2019 IEEE Intelligent Transportation Systems Conference (ITSC)
Auckland, NZ, October 27-30, 2019

Model-free Deep Reinforcement Learning for Urban Autonomous Driving

Jianyu Chen*, Bodi Yuan* and Masayoshi Tomizuka

978-1-5386-7024-8/19/$31.00 ©2019 IEEE

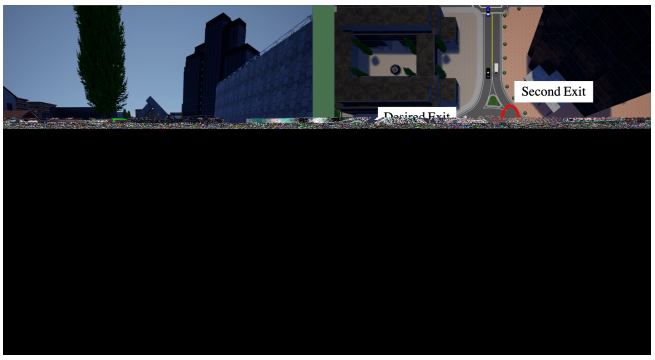

Fig. 4: The simulation environment and experiment scenario. Left is a sample view of the simulation. Right is a bird-view of our roundabout task scenario.

B. Network Architectures

We train our variational auto-encoder (VAE) on 50k bird-view images, which is generated by using a simple controller to drive around the roundabout. Note that we do not need any labels in the training dataset, such as the action commands or ground truth positions of vehicles, but only the raw bird-view images. We also do not require the controller to drive well, it does not matter if the vehicle drives out of lane or collides with other objects. The VAE model has 4 conv-layers of 3×3 kernel size, with 32, 64, 128, and 256 channels separately. The stride is set to 2. Each conv-layer is followed by ReLU activation. The latent space layer of size 64 is then fully connected to the last conv-layer. The VAE model is trained from scratch using Adam optimizer [15], with learning rate 10^{-4} for 100 epochs. Some examples of trained VAE results are given in Figure 3.

During the reinforcement learning process, all networks have the same conv-layers and first dense layer copied from the pretrained VAE model, denoted as visual encoding layers here. To keep the visual encoding stable, the weights of the visual encoding layers are fixed without updating. Now we introduce the remaining layers of the networks used in the three reinforcement learning algorithms as follows:

1) Double Deep Q-Network (DDQN): DDQN has one Q network and a target Q network with the same architecture. After the visual encoding layers, the network is then followed with 5 dense layers, with hidden layers ranging from 256 to 32 nodes. The output size is equal to the number of the possible actions, each representing the Q value of the corresponding action. The networks are trained using Adam optimizer with learning rate of 10^{-3}.

2) Twin Delayed Deep Deterministic Policy Gradient (TD3): TD3 has a policy network and two Q networks. After the visual encoding layers, all of these networks have 4 dense layers, with 64, 200 and 20 hidden units separately, followed by batch normalization and leaky ReLU for each layer. The negative slope of leaky ReLU is set to 0.01. The last dense layer of the policy network, corresponding to the action output, has 2 units, with a tanh activation function to limit the action range. The Q networks are a little different, their latent states from the visual encoding are first concatenated with the corresponding action, and the last layer has only one unit, denoting the Q value. Also, there is no batch normalization and no tanh activation used in Q networks. All the networks are trained using Adam optimizer. The policy network and Q networks are trained with learning rate 10^{-4} and 10^{-3} respectively.

3) Soft Actor Critic (SAC): The soft actor critic has two Q networks, one value network, and a policy network. The Q and value networks has the same architecture with one output unit. The visual encoding layers are followed by 5 dense layers with hidden units ranging from 256 to 32. The policy network's architecture is the same except that for the last layer, it splits to two branches. This is because the policy network of SAC is stochastic. The first branch represents the mean of the action and the second branch represents its variance. All networks are trained using Adam optimizer with learning rate of 3×10^{-4}.

C. Implementation Details

To improve the performance on the proposed task, we designed specific reward functions and added several tricks such as frame skip and new exploration strategies.

1) Rewards Design: After testing a variety of reward combinations, we came up with a five-term reward function which works well. The reward function is given below:

$$r = r_v + r_\alpha + r_c + r_o + c. \tag{16}$$

where r_v is the term encouraging moving forward to make progress. It is set equal to the ego vehicle's speed. Moreover, we set $r_v \leftarrow 10 - r_v$ if $r_v > 5$ to penalize exceeding speed limit. r_α is the term penalizing the magnitude of steering angle α to improve driving smoothness, where $r_\alpha = 0.5 * \alpha^2$. r_c is the term penalizing collision with other surrounding vehicles, where $r_c = -10$ if there is a collision, otherwise $r_c = 0$. r_o is the term penalizing running out of the lane, where $r_o = -1$ if the distance d between the ego vehicle the routing baseline is larger than $2m$, otherwise $r_o = 0$ (note that we also tried to set r_o continuously, such as linear to the distance d or d^2, but it turns out the discrete one works better). The last term c is a small constant set to -0.1, which was used to penalize the ego vehicle for stopping still.

2) Frame Skip: We use the frame skip trick during training, where we keep the action unchanged for k consecutive frames. In other words, during the training, each action made by the ego vehicle will last k frames until the new action starts to be effective. This technique heavily reduces the training complexity, one can regard it as reducing the search depth by a factor of k.

However, k shouldn't be set too large. If k is too large, the correct action space might be too small or even not existed. For example, when we set $k = 10$, it is difficult for the ego vehicle to make turns successfully. In our experiments, we use $k = 4$.

3) Exploration Strategies: We proposed different exploration strategies in our implementation. For DDQN, we use a different kind of epsilon greedy strategy. Generally, epsilon greedy chooses the random actions from uniform distribution. However, the uniform distribution contains no heuristic information, which is inefficient. We thus propose

dramatically. This is mainly because they are easy to suffer from insuf cient exploration. Thanks to a good balance of exploration and exploitation, SAC performs the best and can reach the goal point quite often.

When we look at the failure cases, we found that almost all the failures come from rear-end to the front vehicle. This is because we do not explicitly incorporate velocity of vehicles in our input, which is essential for keeping safe distance with the front vehicle. Instead, we draw historical vehicle states using boxes with fading color, which only implicitly contain the velocity information. This problem is even more serious when the input is passed through an encoding network. As in Fig.3, we can barely see the fading color of some vehicles in the reconstructed images.

TABLE I: Success rate for the roundabout scenarios evaluated on our three models DDQN, SAC and TD3. The value represent percentage of success trials.

Approach	DDQN	TD3	SAC
Entrance	80%	88%	86%
First exit	52%	74%	80%
Second exit	38%	2%	74%
Desired exit	8%	0%	64%
Goal point	0%	0%	58%

VII. CONCLUSIONS

In this paper, we proposed a framework to enable model-free deep reinforcement learning in challenging urban autonomous driving scenarios. We designed a bird-view input representation to reduce the sample complexity, and used visual encoding to capture the low-dimensional latent states. We then applied three state-of-the-art model free deep RL algorithms (DDQN, TD3, SAC) into our framework, with several tricks to improve the performance. We evaluate our method in a challenging roundabout scenario with dense surrounding vehicles in a high-de nition driving simulator. The results showed that that our method has the power to solve the tasks well. Although our method is signi cantly better than the baseline, it doesn't solve task perfectly. In the future, we will use more computation resources to dig the limit of our method, and also improve the learning ef ciency. Furthermore, We will explore the generalization ability to other scenarios.

REFERENCES

[1] K. Arulkumaran, A. Cully, and J. Togelius. Alphastar: An evolutionary computation perspective. arXiv preprint arXiv:1902.01724, 2019.

[2] M. Bansal, A. Krizhevsky, and A. Ogale. Chauffeurnet: Learning to drive by imitating the best and synthesizing the worst. arXiv preprint arXiv:1812.03079, 2018.

[3] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, et al. End to end learning for self-driving cars.