A Method to Effectively Detect Vulnerabilities on Path Planning of VIN

Jingjing Liu[1], Wenjia Niu[1(✉)], Jiqiang Liu[1(✉)], Jia Zhao[1], Tong Chen[1], Yinqi Yang[1], Yingxiao Xiang[1], and Lei Han[2]

[1] Beijing Key Laboratory of Security and Privacy in Intelligent Transportation, Beijing Jiaotong University, Beijing 100044, China
{niuwj,jqliu}@bjtu.edu.cn
[2] Science and Technology on Information Assurance Laboratory, Beijing 100072, China

Abstract. Reinforcement Learning has been used on path planning for a long time, which is thought to be very effective, especially the Value Iteration Networks (VIN) with strong generalization ability. In this paper, we analyze the path planning of VIN and propose a method that can effectively find vulnerable points in VIN. We build a 2D navigation task to test our method. The experiment for interfering VIN is conducted for the first time. The experimental results show that our method has good performance on finding vulnerabilities and could automatically adding obstacles to obstruct VIN path planning.

Keywords: Path planning · Reinforcement learning · VIN
Vulnerable points

1 Introduction

Path planning for robot mainly solves three problems. First, it must ensure that the robot can move from the initial point to the target point. Second, it should allow the robot to bypass the obstacles with a certain method. Third, it tries to optimize the robot running trajectory in the completion of the above tasks on the premise. In general, path planning approach has two different types, the global planning and the local path planning. The global planning method usually can find the optimal solution. The available planning algorithms for this class are framework space method [6], free space method [7] and grid method [8]. But it needs to know the accurate information of the environment in advance, and the calculation is very large. The accuracy of path planning depends on the accuracy of obtaining the environmental information. The local path planning requires only the robot's near obstacle information, so that the robot has good ability to avoid collision. The commonly used local path planning methods are template matching [9] and artificial potential fields [10]. The template matching method is very easy to achieve, but the fatal flaw of this method is to rely on the past experience of the robot, and if there is not enough path template

S. Qing et al. (Eds.): ICICS 2017, LNCS 10631, pp. 374–384, 2018.
https://doi.org/10.1007/978-3-319-89500-0_33

in the case library, it is impossible to find the path that matches the current state. The artificial potential fields method is much flexible to control. However, this method is easy to fall into local optimum, resulting in motion deadlock. Nowadays, researchers start to put their attention on Reinforcement Learning (RL), a typical machine learning approach acquiring knowledge in the action-evaluation environment, for better solutions.

Traditional RL methods are Policy Iteration (PI) [11], Value Iteration (VI) [12], Monte-Carlo Method (MC) [13], Temporal-Difference Method (TD) [14] and Q-learning [1]. However, most traditional methods perform relatively poor when encounter different scenes from the previous training set. In contrast, deep RL, which incorporates the advantages of neural networks, has better performance and accuracy. Deep RL has been considered one of the major human learning patterns all the time. It is widely used in robot automatic control, artificial intelligence (AI) for computer games and optimization of market strategy. Deep RL is especially good at controlling an individual who can act autonomously in an environment and constantly improve individual behavior through interaction with the environment. Well known deep RL works are Deep Q-Learning (DQN) [15] and Value Iteration Networks (VIN) [5].

Among them, VIN has better generalization ability [5]. There is a kind of special value iteration in VIN, not only need to use the neural network to learn a direct mapping from the state to the decision, but also can embed the traditional planning algorithm into neural network. Thus, the neural network learns how to make long-term planning in the current environment, and uses long-term planning to assist the neural network in making better decisions. That is, VIN learns to plan and make decisions by observing current environment. Assuming VIN has already learned a model to predict rewards of future states, the flaw is when such predictions carry on, the errors of observations accumulate and can not be avoided because VIN perform VI over the whole environment. We can use this flaw to study how to obstruct VIN's performance. Apparently, adding some obstacles, which are unnoticed for human but can be detected by robots, to form a fake environment sample will probably make VIN be tricked and do wrong predictions.

Our main contribution is a method for detecting potential attack that could obstruct VIN. We study the typical and most recent works about path planning [2–4] and build our own method. We build a 2D navigation task demonstrate VIN and study how to add obstacles to effectively affect VIN's performance and propose a general method suitable for different kinds of environment. Our empirical results show that our method has great performance on automatically finding vulnerable points of VIN and thus obstructing navigation task.

2 Related Work

2.1 Reinforcement Learning

Traditional Reinforcement Learning methods are Policy Iteration, Value Iteration, Monte-Carlo Method, Temporal-Difference Method and Q-learning.

Policy Iteration and Value Iteration. The purpose of Policy Iteration [11] is to iteratively converge the value function in order to maximize the convergence of the policy. It is essentially the direct use of the Bellman Equation. So Policy Iteration is generally divided into two steps: (1) Policy Evaluation; (2) Policy Improvement. Value Iteration [12] is obtained by Bellman Optimal Equation. Thus, Policy Iteration uses Bellman Equation to update value, finally the convergence of the value is currently under policy value (so called to evaluate policy), to get new policy. And Value Iteration uses the bellman optimal equation to update the value and finally converges to the resulting value. Therefore, as long as the final convergence is concerned, then the optimal policy is obtained. This method is based on updating value, so it's called Value Iteration.

Monte-Carlo Method. Monte Carlo's idea is simple, that is, repeating tests to find the average. The Monte Carlo method is only for problems with stage episode [13]. For example, playing a game is step by step and will end. The Monte Carlo Method cares only for problems that can end quickly. The Monte Carlo method is extremely simple. But the disadvantages are also obvious. It takes a lot of time to do as many tests as possible, and to calculate at the end of each test. AlphaGo [16] uses the idea of Monte Carlo Method in Reinforcement Learning. It only uses the final winning or losing results to optimize each step.

Time Difference can be regarded as a combination of Dynamic Programming and Monte Carlo Method. Time Difference [14] also can directly learn from experience, does not require any dynamic model in the environment, but also a kind of Reinforcement Learning, can learn step by step, and does not need to wait for the end of the entire event, so there is no problem of calculating the peak value.

Q-learning is an important milestone in Reinforcement Learning. Watkins [1] takes the Reinforcement Learning method, whose evaluation function is based on Q value of state-action, as Q-learning. It is actually a change form of the Markov Decision Process (MDP). In Reinforcement Learning, the reinforcement function R and the state transfer function P are unknown, so the Q-learning uses iterative algorithm to approximate the optimal solution. Q-learning does not go forward along the path of the highest Q value at each iteration. The reason why the Q-learning effects is the updating of the Q matrix.

2.2 Deep Reinforcement Learning

Well known deep Reinforcement Learning works are Deep Q-Learning and Value Iteration Networks.

Deep Q-learning is the first deep enhancement learning algorithm proposed by Google DeepMind in 2013 [15] and further improved in 2015 [4]. DeepMind applies DQN to computer games like Atari, which is different from the previous practice, using video information as input and playing games with humans. This is the first introduction of the concept of deep Reinforcement Learning, and it's beginning to develop rapidly. In DQN, since the output value of the value

network is the Q value, if the target Q value can be constructed, the loss function can be obtained by the mean-square error (MSE). But for the value network, the input information is the state s, action a and feedback R. Therefore, how to calculate the target Q value is the key to the DQN, which is the problem that Reinforcement Learning can solve.

Value Iteration Networks. The biggest innovation of this work is that a Planing Module is added to the General Policy. The author believes that joining the motivation of this module is natural, because when solving a space problem, it is not simply solving the problem, but planning in this space. It aims at solving the problem of poor generalization ability in Reinforcement Learning. In order to solve this problem, a Learn to Plan module is proposed [5]. The innovations of VIN are mainly the following points: (1) the reward function and transfer function are also parameterized and can be derived; (2) a spatial assisted strategy is introduced to make policy more generalized; (3) the attention mechanism is introduced into the solution of the strategy; (4) the design of VI module is equivalent to CNN, and the BP algorithm can be used to update the network.

We find that each method has its own superiority. But, VIN is much better at planning in different sample from training set. Thus, we choose VIN as our experiment bases and we will show our analysis in Sect. 3.

3 Method

In this section, we will discuss how we build the method for obstructing VIN's performance. First, we analyze the path planning idea of VIN, and carry out tentative experiments to speculate the factors that might affect VIN path planning. After that, we propose the method of obstructing VIN according to these factors. In Sect. 4, we will use a large number of experiments to verify this method.

3.1 Analysis of VIN

Figure 1 is a sample of VIN path planning, from which we can see that VIN does have the ability to reach the destination but not along the theoretical shortest

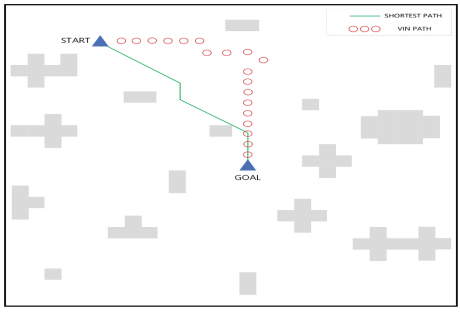


Fig. 1. Diagram of VIN Planning in a map

path. VIN defines a MDP space, M, which consists of a series of tuples, that is, some columns' states, actions, transfers, rewards, data tuples, and M that determines our final strategy. Then, a strategy obtained by data M from this MDP space policy is not a good generalization strategy because the policy is limited to this data space. Therefore, the authors assume that the unknown data space M′ is obtained, and that the optimal plan in this space contains important information about the optimization strategies in the M space. In fact, the assumption is that M is just a sampling trajectory of a part of the MDP space, adding M′ is as a complement to the trajectory in this space.

VIN believes that the reward R′ and the transfer probability P′ in the data space M′ also depend on the observations in the M data space. After doing such hypothesis, it introduces two functions, fR and fP, respectively, for parameterized R′ and P′. fR is the reward mapping: when inputting state, it will calculate the corresponding reward value. For example, the reward value is higher in the state near the target, and the value is lower when the state position is near the obstacle.

3.2 Problem Definition

As we said before, we aim to find those vulnerable points and try to add some obstacles at there to effectively affect VIN's performance. We want to build a method to calculate a kind of value for each point over the entire environment depending on some potential rules. Thus, we can directly derive vulnerable points from our method without exhaustive experiment for every space. So, the problem is transformed into acquiring reasonable formulas for solving the set of interference points.

Based on analysis of VIN and the correct of tentative experimental trials, we summarize three rules that might effectively obstructing VIN: (1) The farther away from the VIN planning path, the less the disturbance to the path. (2) It is often the most successful to add obstacles around the turning points in the path. (3) The closer the adding obstacle position is to the destination, the less likely it is to change the path.

Assuming that we know the entire environment (including obstacles, starting point and destination), and we also know that the robot is using the VIN method to find the path, it is easy to get the VIN planning path and the theoretical shortest path.

Let the points on the VIN path be X set: $\{x_1, x_2, \cdots, x_n\}$, the existed obstacles be B set: $\{b_1, b_2, \cdots, b_n\}$, any other available points be Y set: $\{y_1, y_2, \cdots, y_n\}$. The method only considers the points in Y. We want to calculate a value v considering three rules above for each point of Y, and then sort the values to pick up most wanted points, that is S set: $\{y | v_{yk} \in \max_i V, y \in Y\}$, $V = \{v_{y_1}, v_{y_2}, \cdots, v_{y_k}\}$.

3.3 Method Building

Now, the problem is converted from solving S to solving V. In our method, the value v consists of three parts: v_1, v_2, v_3, which refers to the three rules.

Rule 1: The farther away from the VIN planning path, the less the disturbance to the path.

The formula is:

$$v_{1y_k} = \omega_1 \min\{d_1 | d_1 = \sqrt{(x_r - y_{kr})^2 + (x_c - y_{kc})^2}, \\ (x_r, x_c) = x \in X, (y_{kr}, y_{kc}) = y_k \in Y\} \tag{1}$$

where (x_r, x_c) is the coordinate of x, (y_{kr}, y_{kc}) is the coordinate of y_k, ω_1 is the weight of v_1. The formula considers the Euclidean distance from y_k to VIN path, and use the weight ω_1 to control the attenuation of v_1.

Rule 2: It is often the most successful to add obstacles around the turning points in the path.

First, to get the turning points, we can calculate the gradient of adjacent two points in path and compare adjacent gradients. The point at which the gradient varies is the turning point. Thus, we can get the turning points set T: $\{t_1, t_1, ..., t_n\}$.

The formula is:

$$v_{2y_k} = \omega_2 \min\{d_2 | d_2 = \max(|t_r - y_{kr}|, |t_c - y_{kc}|), \\ (t_r, t_c) = t \in T, (y_{kr}, y_{kc}) = y_k \in Y\} \tag{2}$$

where (t_r, t_c) is the coordinate of t, (y_{kr}, y_{kc}) is the coordinate of y_k, ω_2 is the weight of v_2. The formula considers the Chebyshev distance from y_k to the nearest turning point, and use the weight ω_2 to control the attenuation of v_2.

Rule 3: The closer the adding obstacle position is to the destination, the less likely it is to change the path.

The formula is:

$$v_{3y_k} = \omega_3 \max(|x_{nr} - y_{kr}|, |x_{nc} - y_{kc}|), \\ (x_{nr}, x_{nc}) = x_n, (y_{kr}, y_{kc}) = y_k \in Y \tag{3}$$

where (x_{nr}, x_{nc}) is the coordinate of x_n, the destination of the path, (y_{kr}, y_{kc}) is the coordinate of y_k, ω_3 is the weight of v_3. The formula considers the Chebyshev distance from y_k to the destination, and use the weight ω_3 to control the attenuation of v_3.

For the points in X and B, the value is 0. That is: $v_x = 0, v_b = 0$.

So, the problem now is how to use ω to control the attenuation of each formula. We define ω_3 as constant, because v_3 is the least important among v, that is Rule 3 is not that important compared with the other two. And we define $\omega_i (i = 1, 2)$ as follows:

$$\omega_i = \exp(-\frac{d_i^2}{2\theta_i^2}), i = 1, 2 \tag{4}$$

where the $\omega_i (i = 1, 2)$ is exponential decay, and when it multiplies $d_i (i = 1, 2)$, $v_i (i = 1, 2)$ will grows within a certain range, then decays exponentially. And $\theta_i (i = 1, 2)$ is the parameter to control the peak point and the rate of the decay.

So, the value of each y is:

$$v_{y_k} = \sum_{i=1}^{3} v_{iy_k} \tag{5}$$

And our target $S = \{s_1, s_2, \cdots, s_m\} = \{y | \max_m\{v_{y_k}\}\}$.

Our algorithm based on the method above is:

Algorithm 1. Method for Detecting Vulnerabilities in VIN Path Planning

Input: the training set: D_{map}, the test set: T_{map}
Output: labeled training set: T_{result}

1. Data preprocessing: $T_{map} \rightarrow T_{experiment}$
2. VIN training model: $D_{map} \rightarrow VIN\ Model$
for T_{map}
  for $T_{map(i)}$
    3. Calculate value: $T_{map(i)}(k) \xrightarrow{Method} (T_{map(i)}(k), value_i(k))$
    4. Sort and get the available set:$\max_5(value_i(k)) \rightarrow S = (s_1, s_2, s_3, s_4, s_5)$
    5. Juge and get the final result:
    randomly select $n, (n = 1, 2, 3)$ points from S to add the obstacle
    load $VIN\ Model$
    if $(Path_{VIN}^{after} - Path_{VIN}^{before} \geq 3)$ or $(Time_{VIN}^{after} - Time_{VIN}^{before} \geq 0.00001)$
      $T_{result}(i) = (T_{map(i)}, n, ' sucess')$
    else
      $T_{result}(i) = (T_{map(i)}, n, ' fail')$
output T_{result}

4 Experiment

Our goal in this section is mainly to figure out the following question: Can our method work effectively on adding obstacles automatically to obstruct VIN's performance?

The basis of our experiment is VIN. We apply the available source code [18] provided by VIN's author to train VIN Model. We adopt 28×28 Grid-World domain as the domain input and output.

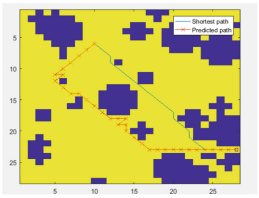
First, we generate $20,000$ maps with random starting and ending points and shortest path. We use $10,000$ maps to train the VIN Model. Then, we preprocess the other $10,000$ domains as the testing set: getting rid of those can't reach the ending point by VIN. And pick $5,000$ domains from them to be the testing set.

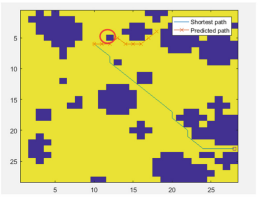
Second, we carry out the experiment on testing set with the method proposed in this paper, and get the label (success or fail) of each domain.

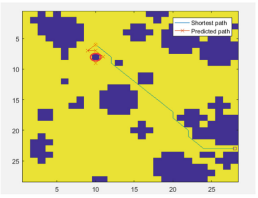
At last, we use the method for random addition of noise as a comparison test. The result shows that our method does have superiority on finding vulnerabilities in VIN path planning and automatically adding obstacles.

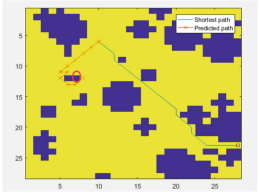
4.1 Single Map Obstructing

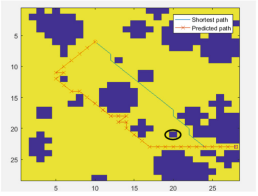
As the pictures of Fig. 2 show, our method does have ability to find vulnerabilities in VIN and thus interfere its performance. Fig. 2(a) is a sample of testing set, and Fig. 2(b) to (f) are the top 5 vulnerable points picked by our method. Four of them can effectively interfere VIN's path.


(a) Sample of Testing Set


(b) Available Obstacle 1


(c) Available Obstacle 2


(d) Available Obstacle 3


(e) Available Obstacle 4

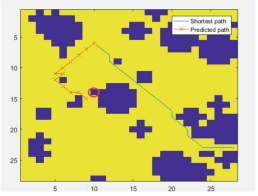

(f) Available Obstacle 5

Fig. 2. Experiment sample

Table 1. success rate comparison

Obstacle number	Our method success rate	Random method success rate
1	34.54%	4.34%
2	37.06%	8.46%
3	66.48%	11.24%

Since there is no such research focusing on interfering VIN, we choose random method as the comparison experiment. Figure 3 is an accuracy comparison of our method and random method. The random method we use is to randomly select points in some fixed areas, like areas around the turning points. As you can see, the trend of accuracy is growing as the number of obstacles increases. But, our method is much better at first and the increment is more obvious. Table 1 shows the value from our experiment by testing 5,000 domains, which is the same value as Fig. 3.

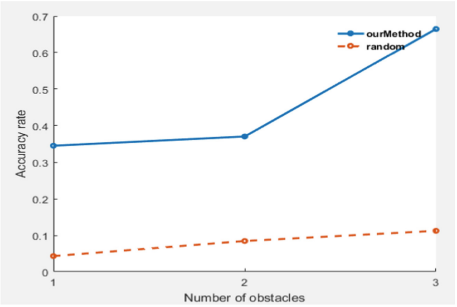
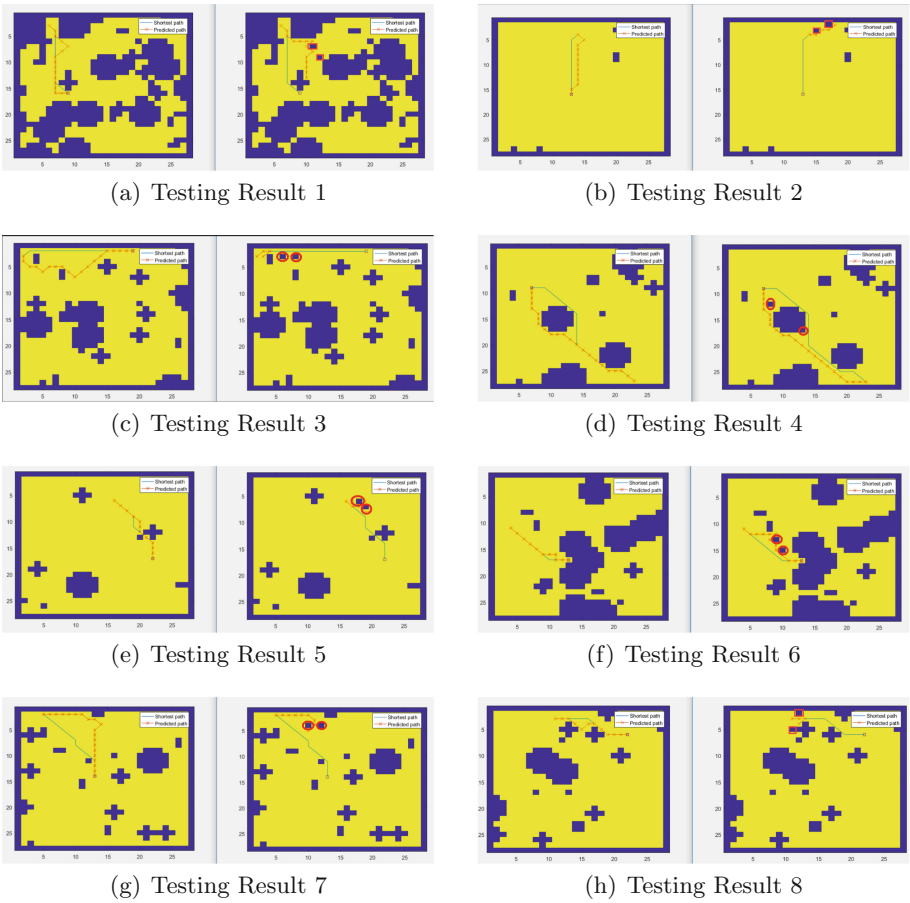


Fig. 3. Accuracy comparison



(a) Testing Result 1 (b) Testing Result 2

(c) Testing Result 3 (d) Testing Result 4

(e) Testing Result 5 (f) Testing Result 6

(g) Testing Result 7 (h) Testing Result 8

Fig. 4. Experiment sample

4.2 Batch Obstructing

This part uses the same method as above, but we show kinds of result in Fig. 4. As you can see, we pick up some well worked samples of adding two obstacles. These obstacles are randomly picked from the top 5 candidates. The results show that our method does have ability to successfully add obstacles in different domains. But you may wonder why the success rates in the Table 1 and Fig. 3 are not very high. That's because of the variety of testing set. For example, the starting and ending point in one sample can be very close, which means no matter what you do on this sample, you could hardly interfere VIN's planning.

5 Conclusion and Outlook

In this paper, we analyze the path planning methods of Reinforcement Learning, especially VIN and propose a method that can effectively find vulnerable points in path planning sample of VIN. We build a 2D navigation task to test our method and the result shows our method has great performance on finding vulnerabilities and automatically adding obstacles to obstruct VIN path planning.

Based on this paper, we believe that Reinforcement Learning methods have commonalities. If our method can work on VIN, it probably will work on other Reinforcement Learning methods. But we could further study methods for generating aggressive samples, like Generative Adversarial Networks (GAN) [17] and improve our method to a more general way.

Acknowledgments. This material is based upon work supported by the National Natural Science Foundation of China (Grant No. 61672092, No. 61502030), Science and Technology on Information Assurance Laboratory (No. 614200103011711), BM-IIE Project (No. BMK2017B02-2), and the Fundamental Research Funds for the Central Universities (Grant No. 2016JBM020, No. 2017RC016).

References

1. Watkins, C.J.C.H., Dayan, P.: Q-learning. Mach. Learn. **8**(3–4), 279–292 (1992)
2. Giusti, A., Guzzi, J., Dan, C.C., et al.: A machine learning approach to visual perception of forest trails for mobile robots. IEEE Robot. Autom. Lett. **1**(2), 661–667 (2016)
3. Levine, S., Finn, C., Darrell, T., et al.: End-to-end training of deep visuomotor policies. J. Mach. Learn. Res. **17**(39), 1–40 (2016)
4. Mnih, V., Kavukcuoglu, K., Silver, D., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
5. Tamar, A., Wu, Y., Thomas, G., et al.: Value iteration networks. In: Advances in Neural Information Processing Systems, pp. 2154–2162 (2016)
6. Hyndman, R.J., Koehler, A.B., Snyder, R.D., et al.: A state space framework for automatic forecasting using exponential smoothing methods. Int. J. Forecast. **18**(3), 439–454 (2002)
7. Brooks, R.A.: Solving the find-path problem by good representation of free space. IEEE Trans. Syst. Man Cybern. **2**, 190–197 (1983)

8. Zhu, Q.B., Zhang, Y.: An ant colony algorithm based on grid method for mobile robot path planning. Robot **27**(2), 132–136 (2005)
9. Terwilliger, T.C.: Automated main-chain model building by template matching and iterative fragment extension. Acta Crystallogr. Sect. D: Biol. Crystallogr. **59**(1), 38–44 (2003)
10. Warren, C.W.: Global path planning using artificial potential fields. In: Proceedings of 1989 IEEE International Conference on Robotics and Automation, pp. 316–321. IEEE (1989)
11. Lagoudakis, M.G., Parr, R.: Least-squares policy iteration. J. Mach. Learn. Res. **4**(Dec), 1107–1149 (2003)
12. Pineau, J., Gordon, G., Thrun, S.: Point-based value iteration: an anytime algorithm for POMDPs. In: IJCAI, vol. 3, pp. 1025–1032 (2003)
13. Dellaert, F., Fox, D., Burgard, W., et al.: Monte Carlo localization for mobile robots. In: Proceedings of 1999 IEEE International Conference on Robotics and Automation, vol. 2, pp. 1322–1328. IEEE (1999)
14. Tesauro, G.: Temporal difference learning and TD-Gammon. Commun. ACM **38**(3), 58–68 (1995)
15. Mnih, V., Kavukcuoglu, K., Silver, D., et al.: Playing Atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
16. Wang, F.Y., Zhang, J.J., Zheng, X., et al.: Where does AlphaGo go: from church-turing thesis to AlphaGo thesis and beyond. IEEE/CAA J. Automatica Sin. **3**(2), 113–120 (2016)
17. Goodfellow, I., Pouget-Abadie, J., Mirza, M., et al.: Generative adversarial nets. In: Advances in neural information processing systems, pp. 2672–2680 (2014)
18. UC Berkeley. https://github.com/avivt/VIN