

A Method of Deep Reinforcement Learning for Simulation of Autonomous Vehicle Control

Anh T. Huynh^{1,4}, Ba-Tung Nguyen^{1,4}, Hoai-Thu Nguyen^{1,4}, Sang Vu^{2,4} and Hien D. Nguyen^{3,4} ^{a,*}

¹*Faculty of Software Engineering, University of Information Technology, Ho Chi Minh City, Vietnam*

²*Faculty of Information Systems, University of Information Technology, Ho Chi Minh City, Vietnam*

³*Faculty of Computer Science, University of Information Technology, Ho Chi Minh City, Vietnam*

⁴*Vietnam National University, Ho Chi Minh City, Vietnam*

Keywords: Autonomous Vehicles, Reinforcement Learning, Policy Gradient, Simulator, Software Engineering.

Abstract: Nowadays autonomous driving is expected to revolutionize the transportation sector. Carmakers, researchers, and administrators have been working on this field for years and significant progress has been made. However, the doubts and challenges to overcome are still huge, regarding not only complex technologies but also human awareness, culture, current traffic infrastructure. In terms of technical perspective, the accurate detection of obstacles, avoiding adjacent obstacles, and automatic navigation through the environment are some of the difficult problems. In this paper, an approach for solving those problems is proposed by using of Policy Gradient to control a simulated car via reinforcement learning. The proposed method is worked effectively to train an agent to control the simulated car in Unity ML-agents Highway, which is a simulating environment. This environment is chosen from some criteria of an environment simulating autonomous vehicle. The testing of the proposed method got positive results. Beside the average speed was well, the agent successfully learned the turning operation, progressively gaining the ability to navigate larger sections of the simulated raceway without crashing.


1 INTRODUCTION

According the report of WHO (2020), the traffic death is one of top 10 causes of death in the world, and the first cause for young people. Clearly, other problems derived from transportation are in terms of Global Warming caused by the gas emissions of transportation. In some countries, it can reach up to 28% of the total emissions that cause the Greenhouse Gas (EPA, 2018). For those reasons, manufactures have been aware that their future is Autonomous Vehicle (AV) development. AV will reduce the number of traffic accidents, reduce traffic jams and hours wasted inside the car. It also will optimize the energy consumption reducing gas emissions, etc. Consequently, the AV will be a technological challenge. There are many technical problems in the development of AV, such as the accurate detection of obstacles, avoiding adjacent obstacles, and automatic navigation through the environment (Marina and Sandu, 2017).

In recent years, Reinforcement Learning (RL) is considered to be an interesting learning technique that requires only performance feedback from the environment (Sutton and Barto, 2015). It is usually used to solve learning problems. There are many fields to apply RL techniques. Agent57 is a deep RL agent can play 57 Atari games (Badia et al., 2020). The optimization of policies of a marketing campaign was determined by using RL algorithms (Perez et al., 2009, Lucarelli and Borrotti, 2020). Deep learning is used to detect diabetic retinopathy in healthcare (Nguyen et al., 2021) and sentiment analysis of sentences (Nguyen et al., 2020a). Is it possible to utilize the advantage and smart of RL to solve these traffic problems human-being are facing.

In this paper, an approach for solving some problems of AV is proposed by using of a RL algorithm, Policy Gradient, to control a simulated car via reinforcement learning. Besides, some criteria of an environment simulating autonomous vehicle have been studied based on simulator's abilities and

* Corresponding author.

 <https://orcid.org/0000-0002-8527-0602>

software evaluation. The proposed method works effectively to train an agent controlling the simulated car in Unity ML-agents Highway, which is a chosen simulating environment by using the simulator's criteria. The results of testing get positive results. Beside the average speed was well, the agent successfully learned the turning operation, progressively gaining the ability to navigate larger sections of the simulated raceway without crashing.

2 RELATED WORK

There are many studies for designing and training of autonomous driving. Using reinforcement learning is a useful approach to solve some problems of this work (Huang et al., 2017, Min et al., 2019). However, those results have not yet mentioned to how those methods work well based on determined criteria.

Huang et al. (2017) proposed longitudinal control of autonomous land vehicles using parametric reinforcement learning. This approach used the parameterized batch actor-critic algorithm to get optimal control policies which adaptively tune the fuel control signals for tracking speeds. Nevertheless, they did not give some simulating criteria to evaluate the effectiveness of autonomous driving.

The results in (Lin, 1992) proposed eight extensions of the reinforcement method, including *adaptive heuristic critic* (AHC), Q-learning and three other extensions for both methods to speed up learning. They mainly focus on Deep Q-Network (DQN) based on learning for agent training. Nonetheless, those results do not perform the application of RL in designing of autonomous vehicle.

Min et al. (2019) defined highways driving policy using the reinforcement learning method. They also proposed a supervisor agent using deep distributional reinforcement learning to enhance the driver assistant systems. The supervisor agent is trained using end-to-end approach that directly maps both a camera image and LIDAR data into action plan. However, they did not show the performance of that method on other simulator.

Reasoning methods are useful for designing of intelligent systems. Those were applied for detecting influencers on social networks (Huynh et al., 2019) and intelligent searching on the knowledge of courses (Pham et al., 2020). In AV control, the reasoning method based on traffic rules is used in the training of driver's behaviors for the device (Talamini et al., 2020).

There are two main algorithms used for reinforcement learning: Q-Learning and Policy Gradient (PG). Q-learning is an off-policy RL algorithm that seeks to find the best action to take given the current state (Hasselt et al., 2015, Watkins and Dayan, 1992). It is considered off-policy because the q-learning function learns from actions that are outside the current policy, like taking random actions, and therefore a policy isn't needed. More specifically, Q-learning seeks to learn a policy that maximizes the total reward. Policy Gradient (PG) is one of the most important techniques in RL (Silver et al., 2014). In this algorithm, the agent through a policy takes some actions within the environment, then it receives from the environment, the reward and the observations of the state. The goal of PG is to find a policy which given some states (inputs) and over some actions (outputs) is able to maximize the expected sum of rewards (Peters and Schaal, 2008), so this method is useful to implement with the AV simulator.

3 DEEP INTO REINFORCEMENT LEARNING

3.1 Reinforcement Learning

Reinforcement learning (RL) is an approach of machine learning (ML). It is different from the other ML techniques due to its objective is to learn various behaviour based on the environment.

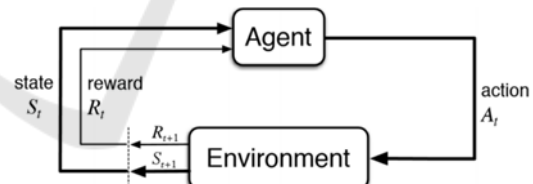


Figure 1: Reinforcement learning schema.

Figure 1 shows a basic schema of an RL setting. *Agent* is the main actor. It is the learning system responsible for observing the environment, choosing and performing Actions. In the agent, Policy is the strategy responsible for choosing the best Actions to get the maximum Reward based on the current state. *Action* is a set of possible moves the Actor can perform in the environment. *Reward* is the reward or penalty (negative reward) gotten from the Environment after performing Actions. It is the feedback telling the success or the failure of the Agent's Action. *Environment* is the physical world where the Agent moves. *State (Observation)* is the

current and concrete situation of the Agent in the environment.

In summary, the Agent observes the Environment, selects and performs Actions, and gets Rewards. Then, in order to get the most Reward over time, the Policy learns by itself what is the best strategy by defining which are the Actions the Agent should choose when it is in a given State.

3.2 Policy Gradient

The Policy is a neural network (Figure 2) which processes the state information through some layers of neurons and ends up with a distribution over all possible actions that you might want to take (Sehnke et al., 2010) Then, from this distribution, it is sampled an action which is the action that would be taken by the agent. Finally, new rewards and states are gotten. This process is repeated until the end with the episode.

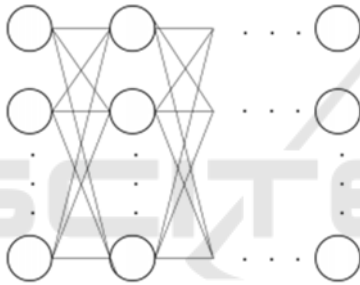


Figure 2: Policy neural network.

Policy Gradient optimizes policy directly. Policy is usually a parameterized function respect to θ , denoted $\pi_\theta(a|s)$. PG optimizes value of θ so that the objective function based on reaches maximum value. The objective function is defined as:

$$J(\theta) = E_\tau[R(\tau)] \tag{1}$$

where, τ is a sequence of states and actions.

$$\tau \equiv (s_0, a_0, s_1, a_1, \dots, s_T, a_T) \tag{2}$$

PG estimates the gradient of the objective function using the following formula:

$$\hat{g} = \nabla_\theta E_\tau[R(\tau)] = \sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'} \tag{3}$$

where, γ is a discount factor, to reduce variance when τ is long, and T is high.

Using baseline to further reduce the variance:

$$\hat{g} = \nabla_\theta E_\tau[R(\tau)] =$$

$$\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(a_t|s_t) \left(\sum_{t'=t}^{T-1} r_{t'} \gamma^{t'-t} - b(s_t) \right) \tag{4}$$

There are different types of baselines. We use the estimate of the discounted sum of rewards here:

$$b(s) \approx v_{\pi, \gamma}(s) = E \left[\sum_{t'=t}^T \gamma^{t'-t} r_{t'} \mid s_t = s \right] \tag{5}$$

It will increase the probability of paths that are better in average and decrease the probability of those that are worst on average. Table 1 shows the PG algorithm.

Table 1: Policy Gradient algorithm.

<p>Initialize policy parameter θ, baseline b</p> <p>For iteration = 1, 2, . . . n do</p> <p style="padding-left: 20px;">Collect a set of trajectories by executing the current policy.</p> <p style="padding-left: 20px;">At each step in each trajectory, compute:</p> <ul style="list-style-type: none"> • The return $R_t = \sum_{t'=t}^{T-1} \gamma^{t'-t} r_{t'}$, and • The advantage estimate $\hat{A}_t = R_t - b(s_t)$ <p style="padding-left: 20px;">Re-fit the baseline, by minimizing $(R_t - b(s_t))^2$, summed over all trajectories and timesteps.</p> <p style="padding-left: 20px;">Update the policy, using a policy gradient estimate \hat{g}, which is a sum of terms $\nabla_\theta \log \pi_\theta(a_t s_t) \hat{A}_t$</p> <p>end for</p>
--

The reward function is defined as sum of 5 following rewards:

- Longitudinal reward: $((\text{vehicle_speed} - \text{vehicle_speed_min}) / (\text{vehicle_speed_max} - \text{vehicle_speed_min}))$.
0: Minimum speed, 1: Maximum speed. Since we expect the agent to maximize the velocity of the controlled vehicles.
- Lateral reward: - 0.5. During the lane change it continuously get lateral reward. Since we expect the agent to minimize the number of lane changes.
- Overtake reward: $0.5 * (\text{num_overtake} - \text{num_overtake_old})$. Since we expect the agent to overtake more other vehicles as it can.
- Violation reward: -0.1. Example: If vehicle do left lane change at left warning, it gets violation reward (Front and right warning also)
- Collision reward: -10. If collision happens, it gets collision reward.

To run an autonomous driving system, Agents can take the following 5 actions: Do nothing, Acceleration, Deceleration, Lane change to left lane, and Lane change to right lane.

The Agents will receive the environment information through LIDAR sensor, which provides the distance of 360 degrees of the vehicle's surroundings. LIDAR is positioned looking forwards. Therefore, the 0 degree corresponds to the distance between the vehicle and an obstacle in front. Figure 3 shows the vector position of the LIDAR degrees.

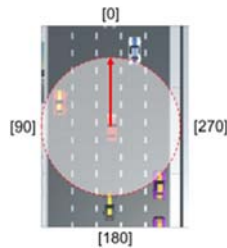


Figure 3: LIDAR's vector's position.

A multilayer perceptron with 1 to 2 hidden layers will be used as policy. The input for it will be 16 states of the LIDAR's range. Figure 4 summaries the process of this simulation.

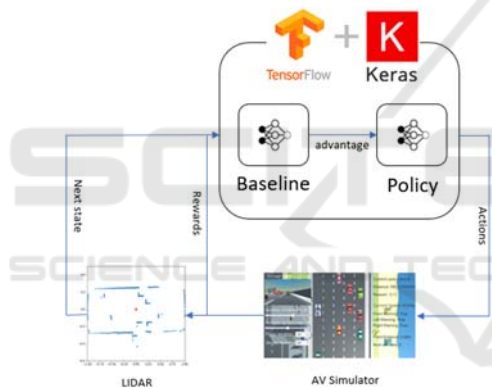


Figure 4: The process of the simulated AV control.

4 SOME CRITERIA OF SIMULATORS FOR AUTONOMOUS VEHICLES

The choice of an adequate environment for simulation is a key step. The simulator will determine the measure got from the model, and therefore it can performance results of an inputted problem. Currently, there exist many of simulators in almost all the fields of robotics, due to its importance in the development of control algorithms. To choose an appropriate simulator, it has to satisfy some specific characteristics which may impact on the development of the simulation:

- **The Abilities of Simulator:**

- *Scenarios:* the scenarios can be controlled by the simulator, such as urban road, highway, city, with other vehicles, pedestrians, etc.
- *Sensing Measurements:* The performance of data from the vehicle or the environment the simulator, such as LIDAR, Radar, Odometer, Cameras, other sensors, position, velocity, etc.
- *Functions:* the status of vehicle which can be controlled by the simulator, such a reset environment, accelerate time, control number of steps, etc.
- *ML Integration:* The ML techniques were integrated in the simulator and how to it support those ML's algorithms/libraries.

- **Criteria for Software Evaluation:**

- *Understandability:* this is one of the most important characteristics of software quality because it can influence the cost or reliability of software evolution in reuse or maintenance.
- *Usability:* this criterion shows the ability to apply for using in the practice.
- *Installation:* the requirements of software and hardware for the simulator, and how straightforward is the installation in a supported system.
- *Portability:* This is the level of difficulty to work with the same project with different computers.

Those criteria help to choose an appropriate simulator to measure models of autonomous vehicles. They can get an adequate simulation environment to experiment adequately, meet the needs of testing and reduce the cost of testing. Based on those criteria, there are some simulators to control an autonomous vehicle on a road as follows:

AirSim Simulator (2021), which is a simulator by Microsoft, comes with a detailed 3D urban environment that includes a variety of diverse conditions, including traffic lights, parks, lakes, and construction sites. It also contains an open world, realistic environments, multi vehicles, etc.

Apollo Simulator (2021) is a product of Baidu. It allows users to input different road types, obstacles, driving plans, and traffic light states. Developers can create realistic scenarios that support the verification of multiple modules such as perception, and planning, plus traffic flows that provide rigorous testing for algorithms.

Carla (2021) has been built in the collaboration of Intel Labs, Toyota Research Institute and CVC Barcelona, for flexibility and realism in the rendering and physics simulation. The environment is composed of 3D models of static objects such as

Table 2: Comparison between simulators.

Simulator		AirSim	Apollo	Carla	Metacar	Unity ML-Agents
Criteria						
Simulator ability	Scenarios	Complex 3D urban environment	Complex 3D environment and deep custom ability	3D models of static objects in high level of complexity	2D urban road with ability to custom	2D urban road with 5 lanes in the same direction
	Sensor	Pose and images. IMU, LIDAR	Laser Point and Image-Based Obstacle Detection	RGB cameras and pseudo-sensors	Front LIDAR	Camera and 360 degrees LIDAR
	Function	Basic level	High level	High level	Basic level (unable to accelerate)	Basic level (unable to accelerate)
	ML Integration	Support	Support	Support	Support	Support
Software Evaluation	Understandability	Complex	Complex	Complex	Easy	Medium
	Usability	High	High	High	Medium	High
	Installation	High computer capacities required	High computer capacities required	High computer capacities required	No installation required	Medium computer capacities required
	Portability	Unable	Unable	Unable	Able	Unable

buildings, vegetation, traffic signs, and infrastructure, as well as dynamic objects such as vehicles and pedestrians. Implemented a basic controller that governs non-player vehicle behaviour.

Metacar Simulator (2021) is a RL environment for self-driving cars in the browser created by Thibault Neveu. The environment has an urban road with two paths with two directions. The library lets you create your own levels and personalize the environment to create your desired scenario.

Unity ML-Agents Highway Simulator (2021) is proposed in (Min et al., 2019). It was built based on Unity ML-Agents in order to test RL algorithms of AV navigation on a highway. The main idea is to reproduce the behaviour of a crowded highway. The environment has a straight highway with five lanes of the same direction.

The results of Table 2 show that AirSim, Apollo and Carla are all powerful simulators. However, the understanding of their implementation are complex. In contrast, Unity ML-Agents Highway Simulator has some advantages:

- *Implementation*: It uses Unity engine, so it can run in many kinds of operator systems.
- *Code language*: This simulator can use Python code, which is a very commonly programming language, as an environment programming.
- *ML oriented*: The implementation using Unity ML-Agent makes it more convenient to implement RL algorithms.

Those advantages are reasons for selecting Unity ML-Agents Highway as an environment simulator to test and experiment the controlling simulated cars via reinforcement learning based on Policy Gradient.

5 EXPERIMENTAL RESULTS

This section presents some experimental results when implementing proposed method was focused on using LIDAR data as the agents input. The implementation uses 06 (six) different configurations for learning rate, batch size, and neural network structure. Each configuration has been trained 3000 episodes ~ 10M steps. Detail information of each configuration are shown as Table 3:

Table 3: Detail information of configurations.

No	Hidden layer 1	Hidden layer 2	Learning rate	Batch size
1	64	None	0.01	10
2	32	32	0.01	10
3	64	None	0.001	1
4	64	None	0.01	1
5	128	None	0.01	10
6	88	None	0.01	10

The first configuration is the very commonly used parameters in PG. The configuration 2 aimed to

evaluate how a “deeper” network affects to performance, so it is added more hidden layer. The configurations 3 and 4 are modified the value of learning rate and batch size to clarify how these factors affect the result. In configuration 5, the Agents were provided with the latest 4 states. With more information from the past, we want to know how the model gets improves. Also, the Agents in the configuration 6 were received additional information from the environment, such as host vehicle speed, front and side warning, distance to front vehicle, speed of the front vehicle.

After 3000 episodes ~ 10M steps training for each configuration, the results are very clear as follows:

Table 4: Results of experiments.

No.	Speed	Mean reward	Lane changes
1	70.94346	0.76279	23.432
2	70.92849	0.74529	17.61
3	70.49962	0.73969	23.494
4	70.67611	0.73749	17.48
5	71.19511	0.75842	16.716
6	71.318	0.77	15.512

As the results in Table 4, the configuration 1 made a pretty good result. In addition, the result of Deep Q-Network using the LIDAR data of the simulator creator reaches the average speed of 71,3758 km/h. However, there are two things to be aware of. First, the result that the creator took is an average of 100K steps ~ 33 episodes. The model, which has trained 3000 episodes, was tested to run 100 episodes twice. The difference of these two results is approximately 1-2 km/h, so 100 or less is not enough to determine quality of the model. Besides, while the configuration 1 only uses LIDAR data, the creator obtains 71,3758 km/h by using not only this data, but also some other information, such as host vehicle speed, front and side warning, distance to front vehicle, speed of front vehicle. Another noteworthy point is that the configuration 1 learned to use action “do nothing” to avoid violation reward when it detects vehicles around.

There is nothing remarkable in the result of the configuration 2. It somewhat learned how to restrict lane change, but did not learn how to avoid violation reward, so it lost a lot of reward compared to configuration 1.

With a small learning rate in the configuration 3, the model quality changes a little, and is more stable compared to configurations 1 and 2. However, the weakness is that its average speed of 70 km/h in episode 2131, compared to configuration 1 in episode 1144 and configuration 2 in episode 978. That means the configuration 3 requires more sample data. It is obviously that with a high learning rate, the algorithm

becomes unstable, the model quality is constantly changing during the training process.

The result of the configuration 5 exceeds the 71 km/h, with a little lane change. Also, that configuration is much more stable than configurations 1 and 2 despite of having the same learning rate and batch size. Another noteworthy point is that the configuration 5 achieved an average speed of 70 km/h in episode 797, much faster than the rest of the configurations.

With additional important information, along with LIDAR data, the configuration 6 obtained the best result among the rest configurations.

Figures 4 and 5 are the summary of average speed and action distribution of each configuration displayed in charts, smoothed by taking an average of 20 episodes to get a more detailed view of the training process:

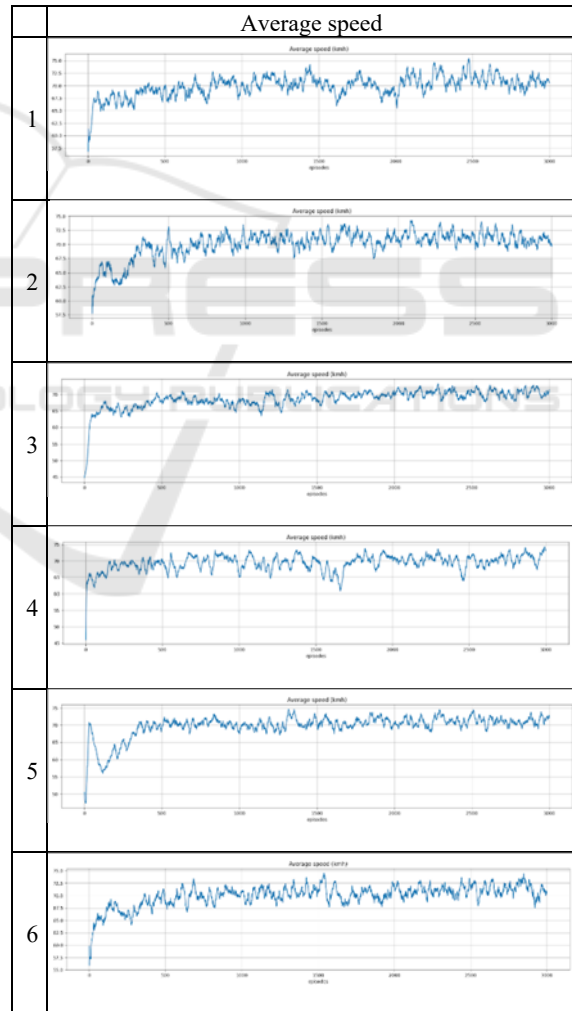


Figure 5: Charts of average speed of each configuration.



Figure 6: Charts of action distribution.

6 CONCLUSION AND FUTURE WORK

In this paper, a method to build an autonomous vehicle navigation system on highways by incorporating autonomous functions using reinforcement learning is proposed. The first step to implementing the RL algorithm is to find a suitable simulator for our project. The choosing is done based on some studied criteria for simulators. After comparison, Unity ML-Agents Highway Simulator is arguably the best for the project. At the next step, the Policy Gradient, which is a RL method, is utilized to implement an AV navigation system on Unity ML-Agents Highway Simulator. This implementation not only includes the basic Policy Gradient algorithm, but

also includes the DQN to overestimate action values under certain conditions.

After training agents with various configurations, it can be seen that the algorithm helps driving the vehicle in the simulator's scenario with an average speed of 71 km/h and have ability to change lane and avoid obstacle naturally and safety.

In the future, the improvement of result quality and reducing of training time would be studied. Some other types of neural network will be used to enhance the performance, such as Long Short Term Memory (Kouris et al., 2020). The proposed method is more studied to adapt for efficient deployment on other platforms. Besides, some methods for integrating of human knowledge into AV navigation system, such as inference rules (Talamini et al., 2020), knowledge of computing (Do and Nguyen, 2015) and ontology integration (Do et al., 2019), will be studied to apply them in real-world. Moreover, the policy gradient method is also able to apply to increase the ability of intelligent systems in e-learning for knowledge searching (Do et al., 2020, Nguyen et al., 2020b).

ACKNOWLEDGEMENTS

This research is funded by University of Information Technology – Vietnam National University HoChiMinh City, under grant number D1-2021-04.

REFERENCES

- AirSim, 2021. <https://microsoft.github.io/AirSim/> (Access on 08 March 2021)
- Apollo, 2021. <https://apollo.auto/platform/simulation.html> (Access on 08 March 2021)
- Badia, A., Piot, B., Kapturowski, S., Sprechmann, P., Vitvitskiy, A., Guo, Z., Blundell, C., 2020. Agent57: Outperforming the Atari Human Benchmark. In *ICML 2020, 37th International Conference on Machine Learning*, vol. 119, April 2020. 507-517.
- Carla, 2021. <https://carla.org/> (Access on 08 March 2021)
- Do, V.N., Nguyen, H.D. 2015. Reducing Model of COKB about Operators Knowledge and Solving Problems about Operators. In: Camacho D., Kim SW., Trawiński B. (eds), *New Trends in Computational Collective Intelligence*, Studies in Computational Intelligence, vol 572. Springer, Cham.
- Do, N.V., Nguyen, H.D., Mai, T. 2019. A Method of Ontology Integration for Designing Intelligent Problem Solvers. *Appl. Sci.* 9(18), 3793.
- Do, N., Nguyen, H., Hoang, L., 2020. Some Techniques for Intelligent Searching on Ontology-based Knowledge domain in E-learning. In *IC3K, 12th International Joint*

- Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management*, Vol. 2, Nov. 2020. SCITEPRESS. 313 – 320.
- Hasselt, H., Guez, A., Silver, D., 2015. Deep reinforcement learning with double Q-learning. In *AAAI 2015, 29th AAAI Conference on Artificial Intelligence*, Jan. 2015. 2094–2100.
- Huang, Z., Xu, X., He, H., Tan, J., Sun, Z., 2017. Parameterized batch reinforcement learning for longitudinal control of autonomous land vehicles. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 49(4), 730 – 741.
- Huynh, T., Zelinka, I., Pham, H., Nguyen, H.D. 2019. Some measures to Detect the Influencer on Social Network Based on Information Propagation. In *WIMS 2019, 9th International Conference on Web Intelligence, Mining and Semantics*, June 2019. ACM.
- Kouris, A., Venieris, S., Rizakis, M., Bouganis, C., 2020. Approximate LSTMs for Time-Constrained Inference: Enabling Fast Reaction in Self-Driving Cars. *IEEE Consumer Electronics Magazine* 9(4), 11 – 26.
- Lin, L.J., 1992. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning* 8(3 – 4), 293–321.
- Lucarelli, G., Borrotti, M., 2020. A deep Q-learning portfolio management framework for the cryptocurrency market. *Neural Comput & Applic* 32, 17229–17244.
- Marina, L., Sandu, A. 2017. Deep reinforcement learning for autonomous vehicles - State of the art, *Bulletin of the Transilvania University of Braşov* 10(59), 195 – 202.
- Metacar, 2021. <https://metacar.scottpletcher.guru/> (Access on 08 March 2021)
- Min, K., Kim, H., Huh, K., 2019. Deep distributional reinforcement learning based high level driving policy determination. *IEEE Transactions on Intelligent Vehicles* 4(3), 416 – 424.
- Nguyen, H., Huynh, T., Hoang, S., Pham, V., Zelinka, I., 2020a. Language-oriented Sentiment Analysis based on the grammar structure and improved Self-attention network. In *ENASE 2020, 15th International Conference on Evaluation of Novel Approaches to Software Engineering*, May 2020. SCITEPRESS. 339–346.
- Nguyen, H.D., Tran, D., Do, H., Pham, V., 2020b. Design an intelligent system to automatically tutor the method for solving problems. *International Journal of Integrated Engineering (IJIE)* 12(7), 211 – 223.
- Nguyen, H., Tran, V., Pham, V., Nguyen, H.D., 2021. Design a learning model of mobile vision to detect diabetic retinopathy based on the improvement of MobileNetV2. *Int. J. Digital Enterprise Technology (IJDET)*, in publishing.
- Perez, G., Guerrero, J., Olivás, E., Ballester, E., Palomares, A., Casariego, N. 2009. Assigning discounts in a marketing campaign by using reinforcement learning and neural networks. *Expert Systems with Applications* 36(4), 8022–8031.
- Peters, J., Schaal, S., 2008. Reinforcement learning of motor skills with policy gradients. *Neural Networks* 21(4), 682–697.
- Pham, X.T, Tran, T.V, Nguyen-Le, V.T, Pham, V.T., Nguyen, H.D. 2020. Build a search engine for the knowledge of the course about Introduction to Programming based on ontology Rela-model. In *KSE, 12th International Conference on Knowledge and Systems Engineering*, Nov. 2020. IEEE. 207 – 212.
- Sehnke, F., Osendorfer, C., Rückstieß, T., Graves, A., Peters, J., Schmidhuber, J., 2010. Parameter-exploring policy gradients. *Neural Networks* 23(2), 551 - 559.
- Silver, D., Lever, G., Heess, N., Degris, T., Wierstra, D., Riedmiller, M., 2014. Deterministic policy gradient algorithms. In *ICML 2014, 31st International Conference on Machine Learning*, vol. 32, June 2014. 387–395.
- Sutton, R., Barto, A. 2015. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, Massachusetts, USA, 2nd edition.
- Talamini, J., Bartoli, A., De Lorenzo, A., Medvet, E. 2020. On the Impact of the Rules on Autonomous Drive Learning. *Appl. Sci.* 10(7), 2394.
- United States Environmental Protection Agency (EPA), 2018. Sources of Greenhouse Gas Emissions <https://www.epa.gov/ghgemissions/sources-greenhouse-gas-emissions> (Access on 08 March 2021)
- Unity ML-Agents Highway, 2021. https://github.com/MLJeuCamp2017/DRL_based_SelfDrivingCarControl (Access on 08 March 2021).
- Watkins, C., Dayan, P., 1992. Q-learning. *Machine Learning* 8(3), 279–292.
- WHO. 2020. <https://www.who.int/news-room/fact-sheets/detail/the-top-10-causes-of-death> (Published on 09 Dec. 2020).