Using RGB Image as Visual Input for Mapless Robot Navigation

Liulong Ma, Yanjie Liu*, and Jiao Chen

***Abstract*—Robot navigation in mapless environment is one of the essential problems and challenges in mobile robots. Deep reinforcement learning is a promising technique to tackle the task of mapless navigation. Since reinforcement learning requires a lot of explorations, it is usually necessary to train the agent in the simulator and then migrate to the real environment. The big reality gap makes RGB image, the most common visual sensor, rarely used. In this paper we present a learning-based mapless motion planner taking RGB images as visual inputs. In the end-to-end navigation network many of the parameters are used to extract visual features. The proposed motion palnner decoupled visual features extracted module from the reinforcement learning network to improve the sample efficiency. Variational Autoencoder (VAE) is used to encode the image, and the obtained latent vector is input as low-dimensional visual features into the network together with the target and motion information. We built and released a set of simulation environments for algorithm comparison. In the test environment, the proposed method was compared with the end-to-end network, which proved its effectiveness and efficiency. The source code is available: *https://github.com/marooncn/navbot*.**

I. Introduction

Robot navigation is an essential capability of mobile robot. It can be roughly described as the ability to plan and follow a path or output the real-time policy to desired target from the current position without colliding with obstacles. Traditional methods have been somewhat fragmented; the navigation part pay more attention to motion planning which seeks a collision-free path in workspace. Motion planning approaches rely on precise geometric model of environment and perfect localization, limiting the use of these methods. There is another related method, simultaneous localization and mapping (SLAM), has undergone rapid development [1] and has been widely used for autonomous driving systems. Manually designed features, like ORB (Oriented FAST and Rotated BRIEF) [2], are extracted to construct an obstacle map and localize the robot. It focuses on building an obstacle map of the navigation environment and localizing the robot itself within the map. But navigation itself is less considered and representations constructed by SLAM are often not suitable for traditional motion planning methods [3].

* This research has been sponsored by the National Key R&D Program of China(No. 2017YFB1303801), the Provincial Funding for National Key R&D Program(Task)(No. GX18A011), the Self-Planned Task (No. SKLRS201813B) of State Key Laboratory of Robotics and System (HIT).

Liulong Ma, Yanjie Liu and Jiao Chen are with the State Key Laboratory of Robotics and System, Harbin Institute of Technology, China. E-mail: LiulongMa@outlook.com, yjliu@hit.edu.cn, Jeffery-Chen@outlook.com.

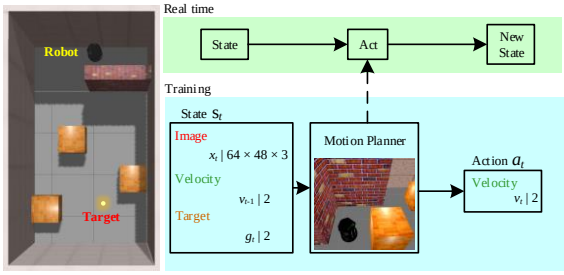

Figure 1. The illustration of our mapless navigation. In the task, the motion planner is trained to navigate a nonholonomic mobile robot to the target.

In contrast, animals including humans can traverse the complex and wide-range environments to the goal location without precise localization or a metric map. In this process, animals build up internal representations of the environments. Similarly, how to navigate without a map, namely mapless navigation, is an important issue. Recent literatures have tried to tackle the task of mapless navigation using Deep Reinforcement Learning (DRL) algorithm. Reinforcement learning (RL) is an algorithm to learn how to map situations to actions so as to maximize the numerical reward signal. The decision-making agent interacts with the environment, within which the agent seeks to achieve the specified goal. Initially, the agent does not know which action should take, but it can find out which one yields the most cumulative reward by trying them. Cumulative reward, evaluated by value function in RL, measures how good the action is in the long run. It's more important than immediate reward because some actions can get high immediate reward, but at the expense of goal. These two characteristics, trial-and-error search and delayed reward, are the two most important distinguishing features of RL [4]. DRL enhances RL by deploying deep neural network as more powerful non-linear value function approximators. DRL can also learn and reason from exploration and corresponding reward signals, and output continuous control policies as the state changes, which makes it suitable for the task of mapless navigation. As of now, reinforcement learning has been applied to a variety of scenarios successfully, including: video game [5], Go [6], robotic manipulation [7] and even object detection [8], image restoration [9].

We list the main contributions of this paper: (1) Propose a sample-efficient mapless motion planner by taking RGB image as visual input and target related information as references. (2) Compare two state-of-the-art deep reinforcement learning algorithms in end-to-end network and choose the better one as the benchmark. (3) Build the simulation environment as robot navigation environment for algorithm comparison. The benchmark was compared with the proposed approach in the built environment.

II. Related Work

A. Deep Reinforcement Learning

Deep reinforcement learning has been rapidly developed since Deep Q-Network (DQN) was successfully applied to Atari games. In this work, deep neural network was used to extend Q-learning [10] to evaluate state-action value functions Q(*s, a*). The network parameters were updated according to the error between calculated target Q-value (called *td-target*) and evaluated Q-value in the randomly selected samples stored in experience memory. Target network and experience replay were two main technologies proposed in DQN to stabilize learning. Although the agent only received the pixels and game score as inputs, it performed so well that even reached a level of professional player in some Atari games, which was impossible in the past [5]. Then various improved algorithms based on DQN were proposed to stabilize training and improve efficiency. Double DQN separated the action slection and evaluation to prevent overoptimistic value estimates and avoid upward bias [11]. For Dueling DQN, the network has two streams to separately estimate state-value and the advantages for each action [12]. Prioritized replay improved experience replay by sampling according to surprise [12]. Rainbow combines six extensions to the DQN [13].

But DQN and these improved algorithms can only deal with discrete and low-dimensional actions. Deep deterministic policy gradient (DDPG) [14] combines technologies from DQN with deterministic policy gradient [15] by actor-critic framework [16] to output continuous actions. Actor-critic integrates the value-based method (critic, learn value function, such as DQN) and policy-based method (actor, learn policy directly). Normalized advantage function (NAF) is another continuous variant of the Q-learning algorithm [17]. NAF represents the Q-value function in such a way that the best policy can be easily determined. Asynchronous advantage actor-critic (A3C) [18] is an asynchronous DRL algorithms with actor-critic framework. The parallelization collects the samples from different actor-learners functioning as a replay memory.

Trust Region Policy Optimization (TRPO) [19] and Proximal Policy Optimization (PPO) [20] are two effective policy-based methods. They take the largest step possible to improve performance while satisfying the constraint on the step size to avoid performance collapse. TRPO reformulated a hard constraint with complex second-order method. PPO instead used first-order method with a few tricks to keep new policies close to old. PPO strikes a balance between performance and generalization. We'll compare the DQN-based planner and PPO-based in the end-to-end mapless navigation experiment in part V.

B. DRL-based Mapless Navigation

DRL-based navigation tries to solve the problem of mapless robot navigation with DRL algorithms. Specifically speaking, tasks include cognitive exploration [21], obstacle avoidance and target reaching. Various sensors, including laser sensor [22], depth camera [23] and monocular camera, were used to perceive the environment information. Various DRL algorithms such as DQN, DDPG, A3C were carried out to get the optimal policy to complete navigation task efficiently [21, 22, 24]. Imitation learning such as inverse reinforcement learning (IRL) [25] and generative adversarial imitation learning (GAIL) [26] were also tried for socially compliant navigation [27, 28].

Some specific technologies and architectures for navigation task were also proposed and got the good performance. Jaderberg et al. [29] proposed UNREAL architecture to augment the DRL agent with auxiliary control and reward prediction tasks. Mirowski et al. [24] greatly improved the performance of their variant of A3C agent by adding LSTM layers to memorize the learned information and using supervision signals from auxiliary tasks. Mirowski et al. [21] proposed a MultiCityNav agent that can be applied on a city scale. Its network architecture includes three parts: a convolutional network to extract visual features, a locale-specific recurrent neural network to memorize the environment and a locale-invariant recurrent network to produce navigation policy. Zhelo et al. [30] utilized intrinsic curiosity module (ICM) [31] to solve the navigation task with sparse reward. Zhu et al. [32] input observation and target image together into the deep Siamese actor-critic model to realize the visual navigation in indoor scenes.

III. Benchmark

A. Problem Definition

In order to implement navigation tasks, the robot receives RGB image as observation and target information, motion information as reference. RGB image information is used to avoid obstacles and remember the environment, the target information tells the agent which direction it should move to get close to the target, and the motion information affects the decision of movement's speed due to inertia. Our goal is to output the navigation policy based on the given information to reach the target, that is, to determine the mapping from input to the next action:

$$v_t = f(x_t, g_t, v_{t-1}) \tag{1}$$

where x_t is the observation with 64×48×3 dimension, g_t is 2-dimensional target position in polar coordinates (distance and angle) with respect to the mobile robot coordinate frame. The robot moves at a relatively low speed, thus we directly took the last action v_{t-1} as the motion information [22].

B. End-to-end Network Structure

Convolutional neural network has been successfully applied to analyzing visual imagery such as image recognition [33]. The convolutional layer is exceptionally valid for image to extract features. The features of RGB images can be extracted by convolutional layers firstly and then the low dimensional features merge with motion and target information. An end-to-end network is designed as Figure 2.

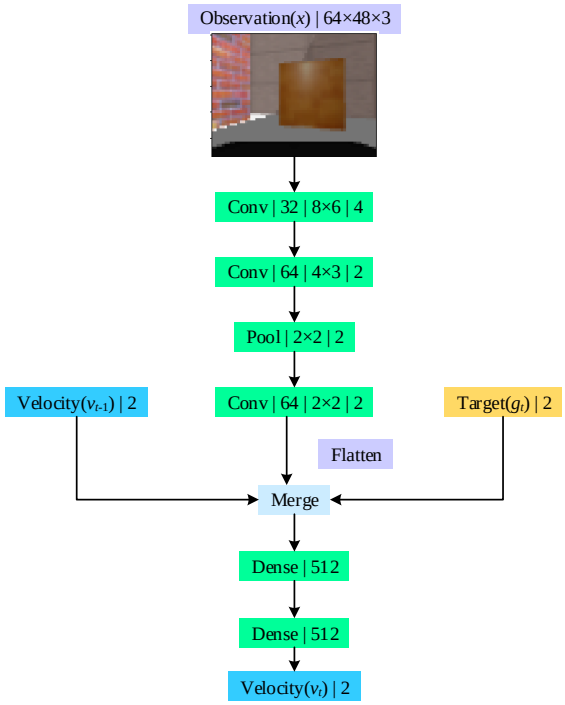

Figure 2. Network structure for continuous action space. Every convolutional layer is represented by its type, channel size, kernel size and stride size. Other layers are represented by their types and dimensions.

In this network three convolutional layers are deployed to extract features of the observation and two fully-connected layers are deployed to evaluate Q-value if the action space is discrete, or choose linear and angular velocity if continuous. Each convolutional layer and fully-connected layer is followed by a Rectified Linear Unit (ReLU) activation function to increase the non-linearity for better data fitting. In the following chapter, we compared two classic DRL algorithms: DQN for discrete action space and PPO for continuous action space, the corresponding networks are named *E2E-DQN network* and *E2E-PPO network*. Fig. 2 is the structure of *E2E-PPO network*, which is the same as *E2E-DQN network* except for the output layer.

C. E2E-DQN Network

As mentioned in II.A, the agent interacts with environment and collects the experience $e_t = \{s_t, a_t, r_t, s_{t+1}\}$ at each time-step t in a data set $D_t = \{e_1, ..., e_t\}$. to perform experience replay. During learning, Q-learning update [10] is applied, on mini-batches of experience $(s, a, r, s') \sim U(D)$. The evaluated *td-target* is:

$$y^{DQN} = r + \gamma \max_{a'} Q(s', a'; \theta) \tag{2}$$

in which γ is the discount factor, θ are the parameters of the Q-network. And the *td-error* is the error of *td-target* and evaluated Q value:

$$\delta^{DQN} = y^{DQN} - Q(s, a; \theta) \tag{3}$$

Then the update step is performed based on the following gradient descent formula with a learning rate of α:

$$\theta \leftarrow \theta - \alpha(\partial(\delta^{DQN}(\theta))^2 / \partial\theta) \tag{4}$$

Three technologies are applied to stabilize learning:

Target-network: It is used to evaluate *td-target*, because *td-target* calculated by formula (2) is not stable with the continuous update of Q-network. The parameters of target-network θ^- are periodically copied from θ.

Double DQN: Use target-network to evaluate *td-target* and use Q-network to choose the action:

$$y^{Double} = r + \gamma Q(s, \arg\max_{a'} Q(s, a'; \theta); \theta^-) \tag{5}$$

Gradient clipping: Limit the gradient for each update to prevent gradient explosions.

D. E2E-PPO Network

The structure of *E2E-PPO network* is shown in Figure. 2, its outputs are linear and angular velocity commands of the mobile robot. PPO is a novel scalable variant of policy gradient reinforcement learning algorithm. Like TRPO, it can perform multiple gradient updates per data sample, thus it is more efficient than traditional policy gradient algorithm. PPO can be seen as an approximate version of TRPO that relies only on first order gradient, making it much simpler to implement and have better generalization ability.

The update step is performed via:

$$\theta \leftarrow \arg\max_{\theta} \mathop{\mathrm{E}}_{s,a\sim\pi_{\theta_k}} [L(s, a, \theta_k, \theta)] \tag{6}$$

L is given by:

$$L(s, a, \theta_k, \theta) = \min(r(\theta)A(s, a), clip(r(\theta), 1-\varepsilon, 1+\varepsilon)A(s, a)) \tag{7}$$

in which $r(\theta) = \frac{\pi_\theta(a \mid s)}{\pi_{\theta_k}(a \mid s)}$, the ratio of the probability under the new and old polices, $A(s, a)$ is the estimated advantage, ε is a small hyperparameter that limits the new policy close to the old.

In part Ⅴ, we will list the experiment details and their performances, and choose the better one as benchmark algorithm.

IV. Proposed Motion Planner

In the above end-to-end network, the first four layers are applied to extract visual features of the observation. In the learning process, parameters of these layers are updated together with parameters of two fully-connected layers, which largely increase the required experiences to accomplish the navigation task in the environment. Thus we propose a new motion planner which decouples visual extractor from the reinforcement learning network. Firstly, the observation is encoded to low-dimensional latent vector $\boldsymbol{z}$ as shown in Figure 3. Then the latent vector $\boldsymbol{z}$ can be input into the reinforcement learning network as visual features together with motion and target information.

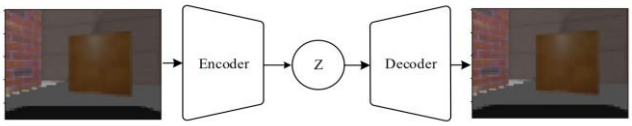

Figure 3. The observation is encoded to latent vector, which can be decoded to the original image.

There are three criteria that the encoder should satisfy:

- Independent training. It can be trained if there are only visual images of the environment.
- Information reserving. The encoded latent vector ***z*** contains almost all the important environment information in the observation. The original image can be reconstructed with latent vector ***z***.
- Ability of generalization. It can not only reconstruct the trained images, but also the unseen style-like images.

Auto-encoder encodes an image into a vector which can then be decoded to the original image, it satisfies the condition "Independent training" and "Information reserving". But it's not good at generalizing. In our test the performance of a trained auto-encoder for unseen style-like image is not good enough while it has excellent performance in the training set. To increase the ability of generalization, we don't encode the image to a fixed vector directly but encode to the mean and deviation vector, one of which represents the average of the distribution and the other represents the standard deviation. The existence of deviation improves the generalization ability. The latent vector can be sampled based on the mean and deviation vector. This new auto-encoder is Variational Autoencoder (VAE) [34]. VAE is more a generative model that can generate new style-like images.

A suitable network as shown in Figure 4 is designed to encode the observation. The number of elements in latent vector is 32, which makes the original image compressed 288 times. There is a constraint on the encoding network, which forces it to generate latent vectors that roughly follow a unit Gaussian distribution, to make sure the generative attribute.

To train the designed network, there are two parts of losses: reconstruction loss and constraint loss. Reconstruction loss evaluates how close the reconstruction image to the original. It is the mean squared error (MSE) of each pixel value between original image and the reconstruction:

$$loss\_r = \frac{1}{m \times n} \sum_{i,j}^{m,n} (x_{i,j} - \overline{x_{i,j}})^2 \tag{8}$$

And the constraint loss measures that to what extent the latent vector match a unit Gaussian by KL divergence:

$$loss\_c = \int p(x) log \frac{p(x)}{q(x)} \tag{9}$$

in which *q(x)* obeys the unit Gaussian distribution.

The training target is to minimize the total loss:

$$loss = loss\_r + loss\_c \tag{10}$$

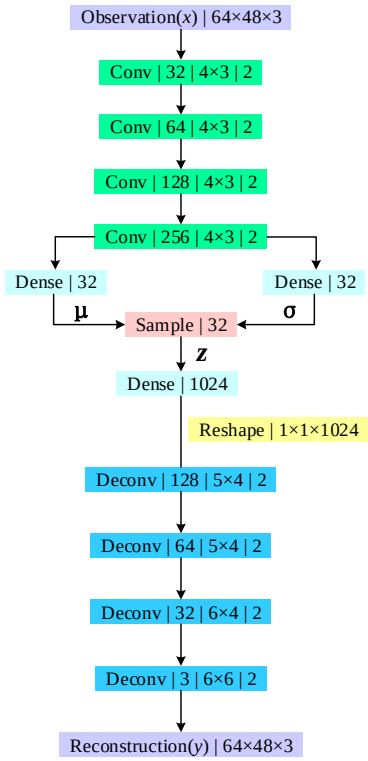

Figure 4. The designed VAE model. 'Sample' means sampling the latent vector *z* by Gaussian distribution N (*u, σ*).

There are more than three million parameters in the designed network. Five million RGB images were collected in the navigation environment introduced in part V to train the network. The network was trained from scratch with the Adam optimizer [35] on a single NVIDIA GeForce GTX 1080Ti GPU and the total loss decreased to 0.0019 in the end. Test the trained model with random unseen images, the result is shown in Figure 5.

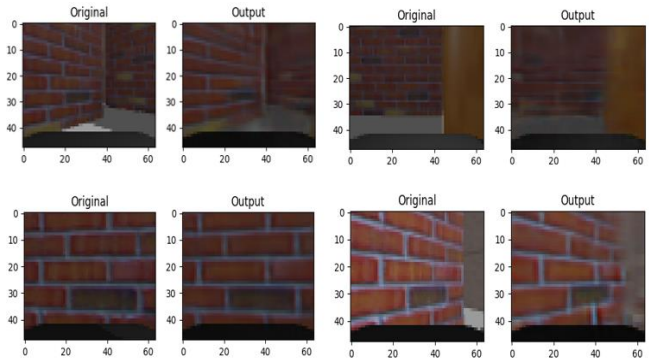

Figure 5. The original images and the corresponding outputs of VAE.

The reconstructed images by latent vector contain almost the same environment information with the original image, even if the original image is not in the training data set. This

model satisfies all the three criteria. Now the problem can be redefined:

$$z_0,\ldots,z_{31} = g_{VAE-encoder}(x_t)$$
$$v_t = h(z_0,\ldots,z_{31},d_t,\alpha_t,v_{t-1},w_{t-1}) \quad (11)$$

The latent vector and target information (d_t, α_t), motion information (v_{t-1}, w_{t-1}) are all low-dimensional data, they can be input together and we use a three fully connected layers to fit the mapping h as shown in Figure 6.

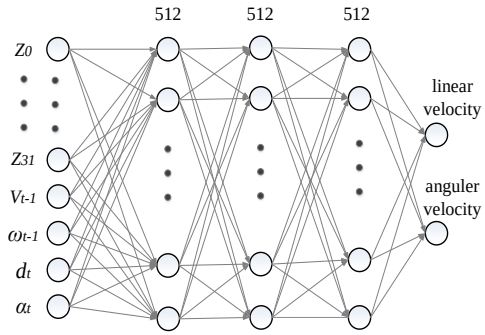

Figure 6. The DRL network.

The following is the whole procedure of the proposed motion planner with PPO as DRL algorithm.

Algorithm 1 Sample-efficient Motion Planner

VAE model trained

Initialize the memory D to store observations, set the samples number T, set the batch size B, set $timestep = 0$
Start the navigation environment
while $timestep < T$ **do**
  Randomly set the mobile robot to a start position
  while not collision **do**
    Capture the RGB image x_t, store x_t in D
    Move with the random linear and angular velocity
  end while
end while
Initialize the parameters of VAE model as θ_0
for $batch = 1, T/B$ **do**
  Select a batch of images D^- randomly from D, and then remove images in D^- from D
  Update θ_0 through a gradient descent procedure on the batch of total loss defined in formula (10)
end for

Motion planner

Initialize the parameters of DRL model as θ_1, load the updated parameters θ_0 of VAE
Set the epochs number K, episodes number N
Start the navigation environment
for $k = 1, 2, \ldots$ **do**
  for $i = 1, N$ **do**
    Start the robot in the initial position
    while not terminal **do**
      Capture the RGB image x_t, encode it to latent vector $\mathbf{z}$ by VAE. Merge action and target information as state s_t
      Input s_t to the network and output action a_t with random noise
      Move with a_t, get the reward r_t
      Collect $\{s_t, a_t, r_t\}$
    end while
  end for
  Compute estimated advantage A_t for all time steps
  Update θ_1 by formula (7) with K epochs
end for

In the following part, the experiments are implemented to compare the proposed motion planner with the benchmark algorithm introduced in part III.

V. EXPERIMENT

A. Environment

To implement the navigation experiment, we built up a collection of navigation environments[1], some of them are shown in Figure 7.

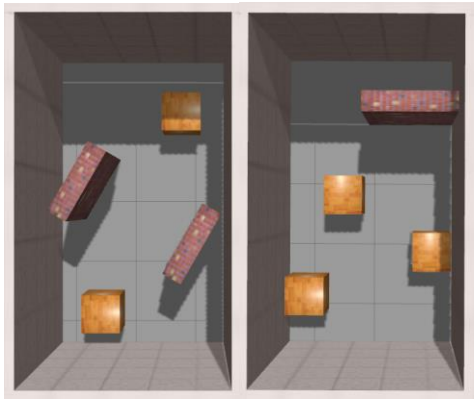

Figure 7. Navigation environment, *maze1*(left) and *maze2*(right).

The built environments have the following properties:

- Diverse complexity. There are both simple and complicated scenes to be chosen.
- Gym-style Interface. The environment interface similar to OpenAI gym [36] is designed.

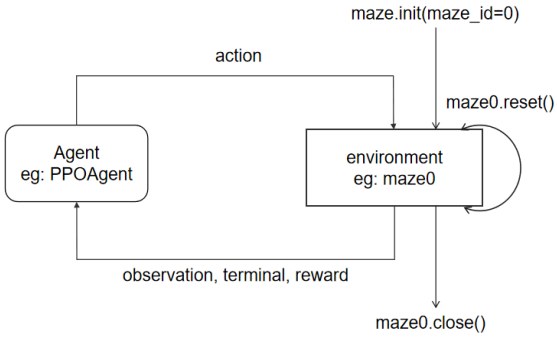

Figure 8. Environment interface.

[1]*https://github.com/marooncn/navbot/tree/master/rl_nav/worlds*

- Robot operating system (ROS) [37] supporting.

The reward function in these environments are designed as:

$$r(s_t, a_t) = \begin{cases} r_{collision}, & if\ collision \\ r_{arrival}, & if\ d_t < c_d \\ c_r(d_{t-1} - d_t) - c_p \end{cases} \quad (12)$$

If the robot arrives at the target position, a positive reward $r_{arrival}$ is arranged, if it collides with obstacle, a negative reward $r_{collision}$ is arranged. Otherwise it's proportional to the difference in the distance relative to the target position, c_p is the time penalty for each step to encourage the agent to find the optimal path.

And the output of the agent is noisy in order to improve generalization performance.

B. Benchmark

The experiment was implemented in *maze1* to compare two end-to-end navigation networks introduced in part III. The result is shown in Figure 9.

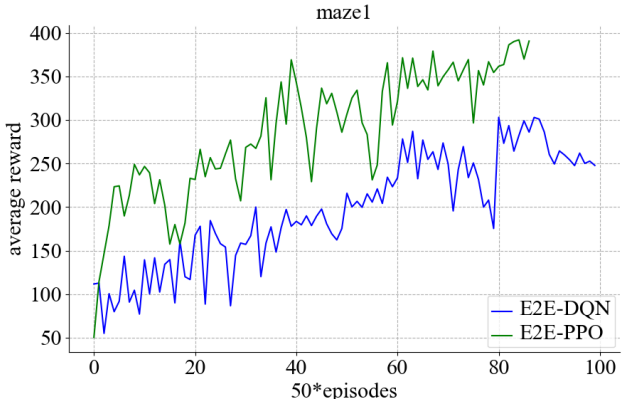

Figure 9. Performance of *E2E_DQN network* and *E2E_PPO network*.

It's obvious that *E2E_PPO network* performs much better than *E2E_DQN network*. What's more, *E2E_PPO* is much easier to converge than *E2E_DQN* in the experiment, with the same hyper-parameters *E2E_DQN* network didn't converge in *maze2* rather than *E2E_PPO* converged quickly.

C. Proposed

Comparing the proposed motion planner with the benchmark, we stopped training when *success rate* reaches 80%. The results are shown in Figure 10 and Figure 11, *average reward* measures the average of total rewards of each 50 episodes and *success rate* means the success rate of navigation task of the last 100 episodes. Whether in *maze1* or *maze2*, the proposed algorithm learned faster, it only used 1/4 samples of the benchmark in *maze1* and nearly 1/3 samples in *maze2* to reach the specified success rate. The proposed algorithm has more sample-efficient. It's effective and efficient to use **Algorithm 1** when using RGB image as visual input in the mapless navigation.

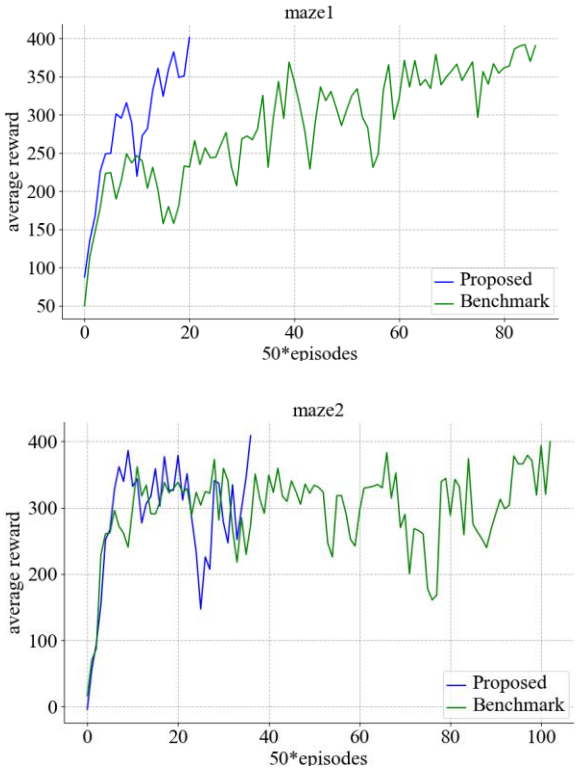

Figure 10. Average reward of proposed motion planner and benchmark.

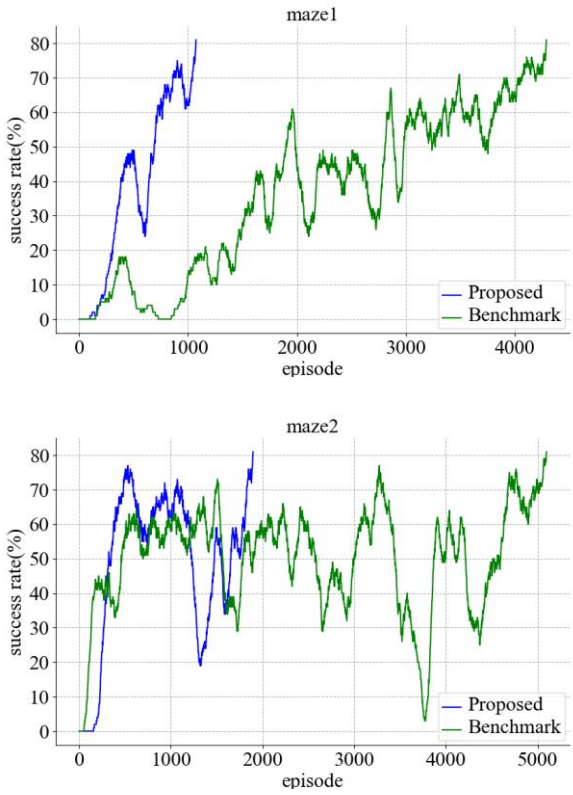

Figure 11. Success rate of proposed motion planner and benchmark.

Trajectories of the vehicle in these two tested environments in one episode are shown as the following picture:

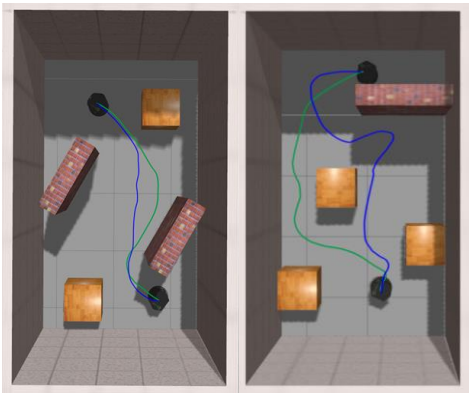

Figure 12. Trajectories in *maze1*(left) and *maze2*(right)

In Figure 12, the trajectories of proposed algorithm are *blue* and the benchmark are *green*. In *maze1*, proposed motion planner has even taken the path that is optimal.

VI. Conclusion

We proposed a sample-efficient motion planner using RGB image as visual input in mapless navigation. It decoupled the visual feature module from DRL network, which can improve the sample efficiency and can easily migrate to the new environment. Two state-of-the art deep reinforcement learning algorithms were compared in end-to-end network and the better one was chosen as the benchmark. The proposed motion planner performed much better than the benchmark in the built navigation environment. It only used 1/4 samples of the benchmark in *maze1* to reach the success rate of 80%. The built environments and source code are released for future research.

References

[1] C. Cadena *et al.*, "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age," *IEEE Trans. Robot.*, vol. 32, no. 6, pp. 1309–1332, Dec. 2016.

[2] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, "ORB-SLAM: A Versatile and Accurate Monocular SLAM System," *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1147–1163, Oct. 2015.

[3] P. Anderson *et al.*, "On Evaluation of Embodied Navigation Agents," *ArXiv180706757 Cs*, Jul. 2018.

[4] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 2018.

[5] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[6] D. Silver *et al.*, "Mastering the game of Go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, Oct. 2017.

[7] T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel, and S. Levine, "Composable Deep Reinforcement Learning for Robotic Manipulation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018, pp. 6244–6251.

[8] D. J. Hemanth and V. V. Estrela, *Deep Learning for Image Processing Applications*. IOS Press, 2017.

[9] K. Yu, C. Dong, L. Lin, and C. Change Loy, "Crafting a Toolchain for Image Restoration by Deep Reinforcement Learning," presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018, pp. 2443–2452.

[10] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Mach. Learn.*, vol. 8, no. 3, pp. 279–292, May 1992.

[11] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-Learning," in *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[12] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling Network Architectures for Deep Reinforcement Learning," *ArXiv151106581 Cs*, Nov. 2015.

[13] M. Hessel *et al.*, "Rainbow: Combining Improvements in Deep Reinforcement Learning," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[14] T. P. Lillicrap *et al.*, "Continuous control with deep reinforcement learning," *ArXiv150902971 Cs Stat*, Sep. 2015.

[15] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic Policy Gradient Algorithms," p. 9.

[16] V. R. Konda and J. N. Tsitsiklis, "Actor-Critic Algorithms," in *Advances in Neural Information Processing Systems 12*, S. A. Solla, T. K. Leen, and K. Müller, Eds. MIT Press, 2000, pp. 1008–1014.

[17] S. Gu, T. Lillicrap, I. Sutskever, and S. Levine, "Continuous Deep Q-Learning with Model-based Acceleration," p. 10.

[18] V. Mnih *et al.*, "Asynchronous Methods for Deep Reinforcement Learning," p. 10.

[19] J. Schulman, "Trust Region Policy Optimization," p. 9.

[20] N. Heess *et al.*, "Emergence of Locomotion Behaviours in Rich Environments," *ArXiv170702286 Cs*, Jul. 2017.

[21] P. Mirowski *et al.*, "Learning to Navigate in Cities Without a Map," *ArXiv180400168 Cs*, Mar. 2018.

[22] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Vancouver, BC, 2017, pp. 31–36.

[23] L. Tai, S. Li, and M. Liu, "A deep-network solution towards model-less obstacle avoidance," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 2759–2764.

[24] P. Mirowski *et al.*, "Learning to Navigate in Complex Environments," *ArXiv161103673 Cs*, Nov. 2016.

[25] A. Y. Ng and S. J. Russell, "Algorithms for Inverse Reinforcement Learning," in *Proceedings of the Seventeenth International Conference on Machine Learning*, San Francisco, CA, USA, 2000, pp. 663–670.

[26] J. Ho and S. Ermon, "Generative Adversarial Imitation Learning," in *Advances in Neural Information Processing Systems 29*, D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, Eds. Curran Associates, Inc., 2016, pp. 4565–4573.

[27] H. Kretzschmar, M. Spies, C. Sprunk, and W. Burgard, "Socially compliant mobile robot navigation via inverse reinforcement learning," *Int. J. Robot. Res.*, vol. 35, no. 11, pp. 1289–1307, Sep. 2016.

[28] L. Tai, J. Zhang, M. Liu, and W. Burgard, "Socially Compliant Navigation through Raw Depth Inputs with Generative Adversarial Imitation Learning," *ArXiv171002543 Cs*, Oct. 2017.

[29] M. Jaderberg *et al.*, "Reinforcement Learning with Unsupervised Auxiliary Tasks," *ArXiv161105397 Cs*, Nov. 2016.

[30] O. Zhelo, J. Zhang, L. Tai, M. Liu, and W. Burgard, "Curiosity-driven Exploration for Mapless Navigation with Deep Reinforcement Learning," *ArXiv180400456 Cs*, Apr. 2018.

[31] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-Driven Exploration by Self-Supervised Prediction," presented at the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2017, pp. 16–17.

[32] Y. Zhu *et al.*, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 3357–3364.

[33] D. Cireşan, U. Meier, and J. Schmidhuber, "Multi-column Deep Neural Networks for Image Classification," *ArXiv12022745 Cs*, Feb. 2012.

[34] D. P. Kingma and M. Welling, "Auto-Encoding Variational Bayes," *ArXiv13126114 Cs Stat*, Dec. 2013.

[35] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization," *ArXiv14126980 Cs*, Dec. 2014.

[36] G. Brockman *et al.*, "OpenAI Gym," *ArXiv160601540 Cs*, Jun. 2016.

[37] M. Quigley *et al.*, "ROS: an open-source Robot Operating System," p. 6.