Depth-based obstacle avoidance through deep reinforcement learning

Wu, Keyu; Mahdi Abolfazli Esfahani; Yuan, Shenghai; Wang, Han

2019

Wu, K., Mahdi Abolfazli Esfahani, Yuan, S. & Wang, H. (2019). Depth-based obstacle avoidance through deep reinforcement learning. ICMRE'19: Proceedings of the 5th International Conference on Mechatronics and Robotics Engineering, 102-106.
doi:10.1145/3314493.3314495

<https://hdl.handle.net/10356/142309>

<https://doi.org/10.1145/3314493.3314495>

© 2019 Association for Computing Machinery. This paper was published in ICMRE'19: Proceedings of the 5th International Conference on Mechatronics and Robotics and is made available with permission of Association for Computing Machinery.

Downloaded on 26 Apr 2023 17:16:31 SGT

Depth-based Obstacle Avoidance through Deep Reinforcement Learning

Keyu Wu

Nanyang Technological University
Singapore
wukeyu@ntu.edu.sg

Shenghai Yuan

Nanyang Technological University
Singapore
SYUAN003@e.ntu.edu.sg

Mahdi Abolfazli Esfahani

Nanyang Technological University
Singapore
MAHDI001@e.ntu.edu.sg

Han Wang

Nanyang Technological University
Singapore
HW@ntu.edu.sg

ABSTRACT

Obstacle avoidance is an indispensable technique for mobile robots to maneuver safely without collision. In this paper, we propose an end-to-end deep neural network to derive control commands directly from the raw depth images using deep reinforcement learning. The convolutional neural networks are used to extract the feature representation from the input depth images and the fully connected neural networks subsequently map the features to Q-values for determination of the optimal action. To improve the performance of the network, we adopt a two-stage method so that noisy fully connected layers are employed at the beginning while conventional ones are utilized during the second stage of training. Compared to the existing method, our proposed model exhibits much better performance in avoiding obstacles and converges faster during training.

CCS CONCEPTS

• **Computer systems organization** → Robotics;

KEYWORDS

Deep Reinforcement Learning, Obstacle Avoidance

1 INTRODUCTION

It is fundamental for autonomous robots to navigate safely without any collision. A large variety of obstacle avoidance approaches have been proposed [1, 8], however, conventional

methods typically requires to tune a number of defined parameters and are difficult to transfer to new situations.

In recent years, deep reinforcement learning (DRL) has achieved impressive successes in a variety of domains, such as video games [7], Go [9], and robotics [4, 5]. Moreover, DRL has also been applied to solve navigation problems. For instance, Tai et al. proposed an asynchronous DRL model to map the laser scans to control commands. Due to the high transferability of laser scans, their model was capable to adapt to real-world applications easily after training in a robotic simulator. However, the application of 2D laser scanners fails to preserve the original 3D information. Therefore, Zhu et al. presented a DRL method to output four command-level actions according to the input RGB images [13]. Although vision-based methods can retain the 3D spatial information, nevertheless, it is challenging for RGB-based models to adapt to real-world scenarios without fine-tuning. The DRL model introduced in [12] addressed this issue by deriving control policies directly from depth images. Besides, Xie et al. proposed to firstly convert the RGB input to depth images through a fully convolutional residual network, and then passed the synthetic depth maps into a variant of Deep Q-Network (DQN) to resolve the optimal actions [11].

As a value-based DRL algorithm, DQN was first proposed by Mnih et al. [6, 7] and has been utilized to tackle various challenges since then [2]. Generally, DQN is featured by two techniques, including experience replay and target network. It stored the experienced transitions in a replay memory to reduce the amount of required interactions while eliminate the temporal correlations of the training data. Besides, the maintenance of a target network alleviate the problem of rapidly fluctuating estimates of target Q-values to achieve more stable training. However, the maximum operation used in the update rule of DQN leads to the overestimation of Q-values. Therefore, the double deep q-network (DDQN)

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

ICMRE'19, February 2019, Rome, Italy

© 2019 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-6365-5.

https://doi.org/10.475/123_4

was introduced to improve the estimation by using a double estimator [10].

In this paper, we propose a two-stage noisy double deep Q-network to address the problem of obstacle avoidance. Rather than adopting the commonly used ϵ -greedy policy, we employ the noisy network [3] to achieve better exploration in a more consistent way during the first stage. The noise components are then deliberately removed in the second stage so that our model is further tuned for better performance. Due to the end-to-end nature, our network is capable of deriving velocity commands directly from the depth images and the experiment results demonstrate its superiority both in terms of effectiveness and training efficiency.

The rest of the paper is organized as follows. The proposed model is described in Section 2 and the experiment results are presented in Section 3. Lastly, the paper is concluded in Section 4.

2 TWO-STAGE NOISY DDQN

2.1 DDQN

A Markov Decision Processes (MDP) is a tuple (S, A, P, R, γ) , where S represents the state space, A is the action space, P denotes the state transition probability function, R indicates the reward function, and γ signifies a discount factor. A policy $\pi(a|s)$ is defined as the probability of choosing action a in state s and its quality can be assessed by the Q-value which is defined as the expected γ -discounted cumulative reward resulted in by executing policy π after taking action a in state s . The Q-value of the optimal policy can then be expressed as $Q^*(s, a) = \arg\max_{\pi} Q^{\pi}(s, a)$.

In DQN, the Q-value of the optimal policy is estimated by minimizing the loss function defined in Eq. (1) with respect to the parameters of the neural network. In Eq. (1), D denotes a replay buffer, θ is the parameters of the online network which is updated at each iteration, whereas θ^- represents those of the target network which is frozen and merely updated after a number of iterations.

$$L(\theta) = \mathbb{E}_{(s,a,r,s') \sim D} [(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta))^2] \quad (1)$$

In order to avoid the overestimation of Q-values, DDQN decouples the action selection from the calculation of target Q-value so that the online network is used to select the optimal action while the target network is adopted to compute the Q-value of the selected action. Therefore, the loss function is re-defined as:

$$L(\theta) = \mathbb{E}_{(s,a,r,s') \sim D} [(r + \gamma Q(s', \max_{a'} Q(s', a'; \theta^-); \theta^-) - Q(s, a; \theta))^2] \quad (2)$$

Figure 1: Architecture of the proposed model. The feature representation is obtained through convolutional layers and mapped to Q-values through fully connected layers. The optimal action is then determined based on the Q-values. Noisy fully connected layers are adopted in the first stage whereas conventional ones are used in the second stage.

2.2 Model and Training

Our two-stage noisy DDQN is an extension of DDQN and its architecture is illustrated in Figure 1. As demonstrated, the input state of the network is a stack of four 60×80 depth images obtained from successive steps and two convolutional neural networks (CNNs) are employed to extract the feature representation from the input images. In specific, the first CNN produces 32 feature maps of size 15×20 using 6×8 kernels and a stride of 4. The second CNN then filters these feature maps through 64 kernels of size 3×3 and a stride of 2. The output is subsequently flattened and fed through the remaining fully connected layers. The output of the network is the Q-values of the eight discretized actions, including three linear velocity commands ($0.5m/s$, $0.35m/s$, and $0.2m/s$) and five angular velocity commands ($-\frac{\pi}{6}rad/s$, $-\frac{\pi}{12}rad/s$, $0rad/s$, $\frac{\pi}{12}rad/s$, and $\frac{\pi}{6}rad/s$). The optimal action is accordingly selected as the one which has the largest Q-value.

During the first stage of training, the noisy fully connected layers are applied rather than the conventional ones. And the weights ω and bias b in equation $y = \omega x + b$ are re-formatted respectively as:

$$\omega_{i,j} = \mu_{i,j}^\omega + \sigma_{i,j}^\omega f(\epsilon_i) f(\epsilon_j), \quad (3)$$

and

$$b_j = \mu_j^b + \sigma_j^b f(\epsilon_j), \quad (4)$$

where $f(\epsilon) = \text{sgn}(\epsilon) \sqrt{|\epsilon|}$ while ϵ_i and ϵ_j are random noise variables. Besides, $\mu_{i,j}^\omega$, μ_j^b , $\sigma_{i,j}^\omega$, and σ_j^b are trainable parameters of the network. During the second stage of training, the noise variables are eliminated so that the weights and bias are changed to:

$$\omega_{i,j} = \mu_{i,j}^\omega, \quad (5)$$

and

$$b_j = \mu_j^b. \quad (6)$$

The weights of the neural network is initialized by the Xavier initializer and the bias are initialized to zero. Regarding the noisy layers, parameters μ are uniformly sampled between $-\frac{1}{\sqrt{N}}$ and $+\frac{1}{\sqrt{N}}$ where N is the number of inputs to the corresponding layer. Besides, parameters σ are initialized to constant $\frac{0.8}{\sqrt{N}}$. At each training iteration, the batch size is set to 32 and the model is trained using the RMSProp optimizer with a learning rate of $2.5e-6$. In addition, the parameters of the target network is updated to those of the online network after every 100 iterations.

Moreover, the instantaneous reward at each step is defined as:

$$r = \begin{cases} -10 & \text{if in collision} \\ \tanh(2v \cos(\omega) - 0.3) & \text{otherwise} \end{cases}. \quad (7)$$

where v represents the linear velocity and ω denotes the angular velocity. In this way, the domain of the reward function is appropriate for training and the agent is supposed to move forward as fast as possible and only turn to avoid obstacles. Besides, an episodes terminates immediately when a collision is detected or the maximum number of steps is reached. In the latter case, the episode terminates without any punishment.

3 EXPERIMENT RESULTS

The proposed two-stage noisy DDQN model is trained and evaluated using a simulated Turtlebot in a Gazebo environment illustrated in Figure 2. The depth images are captured using the depth sensor of a simulated Kinect and the training is implemented using a GeForce GTX 1080 Ti Graphics Card.

The robot is located at the origin of the simulated world at the beginning of each episode. At each training iteration, the robot interacts with the environment by selecting the optimal action according to the Q-values calculated by the online network and transit from the current state to a new state.

Figure 2: The simulated environment for training and evaluation.

This transition is stored in the experience replay memory with a maximum capacity of 2000. In the meantime, a batch of 32 training data is randomly sampled from the replay buffer and the parameters of the online network are updated by backpropagating the loss.

To demonstrate the superiority of our model, a comparison experiment is conducted. Besides the two-stage noisy DDQN model, a DDQN model and a noisy DDQN model are also trained with the same configuration and random seed. The conventional fully connected neural networks are used to map the feature representation to Q-values in the DDQN model while the noisy fully connected layers are employed throughout the learning process in the noisy DDQN model. During the training, the DDQN model adopts the ϵ -greedy policy to achieve the exploration and the value of ϵ is reduced from 0.1 to 0.0001 constantly during the first 200000 iterations. Meanwhile, the noisy DDQN model and the two-stage noisy DDQN model both employ the greedy policy and achieve the exploration via the introduction of noise variables instead. The evaluation is carried out every 5000 iterations to calculate the average episode reward which is defined as the total reward obtained in an episode. More specifically, the episode reward is the accumulation of instantaneous rewards of all steps within that episode. The average total rewards attained by the three models are illustrated in Figure 3.

According to Figure 3, the average episode rewards obtained by the DDQN model is the lowest throughout the experiments. Compared to DDQN model, the noisy DDQN model leads to higher rewards by introducing the noise terms. Moreover, it is worth noting that the convergence speed of the noisy DDQN model is significantly increased. However, although the performance of noisy DDQN model is improved by introducing greater uncertainties to the parameters and tuning the amount of injected noise automatically, the optimization of this model turns to be difficult and unsteadily

Figure 3: The average episode rewards obtained by the three models at different training steps.

at a later phase. On the contrary, it can be observed that the performance and stability of the two-stage noisy DDQN model is much better by eliminating the variability in decisions during the second stage. Hence, the proposed model can achieve higher training and exploration efficiency by introducing the uncertainties at the first stage while it is also able to realize better optimization and stability at the second stage.

In order to demonstrate the effectiveness of the proposed two-stage noisy DDQN model in a more intuitive way, two intermediate processes during an episode are provided in Figure 4 and Figure 5, respectively as two collision avoidance examples. In both figures, four intermediate steps are displayed in sequence. And for each step, the top view of the simulated scenario, the depth and RGB images captured by the Kinect are displayed accordingly. In the first example, the Turtlebot succeeds in avoid the collision from under the table. And in the second example, the Turtlebot changes its maneuver direction to prevent itself from colliding with the wall.

4 CONCLUSION

In this paper, a two-stage noisy DDQN model is proposed to address the obstacle avoidance problem for mobile robots. Our end-to-end model is capable of directly mapping the raw depth images to control commands based on deep reinforcement learning. Specifically, the convolutional neural networks are firstly adopted to extract the feature representation and the fully connected neural networks then derives the Q-values accordingly to determine the optimal action. During the first stage of learning, we adopts the noisy fully connected layers to achieve more efficient exploration and

Figure 4: Example of collision avoidance from under the table. (a), (b), (c), and (d) illustrate the four intermediate steps in sequence. For each step, the left column shows the top view of the scenario, the middle column demonstrates the corresponding depth image, and the right column is the corresponding RGB image.

training while during the second stage, we eliminates the variations in decisions for better performance and stability. Experiment results have demonstrated the superiority of our model compared to existing methods.

ACKNOWLEDGMENT

This work is supported by the ST Engineering-NTU Corporate Lab.

REFERENCES

- [1] Meteb M Altaf, Bassant M El Bagoury, Fahad Alraddady, and Said Ghoniemy. 2015. Enhancing Case-Based Retrieval Engine with Case Retrieval Nets for Humanoid Robot Motion Controller. *International Journal of Machine Learning and Computing* 5, 3 (2015), 235.
- [2] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. 2017. A brief survey of deep reinforcement learning. *arXiv preprint arXiv:1708.05866* (2017).
- [3] Meire Fortunato, Mohammad Gheslaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis

Figure 5: Example of avoiding collision with the wall. (a), (b), (c), and (d) illustrate the four intermediate steps in sequence. For each step, the left column shows the top view of the scenario, the middle column demonstrates the corresponding depth image, and the right column is the corresponding RGB image.

- [9] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *nature* 529, 7587 (2016), 484.
- [10] Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning.. In *AAAI*, Vol. 2. Phoenix, AZ, 5.
- [11] Linhai Xie, Sen Wang, Andrew Markham, and Niki Trigoni. 2017. Towards monocular vision based obstacle avoidance through deep reinforcement learning. *arXiv preprint arXiv:1706.09829* (2017).
- [12] Jingwei Zhang, Jost Tobias Springenberg, Joschka Boedecker, and Wolfram Burgard. 2017. Deep reinforcement learning with successor features for navigation across similar environments. In *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*. IEEE, 2371–2378.
- [13] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. 2017. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 3357–3364.

Hassabis, Olivier Pietquin, et al. 2017. Noisy networks for exploration. *arXiv preprint arXiv:1706.10295* (2017).

- [4] Shixiang Gu, Ethan Holly, Timothy Lillicrap, and Sergey Levine. 2017. Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 3389–3396.
- [5] Tuomas Haarnoja, Vitchyr Pong, Aurick Zhou, Murtaza Dalal, Pieter Abbeel, and Sergey Levine. 2018. Composable Deep Reinforcement Learning for Robotic Manipulation. *arXiv preprint arXiv:1803.06773* (2018).
- [6] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).
- [7] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.
- [8] A Pandey, S Pandey, and DR Parhi. 2017. Mobile robot navigation and obstacle avoidance techniques: A review. *Int Rob Auto J* 2, 3 (2017), 00022.