Learning Adaptive Parking Maneuvers for Self-Driving Cars

Gregory Gorbov[1], Mais Jamal[1], and Aleksandr I. Panov[2]

[1] Moscow Institute of Physics and Technology, Dolgoprudny, Russia
[2] Federal Research Center "Computer Science and Control" of the Russian Academy of Sciences, Moscow, Russia

Abstract. This paper addresses the autonomous parking for a vehicle in environments with static and dynamic obstacles. Although parking maneuvering has reached the level of fully automated valet parking, there are still many challenges to realize the parking motion planning in the presence of dynamic obstacles. One of the most famous autonomous driving platforms is the Baidu Apollo platform. In the Apollo platform, this problem is solved using the classic method hybrid A*. However, this method has two main downsides. Firstly, it generates in some parking scenarios, trajectories that consist of many partitions with different gear types and sizes. Such trajectories are intractable by a self-driving car when testing the Apollo planner on more realistic data coming from a simulator such as SVL. Secondly, the built-in algorithm does not have the ability to interact with dynamic obstacles, which might lead to a collision in some critical parking scenarios. To overcome these issues, we proposed a method based on reinforcement learning, which uses the RL-policy (from POLAMP) allowing us to take into account the kinematic constraints of the vehicle, static and dynamic obstacles. The proposed method was fully integrated into the Apollo platform with developed Cyber RT nodes, which were used for publishing the parking trajectory from our algorithm to the SVL simulator through a ROS/Cyber bridge. The final model demonstrates transferability to the previously unseen experimental environments and flexibility with respect to built-in hybrid A*.

Keywords: Autonomous Parking · Reinforcement Learning · Autonomous Vehicle · Motion Planning

1 Introduction

Over the last decade, many studies have involved algorithms dedicated to planning the motion of autonomous vehicles in a variety of tasks. Valet parking is one of the tasks that is usually described as a forward motion to determine and reach the proper orientation of the vehicle and then a backward motion to reach the parking spot. Baidu Apollo [1] is a well-known platform that supports developing and testing planning algorithms of self-driving cars. Before deploying an algorithm, it is frequently tested on a high fidelity simulator such as SVL [9].

The majority of trajectory tracking errors are caused by an intractable trajectory. One of the main issues with the hybrid A* algorithm is that it generates small pieces of trajectory with different types of gear, which introduces an accumulated trajectory error while planning and, as a result, the autonomous car may miss the parking spot. The second issue with the generated hybrid A* trajectory is that it does not take into account dynamic obstacles encountered while parking, which may result in a collision or an incomplete trajectory. This paper proposes a modern approach to resolving these issues.

Many recent planning algorithms with RL modules [4, 8, 12] demonstrate the appropriate behavior of an autonomous vehicle in dynamic environments. The POLAMP algorithm[2] was chosen after comparing the quality of recently developed algorithms. Because valet parking task is divided into two types—forward motion and backward motion, which is an efficient approach for learning POLAMP local planner—an algorithm for generating end positions for the first type of work is required. This issue is addressed by the POLAMP global planner. Following the implementation of the proposed algorithm, it was integrated into the Apollo platform, which communicates with SVL via nodes and topics. Appropriate Cyber nodes have been designed to publish the trajectory of POLAMP via support scripts and to read some necessary information from Apollo topics. We tested our algorithm on several developed maps with various parking elements such as a parking lot. Our results show that in situations with dynamic obstacles, our approach outperforms the built-in algorithm.

The remainder of this paper is divided into five sections. Section 2 discusses some recent related works. Section 3 presents the proposed methods for adaptive parking maneuver: RL local planner, vehicle model, curriculum learning, and Cyber nodes. Section 4 presents the evaluation after training the algorithm. Finally, Section 5 concludes with a discussion of future work.

2 Related Works

Recent methods, such as [6], are trajectory form-based methods that use a fixed shape of the trajectory to reduce the time of the valet parking task, but they still do not take into account the occurrence of dynamic obstacles, even though they are very effective in the absence of obstacles in terms of time reductions.

Another approach for parking planning is Hybrid A* and its variant algorithms [11]. Hybrid A* is the built-in planner in the Apollo platform. It is based on A* algorithm, where it searches for the goal state in a grid environment. The main difference between A* and hybrid A* is that the nodes, which are chosen from the open set at each iteration of the algorithm, have constraints of a simple bicycle model of kinematics. The movement is specified as a parameter, defining how far the car can travel from the current state to another state during the time $t = 0.1$s, with 10 points (the parameter can be changed in configurations). These 10 points are chosen for forward movement, and another 10 points are chosen for backward movement. In addition to this change, the built-in planner modifies the cost of moving to the next point by applying special penalties: a gear change penalty l_1; a penalty for trajectory curvature l_2; a penalty for changing the steer angle l_3; a penalty for moving forward l_4; and a penalty for reversing l_5.

Because the opening of nodes from an open set does not take into account the movement of other cars, neither hybrid A* nor A* can work with dynamic obstacles. However, dynamic obstacles are common in parking scenarios, which make RL-based algorithms preferred over A* algorithms.

In [3], it was proposed to solve the problem of navigation from point A to point B in the presence of dynamic obstacles using the reward function:

$$R_{\theta r} = \theta_r^T [r_{step} r_{goalDist} r_{collision} r_{turning} r_{clearance} r_{goal}],$$

which is in charge of an agent avoiding obstacles, penalizing it for a collision and rewarding it for getting closer to the goal. The use of an *actor* and a *critic* allows the agent to change the parameters of the planner strategy at each step, rather than only at the end of the episode, as is the case with gradient methods. The critic's task is to change the utility function Q of the state-action iteratively, penalizing the actor at each iteration.

The method described in [8] should be highlighted as one of the more advanced methods. The authors proposed to use reward function:

$$R = R_{goal} + R_{timestep} + R_{collision} + R_{potential} + R_{waypoint},$$

which is fundamentally different from the previous method. Furthermore, the authors put forth a motion planner that divides the global point trajectory from initial to terminal states into waypoints. The reward is set so that the agent balances between the following waypoints while also avoiding pedestrians; this is done so that the local scheduler has some allowable range for walking moves. It was demonstrated that with such a method configuration, trained on a simplified in a two-dimensional environment, the strategy is well transferred to a previously unobserved 3D realm.

It is also necessary to note a number of works in which reinforcement learning is used to build a maneuver trajectory[5, 7, 15] or to navigate a robot[13, 14]. However, these works do not use integration with well-known platforms for unmanned vehicles.

3 Method

In this work, we use the POLAMP algorithm [2] as a local planner that was trained to avoid dynamic obstacles. It is made up of the PPO [10] algorithm and a frame stack. The bicycle model was integrated into the POLAMP simulator, and acceleration constraints were used to make the autonomous car move more realistically and closer to Apollo car dynamic.

3.1 RL Local Planner

We consider partially observable Markov decision process (S, A, O, T, r, γ), where S is state space, A—actions space, O—observation space, $T : S \times A \times S \longrightarrow R$

is the state transition model, r—reward function, and $\gamma \in [0, 1)$—the discount factor. We assume that the state is not directly observable and learn a policy $\pi(a|o)$ conditioned on observations $o \in O$. The agent following the policy π. In addition to the decision process, we have a planning system $\Sigma = (S, A, T)$ and a planning task $P = (\Sigma, s_0, g)$ that process the sequence of actions $\{a_1, a_2, ..., a_n\}$. We propose that policy π produce the actions then $a_1 = \pi(o_0), ..., g = \pi(o_{n-1})$. Assuming that the policy is parameterized by θ, a policy gradient algorithm optimizes θ to maximize the expected future return and minimize the length of the trajectory:

$$\pi^* : \begin{cases} \pi^* = \underset{\pi(\theta)}{\operatorname{argmax}} \; E\left[\sum_{t=0}^{\infty} \gamma^t r(s_t)|\pi\right] \\ \pi^* = \underset{\pi(\theta)}{\operatorname{argmin}} \; \|\{\pi(o_0), \pi(o_1), ..., \pi(o_{n-1})\}\| \\ \text{where } g = \pi(o_{n-1}) \end{cases}$$

Because the environment is partially observed at each timestamp t, action a_t is generated from distribution $\pi^*(o_t)$ with respect to observation o_t. Observation $o_t = (\Delta x_g, \Delta y_g, \Delta\theta, \Delta v, \Delta\gamma, \theta, v, \gamma, flag, l_1, l_2, l_3, ..., l_n)$ where $\Delta\theta$ is the heading angle, Δv—difference between vehicle speed and speed constraint on parking place, and $\Delta\gamma$—difference between vehicle steer angle and steer angle constraint on parking place, $flag$ the information of the current task (forward task, reversing task) this information was added to simplify learning, $l_1, l_2, l_3, l_4, ..., l_n$ is lidar data information consisting of n beams and $\Delta x_g, \Delta y_g$ are showed in Fig. 1.

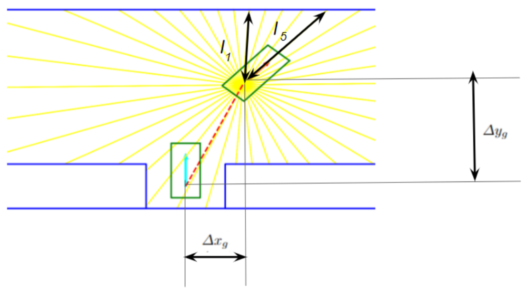


Fig. 1: Observation elements of RL local planner

We changed the agent action to (a, ϵ) in order for the RL algorithm to generate an appropriate speed and acceleration profile for the trajectory, where a is linear acceleration and ϵ is angular acceleration. The reward function has been modified to:

$$R = w_r^T[r_{collision}, r_{timestep}, r_{distance}, r_{overspeeding},$$
$$r_{oversteering}, r_{action_a}, r_{action_{Eps}}],$$

where $r_{collision}$ is -20 when the agent collides with obstacles and 0 otherwise, r_{step}—a constant penalty step with value 1, $r_{distance}$—Euclidian distance to the goal, $r_{overseeding}$, $r_{oversteering}$ equal to -5 when the agent tries to exceed the speed limit and $r_{action_a}, r_{action_\epsilon}$ are penalties for the absolute value of linear and angular acceleration, which equal $-0.5 * a_t - 0.5 * \epsilon_t$ at timestamp t.

We use the POLAMP local planner named PPO. It has an actor that is a neural network that takes an observation o_t and returns the expectation and dispersion of linear acceleration and angular velocity. During training, the actor updates policy π weights with respect to gradient:

$$\Delta_w J(w) = E_{\pi_w} \Delta_w log \pi_w(s,a) A^{\pi_w}(s,a)$$

The critic is used to predict the Advantage function $A^\pi(o_t, a_t)$, which states how preferable it is to choose action a_t with o_t. Also, the actor is updating its weights in relation to the loss function:

$$\begin{gathered} L(s,a,w_k,w) = min(\pi_w(a|s)/\pi_{wk}(a|s) * A^{\pi_{wold}}(s,a), \\ clip(\pi_w(a|s)/\pi_{wold}, 1-\epsilon, 1+\epsilon) A^{\pi_{wold}}(s,a)) \end{gathered}$$

The critic's weights are updated by estimating $A^\pi(s_t, a_t)$ from generated trajectories.

3.2 Vehicle Model

We changed the agent dynamic model to a bicycle model with a reference point in the center of the vehicle's rear axle.

$$\begin{aligned} \dot{x_r} &= v * \cos(\theta) \\ \dot{y_r} &= v * \sin(\theta) \\ \dot{\theta} &= \frac{V * tan(\delta)}{L} \end{aligned}$$

The kinematic constraints were modified as follows:

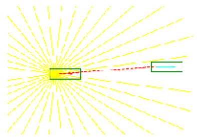
$$\begin{gathered} -2\ m/s <= v <= 2\ m/s \\ -1.5\ m/s^2 <= a <= 1.5\ m/s^2 \\ -1.5\ rad/s^2 <= \epsilon <= 1.5\ rad/s^2, \end{gathered}$$

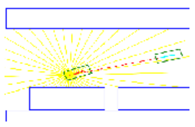
and the constraints for the parking place(terminal constraints) are:

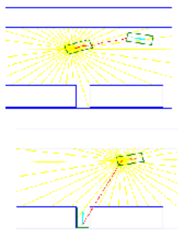
$$\begin{gathered} distance <= 0.5\ m \\ -0.5\ m/s <= v <= 0.5\ m/s \end{gathered}$$

3.3 Curriculum Learning

We employed a curriculum-based learning approach to educate agents how to park in critical situations with static and dynamic obstacles. For learning, four major stages were chosen (Fig. 2): learning in an empty environment with static obstacles, union learning when we assign the parking place as the goal after the agent has completed the first goal. There are also multiple consecutive learnings within each stage. For example, for the agent to learn to reach the goal with distance, heading angle, and speed limitations, the agent was first taught with distance constraints only; otherwise it is difficult for the agent to conclude the episode. The agent was learned in four stages, but the number of learning at each


(a) empty


(b) static


(c) union task

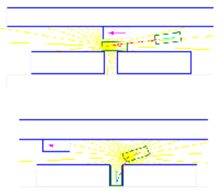

(d) dynamic

Fig. 2: Curriculum learning stages

stage may vary. For example, to assist the agent in reaching the parking spot with all constraints, the agent was first learned in an empty environment with only one condition $distance <= 0.5m$, then we added angle condition $angle <= 15°$ and speed condition $speed <= 0.5m/s$.

Fig. 3 depicts the architecture of the neural network used for training. We combine the last three observations o_{t-1}, o_{t-2}, and o_{t-3} with the current observation o_t. Because $n = 39$ lidar beams are used, the observation vector has a length of 48. When we combine the previous observations, we get a vector with the length $48 * 4 = 192$. A batch of 8000 is fed into neural network to train the actor. To train the critic, the action of the agent was added to the input vector so its size is: 194.

3.4 Cyber Nodes

The POLAMP planner and the Apollo simulator interact with developed Cyber topics, readers, and writers (Fig. 4). To begin with, information regarding static obstacles in the environment and task boundaries where the agent could go is required for the POLAMP simulator initialization. This information is gathered by the Apollo algorithm's ROI (Region Of Interest) decider. The dynamic obstacles' information is obtained in the ROI decider via the Apollo Perception module, which obtains publishes dynamic obstacles' positions and speed vectors every

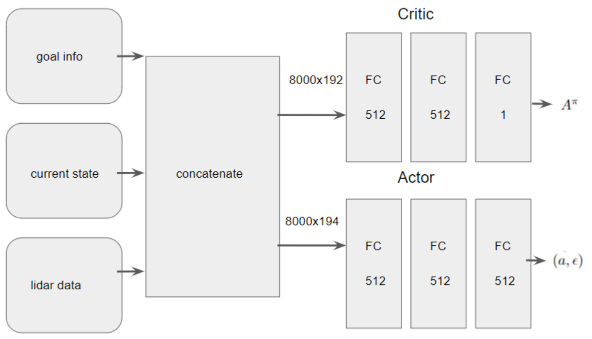


Fig. 3: Actor critic architecture

0.1 second. When information shows in the POLAMP planner, the POLAMP simulator is launched. Following that, the agent generates one episode under the premise of uniform rectilinear motion of the dynamic obstacles, saves trajectory points as tuples (x, y, heading angle, steering angle, linear acceleration, angular acceleration), and provides trajectory to Apollo via the created topic.

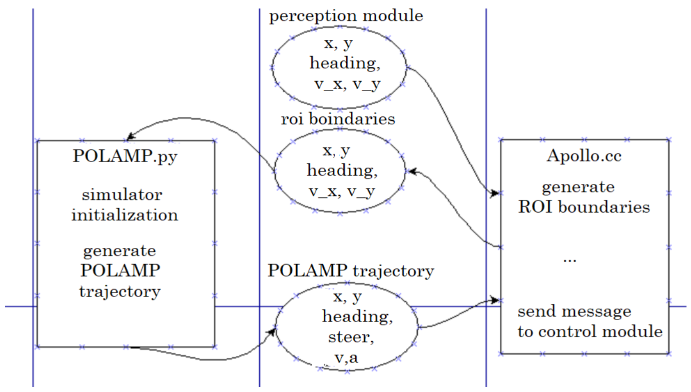


Fig. 4: Perception module, roi boundaries, POLAMP trajectory are cyber topics.

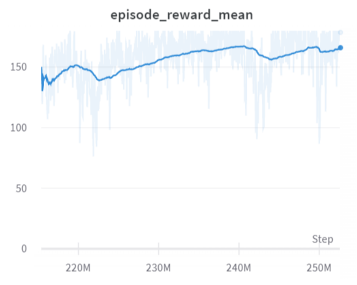
The speed and acceleration profiles of the trajectory were smoothed in Apollo because the speed at the end point of each POLAMP trajectory for forward and backward motion might range from -0.5 to $0.5m/s$. If the current trajectory's end point has index i and if the parking spot's end point has index j, then the new linear acceleration and speed for this point is calculated as follows:

$$a(i-1) = -v(i)/dt$$
$$v(i) = 0$$
$$a(i) = v(i+1)/dt$$

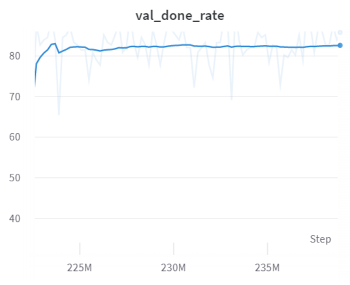
$$a(j-1) = -v(j-1)/dt$$
$$v(j) = 0$$
$$a(j) = 0$$

4 Evaluation

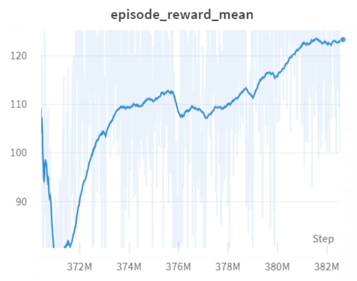
Fig. 5 shows the results that were obtained after training the algorithm in the POLAMP simulator. The main metrics are as follows: episode reward mean—the



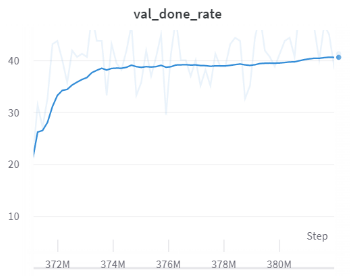
(a) static reward



(b) static val



(c) dynamic reward



(d) dynamic val

Fig. 5: POLAMP learning results

mean reward for the entire batch (includes 8,000 tasks) and val done rate—the number of tasks completed on the validation dataset (includes 600 tasks). When the roads are narrow, the main cases where the agent does not complete the episode are situations where the agent has to fix the direction of the movement vector to achieve the parking place with all constraints, which is uncritical for common parking scenarios.

5 Conclusion

We trained and tested the algorithm using the Apollo platform and the SVL simulator. If a collision with the agent is possible, the algorithm has successfully learned to avoid dynamic obstacles. If there are no obstacles, the algorithm finds the optimal trajectory, as illustrated in Fig. 6.

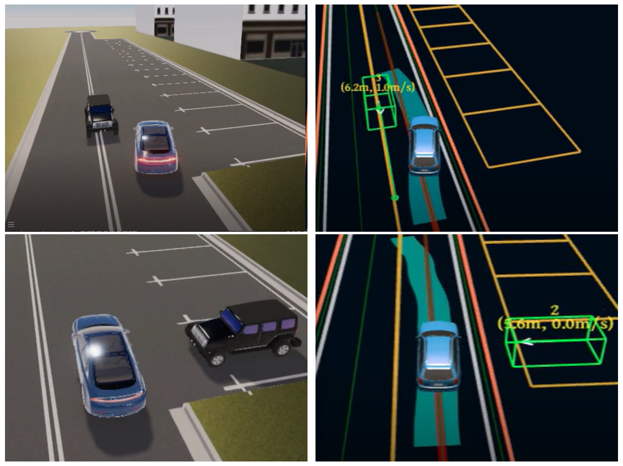


Fig. 6: Various parking scenarios with dynamic obstacles were tested in SVL simulator(the left figures) and in Apollo (the right figures).

Among the remaining uncovered cases of algorithm execution, an autonomous vehicle makes a trajectory when reversing with some error. This issue arises because the built-in module Control is unable to track the profile of speeds and accelerations provided by the POLAMP algorithm.

References

1. Baidu Apollo team (2017), Apollo: Open Source Autonomous Driving, howpublished = https://github.com/apolloauto/apollo, note = Accessed: 2019-02-11
2. Angulo, B., Yakovlev, K., Panov, A.: Policy optimization to learn adaptive motion primitives in path planning with dynamic obstacles (2022)
3. Chiang, H.T.L., Faust, A., Fiser, M., Francis, A.: Learning navigation behaviors end-to-end with autorl. IEEE Robotics and Automation Letters **4**(2), 2007–2014 (2019)
4. Chiang, H.T.L., Hsu, J., Fiser, M., Tapia, L., Faust, A.: Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies. IEEE Robotics and Automation Letters **4**(4), 4298–4305 (2019)
5. Jamal, M., Panov, A.: Adaptive Maneuver Planning for Autonomous Vehicles Using Behavior Tree on Apollo Platform. In: Bramer, M., Ellis, R.

(eds.) Artificial Intelligence XXXVIII. SGAI 2021. Lecture Notes in Computer Science, vol. 13101, pp. 327–340 (2021). https://doi.org/10.1007/978-3-030-91100-3_26, https://link.springer.com/10.1007/978-3-030-91100-3_26 https://www.scopus.com/record/display.uri?eid=2-s2.0-85121900442&origin=resultslist&sort=plf-f

6. Liyang, S., Yu, H., Xuezhi, C., Changhao, J., Miaohua, H.: Path planning based on clothoid for autonomous valet parking. In: 2020 17th International Computer Conference on Wavelet Active Media Technology and Information Processing (ICCWAMTIP). pp. 389–393. IEEE (2020)
7. Martinson, M., Skrynnik, A., Panov, A.I.: Navigating Autonomous Vehicle at the Road Intersection Simulator with Reinforcement Learning. In: Kuznetsov, S.O., Panov, A.I., Yakovlev, K.S. (eds.) Artificial Intelligence. RCAI 2020. Lecture Notes in Computer Science. vol. 12412, pp. 71–84. Springer International Publishing (2020). https://doi.org/10.1007/978-3-030-59535-7_6, https://link.springer.com/chapter/10.1007/978-3-030-59535-7_6 https://www.scopus.com/record/display.uri?eid=2-s2.0-85092200949&origin=resultslist
8. Pérez-D'Arpino, C., Liu, C., Goebel, P., Martín-Martín, R., Savarese, S.: Robot navigation in constrained pedestrian environments using reinforcement learning. In: 2021 IEEE International Conference on Robotics and Automation (ICRA). pp. 1140–1146. IEEE (2021)
9. Rong, G., Shin, B.H., Tabatabaee, H., Lu, Q., Lemke, S., Mozeiko, M., Boise, E., Uhm, G., Gerow, M., Mehta, S., Agafonov, E., Kim, T.H., Sterner, E., Ushiroda, K., Reyes, M., Zelenkovsky, D., Kim, S.: LGSVL simulator: A high fidelity simulator for autonomous driving. CoRR **abs/2005.03778** (2020), https://arxiv.org/abs/2005.03778
10. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017)
11. Sedighi, S., Nguyen, D.V., Kuhnert, K.D.: Guided hybrid a-star path planning algorithm for valet parking applications. In: 2019 5th international conference on control, automation and robotics (ICCAR). pp. 570–575. IEEE (2019)
12. Staroverov, A., Panov, A.I.: Hierarchical landmark policy optimization for visual indoor navigation. IEEE Access **10**, 70447–70455 (2022). https://doi.org/10.1109/ACCESS.2022.3182803
13. Staroverov, A., Yudin, D.A., Belkin, I., Adeshkin, V., Solomentsev, Y.K., Panov, A.I.: Real-Time Object Navigation with Deep Neural Networks and Hierarchical Reinforcement Learning. IEEE Access **8**, 195608–195621 (2020). https://doi.org/10.1109/ACCESS.2020.3034524, https://ieeexplore.ieee.org/document/9241850/
14. Staroverov, A., Vetlin, V., Makarenko, S., Naumov, A., Panov, A.I.: Learning embodied agents with policy gradients to navigate in realistic environments. In: Kryzhanovsky, B., Dunin-Barkowski, W., Redko, V., Tiumentsev, Y. (eds.) Advances in Neural Computation, Machine Learning, and Cognitive Research IV. NEUROINFORMATICS 2020. Studies in Computational Intelligence. vol. 925, pp. 212–221. Springer International Publishing (2021). https://doi.org/10.1007/978-3-030-60577-3_24, https://link.springer.com/chapter/10.1007/978-3-030-60577-3_24 https://www.scopus.com/record/display.uri?eid=2-s2.0-85093080089&origin=resultslist
15. Yudin, D.A., Skrynnik, A., Krishtopik, A., Belkin, I., Panov, A.I.: Object Detection with Deep Neural Networks for Reinforcement Learning in the Task of Autonomous

Vehicles Path Planning at the Intersection. Optical Memory and Neural Networks **28**(4), 283–295 (2019). https://doi.org/10.3103/S1060992X19040118, https://link.springer.com/article/10.3103%2FS1060992X19040118
https://www.scopus.com/record/display.uri?eid=2-s2.0-85079294637