End-to-end autonomous driving using the Ape-X algorithm in Carla simulation environment

Maxence Hussonnois
Applied Artificial Intelligence Institute
Deakin University, *Burwood, Australia*

Jae-Yun Jun
Applied Mathematics
ECE Paris, *France*
jaeyunjk@gmail.com

***Abstract*—Despite great advances in controlling vehicles for autonomous driving and in deep reinforcement learning (DRL) techniques, designing an end-to-end architecture that supports autonomous driving using DRL techniques while facing uncertainties in complex and dynamic environments still remains challenging. By examining the state-of-the-art works in the domain of DRL for autonomous driving and inspired from the work of [1], we have designed an end-to-end autonomous driving system using the Ape-X algorithm [2] in Carla simulation environment [3] and have evaluated the performance by comparing its results to those that are obtained using other DRL techniques.**

***Index Terms*—Autonomous Driving; Deep Reinforcement Learning; End-to-End Architecture; Neural Networks**

I. Introduction

In the context of autonomous driving, we are interested in responding to the research question of how to realize a robust end-to-end system able to learn to drive autonomously facing uncertainties in complex and dynamic environments (e.g., an urban area). Such a system can be designed using the imitation-learning approach [6]. Bojarski et al. [4] proposed an end-to-end learning architecture for self-driving based on a convolutional neural network (CNN) [5], which receives images taken from cameras as its inputs and outputs the steering angles as action commands to control the vehicle's motion. The CNN is learned from the training data that consist in input images and the corresponding steering angles. In fact, the steering angles are returned by some expert remote driver. Hence, in this case, the autonomous driving task is approached by an imitation-learning method to optimize the CNN parameters. Codevilla et al. [7] observed that, while imitation learning is useful for self-driving during the training time, it cannot be used to control the vehicle during test time. To overcome this issue, the authors proposed an end-to-end system with a method that conditions the imitation learning with the possibility to have the policy function during test time.

On the other hand, the end-to-end autonomous driving system can also be designed using the deep reinforcement learning (DRL) approach. Jaritz et al. [8] proposed an end-to-end race driving method using a DRL algorithm (instead of using the imitation-learning approach as the previous authors did). In particular, they chose the Asynchronous Advantage Actor-Critic (A3C) algorithm [9] to train their architecture because it is considered to be well suited for experience decorrelation. Hu et al. [10] also worked on the development of an end-to-end system using DRL approach but for a more specific task, which is the automated lane-changing task. They realized that most methods employed for automated lane-changing task are rule-based, which are often unsuitable for unpredictable scenarios. The automated lane-changing task requires the coordination of both lateral and longitudinal controls of the vehicle while considering the vehicle state (i.e., its position and velocity) and the state of the surrounding environment. For this task, instead of using rule-based methods, the authors proposed an end-to-end method based on LIDAR data using a DRL algorithm called the Deep Deterministic Policy Gradient (DDPG) approach [11]. Within this framework, the reward function is composed in safety, comfort, and efficiency. Depending on the importance given among these components, they considered two different driving styles and measured the performance of lane-changing task in terms of the required time for achieving the task.

In the literature, we also find survey works on DRL for autonomous driving such as [12], [13]. For autonomous driving, the direct application of reinforcement learning algorithms with noisy images coming from visual sensors often adds difficulties to find optimal solutions. Indeed, the noisy images contain complex information such as the road texture, obstacles, weather conditions, the luminosity, and more. To reduce this complexity, Toromanoff et al. [1] proposed a method called *Implicit Affordance*, which consists of separating the training stage in two phases. During the first phase, an encoder is trained to predict a situation having for instance traffic lights or the distance between the road centerline and the vehicle position. Then, during the second phase, the encoded features are recovered and fed as observations to a reinforcement learning algorithm.

Although there have been great advances in controlling vehicles for autonomous driving and in DRL techniques, designing end-to-end autonomous driving architectures based on DRL techniques while facing uncertainties in complex and dynamic environments still remains challenging. In this work, motivated by the work of [1], we designed an end-to-end autonomous driving system using the Ape-X algorithm [2] in Carla simulation environment [3]. We compare the results obtained from the Ape-X algorithm to other state-of-the-art DRL techniques and show the benefits of designing the end-to-end autonomous driving architecture with this DRL technique.

Fig. 1. *Town02* Map in Carla environment [3] with a desired path that the vehicle should follow. The path is defined using waypoints represented on the map by blue bars with pink circles on them.

II. Related works

Recently, there have been great advances in the domain of deep reinforcement learning to learn optimal (or sub-optimal) actions for a variety of tasks, among which the autonomous driving. Mnih et al. [14] proposed the Deep Q-Network (DQN), which is a value-based, off-policy and model-free deep reinforcement learning algorithm. This proposal was motivated by the fact that most reinforcement learning algorithms that existed by then (such as Q-learning) suffered from instabilities and divergences (for instance, when nonlinear function approximators such as neural networks were used to represent the action-value function (i.e., the Q-function)). Such instabilities seem to be caused by: 1) the correlations that exist between the observations, 2) the fact that Q-value updates may significantly change the policy (and therefore the data distribution), 3) and the correlation between the action-values (i.e., Q) and the temporal-difference target values. The DQN endeavors to solve these issues by using the experience replay [15] (which randomizes the data to remove correlations in the observations and to smooth over changes in the data distribution) and by updating the action-values (i.e., Q) towards target values at a certain period (which reduces the correlation between the action values and the target values).

However, DQN tends to overestimate the rewards returned from noisy environments rendering training results that are not in fact optimal. Further, in DQN, a same neural network is responsible for both the action choice and the evaluation of such choice, and this often causes the instability during training stage. In order to overcome these issues, Van Hasselt et al. [16] proposed an improved version of the DQN, called the Double Deep Q-Network (DDQN). DDQN attempts to reduce the overestimation by decomposing the max operation corresponding to the target network in 1) action selection and then 2) selection evaluation. That is, the action is chosen by the principal network, and the action-value (i.e., Q) is evaluated by the target network.

Wang et al. [17] pursuit to further improve, through the Dueling DDQN, the performance of the DDQN by reformulating the action-value function (i.e., Q) using the state-value function (i.e., V) and the advantage function (i.e., A):

$$Q(s,a) = V(s) + A(s,a). \tag{1}$$

In accordance to this reformulation, the Q-network is now designed in two separate networks that represent the state-value function and the advantage function. The outputs of these two networks are aggregated in a posterior layer to obtain the action-value function. However, this can cause instability in training with large changes in Q-values. To smooth this, the regularized version (with the average of advantages) is proposed to allow smaller changes in Q-values:

$$Q(s,a) = V(s) + A(s,a) - \frac{1}{|A|}\sum_{a'} A(s,a'). \tag{2}$$

Fortunato et al. [18] further improves the performance of DQN and Dueling DDQN by adding a noisy network to their architectures. In the existing deep reinforcement learning algorithms, the exploration-exploitation trade-off is often modeled using the simple epsilon-greedy approach. The authors of this work endeavor to explore new solutions using a noisy network, and they effectively obtained better results than using the simple epsilon-greedy approach.

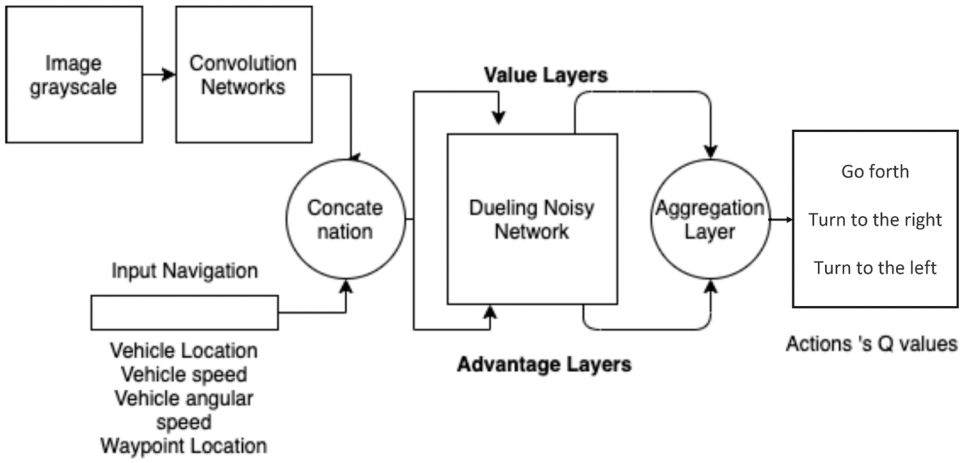
As mentioned earlier, one of the contributions of the DQN is the experience replay, from which a number of transition samples are randomly chosen following the uniform distribution. Schaul et al. [19] realized that the performance of the DQN can be further increased by prioritizing certain transition samples from the experience replay. Such method is known as the Prioritized Experience Replay (PER). The prioritized transition samples are those that correspond to large temporal difference errors, meaning that these transition samples correspond to those that need to be "significantly" improved (in opposition to those whose temporal difference errors are small, for which there is only a small gain remaining for learning).

Horgan et al. [2] go beyond the work of prioritized experience replay [19]. Inspired from the work of [20], the acting phase is decoupled from the learning phase. The actors select actions to interact with the environment through a shared neural network, while the learner replays transition samples prioritizing the most significant ones generated by the actors and updates the neural network. This approach is called Ape-X and demonstrates even further improvement with respect to other state-of-the-art deep reinforcement learning algorithms.

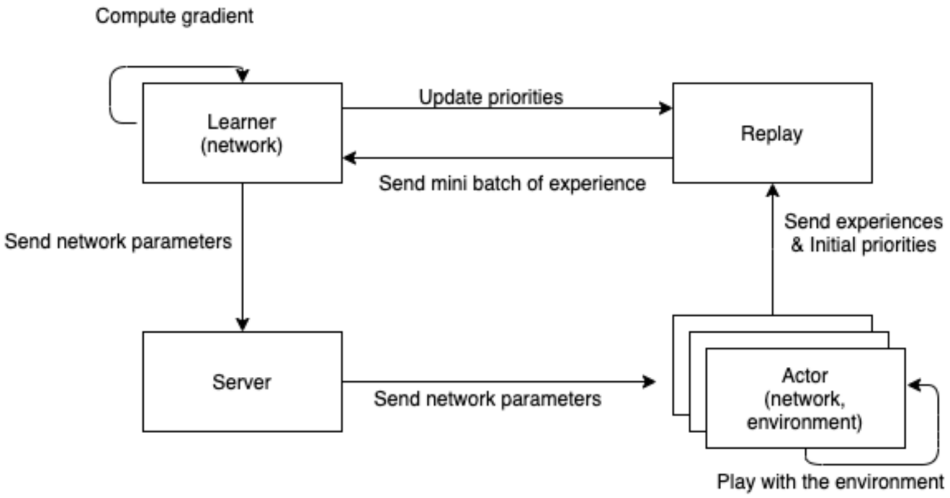
On the other hand, there exist several simulators to develop, train, and validate autonomous driving [21]: Carla [3], Gazebo [22], LG Silicon Valley Lab (LGSVL) [23], and others. For the present work, we used Carla, while we could have used any other simulator listed hereby. Carla is an open-source 3D simulator that supports development, training, and validation of autonomous driving systems with diverse sensor types (such as RGB cameras, LIDAR, radar, collision detectors, line-change detectors, etc.) under various environmental conditions over numerous maps [3].

III. Methods

By examining the state-of-the-art works listed in Section II and also inspired from the work of [1], we designed an end-



(a) Distributed Prioritized Experience Replay (Ape-X) formed by a convolution neural network, a concatenation layer, a dueling noisy network, and a aggregation layer to get desired actions from vehicle and environment states.



(b) Details of Distributed Prioritized Experience Replay (Ape-X) consisting of a server, a learner network, an experience-replay memory, and actor networks.

Fig. 2. Distributed Prioritized Experience Replay (Ape-X) [2] and its details

to-end autonomous driving system using the Ape-X algorithm in Carla simulation environment. Initially, we suppose that the desired path is given in urban areas (i.e., suppose that the desired path is obtained through Apps such as Google Map), and we solely focus on controlling the vehicle's movement to follow the waypoints that define the desired path. We formulate this path-following problem as a reinforcement learning problem. That is, our problem can be described within the Markov Decision Process (MDP) framework $(S, A, P_{s,a}, R, \gamma)$, where $S, A, P_{s,a}, R, \gamma$ respectively represent the state space, the action space, the state-transition distribution, the instantaneous reward, and the discount factor.

The environment state is represented from the agent's (i.e., the vehicle's) perspective. That is, by images of size 150×150 pixels taken from an RGB camera mounted frontally on the vehicle. These images are then converted into grayscale images in order to reduce the data size that need to be processed later using neural networks, which in turn enables to quickly make decision based on the received information. Next, the agent's state is defined by its position, velocity, and angular velocity. We also use a bumper sensor on the vehicle to detect whether it collides with objects that are present on the map (i.e., the collision state). In this work, the agent can take three possible actions: go forth, turn to the right, and turn to the left. These desired actions are expressed with some constant forward and angular speeds. Further, the state transition through an action taken by the agent is represented by the dynamic simulation engine of the Carla, which returns the next state of both the environment and the agent. Within this environment, for a given map, a desired path is defined using the *waypoints* (see Fig. 1). A waypoint is a point's Cartesian coordinates that Carla uses to facilitate the agents' navigation. Among the waypoints that define a path, a waypoint is sequentially chosen to be the objective that the vehicle must reach. Once it is reached, then a next waypoint is recovered (from the path sequence) to make it be the next objective for the

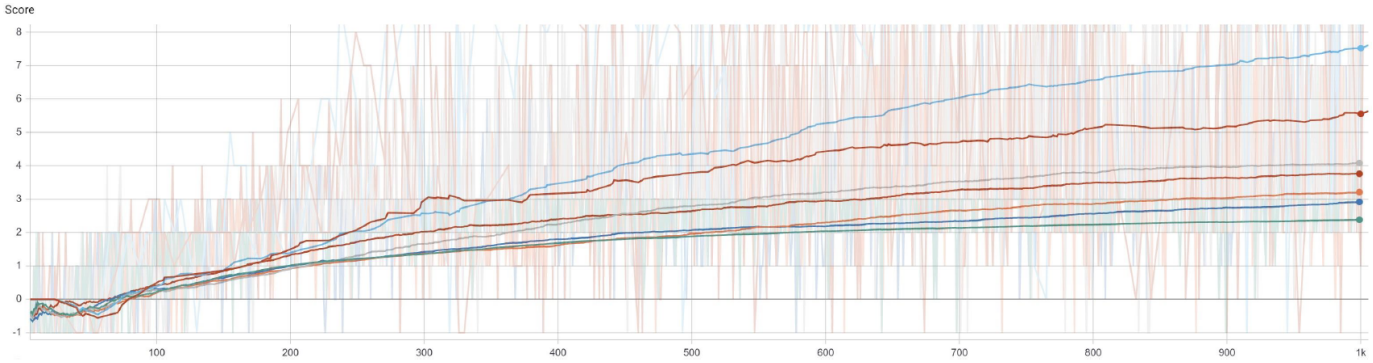


Fig. 3. Algorithm comparison results: DQN (green), DDQN (grey), Dueling DDQN (orange), Noisy Network (blue), PER (purple), Ape-X with one actor (cyan), and Ape-X with two actors (bordeaux). The colors are ordered from low to high performance results at the end of 1000 episodes.

vehicle and so on. In our case, the distance between two consecutive waypoints is 20 m. The reward function is defined by considering the following three aspects: if the agent collides with any object in the environment, a negative reward (i.e., -1) is assigned; if the agent reaches a given waypoint, the agent gains a positive reward (i.e., 1); and, if the agent reaches the goal position, it gains a positive reward (i.e., 1). As far as the discount factor, we have chosen it heuristically. Finally, the stop condition for a training run (i.e., an episode) is satisfied if at least one of the following events are triggered: the agent collides with an object; the agent does not reach a waypoint within a time threshold (i.e., 8 seconds); or the agent reaches the goal position.

Fig. 2(a) shows the Distributed Prioritized Experience Replay (Ape-X) that we used in the core of our end-to-end autonomous driving system. The images that come from an RGB camera are converted into grayscaled ones and then processed by a CNN. The output of the CNN is concatenated with the vehicle state (i.e., position, linear and angular speeds) and the desired waypoint location. The result of this concatenation is fed to a dueling noisy network, which in turn returns estimated state-values and advantage values. Then, by aggregating these values, action-values can be estimated, which in turn are used to choose the desired action.

Fig. 2(b) shows a more detailed view of Ape-X. Ape-X uses a Prioritized Experience Replay [19]. The learner retrieves transition samples from this memory and updates the parameters of the learner's neural network. On the other hand, the parameters of actor networks are periodically updated with the last parameters of the learner network. The server periodically receives the parameters optimized by the learner to stock them. Each actor network periodically sends the accumulated transitions to the Prioritized Experience Replay. Ape-X grants to increase the number of actors that interact with the environment using a parallel-programming scheme. In this way, the number of collected data can be drastically increased and reduce the overall training time. In this work, we considered both one-actor and two-actor cases and compared their respective performance results, which are shown in Section IV.

Hyperparameter	Symbol	Value
Discount factor	γ	0.99
Initial exploration rate	ϵ	0.9
Optimizer		Adam
Replay memory size		80 000
Minimum ϵ value	ϵ_{min}	0.01

TABLE I
HYPERPARAMETER VALUES

As far as the hyperparameters, the following ones are employed in our architecture: γ (the discount factor), ϵ (epsilon-greedy), ϵ-decay (the rate (between 0 and 1) at which the ϵ is reduced as function of the number of episodes), ϵ_{min} (the minimum value that ϵ can reach), the size of the replay memory, the size of mini-batches, the update rate for the target-networks' parameters, the minimum replay memory size before the training procedure is launched, the learning rate, the optimizer type, the number of layers in neural networks, the number of neurons per layer in the neural networks. The hyperparameter values employed in this work are heuristically chosen and some of their values are shown in Table I.

IV. RESULTS

The end-to-end autonomous driving system described in Section III is trained over 1000 episodes using the Lambda Quad Max Deep Learning server (with 128GB RAM, 12-core Intel i9-10920X CPU (2 threads per each core), 2 disks of 4 TBytes, 4 NVIDIA TITAN V GPUs) with Ubuntu 18.04 as its operating system. We wrote our code in Python 3.7 using PyTorch [24] as the deep-learning library. The backpropagation to optimize the neural networks' parameters is executed using the *autograd10* library that is available in PyTorch. The visualization of the learning results is done using Tensorboard. The Python's multiprocessing module allowed us to run our code in parallel. Ape-X is a distributed algorithm with various actor networks, and the parallel execution mechanism is crucial for the good performance of the algorithm.

In the sequel, we show two types of results: 1) the comparison of the performance results obtained by following the

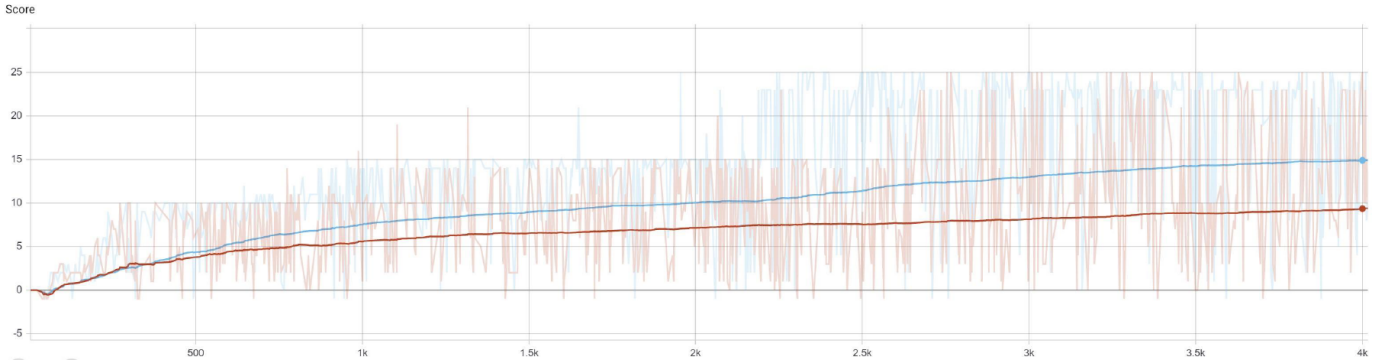


Fig. 4. Ape-X results: blue score curve ($\epsilon_{\text{min}} = 0.01$) and red score curve ($\epsilon_{\text{min}} = 0.05$) over Town02 map.

path indicated on the map shown in Fig. 1 using various DRL algorithms; 2) the influence of the exploration-curve decay within the Ape-X framework.

A. Comparison of the performance results obtained from various DRL algorithms

The DRL algorithms mentioned in Section II are employed to compare their performance to autonomously follow the path shown on the map in Fig. 1. These algorithms are DQN, DDQN, Dueling DDQN, Noisy Network, Prioritized Experience Replay (PER), and Ape-X. The same number of episodes are used to train the respective model parameters (i.e., 1000 episodes). Fig. 3 shows these comparison results. The seven curves shown in this figure represents the averaged score curves for seven different cases: DQN (green), DDQN (grey), Dueling DDQN (orange), Noisy Network (blue), PER (purple), Ape-X with one actor (cyan), and Ape-X with two-actors (bordeaux). The shape of the averaged score curves are similar across all the DRL algorithms. The Dueling DDQN and Noisy Network respectively have about 2 and 4 times more parameters than the DDQN model. Therefore, these two former models need more time to optimize their parameters. However, their performance results are among the worst algorithms. The most performing algorithm was the Ape-X regardless of the number of actors considered in the experiment. We found that, for the problem that we considered, the best performing algorithm is the Ape-X with two actors.

B. Influence of the exploration-curve decay within the Ape-X framework

Fig. 4 shows the performance results obtained with the Ape-X algorithm with two ϵ_{min} values: 0.01 and 0.05. For each of these two cases, we performed 4000 episodes to train the Ape-X algorithm with about 2.5 million parameters. Using the hardware and software specifications given earlier, the training procedure for each case took in average about 46 hours. The two score curves shown in Fig. 4 reveal us that ϵ_{min}=0.01 gives better results than ϵ_{min}=0.05.

C. Testing the optimized end-to-end autonomous driving system

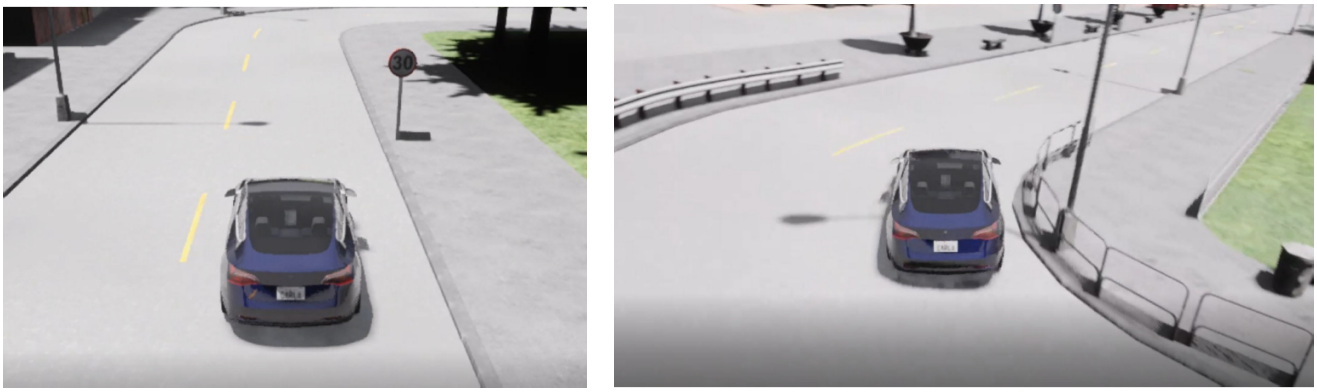
Fig. 5 shows two images that illustrate the outcomes of learning to drive autonomously using the Ape-X approach in Carla environment as described in Section III. We observed that the optimized Ape-X architecture allowed the vehicle to follow the path indicated on the map shown in Fig. 1. However, we also observed that the vehicle zigzagged around the road centerline, although it does not go laterally beyond the road boundaries. This is something that we still need to improve and may be achieved by modifying the reward function so that it would take into account the lateral distance between the vehicle position and the road (or the lane) boundaries.

V. Conclusions

In this work, we have shown an end-to-end system able to learn to autonomously follow a path using the Ape-X algorithm in Carla simulation environment. We evaluate its performance by comparing its results to those that are obtained using other deep reinforcement learning algorithms such as DQN, DDQN, Dueling DDQN, Noisy Network, and Prioritized Experience Replay (PER). We have shown that Ape-X with two actors returns the best performance results among all the DRL algorithms considered and for the map employed in this study. In addition, we also observed that slowing down the exploration decay (rather than a fast exploration decay) within the Ape-X framework helps obtain better performance results. The resulting behavior of the vehicle still needs to be improved as it zigzags about the centerline of the road. In the future, we will improve the definition of the reward function with the purpose to obtain more stable solutions.

Acknowledgment

At the time of the realization of this work, Maxence Hussonnois was a research intern at ECE Paris, France. The authors appreciate the ECE Paris for financing the purchase of the Lambda Quad Max Deep Learning server to obtain the results illustrated in the present work.



(a) Going straight in Carla environment

(b) Turning to the right in Carla environment

Fig. 5. Two footages that illustrate the outcomes of learning to drive autonomously using the Ape-X approach described in Section III.

REFERENCES

[1] M. Toromanoff, E. Wirbel, and F. Moutarde, "End-to-end model-free reinforcement learning for urban driving using implicit affordances," *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7153–7162, 2020.

[2] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel, H. Van Hasselt, and D. Silver, "Distributed prioritized experience replay," *International Conference on Learning Representations (ICLR)*, 2018.

[3] A. Dosovitskiy, G. Ros, F. Codevilla, A. López, and V. Koltun, "CARLA: An open urban driving simulator," *Proceedings of the 1st Annual Conference on Robot Learning*, pp. 1–16, 2017.

[4] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[6] D. C. Halbert, "Programming by example," Ph.D. dissertation, University of California, Berkeley, 1984.

[7] F. Codevilla, M. Müller, A. López, V. Koltun, and A. Dosovitskiy, "End-to-end driving via conditional imitation learning," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4693–4700, 2018.

[8] M. Jaritz, R. De Charette, M. Toromanoff, E. Perot, and F. Nashashibi, "End-to-end race driving with deep reinforcement learning," *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2070–2075, 2018.

[9] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning (ICML)*. PMLR, 2016, pp. 1928–1937.

[10] H. Hu, Z. Lu, Q. Wang, and C. Zheng, "End-to-End automated lane-change maneuvering considering driving style using a deep deterministic policy gradient algorithm," *Sensors*, vol. 20, no. 18, p. 5443, 2020.

[11] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *International Conference on Machine Learning (ICML)*. PMLR, 2014, pp. 387–395.

[12] B. T. T. C. Grigorescu, Sorin and G. Macesanu, "A survey of deep learning techniques for autonomous driving," *Journal of Field Robotics*, vol. 37, no. 3, pp. 362–386, 2020.

[13] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[15] J. O'Neill, B. Pleydell-Bouverie, D. Dupret, and J. Csicsvari, "Play it again: reactivation of waking experience and memory," *Trends in Neurosciences*, vol. 33, no. 5, pp. 220–229, 2010.

[16] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," *In Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[17] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International Conference on Machine Learning (ICML)*. PMLR, 2016, pp. 1995–2003.

[18] M. Fortunato, M. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg, "Noisy networks for exploration," *International Conference on Learning Representations (ICLR)*, 2017.

[19] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *International Conference on Learning Representations (ICLR)*, 2016.

[20] A. Nair, P. Srinivasan, S. Blackwell, C. Alcicek, R. Fearon, A. De Maria, V. Panneershelvam, M. Suleyman, C. Beattie, S. Petersen, S. Legg, V. Mnih, K. Kavukcuoglu, and D. Silver, "Massively parallel methods for deep reinforcement learning," *arXiv preprint arXiv:1507.04296*, 2015.

[21] P. Kaur, S. Taghavi, Z. Tian, and W. Shi, "A survey on simulators for testing self-driving cars," in *Fourth International Conference on Connected and Autonomous Driving (MetroCAD)*. IEEE, 2021, pp. 62–70.

[22] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3. IEEE, 2004, pp. 2149–2154.

[23] G. Rong, B. H. Shin, H. Tabatabaee, Q. Lu, S. Lemke, M. Možeiko, E. Boise, G. Uhm, M. Gerow, S. Mehta *et al.*, "LGSVL simulator: A high fidelity simulator for autonomous driving," in *IEEE 23rd International Conference on Intelligent Transportation Systems (ITSC)*. IEEE, 2020, pp. 1–6.

[24] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf