Proceedings of CONF-MLA 2024 Workshop: Semantic Communication Based Complexity Scalable Image Transmission System for Resource Constrained Devices
DOI: 10.54254/2755-2721/83/2024GLG0073

Comparative Analysis of Value Iteration and Policy Iteration in Robotic Decision-Making

Yichen Qiu

Department of Mathematics, University College London, 25 Gordon Street, London, WC1H 0AY, UK

yichen.qiu.23@ucl.ac.uk

Abstract. This study focuses on Markov Decision Processes (MDPs) as a framework for decision-making in robotics. MDPs are crucial for enabling autonomous systems to operate in dynamic environments. MDPs enable the development of optimal strategies that balance immediate and future rewards, making them essential for intelligent robotic behavior. The purpose of this study is to evaluate the performance differences of value iteration and strategy iteration algorithms in different robot tasks. The study highlights the strengths and weaknesses of each algorithm. This comparative analysis uniquely addresses a gap in existing research by evaluating both Value Iteration and Policy Iteration side by side, offering critical insights into their respective performances across diverse robotic tasks. Results indicate that Policy Iteration converges faster and adapts better in complex environments, making it ideal for real-time applications, while Value Iteration is more efficient in smaller state spaces.These findings provide critical insights for selecting algorithms tailored to specific robotic applications, highlighting their role in advancing intelligent robotic systems.

Keywords: Markov Decision Process, Value Iteration, Policy Iteration.

1. Introduction

Markov Decision Processes (MDPs) serve as a fundamental framework in decision-making under uncertainty, particularly in robotics. The importance of MDPs lies in their ability to model complex environments and generate optimal policies, allowing robots to make informed decisions autonomously. As robotics continues to evolve, the selection of appropriate decision-making algorithms has become crucial for achieving intelligence and autonomy. MDPs, through algorithms such as Value Iteration and Policy Iteration, provide the necessary tools to address these challenges. Although significant advancements have been made in reinforcement learning and autonomous systems, most existing research primarily focuses on single algorithms, lacking comparative analysis between Value Iteration and Policy Iteration, which limits a comprehensive understanding of algorithm performance across different applications. Puterman provided a comprehensive exploration of MDPs focusing on discrete stochastic dynamic programming[1]. Leslie Pack Kaelbling and Tomás Lozano-Pérez talked about integrated task and motion planning in belief space, which helps robots make decisions in the face of uncertainty[2]. Richard S. Sutton introduced reinforcement learning principles and algorithms, emphasizing their applications in AI and robotics[3]. Two reserachers provided basic theories and techniques in order to explore policy iteration methods for reinforcement learning in continuous time

DOI: 10.54254/2755-2721/83/2024GLG0073

and space[4]. In 2024, a group of researchers used integrated reinforcement learning to provide a value iteration-based adaptive fuzzy backstepping optimal control for modular robot manipulators[5]. Three researchers introduced a theory of regularized MDPs, providing a theoretical framework for their application[6]. Dimitri Bertsekas wrote a book that covers the intersection of reinforcement learning and optimal control, focusing on the development of advanced algorithms for decision-making[7]. Mikko Lauri, David Hsu, and Joni Pajarinen did a survey that reviews the application of Partially Observable MDPs (POMDPs) in robotics, highlighting recent advancements and challenges[8]. Dimitri Bertsekas wrote a book in 2022 that includes abstract dynamic programming, expanding on the theoretical underpinnings of decision-making processes[9]. A group of researchers introduced the development of an AI that mastered the game of Go using deep neural networks and tree search[10]. A paper in 2016 introduced a deep reinforcement learning approach that achieved human-level control in various Atari games[11]. Deep reinforcement learning methods for motion planning and control in autonomous cars were examined in a survey, which included current developments and difficulties[12]. The existing research primarily focuses on either Value Iteration or Policy Iteration in isolation, but leave a gap in understanding the comparison between these algorithms in various robotic applications. This study compares the performance of Value Iteration and Policy Iteration algorithms in various robotic tasks, addressing this gap and providing a theoretical basis for selecting the appropriate algorithm. This paper will first introduce the fundamental principles of MDPs, then analyze the strengths and weaknesses of both algorithms, and finally present specific application examples to demonstrate their practical performance in robotic decision-making.

2. Background on MDP

Components of MDP: states(S), actions(A), transition probabilities(P), rewards(R), and discount factor(γ)[1].

States: A finite set that contains the various circumstances or configurations in the environment. These can include the robot's position, orientation, speed, and sensor readings, as well as the status of external factors such as obstacles or other agents in the environment. For example, in robotic navigation, states might represent the robot's location on a map grid.

Actions: A finite set of actions that are available to the decision-maker in every state. These actions could include moving forward, turning, or grasping an object. Each action taken in a state leads to a new state, depending on the dynamics of the environment and the robot's capabilities. For a mobile robot, actions might involve moving in different directions (e.g., forward, backward, left, right) or changing speed.

Transition probabilities: The probability of changing states after executing a particular action. This can be written as $p(s' \mid s, a)$, where s represents the current state, a is the action that was made, and s' denotes the resulting state. In robotics, transition probabilities are crucial for modeling uncertainty. Accurately modeling these probabilities allows the robot to make more reliable decisions by anticipating various possible outcomes and planning accordingly, thereby increasing its robustness in unpredictable environments.

Rewards: a function that determines the immediate gain or loss from doing an action in a specific state and assigns a numerical reward for each state-action pair or state. In robotic path planning, the reward might be high for reaching a target location and low (or negative) for hitting obstacles.

Discount factor: A number ranging from 0 to 1 used to discount future rewards, reflecting a preference for immediate rewards over those in the future. In long-term tasks such as navigation, a higher discount factor might be used to ensure the robot considers the long-term effects of its actions, such as designing a route that avoids obstacles far ahead. In contrast, in tasks requiring quick, short-term decisions, such as reactive obstacle avoidance, a lower discount factor might be preferred to prioritize immediate safety over distant goals.

The theoretical basis of MDP includes Markov properties, Bellman equations and dynamic programming, which not only provide a solid mathematical foundation for MDP, but also ensure the transparency and interpretability of the decision process. The Markov property is a stochastic process's

DOI: 10.54254/2755-2721/83/2024GLG0073

memoryless nature, which means that its future state is decided solely on its current state and not on the sequence of events that came before it. Bellman equations provide a recursive decomposition of the value of a decision problem, breaking it down into simpler subproblems. Dynamic programming is an optimisation technique that breaks down large problems into simpler subproblems in order to solve them. They ensure that decision processes are both efficient and interpretable, making them invaluable in fields like robotics, where clear, optimal decision-making is essential. This solid theoretical basis allows for the development of algorithms that are both practically useful in challenging real-world situations and mathematically sound.

Applications of MDPs in robotics: Robotic applications include navigation, tracking, autonomous driving, multi-robot systems, manipulation, and search and rescue all make extensive use of MDP. For example, in robot navigation, MDP can help the robot decide what actions to take at evey time step to maximize the probability of reaching the target location or minimize the time[8].

Challenges in robotics: High-dimensional state spaces and real-time constraints are two challenges in these tasks, and MDPs are particularly useful to ensure robots can operate efficiently and safely[2]. High-dimensional state spaces arise when a robot must consider a large number of variables to make decisions, such as the position, velocity, and orientation of each joint in a robotic arm, or the environment's layout in autonomous navigation. To solve this problem, MDP simplifies decision-making to only current state according to the Markov property. Real-time constraints requires robots to make decision within short time. By using effitient algorithms that prioritize faster convergence, MDP can meet the real-time constriants.

3. Policy Iteration

3.1. Algorithm overview

There are two steps for Policy iteration, which are policy evaluation and policy improvement respectively. Policy evaluation is a process to find a given policy's value function. It calculates each state's expected return under the specified policy. The policy evaluation formula is:

$$v_\pi(s) = E_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s] = E_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$
$$= \sum_a \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_\pi(s')] \quad (1)$$

Where $v_\pi(s)$ is the expected return from state s and under policy π. It can be iterated using the formula:

As k approaches infinity, the value function converges to a true value function.

$$v_{k+1}(s) = E_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s] = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \quad (2)$$

Policy improvement is to find a better policy under the current policy from state s, that is $v_\pi(s)$. Selecting an action a in state s and following the existing policy π, then follow the formula:

$$q_\pi(s, a) = E\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s, A_t = a] = \sum_{S',r} p(s', r|s, a)[r + \gamma v_\pi(s')] \quad (3)$$

If it is higher, choosing *a* each time *s* is encountered will be preferable, and the new policy will be superior. That is a special scenario of *the policy improvement theorem*, a generic result. Suppose there are two policies π and π' such that, for all $s \in S$, if: $q_\pi(s, \pi'(s)) \geq v_\pi(s)$. In that case, the policy π' will be at least as good as, π. In other words, it must get a higher or equivalent expected return from every state in S. It follows naturally to take into account modifications at all states and to all feasible courses of action, choosing the action that seems most appropriate at each state based on $q_\pi(s, a)$. Consider the new policy, which is called the greedy policy:

DOI: 10.54254/2755-2721/83/2024GLG0073

$$\begin{aligned}\pi'(s) &= \underset{a}{\operatorname{argmax}}\, q_\pi(s,a) \\ &= \underset{a}{\operatorname{argmax}}\, E[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t=s, A_t=a] \\ &= \underset{a}{\operatorname{argmax}} \sum_{s',r} p(s',r|s,a)[r+\gamma v_\pi(s')] \end{aligned} \quad (4)$$

The greedy policy is at laest as good as the original policy.

After using v_π to enhance policy π and produce a superior policy π',we can proceed to compute $v_{\pi'}$ and refine it once more to produce an even superior π''. As a result, we can produce a series of policies and value functions that improve monotonically[3].

The full procedure for policy iteration is shown on Figure 1.

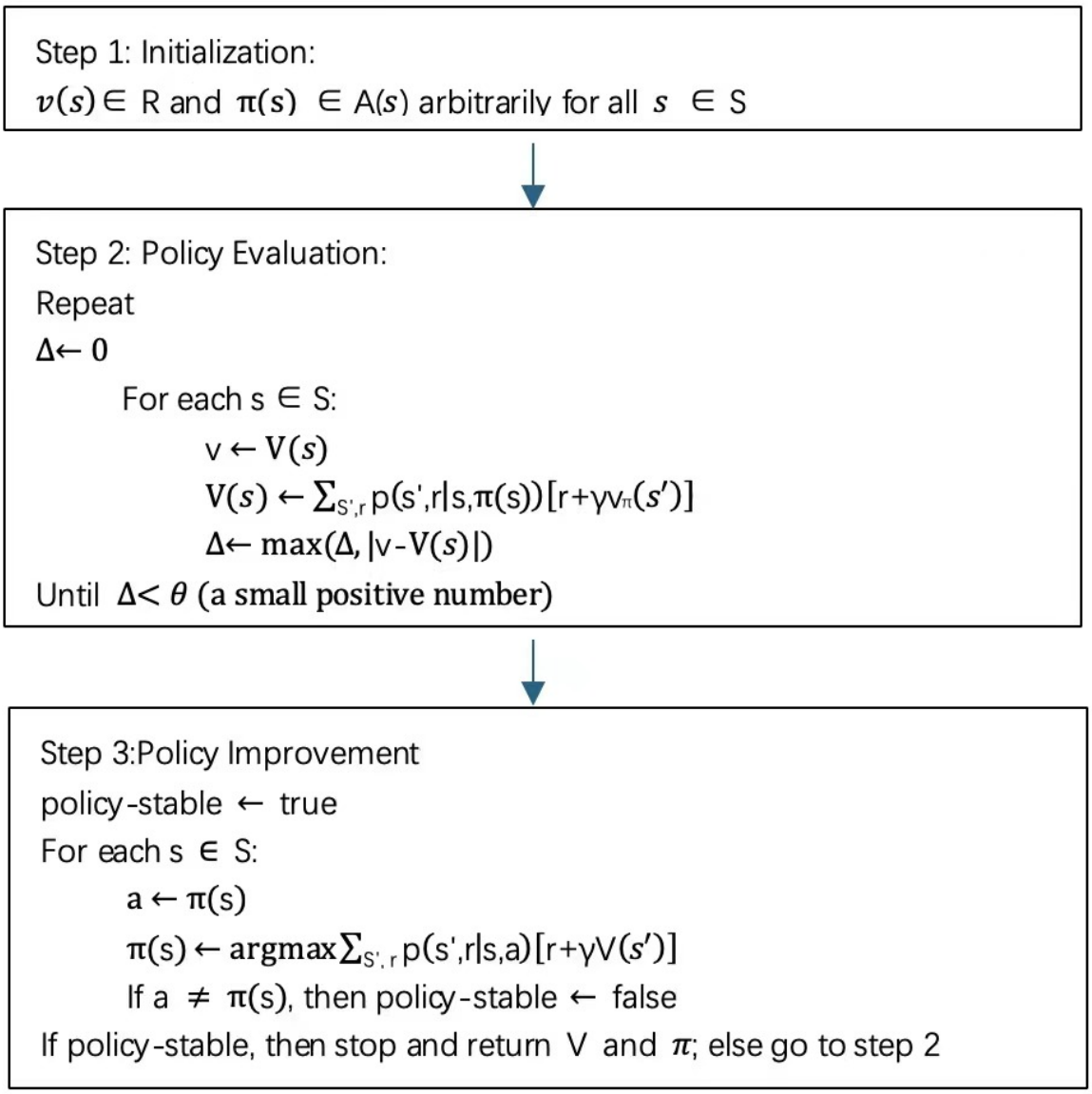

Figure 1. Flow chart for Policy Iteration

3.2. Example application in robotics

One example of policy iteration in robotics is to use it in continuous time and space (CTS) systems. More techniques can be extended by policy iteration, including differential PI (DPI) and Improved PI

DOI: 10.54254/2755-2721/83/2024GLG0073

(IPI), which are applicable to real-time, continuous systems like those found in robotics, demonstrating their effectiveness through simulations[4].

3.3. Advantages of policy iteration

It usually converges faster than value iteration, especially when only a small number of iterations is required. This makes it more suitable for tasks with larger state spaces or where faster convergence is necessary, such as autonomous driving or high-dimensional manipulation.

3.4. Disadvantage of policy iteration

Policy evaluation is involved in each iteration, and this step can be computationally demanding, which is not suitable for time-sensitive robotic applications. Another disadvantage is that there is a risk of policy iteration converging to suboptimal policies if the policy evaluation stage is not carried out accurately[6].

4. Value Iteration

Value iteration is an algorithm that finds the best course of action by iteratively updating each state's value until convergence in reinforcement learning. It does so by computing each state's largest expected utility of future rewards, helping to determine the best action to take from each state to maximize long-term rewards.

4.1. Algorithm Overview

Value iteration is derived from policy iteration. To be specific, there are multiple approaches to shorten the policy evaluation phase of policy iteration without sacrificing policy iteration's convergence guarantees. Value iteration is an algorithm that the policy evaluation stops after just one steep. It can be expressed as a mixture of abbreviated policy evaluation and policy improvement[3]:

$$v_{k+1}(s) = \max_{a} E[\, R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s,\, A_t = a \,]$$
$$= \max_{a} \sum_{s',r} p(s', r \mid s, a)[r + \gamma v_k(s')] \quad (5)$$

Similar to policy iteration, value iteration terminates when it takes an unlimited number of iterations to converge precisely to v_*. In practice, we cease when the value function only slightly shifts in a sweep[3].

4.2. Example application in robotics

Value Iteration is central to optimizing the control policy for modular robot manipulators. It works with adaptive fuzzy systems and reinforcement learning to provide a robust and adaptive control strategy that can handle the complexities of robotic manipulation tasks[5].

4.3. Advantages of value iteration

It requires less memory compared to policy iteration because it focuses on updating the value function directly rather than storing and evaluating entire policies[6]. Additionally, value iteration is simple and easy for implementation. It is particularly effective in problems with small to medium state spaces.

4.4. Disadvantage of value iteration

In big state spaces, in particular, it can be computationally expensive. The need to iterate over all states and actions multiple times until convergence may be demanding[7].

DOI: 10.54254/2755-2721/83/2024GLG0073

5. Comparison Analysis

5.1. Comparative Performance of Policy Iteration and Value Iteration

5.1.1. Speed of convergence of various robotic tasks

Policy iteration is known for its rapid convergence, especially in problems where the initial policy is close to optimal. The algorithm changes between evaluating a policy and improving it, which generally leads to faster convergence compared to Value Iteration, particularly when dealing with large state spaces[9]. In robotic tasks, this means that Policy Iteration can quickly refine control policies, making it suitable for applications where quick adaptation is critical, such as dynamic path planning and obstacle avoidance. Value Iteration, on the other hand, can be slower to converge, especially in environments with large or continuous state spaces. The algorithm iterates over all possible actions and states, updating the value function until it stabilizes, which can be computationally intensive[3]. This slow convergence can be a disadvantage in time-sensitive robotic tasks, where quick policy updates are necessary to respond to changing environments, such as in real-time robotic manipulation tasks.

Example: In a study comparing the two algorithms in robotic navigation tasks, it was observed that Policy Iteration outperformed Value Iteration in terms of convergence speed, particularly in environments with complex state spaces[10]. However, in simpler tasks, such as grid-based path planning, the difference in convergence speed was less obvious, with Value Iteration providing a competitive alternative due to its straightforward implementation[11].

5.1.2. Impact on real-time decision-making in robotics

The faster convergence of Policy Iteration is generally regarded as better real-time performance in robotics, as it allows for the development of policies that can adjust quickly to new information and changing environments. This is particularly useful in tasks that require continuous and dynamic updates, such as autonomous navigation and human-robot interaction. Recent studies have shown that Policy Iteration can effectively handle complex state spaces in real-time scenarios, offering robust performance in dynamic environments[12]. Additionally, the ability to handle larger state spaces more efficiently makes Policy Iteration a better choice for complex decision-making tasks in robotics.Value Iteration is less commonly used in real-time applications due to its slower convergence. But in some scenarios where decision-making processes can be precomputed or where real-time updates are not as critical, Value Iteration can still be considered to be used.

Example: In scenarios where a robotic arm had to adjust its movements rapidly in response to unpredictable obstacles or changes in task requirements, Policy Iteration consistently outperformed Value Iteration. This superior performance was attributed to Policy Iteration's ability to converge more quickly to an optimal policy, enabling the robotic system to make more timely decisions[12].

5.2. Applicability of Value Iteration and Policy Iteration in Various Robotic Tasks

5.2.1. Types of tasks where each algorithm excels

Policy Iteration is more suitable for larger state spaces where it may not be possible to compute the value function directly for every state. It is beneficial in environments where the agent can improve its policy incrementally through policy evaluation and policy improvement steps. This method is advantageous when the agent can leverage the current policy to guide the improvement process, making it more effective in complex tasks with high-dimensional state spaces. Value Iteration tends to excel in tasks where the environment's dynamics are well-understood, and in small state space. It works especially well in situations when the agent can quickly calculate the greatest expected utility for all possible

DOI: 10.54254/2755-2721/83/2024GLG0073

actions efficiently. Value iteration is also simpler in terms of implementation and is known for its computational efficiency when dealing with smaller state spaces.

Example: In scenarios where a robot must navigate through a constantly changing environment, such as a warehouse with moving obstacles or an outdoor setting with varying terrain, Policy Iteration is highly effective. Its ability to update policies rapidly allows the robot to adjust its path in real-time, ensuring safe and efficient navigation[12].

5.2.2. Considerations for making choice between Value Iteration and Policy Iteration

The following need to be considered when deciding which to choose between two algorithms, namely Value Iteration or Policy Iteration:

State Space Size: For smaller, discrete state spaces, value iteration might be more efficient. For larger or continuous state spaces, policy iteration could be more appropriate.

Convergence Speed: Compared to value iteration, policy iteration may converge to an optimal policy more quickly because it directly improves the policy through evaluation and improvement steps.

Computational Resources: Value iteration might be preferred when computational resources are limited due to its typically simpler implementation. However, policy iteration could be chosen if more complex computations are feasible.

Policy Evaluation: If the environment allows for efficient policy evaluation, policy iteration could be advantageous as it includes this step in its process.

Real-time Decision-Making: In scenarios requiring real-time decisions, the potentially faster convergence of policy iteration might be preferred.

Model Knowledge: A model of the environment is necessary for both algorithms. In the event that a precise model is accessible, both algorithms could be utilised efficiently.

The differences between Policy Iteration and Value Iteration is listed in Table 1.

Table 1. Comparison of Policy Iteration and Value Iteration

	Policy Iteration	Value iteration
Algorithm Type	Alternates between policy evaluation and policy improvement until the policy stabilizes	Iterative process that updates the value of each state until convergence.
Convergence Speed	Typically faster, especially when the initial policy is close to optimal.	Generally slower, especially in large state spaces.
Memory Usage	Requires more memory due to storing and evaluating entire policies.	Requires less memory as it only stores the value function.
Complexity	More complex, particularly suited for high-dimensional or larger state spaces.	Simpler to implement, more straightforward in small to medium-sized state spaces.
Application Suitability	Better suited for complex, dynamic environments requiring fast adaptation.	Effective in smaller or well-understood environments where state space is limited.
Real-Time Decision-Making	Better for real-time applications due to quicker convergence and adaptability.	Less suited for real-time applications due to slower convergence.

6. Conclusion

In conclusion, both Value Iteration and Policy Iteration are critical algorithms for solving MDPs, particularly in the domain of robotics. Policy Iteration typically converges faster, making it suitable for complex dynamic environments that require real-time decision-making. In contrast, Value Iteration has

DOI: 10.54254/2755-2721/83/2024GLG0073

a simpler implementation and lower memory requirements, making it more effective in scenarios with smaller or well-understood state spaces. The choice between these algorithms should consider factors such as state space size, the need for real-time decisions, and available computational resources. Ultimately, both algorithms play a significant role in advancing intelligent robotic systems capable of autonomous operation in uncertain environments. Although these two algorithms are effective in solving MDPs, they still have limitations in large state spaces and real-time constraints that require further research. The future of robotics will likely witness continued innovations in MDP algorithms, focusing on enhancing scalability, efficiency, and adaptability. These advancements are expected to drive further progress in autonomous systems, enabling more intelligent, responsive, and robust robotic behavior in increasingly complex and dynamic environments. This research establishes a foundational basis for future investigations into hybrid methods or adaptive algorithms that integrate the strengths of both approaches. Such innovations have the potential to significantly enhance the scalability of algorithms, thereby facilitating their application in more complex, uncertain, and real-time robotic tasks. Ultimately, this advancement will broaden their applicability across a wider array of autonomous systems.

References

[1] Puterman M L. Markov decision processes: discrete stochastic dynamic programming[M]. John Wiley & Sons, 2014.
[2] Kaelbling L P, Lozano-Pérez T. Integrated task and motion planning in belief space[J]. The International Journal of Robotics Research, 2013, 32(9-10): 1194-1227.
[3] Sutton R S, Barto A G. Reinforcement learning: An introduction[M]. MIT press, 2018.
[4] Lee J, Sutton R S. Policy iterations for reinforcement learning problems in continuous time and space—Fundamental theory and methods[J]. Automatica, 2021, 126: 109421.
[5] Dong B, Jiang H, Cui Y, et al. Value Iteration-Based Adaptive Fuzzy Backstepping Optimal Control of Modular Robot Manipulators via Integral Reinforcement Learning[J]. International Journal of Fuzzy Systems, 2024: 1-17.
[6] Geist M, Scherrer B, Pietquin O. A theory of regularized markov decision processes[C]//International Conference on Machine Learning. PMLR, 2019: 2160-2169.
[7] Bertsekas D. Reinforcement learning and optimal control[M]. Athena Scientific, 2019.
[8] Lauri M, Hsu D, Pajarinen J. Partially observable markov decision processes in robotics: A survey[J]. IEEE Transactions on Robotics, 2022, 39(1): 21-40.
[9] Bertsekas D. Abstract dynamic programming[M]. Athena Scientific, 2022.
[10] Silver D, Huang A, Maddison C J, et al. Mastering the game of Go with deep neural networks and tree search[J]. nature, 2016, 529(7587): 484-489.
[11] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. nature, 2015, 518(7540): 529-533.
[12] Ye F, Zhang S, Wang P, et al. A survey of deep reinforcement learning algorithms for motion planning and control of autonomous vehicles[C]//2021 IEEE Intelligent Vehicles Symposium (IV). IEEE, 2021: 1073-1080.