International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 19 (2018) pp. 14447-14450

Exploring Navigation using Deep Reinforcement Learning

Tegg Taekyong Sung[1], Changhyung Kim[1], Kyunghak Lee[2] and Chae-Bong Sohn[1]

[1]Department of Electronics and Communications Engineering, Kwangwoon University
[2]IACF, Kwangwoon University, 20, Gwangun-ro, Nowon-gu, Seoul, 01897, Republic of Korea.

Abstract

This paper discusses a navigation system with deep reinforcement learning approach. Reinforcement learning maximizes designed reward function and can be applied diverse domains, such as vision, language, or robotics. Especially, one of the methods, model-free learns how to maximize the objective without achieving any environment information as a trial-and-error. We review recent methodologies of navigation using reinforcement learning and discuss the impact of different observation spaces from the agent. Furthermore, we experiment the navigating robot using the model-free algorithm and a physical simulator.

Keywords: Deep learning, Reinforcement learning, Navigation, Planning

INTRODUCTION

Reinforcement learning (RL) aims to make an optimal decision by maximizing the designed objective. Thinking of navigation application, an automobile which corresponds to an agent is going to select an action within directional moves and speed, and the objective would safely arrive at the target destination. Previously used, the traditional dynamic programming (DP) is scheduled a mapping and a pathfinding separately. For instance, popularly known algorithm, SLAM performs to cartograph an environment and then DP planning is used for finding the path. However, this two-way takes much time for performing the separate stages. Instead, RL leverages deep neural networks to operate the mapping and pathfinding as end-to-end that the progress operates in a more structural way.

Conventionally, RL is developed from the Bellman equations in DP, and the arguments expressing by deep neural networks take discrete sequences. However, the agent settings are varied in observation spaces. Especially, for the continuous domains, the input sequences are needed to be discretized into countable values. To evaluate performance in different observation spaces, we experimented with the same environment and algorithm setting and verify their advantages and drawbacks.

Also, the method of RL can be divided as leveraging model dynamics into two categories: model-based and model-free. We review recent methods for both approaches and the navigation methods using Deep RL. We experiment a model-free algorithm to navigation using a physical simulator. The advantage of training the agent using simulator is that the trained model can be deployed to an actual robot [1]. Comparing to the traditional DP, Deep RL can be saved training time and avoid the risk of hardware damage.

BACKGROUNDS

A. Model-based reinforcement learning

The model-based method is to train an agent leveraging a dynamic model that represents an environment. The agent collects a large number of observed trajectories and estimates dynamics, rewards, and transition probabilities. We pretend the observed model is correct, and the RL algorithm is optimized using dynamic programming. Usually, the model-based methods require a large number of observations which causes data-inefficiency. The World Models [2], for instance, creates the dynamic model by variational auto-encoder [3], passes through Mixed Density Network with Recurrent Neural Networks (MDN-RNN) [4] to attain sequential information, and decides an optimal action using the RL or an evolutionary algorithm. This mechanism resembles the flow of vision, memory, and control from a human, and work efficiently by reducing the required parameters. Extending this work, [5] applies the World Models to a real application, successfully controlling an autonomous driving.

B. Model-free reinforcement learning

On the other hand, the model-free method does not require the observed models, and the agent does not estimate the reward and the dynamics. Instead, it directly learns value function or optimal policy with given observations. Such algorithms are Deep Q-Networks (DQN) [6], Deep Deterministic Policy Gradient (DDPG) [7], and Trust Region Policy Optimization (TRPO) [8]. A high-dimensional observation space can be efficiently approximated by deep neural networks and by approximating the objective functions, dynamic programming can be successfully applied to pixel-based environments in games or robotics. DQN is widely used as a baseline in the field of the Deep RL. Additionally, a replay buffer decouples the correlation among input experiences, and let the target and the estimate functions be separated to tackle a non-stationary problem.

NAVIGATION

A. Cognitive Mapper and Planner

The navigation is composed of mapping and planning. In a traditional method, the SLAM algorithm is used to build the environment information and pathfinding algorithm is applied to the map. On the other hand, Cognitive Mapper and Planner (CMP) joints the mapping and the planning and form a visual navigation [9]. First, the agent creates a confidence and a belief

map by considering uncertainty about the world. These maps are differentiable and sequentially updated by the latent representation extracted by auto-encoder. Then, the maps are multi-scaled and hierarchically stacked and based on the maps, the agent selects the optimal action using Value Iteration Networks [10]. The mapper works as an online manner without requiring a pre-constructed map and can be explored with statistical exploration, bringing robustness from errors.

B. Unsupervised Reinforcement and Auxiliary Learning

The objective of RL is to maximize cumulative rewards. With this extrinsic reward, UNsupervised REinforcement and Auxiliary Learning (UNREAL) incorporates unsupervised auxiliary information (*i.e.* changing the pixels in the input image or the activation of the hidden unit of agent's neural networks) which represent the pseudo-reward function [11]. The authors use an off-policy RL algorithm, A3C [12] combined with auxiliary pixel control, reward prediction, and replayed value losses to learn different pseudo-rewards simultaneously in parallel from a single experience. Controlling pixel changes let the agent better predict immediate pixel changes and reconstruct the input. That these additional tasks are provided, the agent can observe more information about the environment and rapidly adapt to the aspects of various tasks and achieves human-level performance in three-dimensional *Labyrinth* tasks. We now conjecture that receiving more information for the same experience allows easy learning of the policy.

EXPERIMENTAL RESULTS

In this section, we train a navigation robot using a physical simulator. Differing the traditional DP, the RL can experiment with the learning algorithm before testing an actual application. In this section, we introduce the simulated environment and verify different state observations with the experimental results.

A. Benchmark Environments

The deep RL could be drastically improved by the development of the physical environment. That researches are usually experimented in a simulation let us efficiently examine the interaction between an agent's action and an environment. Moreover, by manipulating the simulator, the training time can be decreased, and the risk of hardware damage can be avoided. Practically, the simulation-based trained model can be deployed to an actual application, working as the Sim2Real methodology [1]. In this paper, we select Gazebo [13] as a simulator. It provides a simple API for creating new robots, actuators, sensors, and arbitrary objects and interacts with clients. It uses OpenGL and OpenGL Utility Toolkit [14] for simulating dynamics and Open Dynamics Engine [15] for the physics engine. Also, the simulator provides environmental factors such as a gravity and a lighting which decrease the gap between the simulation and the real world.

B. Experimental Setup

We choose the training agent to Turtlebot3 [16]. The simulation sees in a top-down view and constructed in a square-shaped map. The randomly spawned red-color zone is the target area and a goal of the agent. We manually set the +200 reward for successfully arriving the destined area and -200 rewards when collided to the wall or the episode time-step is over which we set to 500. After the agent gets the reward, the environment starts over and repeated the progress for 2000 episodes. Since the LiDAR sensor has continuous values, we discretize the values to 26 sections and 360 sections. Each section represents the observation space of the agent. Moreover, we manually set 5 directional actions from -6.28 degrees to +6.28 degrees in an egocentric view. The corresponding set up is shown in Figure 1.

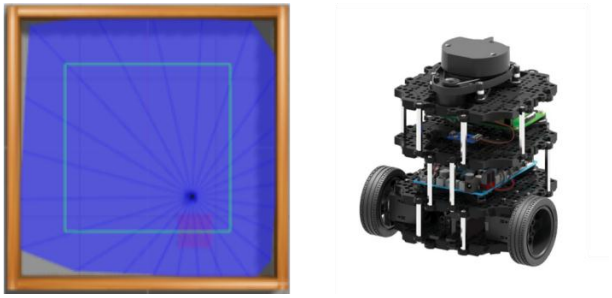

Figure 1: **Left** shows the simple square environment with the 26 discretized observation spaces. **Right** shows the actual robot, Turtlebot3 burger [16].

C. Results

We applied DQN at the same environment setting with different sizes of discretization in observation spaces. We performed twice per each experiment and took about 72 hours for each case. The result is shown in Figure 2.

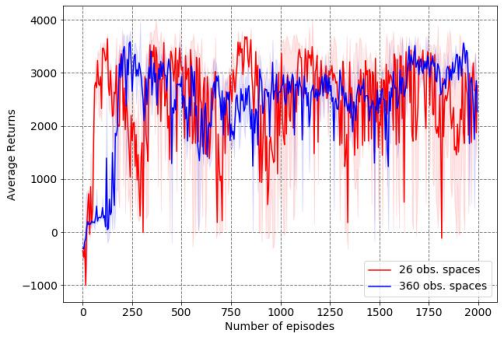

Figure 2: The result of DQN algorithm with the different observation spaces, 26 and 360.

The graph shows that too finely discretized observation spaces lose the ability to generalize. On the other hand, low discretized observation spaces result in high variances in the performance. Note that in the first 200 episodes, the 360 observation spaces

achieved 3000 average rewards which are slower than the 26 observation spaces. We see the reason as the higher observation values achieve the increased size of the training data that we need. Evaluating this result, the discretizing values need to be careful to not lose generalization and decrease the variance of performance.

D. Scalability

To apply an algorithm to different environment settings, the performance has to be compatible. The experimented environment is usually limited in the size of the map, agents, or obstacles. Therefore, scalability is one of the main considerations in the RL world. Also, the scalable is closely related to generalization. In this section, we especially focus on the scalability in the number of agents. Using Gazebo tool, we created the customized environment and launched multiple robots. Figure 3 shows the simple laboratory and two robots are set.

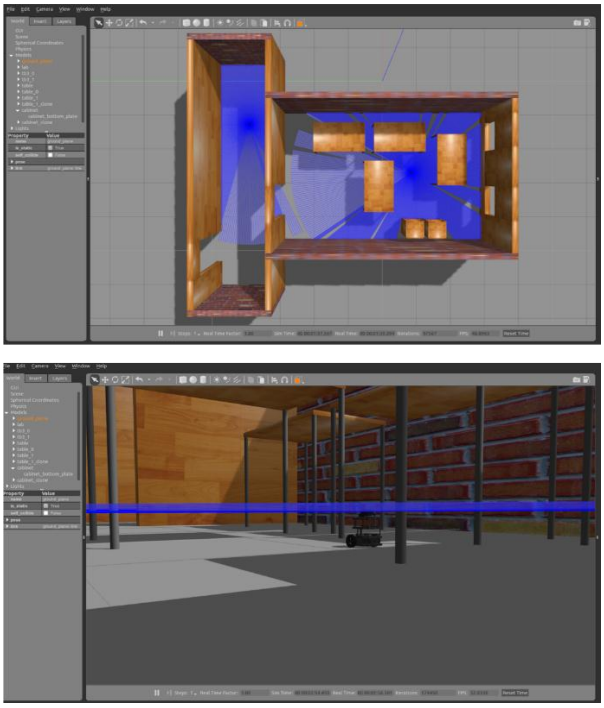

Figure 3: Customized laboratory environment with multiple Turtlebot3 robots. **Top** shows a top-down view. **Right** shows an egocentric view.

DISCUSSION

Accompanying with recent researches, Deep RL can apply to navigation models. However, adapting to the real industry still requires several issues, such as safety. Tackling this matter, RL community also studies uncertainty and probability theory to training algorithm. Haarnoja et, al. [17] incorporates the entropy-based model with Deep RL algorithm to introduce safe RL. Sung et, al. [18] experiments dropout regularization to discuss the exploration and exploitation considering the uncertainty.

The physical simulator, Gazebo, also meet slow training performance by leveraging simulated results. In every time-step, to update the learning algorithm, the agent has to wait for actions which takes much longer time than updating. Another physical simulation, Gym [19] developed by OpenAI provides on/off rendering operation which benefits saving time. To decrease the learning time, the simulator needed to work in the background that can speed the training up.

CONCLUSION

We view two RL approaches, model-based and model-free, and navigation methods, CMP and UNREAL. With this approach, the learning method can be performed from the simulated world as pre-training from the real world. This way can save costs and avoid unexpected situations, compared to traditional DP. Moreover, we experimented the model-free algorithm, DQN with different sizes of observation spaces to discover effects of each case. To consider scalability, we built the simulated environment and multiple robots. We believe the training in simulation to the real world would be a promising way.

ACKNOWLEDGMENT

This material is based upon work supported by the Ministry of Trade, Industry & Energy (MOTIE, Korea) under Industrial Technology Innovation Program. No.10077659, 'Development of artificial intelligence based mobile manipulator for automation of logistics in manufacturing line and logistics center'

REFERENCES

[1] Sadeghi, F., Toshev, A., Jang, E., and Levine, S. (2017) Sim2real view invariant visual servoing by recurrent control. *arXiv preprint arXiv*:1712.07642.

[2] Ha, D., and Schmidhuber, J. (2018) World models. *arXiv preprint arXiv*:1803.10122.

[3] Kingma, D., and Welling, M. (2013) Auto-encoding variational bayes. *arXiv preprint arXiv*:1312.6114.

[4] Graves, A. (2013) Generating sequences with recurrent neural networks. *arXiv preprint arXiv*:1308.0850.

[5] Kendall, A., Hawke, J., Janz, D., Mazur, P., Reda, D., Allen, J. M. Lam, V. D., Bewley, A., and Shah, A. (2018) Learning to Drive in a Day. *arXiv preprint arXiv*:1807.00412.

[6] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013) Playing atari with deep reinforcement learning. *arXiv preprint arXiv*:1312.5602.

[7] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2015) Continuous control with deep reinforcement learning. *arXiv preprint arXiv*:1509.02971.

[8] Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015) Trust region policy optimization. In *Proceedings of the 32nd International Conference on Machine Learning (ICML)*, 15, 1889-1897.

[9] Gupta, S., Davidson, J., Levine, S., Sukthankar, R., and Malik, J. (2017) Cognitive mapping and planning for visual navigation. *arXiv preprint arXiv*: 1702.03920.

[10] Tamar, A., Yi W., Garrett T., Sergey L., and Pieter A. (2016) Value iteration networks. In *Advances in Neural Information Processing Systems*, 2154-2162.

[11] Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. (2016) Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv*:1611.05397.

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

[13] Koenig, N. P., and Howard, A. Design and use paradigms for Gazebo, an open-source multi-robot simulator. (2004) In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 4, 2149-2154.

[14] Kilgard, M. J. The OpenGL Utility Toolkit (GLUT) Programming Interface. (1996).

[15] Smith, R. Open dynamics engine. (2005).

[16] Thai, C. N. (2017) ROBOTIS'Robot Systems. In *Exploring Robotics with ROBOTIS Systems*, 5-21.

[17] Haarnoja, T., Tang, H., Abbeel, P., and Levine, S. (2017) Reinforcement learning with deep energy-based policies. *arXiv preprint arXiv*:1702.08165.

[18] Sung, T. T. Kim, D., Park, S. J., and Sohn C. B. (2018) Dropout Acts as Auxiliary Exploration. In *International Journal of Applied Engineering Research*, 13(10), 7977-7982.

[19] Brockman, G., Vicki C., Ludwig P., Jonas S., John S., Jie T., and Wojciech Z. (2016) Openai gym. *arXiv preprint arXiv*:1606.01540.