Mobile Robot Navigation and Obstacles Avoidance based on Policy Gradient

Shuai Guo
dept. College of Computer Science and Technology
Zhejiang University
Hangzhou, China
gs98@zju.edu.cn

Hongliang He
dept. College of Computer Science and Technology
Zhejiang University
Hangzhou, China
3160103176@zju.edu.cn

***Abstract*—Past robotics often used more traditional algorithms and control theory. Over the past few years, however, deep RL has shown amazing performance in video games due to the massive increase in computing power. This paper explores the most commonly used deep reinforcement learning method, Policy Gradient, in the control of mobile robots, confirming the feasibility of policy gradient in simple robot skills learning. The framework let the agent learn in a custom simulation environment by the REINFORCE algorithm and eventually achieves a relatively satisfactory result in moving and obstacle avoidance tasks.**

***Index Terms*—robotics, reinforcement learning, policy gradient**

I. Introduction

In the past few years, robots have become increasingly popular with humans for their outstanding abilities to perform their tasks. A wide variety of robots have penetrated into all walks of life, they can perform complex operations, imitate humans, and even surpass humans in some areas. The realization of these goals is based on the skills of the robot, which can be defined as the abilities to complete a task, make an action or make a decision. The skill learning of the robot is very important. Only by learning the robot skills, the robot can grasp the corresponding ability, and then help the human to achieve a certain goal.

At present, the learning of robot skills has achieved remarkable results. In the field of mobile robots, For example, the autonomous navigation, obstacle avoidance, take-off and hover skills of flying robot and the small soccer robot's reaching goals, obstacle avoidance and shooting skills. Regarding decision-making robots, such as in 2016, Alpha Go [1] defeated the best Go players in the world, and then Alpha Go Zero completely surpassed Alpha Go with a 100:0 record. Behind these skills, the quality of an algorithm directly affects the effect of skill learning. This paper will focus on the research of mobile robot navigation skills.

Navigation technology is the foundation and core of mobile robot technology. It allows the robot to autonomously follow the internal predetermined information, or according to the sensor to obtain the external environment for guidance, then plan a path suitable for the robot to move in the environment. In the past, heuristic search was generally used to make the path planning of robots and made it more in line with human thinking. But the problem is that the search is too time-consuming, space-consuming, and requires human thinking to deal with all possible conflicts. In recent years, Machine Learning (ML) has developed rapidly. The use of ML to achieve navigation planning has gradually become the mainstream, especially the Reinforcement Learning (RL) methods, which can better adapt to the environment and simplify work, and thus more versatile.

This paper is devoted to learning the navigation and obstacle avoidance skills of mobile robots based on the application of Policy Gradient (PG) methods, which is a kind of Deep Reinforcement Learning (deep RL). We imitate the RoboCup Small Soccer Robot Competition to built a simple simulation environment and tested on it.

The rest of the paper is structured as follows: We introduce the related work in section 2. In section 3 we will detail the PG methods, architecture and the implementation we have proposed. The section 4 illustrates the experimental simulation environment and experimental results. Finally, the conclusion is drawn and some future works are presented in section 5.

II. Related work

The inevitable problems that robots will face are real-time path planning and trajectory planning during the movement of mobile robots. At present, there are many mature algorithms that has been applied to the real-time path planning of mobile robots, such as A* algorithm [2], Artificial Potential Field method [3], Rapidly Exploring Random Tree (RRT) algorithm and so on. Later, a large number of new algorithms and combinations and variants of existing algorithms have emerged. For example, RRT* algorithm, improved iterative RRT with path smoothing algorithm [4], genetic algorithm [5], etc. These algorithms are based on human perception of the environment and a comprehensive consideration of the problems and treatments that mobile robots may meet and need during the movement. They all have clear advantages and disadvantages, so we need flexible choice and application.

The above methods can be called traditional methods and they need to be selected according to a specific environment, which increases the workload of developers. And navigation planning usually needs to combine path planning and trajectory planning, which limits the performance of the algorithm

and thus has strong limitations. The application of the RL algorithm will break this mode, allowing the robot to adapt to the environment and learning strategies autonomously, which not only simplifies the work of developer, but also has stronger applicability and versatility.

Taking the navigation and obstacle avoidance of small soccer robots as an example. The soccer robot is a kind of mobile robots and it is based on navigation planning and then completes a series of complicated actions. In paper [6], the author trained the skills of going to ball, aiming and shooting, and applied the Deep Deterministic Policy Gradient (DDPG) algorithm, which produced significant results, but we also expect the mobile robot to have obstacle avoidance skill. In addition to deep RL algorithms, other ML methods can also achieve expectations well, such as the application of Support Vector Machine (SVM) and Neural Network (NN) in the paper [7] to achieve Direct Kick, Pass to Opposite Attacker and other drills, but may need to depend on the data set. Therefore, our group is more inclined to use deep RL methods to complete our skills learning on mobile robots.

III. Approach

We proposed a fundamental RL model to achieve the task of reaching goals and obstacle avoidance. It uses a parametric neural network to approximate the stochastic policy $\pi_\theta(a|s)$ and **Monte Carlo Policy Gradient** (a.k.a. **REINFORCE**) algorithm to update the parameters of the policy net. The reward function $R(s', s, a)$ is also well defined so that the agent can perform well in the environment.

A. State, Action and Reward

In an end-to-end deep RL settings, the input state is usually the raw sensory data (e.g., the raw image pixels in [8]). In that configuration, you do not need to know the prior knowledge (dynamics) of the environment, just get raw data from the sensor. But it introduces a lot of redundant information, making the neural network larger. Since we build the environment from scratch, it is assumed that the information about the position, angle, and so on of the agent and the target has been extracted from the sensory data. We take them directly as the states. This reduces the dimensions of the neural network. Our major aim is to demonstrate the practicality of the gradient algorithm in these tasks.

State. The state is a 4-dimensional vector $[dist_t, angle_t, dist_o, angle_o]^\top \in \mathbb{R}^4$ with each element meaning:

1) $dist_t \in \mathbb{R}$: The distance from the robot to the target.
2) $angle_t \in [-\pi, \pi)$: The angle at which the robot needs to rotate in order to point to the target.
3) $dist_o \in \mathbb{R}$: The distance from the robot to the nearest obstacle.
4) $angle_o \in [-\pi, \pi)$: The angle at which the robot needs to rotate in order to point to the nearest obstacle.

Action. Now consider the action. In reality, the action of the robot is in a continuous space. But that makes it almost impossible to sample an action from a policy net. Because the policy net should assign a probability of each action. So we choose to discretize the action. To make the action space smaller, we fix the robot's forward speed so that it has only three actions - $\{forward, left, right\}$.

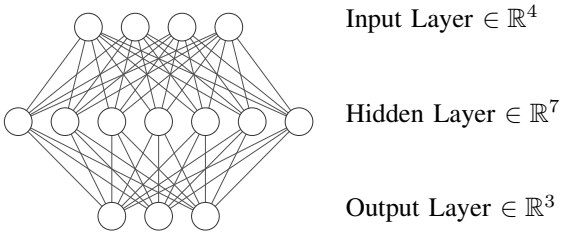


Fig. 1. The NN architecture.

Reward. The design of the reward function often determines the performance of the robot. We propose a simple but intuitive reward that is dependent on the state vector - the closer the robot is to the target, or the more toward the target, the greater the reward. As for obstacles, the farther away from the obstacle, and the farther the angle is from the obstacle, the smaller the reward value (the greater the penalty). Combining this reward and punishment in some way becomes a total reward. Express them with a math expression:

$$r \propto \frac{dist_o \cdot \cos(angle_t)}{dist_t \cdot \cos(angle_o)}$$

B. Network Architecture

The neural network acts as the policy $\pi(a|s)$ so the input is state $S_t \in \mathbb{R}^4$. It has only one hidden layer consisted of 7 neurons with *ReLU* as the activation function. The output layer has 3 neurons representing the log-probability of each actions so we apply a *Softmax* to get the probability distribution over action space. The corresponding action is sampled from this distribution to guide the robot. The overall structure of the MLP is shown in Fig 1.

C. Algorithm (Policy Gradient)

The model use *policy gradient* to update the parameters of the policy network. Policy Gradients are a special case of a more general *score function gradient estimator*. The general case is that when we have an expression of the form $E_{x\sim p(x|\theta)}[f(x)]$. Here in RL, the $f(x)$ will be the performance $J(\theta)$ and the $p(x|\theta)$ will be the policy network $\pi_\theta(a|s)$. Then we can use gradient ascend to optimize the parameters to maximize the objective function.

$$\nabla_\theta E_x[f(x)] = E_x[f(x)\nabla_\theta \log p(x)]$$

There are quite many variants of policy gradient, each of which has a different way to optimize the policy net. All of these methods are compatible with a general form of policy gradient methods proposed in the GAE (general advantage estimation) paper [9]. The paper claimed that there are several

related expressions for the policy gradient, which have the form

$$g = \mathbb{E}\left[\sum_{t=0}^{\infty} \Psi_t \nabla_\theta \log \pi_\theta(a_t|s_t)\right]$$

where Ψ_t can be some criterion about the performance of the agent. However, we use the naive REINFORCE (Monte-Carlo policy gradient) to achieve our tasks, in which $\Psi_t = Q^\pi(s_t, a_t)$. So the REINFORCE method can be expressed by

$$\begin{aligned}\nabla_\theta J(\theta) &= \mathbb{E}_\pi[Q^\pi(s,a)\nabla_\theta \ln \pi_\theta(a|s)] \\ &= \mathbb{E}_\pi[G_t \nabla_\theta \ln \pi_\theta(A_t|S_t)]\end{aligned}$$

Then the algorithm is easily implemented as:

1) Initialize the policy parameter θ at random.
2) Generate one trajectory on policy π_θ: $, A_1, R_2, S_2, A_2, \ldots, S_T$
3) For $t = 1, 2, \ldots, T$
   a) Calculate the return $G_t = R_{t+1} + \cdots + R_T$
   b) Update policy parameters:

$$\theta \leftarrow \theta + \alpha\gamma^t G_t \nabla_\theta \ln \pi_\theta(A_t|S_t)$$

We also adopted the widely used trick - subtracting a *baseline* value from the return G_t to reduce the variance of gradient estimation while keeping the bias unchanged [10]. The baseline can takes several different forms. For example, it is useful to do a normalization over returns G_t (e.g. subtract its mean, divide by its standard deviation) before we use them to update the parameters. In this way were always encouraging and discouraging roughly half of the performed actions. Mathematically these tricks can be interpreted as a way of controlling the variance of the policy gradient estimator [9].

IV. Experiment and Results

A. Simulation Environment

Since there is no existing simulation environment for small soccer robots (such as the famous **Gym** by *OpenAI*), we wrote a simple simulation environment ourselves using **pyglet**.

The environment involves an agent whose velocity v and angular velocity ω can be set. The user can set the position of the target (the star) by *left-clicking* the mouse anywhere in the field. You can also add obstacles (enemy robots) by *right-clicking* the mouse anywhere and *click the middle mouse button* to clear the obstacle under the cursor. The trajectory of the agent is also tracked so that you can better understand how the robot moves. The overview of the environment is shown is Fig 2. The hyperparameters we use are listed in Table I.

TABLE I
Hyperparameters

Name	Value
learning rate	0.02
batch size	10
gamma	0.99

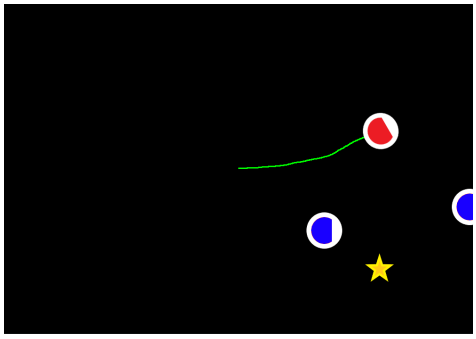


Fig. 2. A simple simulation environment.

B. Results

The experimental results are divided into two parts. At first we trained the skill of reaching target points of mobile robot without obstacles. This seems to be a rather simple task and we can remove the variables related to obstacles in the state of the robot and make it a 2-d vector. We used the REINFORCE algorithm mentioned above and it converged in a very short time. While training, every time the robot had performed 100 tasks, we would count the number of times it successfully reached the target point. The rate of success is shown in Fig 3. During the test, we let the target point randomly change position in the entire field, and the robot could always reach it no matter where it is. The test result is shown in Fig 4. We also tried the deep Q network algorithm, but it converged with a slow speed and often converged to local minima.

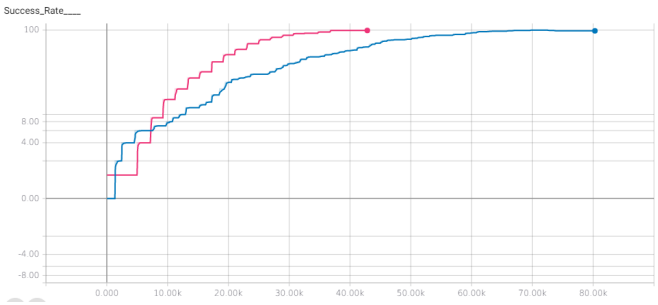


Fig. 3. Success rate of reaching target point. The robot can reach the end point steadily after training. The color difference: LR=0.02 train for about 30min; LR=0.01 train for about 50min

Then we added obstacles during the training process, making the state of the robot a 4-d vector mentioned in section 3. We set up 1 or 2 obstacles during training and let the obstacles position change randomly. Then we got a model and trained it again with more obstacles. More obstacles can be set during the test and the obstacle avoidance results are shown in Fig 5. Because the design of robot states and reward are not optimal, it is still possible to hit an obstacle sometimes and may take extra distance to reach target point. And we found that if the robot needs to take a huge arc to reach the target, then it is more likely to fail, as shown in Fig 5(d). It is expected that engineering improvements will be made in

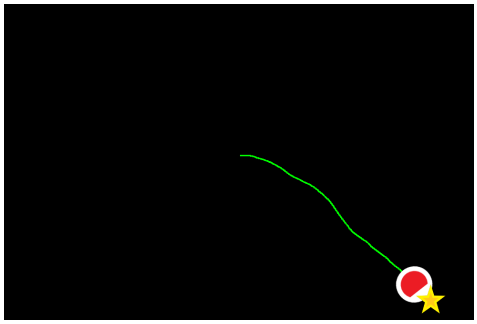


Fig. 4. Test result of reaching target point.

the later work through the optimization of algorithms, neural network structures, robot states and reward settings.

V. Conclusion and Future Work

In this work, we proposed a fundamental RL framework for robot navigation and obstacle avoidance. The experimental results prove the feasibility of this RL method. It is totally different from classical explicit navigation planning algorithms - it makes robots learning how to move from interaction with the environment and thus obtains a better generalization than hand-crafted algorithms when facing a complex environment.

If we also learn a value function during training, it can assist the policy update, and that's exactly what the **Actor-Critic** method does. It uses a *Critic* to maintain a value function and help to *Actor* to update the policy. It is an improvement over REINFORCE because it reduces the variation of parameters' updating a lot.

Both REINFORCE and the vanilla version of actor-critic method are *on-policy*: training samples are collected according to the target policy the very same policy that we try to optimize for. *Off-policy* methods, however, result in better exploration by using a known *behavior* policy and updating the *target* policy. The trained target policy can approach a deterministic policy [10] so it can approximate an optimal policy.

To accelerate the training process, it is preferred to run multiple actors to collect trajectories. That's what Asynchronous Advantage Actor-Critic (**A3C**) does [11]. Besides more efficient algorithms, it is also vital to design a proper reward function to guide the robot to our desired actions.

Acknowledgment

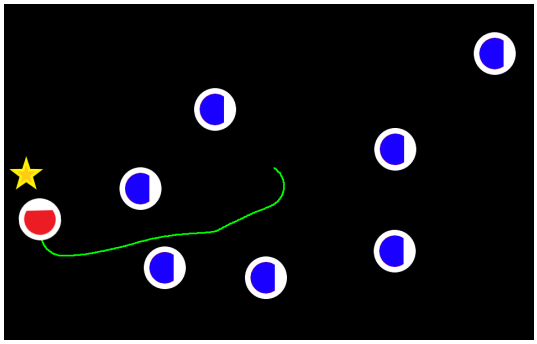
Thanks to Xiong Rong, Shen Zebang, Qian Hui and Zhou Chunlin.

References

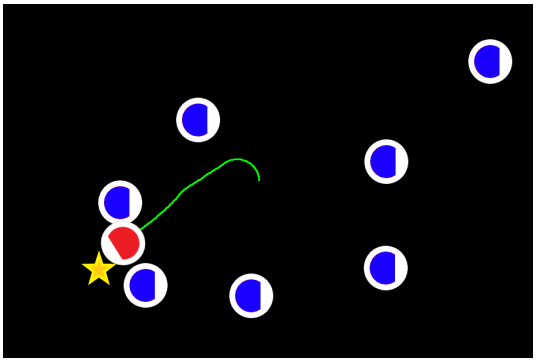
[1] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[2] T. L. P. U. Kayrak, "Algorithm for path planning of robotic soccer player," 2011.

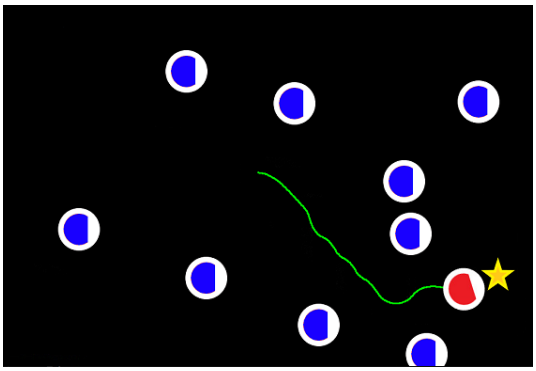
[3] K. F. Uyanik, "A study on artificial potential fields," *Middle East Technical University, Ankara, Turkey*, vol. 2, no. 1, 2011.



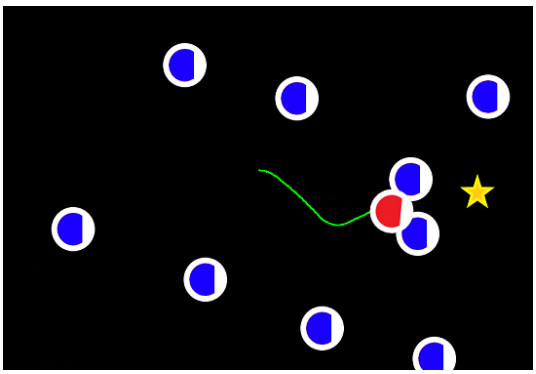
(a) scene 1, success



(b) scene 1, success



(c) scene 2, success



(d) scene 2, fail

Fig. 5. Test result of reaching target point with obstacles.

[4] R. H. Abiyev, N. Akkaya, E. Aytac, I. Günsel, and A. Çağman, "Improved path-finding algorithm for robot soccers," *Journal of Automation and Control Engineering*, vol. 3, no. 5, 2015.

[5] R. T. U. Albab, I. K. Wibowo, and D. K. Basuki, "Path planning for mobile robot soccer using genetic algorithm," in *2017 International Electronics Symposium on Engineering Technology and Applications (IES-ETA)*. IEEE, 2017, pp. 276–280.

[6] D. Schwab, Y. Zhu, and M. M. Veloso, "Learning skills for small size league robocup," 2018.

[7] C. Quintero, S. Rodríguez, K. Pérez, J. López, E. Rojas, and J. Calderón, "Learning soccer drills for the small size league of robocup," in *Robot Soccer World Cup*. Springer, 2014, pp. 395–406.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv: Learning*, 2013.

[9] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[10] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[11] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.