning: An Overview." *Braverman Readings in Machine Learning. Key Ideas from Inception to Current State*. Springer, Cham, 2018: pp. 298-328.

[32] R. Bellman. "On the theory of dynamic programming." *Proceedings of the National Academy of Sciences of the United States of America* 38.8 (1952): pp. 716-719.

[33] R. Bellman. "A Markovian decision process." *Journal of mathematics and mechanics* (1957): pp. 679-684.

[34] R. A. Howard. "Dynamic programming and Markov processes." (1960).

[35] D. P. Bertsekas, et al. *Dynamic programming and optimal control*. Vol. 1. No. 2. Belmont, MA: Athena scientific, 1995.

[36] R. E. Kalman. "A new approach to linear filtering and prediction problems." *Journal of basic Engineering* 82.1 (1960): pp. 35-45.

[37] M. Sniedovich. "A new look at Bellman's principle of optimality." *Journal of Optimization Theory and Applications* 49.1 (1986): pp. 161-176.

[38] C. Andrieu, et al. "An introduction to MCMC for machine learning." *Machine learning* 50.1-2 (2003): pp. 5-43.

[39] R. S. Sutton. "Learning to predict by the methods of temporal differences." *Machine learning* 3.1 (1988): pp. 9-44.

[40] G. A. Rummery, and M. Niranjan. *On-line Q-learning using connectionist systems*. Vol. 37. Cambridge, England: University of Cambridge, Department of Engineering, 1994: p. 21.

[41] C. JCH Watkins, and P. Dayan. "Q-learning." *Machine learning* 8.3-4 (1992): pp. 279-292.

[42] G. Brockman , et al. "Openai gym." *arXiv preprint arXiv:1606.01540* (2016).

[43] V. Kumar. Reinforcement Learning: Temporal Difference.
URL: https://towardsdatascience.com/reinforcement-learning-temporal-difference-sarsa-q-learning-expected-sarsa-on-python-9fecfda7467e (2019). Accessed on: 10/08/2019

[44] H. Van Hasselt, A. Guez, and D. Silver. "Deep reinforcement learning with double q-learning." *Thirtieth AAAI conference on artificial intelligence*. 2016.

[45] F. Schuldt. *Ein Beitrag für den methodischen Test von automatisierten Fahrfunktionen mit Hilfe von virtuellen Umgebungen*. Ph.D. dissertation, Technische Universität Braunschweig, Braunschweig, 2017.

[46] G. Bagschik, T. Menzel, and M. Maurer. "Ontology based scene creation for the development of automated vehicles." *2018 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2018.

[47] M M Minderhoud., and P. HL Bovy. "Extended time-to-collision measures for road traffic safety assessment." *Accident Analysis & Prevention* 33.1 (2001): pp. 89-97.

[48] Forschungsgesellschaft für Straßen- und Verkehrswesen e. V: Richtlinien für die Anlage von Autobahnen – RAA08. Köln: FGSV-Verlag, 2008.

[49] C. Roghardt. “Entwicklung und Programmierung herausfordernder Szenarien zur Absicherung der Trajektorienplanung autonomer Fahrzeuge“. Master Thesis at Technische Universität München, 2018.

[50] A. Schuster, et al. “Bestimmen der aktuellen Abmessungen differenzierter Personen-Bemessungsfahrzeuge“. Westsächische Hochschule Zwickau. Institut für Verkehrssystemtechnik. Technische Universität Braunschweig, 2011: p. 10.

[51] EU Directive 96/53: Council Directive 96/53/EC of 25 July 1996 laying down for certain road vehicles circulating within the Community the maximum authorised dimensions in national and international traffic and the maximum authorised weights in international traffic, Brussels 2006.

[52] S. Glück, J. Möller-Töllner. “Was zeichnet die in Deutschland zehn bestverkauften Motorräder aus? Warum liegen sie in der Käufergunst vorne? Preis, Design, Technik, Emotion? Eine Analyse des Motorradmarktes bis September 2015“. Motor Presse Stuttgart GmbH & Co. KG, 2015. Retrieved from: https://www.motorradonline.de/recht-verkehr-branche/bestseller-der-zehn-groessten-motorrad-hersteller.699874.html

[53] M. Treiber, A. Hennecke, and D. Helbing. "Congested traffic states in empirical observations and microscopic simulations." *Physical review E* 62.2 (2000): pp. 1805-1824.

[54] Retrieved from: https://www.mathworks.com/help/reinforcement-learning/ug/train-biped-robot-to-walk-using-ddpg-agent.html

[55] D.P. Kingma, and J. Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980* (2014)

[56] A. Svensson. *A method for analysing the traffic process in a safety perspective*. Doctoral Dissertation at Lund Institute of Technology, 1998: p. 26.