Visual Autonomous Driving with Reinforcement Learning control

Francisco C. Vazquez Donaire[1*], José M. Cañas Plaza[1†] and Roberto Calvo-Palomino[1†]

[1*]RoboticsLabURJC, Universidad Rey Juan Carlos, Madrid, Spain.

*Corresponding author(s). E-mail(s): franciscoc.vazquez@urjc.es;
Contributing authors: josemaria.plaza@urjc.es; roberto.calvo@urjc.es;
†These authors contributed equally to this work.

Abstract

Autonomous vehicle driving has gained attention in the recent years due to the progress of artificial intelligence technologies, and also to the great interest of the society in automating driving. However autonomous driving presents a complex, unfriendly and dynamic scenario from the driver point of view. Different techniques can be used to create a smart driving behaviour to autonomously control the vehicle. In this paper, we focus on autonomous driving systems which integrate computer vision and reinforcement learning techniques to provide the follow-line and the follow-lane behaviours. We propose to use AI-based computer vision techniques for the perception module of the behaviour, which is in charge of the robust detection the lane of the road, and to use the Q-learning reinforcement learning for the control module. Our solutions have been experimentally validated under different conditions in the CARLA simulator, a widely used tool in the community. This work contributes to the field of autonomous driving by leveraging AI-based computer vision and reinforcement learning techniques to improve the capabilities of self-driving. The results obtained pave the way for safer and more efficient autonomous vehicles, bringing us closer to widespread adoption of autonomous driving systems.

Keywords: Autonomous Driving, Reinforcement Learning, Deep Learning

1 Introduction

According to the latest publications of the National Highway Traffic Safety Administration (NHTSA), 94% of car accidents are caused by humans. First step to solve the latter is to incorporate and advanced driver-assistance system (ADAS) which includes technologies that assist and help drivers for driving safely. This technology still needs the human supervision all the time, but it is the core to built systems that allow the car navigate autonomously. Fully autonomous systems have gained attention in the research/industry community thanks to the great advances of computer vision and deep learning technologies. These systems are created with the purpose of reducing the number of accidents, as well as CO_2 emissions and the stress involved in driving through crowded places on a daily basis [1].
These systems require a very precise perception in order to identify the elements of the environment and act accordingly. Moreover, the problem is very complex because the environment can sometimes be uncertain, as it depends on the decisions made by other road users. An example of such a situation is the crossing between two vehicles on a narrow road. For this reason, ADSs must be able to interpret the signals of other drivers and be interpreted by them [2].
At present, the companies that leads the development of these ADSs are primarily Tesla, with its Autopilot, Mercedes, Cruise (General Motors) and Waymo, a company that is part of Alphabet (Google). Even if these companies have successfully developed full autonomous systems and their cars are driving autonomously in real scenarios (Waymo and Cruise offers robotaxi service in San Franscisco), there is still work to do. Tesla is under investigation by National Highway Traffic Safety Administration (NHTSA) due to Tesla autopilot was involved in nearly 750 crashes since 2019, including 17 fatalities[1]. Waymo and Cruise robotaxis have been the main responsible of light accidents and traffic jumps in San Francisco due to software malfunction[2]. Nevertheless the development and self full-driving solutions of these companies shed light on the future of autonomous driving.

This paper aims to investigate and analyze the use of AI-based techniques for developing two different self full-driving behaviours: line and lane following without traffic. Specifically, we focus on computer vision techniques for the perception and Reinforcement Learning for the control. The latter will command the correct decisions to keep the vehicle safely in the lane of the road. We rely on CARLA[3], the open-source autonomous driving simulator widely-accepted in the research community, for experimental validation. Results confirm that these techniques are good candidates to build safe autonomous driving systems. Fig. 1 shows the workflow diagram we have followed in order to implement these agents, note that the most important part is the training of the agent.

[1] https://www.caranddriver.com/news/a44185487/report-tesla-autopilot-crashes-since-2019/
[2] https://www.sfchronicle.com/bayarea/article/sffd-says-two-robotaxis-blocked-ambulance-18343808.php
[3] https://carla.org/

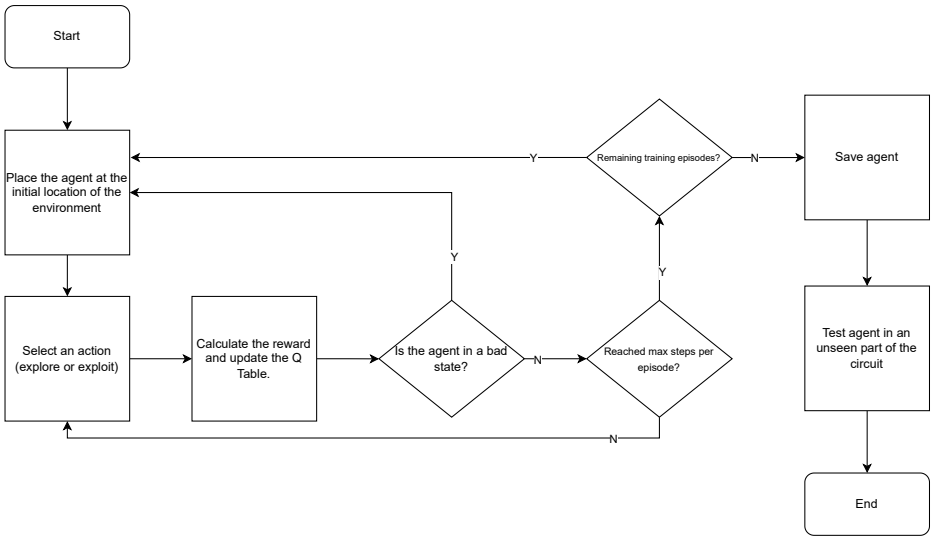

Fig. 1 Workflow diagram that illustrates the complete pipeline of the proposed approach.

2 Related Work

In Machine Learning, the aim for a model is to 'learn' from experience in order to improve its performance in a specific task. In the case of Reinforcement Learning (RL), the agent learns to perform its task through interactions with the environment. These agents are not explicitly programmed but are evaluated using a *reward function*, which provides greater rewards when they take the correct actions for each possible state.

The objective of this agent is to accumulate as many rewards as possible, achieved by exploiting its previously acquired knowledge and selecting actions that yield the highest rewards. However, in order to discover the actions that maximize rewards, the agent must take risks and explore other actions that could potentially result in even greater rewards. The RL agent must balance the exploitation of known strategies to obtain rewards with the exploration of unknown possibilities to uncover potentially better rewards in the future. A trade-off between exploitation and exploration must be achieved [2].

Within reinforcement learning there are several families of methods. On the one hand, we have the value-based methods; among others, Q-learning. And on the other hand, the policy-based methods, among which Proximal Policy Optimization (PPO) or Trust Region Policy Optimization stand out TRPO.

2.1 Value-based methods

One of the most well-known algorithms in the field of RL is the *Q-learning* algorithm. This algorithm utilizes the information gained from taking a step, which involves selecting an action, and uses this information to update the current *Q-value*. In *Q-learning*, the estimation of the *Q-value* is obtained through the following iterative process [3].

$$Q_{t+1}(x, a) = Q_t(x_t, a_t) + \alpha \cdot [r_t + \gamma \cdot V(y) - Q_t(x_t, a_t)] \tag{1}$$

where r_t is the reward obtained, γ is the discount factor, $V(y)$ will be calculated as the maximum reward obtained when switching to the new state and α is the learning rate.

Other methods have emerged in the literature as extensions to the aforementioned approach. In [4], a novel algorithm called *Double Q-learning* is proposed, which demonstrates improved performance in highly stochastic environments where *Q-learning* shows limitations. These limitations arise from the fact that *Q-learning* approximates the action with the highest rewards as the action that will yield the maximum reward. The authors address this approximation issue by introducing a double estimator, applied to *Q-learning* to yield *Double Q-learning*. This enhanced algorithm demonstrates the ability to converge to the optimal policy in situations where *Q-learning* previously exhibited poor performance. Experimental evaluations were conducted on roulette and maze games, yielding promising and faster results.

Furthermore, other methods have been developed that combine neural networks with classic RL algorithms. Many significant advancements in *Deep Reinforcement Learning* have been achieved by scaling previous work to higher-dimensional problems. A prominent example of such an algorithm is the *Deep Q Network (DQN)*, introduced by researchers at the *DeepMind* lab [5]. The *DQN* algorithm is capable of learning optimal policies from input signals with very high dimensions. The research demonstrated that agents of this kind, which only receive pixel data and game outcomes as input, surpassed all previously tested algorithms, reaching the skill level of professional Atari video game players. Moreover, this algorithm has been recently improved in [6], combining *DQN* with *Double Q-learning*. Again, this algorithm has been tested on Atari 2600 games, surpassing the results obtained in the previous work [5].

One of the prevailing challenges in the field of robotics revolves around the issue of navigating through unfamiliar environments. As evidenced by Ruan et al. [7], RL appears to offer a robust and end-to-end solution to this problem. In this study, the researchers employ an RGB-D image captured by a Kinect camera and introduce a novel algorithm named Dueling Double DQN (D3QN). The findings presented in the paper strongly support the notion that autonomous learning from the environment, devoid of external supervision, is highly effective. The robot can autonomously navigate the environment while concurrently avoiding collisions with obstacles.

2.2 Policy-Based Methods

Policy-based methods focus on learning a policy directly, rather than estimating value functions. A policy defines the mapping from states to actions, indicating the best action to take in each state. These methods aim to optimize the policy by adjusting its parameters, often using gradient ascent techniques.

PPO, or Proximal Policy Optimization, stands as one of the most popular algorithms in this field. It has found practical applications in diverse areas, including the stabilization control of quadcopters. In a study conducted by Cano et al. [8], PPO was employed to enable the agent to acquire an effective control policy for precise stabilization of a drone with these specifications within a simulated environment. The results demonstrated a substantial improvement compared to prior research efforts ([9]), notably reducing convergence time while maintaining high performance.

Another algorithm that utilizes policy-based reinforcement learning along with neural networks is DDPG (Deep Deterministic Policy Gradient). While DDPG has found applications in various problem domains, noteworthy is the work done by DeepMind researchers in [10], who employed this algorithm for trajectory planning in mobile robots. The researchers identified weaknesses in DDPG, such as slow training efficiency and convergence. To address these limitations, they introduced enhancements to the algorithm, incorporating a small amount of prior knowledge to reduce trial and error. Additionally, they adopted an adaptive exploration policy based on the epsilon-greedy algorithm. In their study, the performance of this modified algorithm was compared to Q-learning and SARSA, with results showing superior path planning, reduced computational time, and faster convergence.

2.3 Reinforcement Learning in Autonomous Driving

Inside Autonomous Driving, some tasks where RL algorithms have been successfully applied are: Controller optimization, Path planning, Path optimization, Motion planning and Dynamic motion planning.
In [11], the *Q-learning* algorithm is utilized to develop a controller for enabling a robot to solve the line-following application. More specifically, they employ a technique called *SA-based Q-learning* to address the exploitation-exploration trade-off. The results of their study demonstrate that this technique outperforms both the e-greedy algorithm and the P-controlled algorithm when trained for a sufficient number of episodes. The results obtained in this research are illustrated in Fig. 2.

The best score out of five tries

Algorithm	Score
P controlled	516.32
ε-greedy Q-learning MISO	432.93
SA-based Q-learning MISO	433.34
SA-based Q-learning MIMO	518.95

Fig. 2 Experimental results in the complex circuit [11]

When applying RL algorithms to the autonomous driving problem, one of the major concerns arises when the *agent* encounters scenarios that did not appear during its training. In such cases, the behavior of the *agent* can become unpredictable. To address this issue, a solution is proposed in [12], where the authors suggest combining deep reinforcement learning with safety-based control techniques. These techniques have demonstrated effective performance in avoiding collisions with nearby vehicles.

One framework very useful in recent years to solve problems related to autonomous driving with RL is AWS DeepRacer, which has its own competitive league in which developers can test their ML systems, both on their physical device and in their 3D simulator. One illustrative work using this environment is Jacob et al. [13], in which they manage to improve the default models of the platform when finding the optimal route while avoiding obstacles, mixing algorithms such as A* and LoS with reinforcement learning.

Another work that takes advantage of this platform is Zhu et al. [14] where the authors applied path planning into DeepRacer based on RL. They proposed a novel system framework called VNARM (Vehicle Network Autonomous Racing Model), which allows multiple DeepRacer cars to cooperate with each other to achieve a common goal. VNARM was able to outperform state-of-the-art autonomous driving algorithms in terms of racing time and completion rate. Specifically, the vehicle's performance of finishing one lap was increased from nearly 30 seconds to less than 9 seconds, while maintaining a high percentage of completion.

In paper [15], the authors argue that the price of training a model in DeepRacer Console is too expensive for beginners, so they present a new simulation process in DeepRacer built on the EC2 platform [16], which is more economical.

In the work presented in [17], Recurrent Neural Networks (RNNs) are incorporated with attention models with the objective of reducing the computational complexity in order to deploy the autonomous driving system on embedded hardware. More specifically, the authors develop an end-to-end framework consisting of using RNNs in Deep Reinforcement Learning to consider POMDP scenarios. Specifically, they succeed in developing a framework for lane keeping that was tested in the TORCS simulator.

3 Follow-line driving

The first Autonomous Driving application we coped with RL, more specifically, with the algorithm Q-learning, is the line following. The first step in solving it was to find a circuit on which to train and test the solution. Given that CARLA presents very realistic environments, we conducted an exhaustive search in all the towns that this simulator offers. Ultimately, we selected a segment in *Town07*. The reason for this choice was that this circuit offered favorable conditions in terms of line perception.

3.1 Perception

To tackle this task, we employed a color filter centered on yellow (the color of the line), followed by successive morphological operations of erosion and dilation. In order to increase the robustness of the image processing algorithm, the image has been processed in the HSV color space.

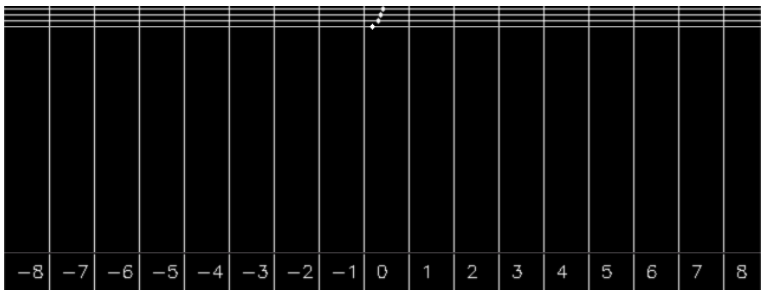

Fig. 3 Example of a processed frame where $state = 0, 0, 0, 0$

Fig. 4 Raw image obtained from the vehicle's on-board camera (top). Image processed with a color filter and various morphological operations (bottom).

3.2 Decision Making

With that perception the RL states and the reward function were defined for the Q-Learning algorithm. In this case, we divided the image into n equidistant vertical strips. Next, using the image provided by the (segmented) perception algorithm, we calculated the midpoint of the line at different image heights, as shown in Fig. 5. This midpoint defines the current state. Specifically, in this work we have used 4 perception lines to define the states.

Fig. 5 Example of a processed frame in which we calculated the midpoint at four different heights of the image.

The reward function was defined as follows. Let C_i be the centers of the line at different heights, with $i \in \{1, ..., 4\}$, and being $\bar{C} = (C_1, C_2, C_3, C_4)$. First, we need to define an auxiliary function, f.

$$f(C) = \begin{cases} 15 & \text{if } 0 \leq |320 - C| \leq 38 \\ 12 & \text{if } 38 \leq |320 - C| \leq 2 \cdot 38 \\ 10 & \text{if } 2 \cdot 38 \leq |320 - C| \leq 3 \cdot 38 \\ 2 & \text{if } 3 \cdot 38 \leq |320 - C| \leq 5 \cdot 38 \\ 1 & \text{if } 5 \cdot 38 \leq |320 - C| \leq 6 \cdot 38 \\ -100 & \text{if } 6 \cdot 38 \leq |320 - C| \end{cases} \tag{2}$$

Let v be the speed of the car in m/s, given by the simulator. Let α be the steering angle of the selected action. Let be $\lambda = 1.02$. Then, our reward function is given by

the expression (3)

$$R(\bar{C}, v, \alpha) = \sum_{j=1}^{4} \frac{1}{4} f(C_j) + \frac{1}{2} v - \lambda\alpha \tag{3}$$

This function gives a higher reward to the agent for placing himself in central positions with respect to the line. In addition, we will reward this agent for reaching higher speeds, and penalize him for making too many turns, in order to avoid zig-zagging. It is worth mentioning that this reward function is based in the one proposed in [18], with some modifications which have improved its performance.

The actions configured for this algorithm are shown in the following table:

Action	0	1	2	3	4
Throttle	0.32	0.7	0.32	0.2	0.2
Sterring Angle	-0.2	0.0	0.2	-0.4	0.4

3.3 Experimental results

With this configuration of states, actions and rewards, we get the agent to learn to solve the task. The results of training with all of these parameters can be seen in the following video, published in https://www.youtube.com/watch?v=-yCdYJnClME.

The correct operation of the agent is highly conditioned by the selection of hyper-parameters, which have been experimentally chosen: α, the learning rate, is set to 0.8; γ, the discount factor, to 0.9; ε, the exploration factor, to 0.9999; and ϕ, that represents the reduction factor applied to ε, is set to 0.9998.

In order to ensure the proper functioning of the algorithms, a series of metrics have been extracted from the conducted training sessions. The most important metric in order to decide when to stop a training session is the accumulated reward. This metric consists of the sum of the rewards that the agent has been obtaining in a given episode each time it takes a *step*, i.e., each time it takes an action in a specific state. Ideally, this accumulated reward per episode should increase as the training progresses until convergence is reached.

Fig. 6 shows a graph that reflects the accumulated reward throughout all the epochs of a training session. The Fig. shows that the accumulated reward is set to a fixed value.

One aspect that is important to highlight is that, initially this algorithm was trained without taking into account the synchrony between the simulator and the client. This introduced problems when calculating the reward obtained by performing an action in a certain state, since we were calculating the reward of a past state and not the current one. First, we tried to solve this problem by changing the states, the rewards

and the hyperparameters. Finally, we realized that the problem was in the lack of synchrony in CARLA. Enforcing the synchronous mode in CARLA the results went from a graph that did not converge to any established value to what is shown in the Fig. 6.

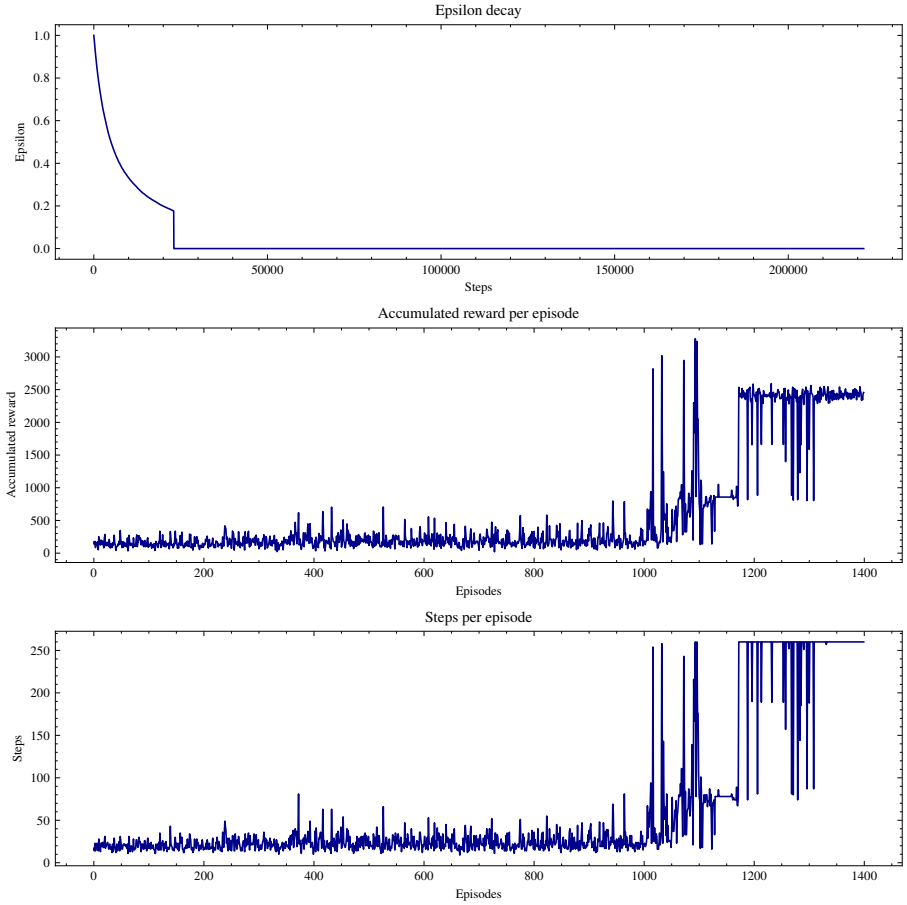

Fig. 6 Graphs showing the convergence of the algorithm. First graph, show the epsilon decay along every step of the training session. Second, the accumulated reward per episode. And last, the number of total steps per episode.

4 Follow-lane driving

The second Autonomous Driving application we coped with the Q-learning algorithm was lane following. Although this task may seem very similar to the previous one, we must emphasize that they are very different tasks. In the case of *follow-line*, we can perform a simple image processing, based on a color filter in *HSV* space that always or almost always provides good results when calculating the current state of

the agent. In addition, having a line as a reference, we can give more slack to the agent, since it has to deviate a lot to stop seeing completely the line it has to follow. In the case of *follow-lane*, we must use more complex visual techniques and be much stricter when restarting the agent.

4.1 Perception

Perception is one of the most important elements that allow this type of algorithms to work correctly. If we want to make good decisions, we must make sure that the perception that the agent is making is robust and correct. That is why in this work we have used neural networks. Specifically, we have used a convolutional neural network (CNN) trained for segmentation. This network receives the image captured by the agent's camera and gives as a result two images: the image with the pixels belonging to the left line of the lane in white, and the same image but with the pixels of the right line of the lane in white.

In this way, we can use the output of this CNN to calculate the midpoint of the lane at different heights of the image, so that, as we did in the case of follow-line, we can calculate the state in which the agent is.

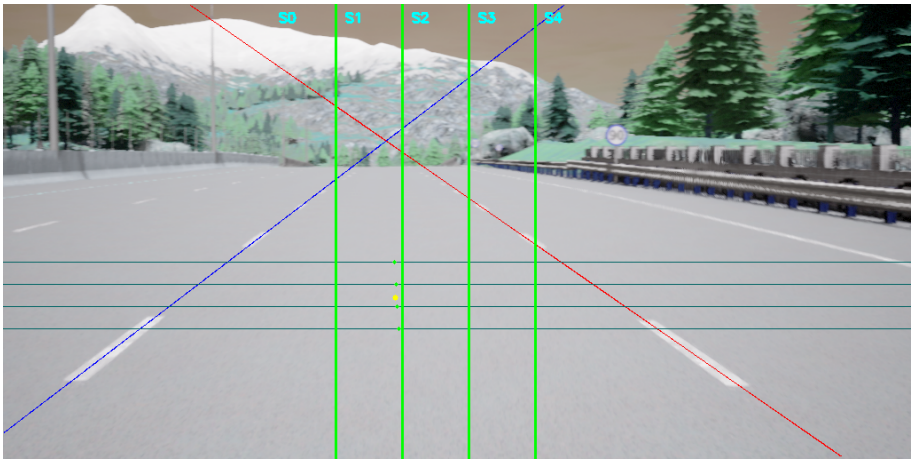

Fig. 7 Example of a processed frame where $state = s1$

In Fig. 7, we can observe several key elements:

- The green vertical lines delineate different states.
- Four lane centers are indicated by green circles.
- The current state, where the agent is located, is highlighted in yellow.
- The output of processing the frame with the CNN is represented by the blue and red lines, which correspond to the lane boundary pixels.

4.2 Decision making

Just as we did for the follow-line application, we have developed a visual controller based on the Q-learning algorithm. In this case, we defined five different states, as illustrated in Fig. 7. These states are determined by calculating the average x-coordinate of four distinct lane centers at various rows.

The reward function utilizes the average x-coordinate, as well as the four points representing the lane center. It calculates a reward based on the distance and angle between the lane and the vehicle, resulting in different rewards. Formally, we defined the reward function:

Let C be the x coordinate of the point marking the center of the lane (in the Fig., highlighted in yellow), calculated as the mean of the four points marked in green in that Fig. It is noteworthy that these points are obtained from the output of the trained network for lane detection. Then, a auxiliary function, f^*, is defined as follows:

$$f^*(C) = \begin{cases} 10 & \text{if } 0 \leq |512 - C| \leq 15 \\ 5 & \text{if } 15 \leq |512 - C| \leq 2 \cdot 15 \\ 2.5 & \text{if } 2 \cdot 15 \leq |512 - C| \leq 3 \cdot 20 \\ 1 & \text{if } 3 \cdot 20 \leq |512 - C| \leq 4 \cdot 20 \\ -10 & \text{if } 4 \cdot 20 \leq |512 - C| \end{cases}$$

Let be $p_1 = (512, 512)$ and $p_2 = (512, 0)$. And let be p_3, p_4, two of the middle of the lane points detected in the image. It is also necessary to determine the angle between a vertical line in the image, defined by the vector $\overrightarrow{p_1 p_2}$, and the line defined by the vector $\overrightarrow{p_3 p_4}$.

With this information, the auxiliary function g^* is defined to calculate this angle as

$$g^*(p_1, p_2, p_3, p_4) = \arctan\left(\frac{p_{4_y} - p_{3_y}}{p_{4_x} - p_{3_x}}\right) - \arctan\left(\frac{p_{2_y} - p_{1_y}}{p_{2_x} - p_{1_x}}\right)$$

In this function, two terms can be observed. The first of them calculates the angle formed by the vector $\overrightarrow{p_3 p_4}$ with the horizontal, and the second one calculates the angle formed by the vector $\overrightarrow{p_1 p_2}$ with the same horizontal. Subtracting both terms yields the desired angle.

Now, let a be the action selected at a specific instant, chosen from among those described in the previous section. Under these conditions, the sought reward function, R^*, can be defined as

$$R^*(C, a, p_1, p_2, p_3, p_4) = \begin{cases} f^*(C) & \text{if } g^*(p_1, p_2, p_3, p_4) > \zeta \\ 2f^*(C) & \text{if } g^*(p_1, p_2, p_3, p_4) \leq \zeta \wedge a = 3 \\ \dfrac{f^*(C)}{2} & \text{if } g^*(p_1, p_2, p_3, p_4) \leq \zeta \wedge a \neq 3 \end{cases}$$

As can be observed, this reward function grants a higher reward to the agent when it is aligned with the lane, that is, when $g^*(p_1, p_2, p_3, p_4) < \zeta$, and it has selected the action to go straight, meaning $a = 3$.

The parameter ζ has been set to 6, and in this case, it has been determined experimentally.

This function is a novel contribution of the paper, as we have not founnd any similar one in the literature.

Now that we have thoroughly examined the reward function, we will proceed to discuss the available actions for the agent. Specifically, the following five actions have been set up:

Action	0	1	2	3	4
Throttle	0.35	0.35	0.3	0.6	0.3
Sterring Angle	-0.04	0.04	-0.08	0.0	0.08

The selection of actions is carefully tailored to the agent's environment, which is anticipated to primarily involve highway driving scenarios. Consequently, the chosen actions include high accelerations, which give the agent an average speed of 30 km/h.

4.3 Experimental results

The main objective of this application is to keep the car inside the lane while driving smoothly. The model has been trained with the hyper-parameters: α, is set to 0.8, γ to 0.9, ε to 0.9999, ϕ to 0.9998.

To ensure the convergence of the algorithm and to terminate its training, we used the cumulative reward per episode as a metric. This cumulative reward is reflected in Fig. 8 (middle) in which it can be seen that after about 550 episodes the system converges getting the maximum reward.

As we have discussed in section 3.3, to achieve convergence in this problem we have also needed to activate the CARLA synchronous mode. The difference between using

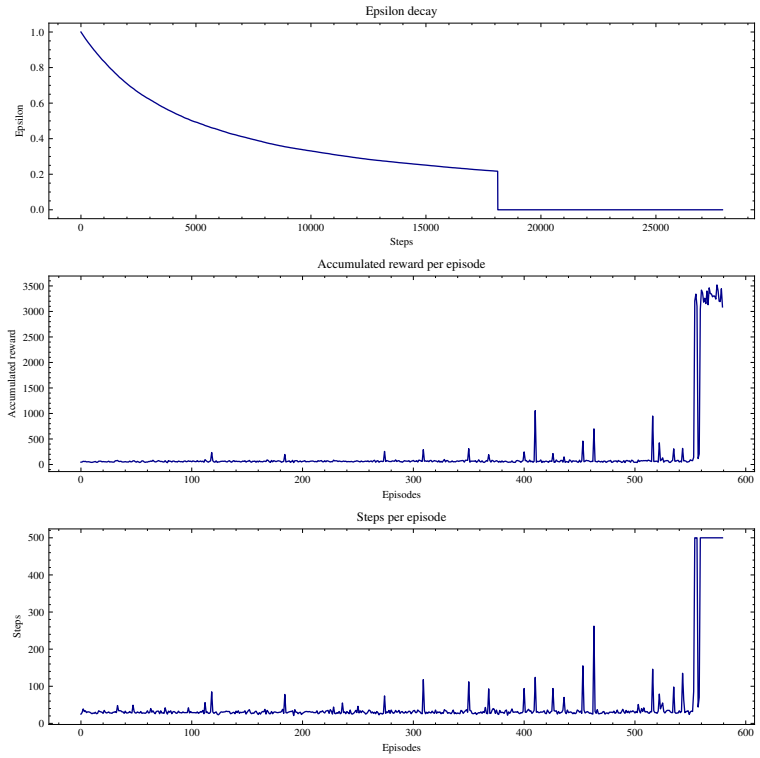

Fig. 8 Graphs showing the convergence of the algorithm. First graph, show the epsilon decay along every step of the training session. Second, the accumulated reward per episode. And last, the number of total steps per episode

this mode and not doing so becomes noticeable in the cumulative reward plot, since the use of this mode makes possible a convergence of the cumulative reward to a particular value.
In the Fig. 9 it can be seen how the vehicle is able to navigate properly though the lane. For a better understanding of the behaviour and scenario an illustrative video of the operation of this agent can be found in https://www.youtube.com/watch?v=_2ma1SqN1MY.

5 Conclusions

We have studied the problem of vision-based autonomous driving in a realistic simulator (CARLA) and developed two different prototype behaviours: follow-line and follow-lane. In both, reinforcement learning (Q-learning) approach has been used for decision making. We have programmed our own implementation of this algorithm in CARLA for both line and lane following applications. We have experimentally validated the convergence of the algorithms and their safely driving along the road. The trained agent is able to control the vehicle solely using visual information. One

Fig. 9 Vehicle navigating along Town04 in curves situation (left) and in a straight line (right).

key factor for convergence is the activation of the synchronous mode on the CARLA simulator. This work shows that vision-based RL is a robust approach for several simple tasks in Autonomous Driving domain. However, it is worth to mention that these visual-controllers are highly dependent on the implemented perception, which will be less robust in real scenarios. In the case of lane following, it would be desirable to retrain the lane detection network with real images, in order to reduce this simulation-reality gap.

One main future line of this work is extending the capabilities of the agent, making the vehicle to successfully drive in environments where other agents are also present. This could be addressed with other more complex deep reinforcement learning algorithms, such as DQN or DDPG.

6 Acknowledgments

This research work has been supported by the following projects:

- TED2021-129162B-C22, funded by the Recovery and Resilience Facility program from the NextGenerationEU Plan of the European Union and the the Spanish Research Agency.
- PID2021-128362OB-I00, funded by the Spanish Plan for Scientific and Technical Research and Innovation of the Spanish Research Agency.

References

[1] Yurtsever, E., Lambert, J., Carballo, A., Takeda, K.: A survey of autonomous driving: Common practices and emerging technologies. IEEE Access **8**, 58443–58469 (2020) https://doi.org/10.1109/ACCESS.2020.2983149

[2] Kiran, B.R., Sobh, I., Talpaert, V., Mannion, P., Sallab, A.A.A., Yogamani, S., Pérez, P.: Deep reinforcement learning for autonomous driving: A survey. IEEE Transactions on Intelligent Transportation Systems **23**(6), 4909–4926 (2022) https://doi.org/10.1109/TITS.2021.3054625

[3] Chen, S.-L., Wu, H.-Z., Han, X.-L., Xiao, L.: Multi-step truncated q learning algorithm. In: 2005 International Conference on Machine Learning and Cybernetics, vol. 1, pp. 194–198 (2005). https://doi.org/10.1109/ICMLC.2005.1526943

[4] Hasselt, H.: Double q-learning. In: Lafferty, J., Williams, C., Shawe-Taylor, J., Zemel, R., Culotta, A. (eds.) Advances in Neural Information Processing Systems, vol. 23. Curran Associates, Inc., ??? (2010). https://proceedings.neurips.cc/paper/2010/file/091d584fced301b442654dd8c23b3fc9-Paper.pdf

[5] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M.A., Fidjeland, A., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., Hassabis, D.: Human-level control through deep reinforcement learning. Nature **518**, 529–533 (2015)

[6] Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. Proceedings of the AAAI Conference on Artificial Intelligence **30**(1) (2016) https://doi.org/10.1609/aaai.v30i1.10295

[7] Ruan, X., Ren, D., Zhu, X., Huang, J.: Mobile robot navigation based on deep reinforcement learning. In: 2019 Chinese Control And Decision Conference (CCDC), pp. 6174–6178 (2019). https://doi.org/10.1109/CCDC.2019.8832393

[8] Cano Lopes, G., Ferreira, M., Silva Simões, A., Luna Colombini, E.: Intelligent control of a quadrotor with proximal policy optimization reinforcement learning. In: 2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE), pp. 503–508 (2018). https://doi.org/10.1109/LARS/SBR/WRE.2018.00094

[9] Hwangbo, J., Sa, I., Siegwart, R., Hutter, M.: Control of a quadrotor with reinforcement learning. IEEE Robotics and Automation Letters **PP**, 1–1 (2017) https://doi.org/10.1109/LRA.2017.2720851

[10] Dong, Y., Zou, X.: Mobile robot path planning based on improved ddpg reinforcement learning algorithm. In: 2020 IEEE 11th International Conference on Software Engineering and Service Science (ICSESS), pp. 52–56 (2020). https:

//doi.org/10.1109/ICSESS49938.2020.9237641

[11] Saadatmand, S., Azizi, S., Kavousi, M., Wunsch, D.: Autonomous control of a line follower robot using a q-learning controller. In: 2020 10th Annual Computing and Communication Workshop and Conference (CCWC), pp. 0556–0561 (2020). https://doi.org/10.1109/CCWC47524.2020.9031160

[12] Xiong, X., Wang, J., Zhang, F., Li, K.: Combining deep reinforcement learning and safety based control for autonomous driving (2016). https://arxiv.org/abs/1612.00147

[13] McCalip, J., Pradhan, M., Yang, K.: Reinforcement learning approaches for racing and object avoidance on aws deepracer. In: 2023 IEEE 47th Annual Computers, Software, and Applications Conference (COMPSAC), pp. 958–961 (2023). https://doi.org/10.1109/COMPSAC57700.2023.00129

[14] Zhu, W., Du, H., Zhu, M., Liu, Y., Lin, C., Wang, S., Sun, W., Yan, H.: Application of reinforcement learning in the autonomous driving platform of the deepracer. In: 2022 41st Chinese Control Conference (CCC), pp. 5345–5352 (2022). https://doi.org/10.23919/CCC55666.2022.9902325

[15] Li, J., Abusharkh, M., Xu, Y.: Deepracer model training for autonomous vehicles on aws ec2. In: 2022 International Telecommunications Conference (ITC-Egypt), pp. 1–5 (2022). https://doi.org/10.1109/ITC-Egypt55520.2022.9855675

[16] Services, A.W.: Amazon Elastic Compute Cloud (EC2). https://aws.amazon.com/ec2/ Accessed 2023-09-22

[17] Sallab, A., Abdou, M., Perot, E., Yogamani, S.: Deep reinforcement learning framework for autonomous driving. Electronic Imaging **2017**, 70–76 (2017) https://doi.org/10.2352/ISSN.2470-1173.2017.19.AVM-023

[18] Amazon Web Services: Amazon DeepRacer Documentation. https://docs.aws.amazon.com/deepracer/latest/developerguide/deepracer-reward-function-examples.html