Model Predictive Control-Based Value Estimation for Efficient Reinforcement Learning

Qizhen Wu and Kexin Liu, *Beihang University, Beijing, 100191, China*
Lei Chen, *Beijing Institute of Technology, Beijing, 100081, China*

Reinforcement learning (RL) suffers from limitations in real practices primarily due to the number of required interactions with virtual environments. It results in a challenging problem because we are implausible to obtain a local optimal strategy with only a few attempts for many learning methods. Hereby, we design an improved RL method based on model predictive control that models the environment through a data-driven approach. Based on the learned environment model, it performs multistep prediction to estimate the value function and optimize the policy. The method demonstrates higher learning efficiency, faster convergent speed of strategies tending to the local optimal value, and less sample capacity space required by experience replay buffers. Experimental results, both in classic databases and in a dynamic obstacle-avoidance scenario for an unmanned aerial vehicle, validate the proposed approaches.

The increasing applications of reinforcement learning (RL) in fields such as game playing,[1] natural language processing,[2] and robotics[3] have gained much attention due to its advancements in artificial intelligence. However, the low sample utilization of RL is challenging in applications because agents have limited interactions with the environment. It means that n-step temporal difference (n-TD),[4] as a typical representation of model-free RL aiming to enhance the learning efficiency with mass environmental interactions, is not adaptive to improve sample utilization.[5]

To overcome this difficulty, model-based RL (MBRL) generates virtual data[6] and combines with model predictive control (MPC) to achieve decision making with fewer attempts and computations.[7] The improved RL methods based on MPC apply multistep prediction for the interactions with virtual environments, which generates additional data to improve RL efficiency. Model-based policy optimization uses the probabilistic ensemble approach to approximate the environment model.[8] It implements stepwise inference of the environment model based on MPC and adopts the model data obtained from the inference for policy training. Furthermore, masked model-based actor–critic implements a masking mechanism based on the model's uncertainty to determine the availability of its prediction.[9] This approach takes complete account of the complexity of the model but reduces computational speed due to the large number of neural networks.

In addition to generating more samples, MPC-based RL can better utilize virtual environments by improving single-step value estimation of RL.[10] It leverages the iterative data generated from virtual environments to construct an intensified value, which suffers deeply from inaccurate models.[11] Therefore, TD-MPC employs a neural network that simultaneously approximates the Q-function and the environment model.[12] A novel MBRL approach adopts filtered probabilistic MPC to model the unobservable disturbances of the environment.[13] Most of the existing ways to improve the estimation of the value function are based on short-term rewards but do not well balance the impact of long-term rewards.

In this article, we present a novel MPC-based RL method that focuses on both improving the value estimation and modeling the environment to enhance the learning efficiency and sample utilization of intelligent agents. Our method performs multistep prediction to estimate value function and optimize the policy. It uses a deterministic model-based approach to approximate the environment and applies a rolling optimization approach[14] to maximize the cumulative return for each prediction interval. In applications, we conduct experimental comparisons with baselines in classic simulation environments and a practical RL problem of

1541-1672 © 2024 IEEE
Digital Object Identifier 10.1109/MIS.2024.3386204
Date of publication 8 April 2024; date of current version 11 June 2024.

unmanned aerial vehicle (UAV) dynamic obstacle avoidance. By comparison, our method leads the strategy to quickly converge to the local optimal value based on fewer interaction data. We verify that the state transition and reward function models we trained approximate the environment model well, especially in the case of low-dimensional state and action spaces. Furthermore, a poorly fitted environment model may cause the policy to converge to a suboptimal rather than a global optimal value more quickly in the high-dimensional problem.

We organize the rest of the article as follows. The "Main Results" section presents our proposed method. The "Experimental Results" section describes the experiments of our method. Finally, in the "Conclusion" section, we present our conclusions.

MAIN RESULTS

Presentation of Basic Algorithms

Let $\mathbb{R}$ denote the set of real numbers. $\mathbb{E}$ denotes the mathematical expectation. RL can be modeled by a Markov decision process (MDP). It is represented by a five-tuple (S, A, P, R, γ), where $S \subseteq \mathbb{R}^{m_s}$ is the environment state space and $A \subseteq \mathbb{R}^{m_a}$ is the action space. m_s and m_a denote the state and action space dimensions, respectively. $P : S \times A \to \mathbb{R}^{m_s}$ denotes the state-transition function, $R : S \times A \to \mathbb{R}$ denotes the reward function, and γ denotes the discount factor. The purpose of RL is to optimize the policy $\pi : S \to A$ such that the cumulative reward $\mathbb{E}_{\Gamma\sim\pi}[\sum_{t=1}^{\infty}\gamma^t r_t], r_t \sim R(s_t, a_t)$ is maximized. $\Gamma = (s_0, a_0, s_1, a_1, ...)$ is the trajectory of the agent interacting with the environment under the policy, where $a_t \sim \pi(s_t)$ and $s_{t+1} \sim P(s_t, a_t)$ denote the selected action and reaching state at each decision step t, respectively.

In the temporal difference method,[5] value-based RL approximates the cumulative rewards $\sum_{t=1}^{\infty}\gamma^t r_t$ by $R(s,a) + \gamma \max Q^*(s', a')$. It estimates the optimal state-action value function $Q^* : S \times A \to \mathbb{R}$ through a Q table or a parameterized neural network $Q_\omega(s,a) \approx Q^*(s,a) = \mathbb{E}[R(s,a) + \gamma \max Q^*(s', a')], \forall s \in S$, where s', a' are the state and action at the following step, respectively. ω is the weighting factor of the neural network. For $\gamma \approx 1$, Q^* estimates the discounted returns of the local optimal strategy over an infinite range. For value-based RL, the value estimation of $Q(s,a)$ can be described as the following form:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s', a') - Q(s,a)] \tag{1}$$

where α is the learning rate of policy training. For value-based deep RL, Q^* can be approximated by an iteratively fitting Q_ω, and the loss function is described as

$$L_\omega = \mathbb{E}_{(s,a)\sim\mathfrak{B}}||Q_\omega(s,a) - y||^2 \tag{2}$$

where $y = R(s,a) + \gamma \max Q_{\omega^-}(s', a')$ is the Q-target. Subscript ω^- is a slow-moving online average. ω^- is updated with the $\omega_{k+1}^- \leftarrow (1-\zeta)\omega_k^- + \zeta\omega_k$ rule at each iteration using a constant factor $\zeta \in [0, 1)$. $\mathfrak{B}$ is a replay buffer that iteratively grows as data are updated.

In the actor–critic algorithm, π is usually a policy parameterized by a neural network π_β. It learns the $\pi_\beta(s) \approx \arg\max_a \mathbb{E}[Q_\omega(s,a)], \forall s \in S$ approximation, which is the local optimal policy. The loss function is

$$L_\beta = -\mathbb{E}_{(s,a)\sim\mathfrak{B}}[Q_\omega(s,a)]. \tag{3}$$

In control theory, we formulate the state-space equation as follows:

$$\dot{s} = As + Ba \tag{4}$$

where s and a are the state and action vectors, respectively. MPC obtains a local solution to the trajectory optimization problem at each step t by estimating the local optimal action $a_{t:t+N}$ on a finite horizon N and executing the first action

$$\pi_1^{\mathrm{MPC}}(s_t) = \arg\min_{a_{t:t+N}} \mathbb{E}\left[\sum_{i=t}^{t+N-1}\left(s_i^T G\, s_i + a_i^T E\, a_i\right)\right] \tag{5}$$

$$\pi_2^{\mathrm{MPC}}(s_t) = \arg\min_{a_{t:t+N}} \mathbb{E}\left[\sum_{i=t}^{t+N-1}\left(s_i^T G\, s_i + a_i^T E\, a_i\right) + s_N^T P\, s_N\right] \tag{6}$$

where G, E, and P are the error weighting, control weighting, and terminal error weighting matrix, respectively. Equation (5) is the basic MPC and (6) is the basic MPC with added terminal cost.[14]

The Improved MPC-Based RL Algorithm

To enhance learning efficiency and reduce the number of interaction data required, we propose an online training method that focuses on improving the value estimation and modeling of the environment. Our method performs multistep prediction to estimate the value function and optimize the policy. It learns local optimal strategies by modeling the environment's state transition and reward functions using a data-driven approach. Moreover, it employs a rolling optimization approach to maximize the cumulative return for each prediction interval.

MPC-based RL replaces process and terminal cost with rewards and terminal value functions. By

transforming the problem from least cost to most benefit in (5) and (6), it obtains the following forms:

$$\pi_1^{\mathrm{MPC-RL}}(s_t) = \arg\max_{a_{t:t+N}} \mathbb{E}\left[\sum_{k=t}^{t+N-1} \gamma^k R(s_k, a_k)\right] \tag{7}$$

$$\pi_2^{\mathrm{MPC-RL}}(s_t) = \arg\max_{a_{t:t+N}} \mathbb{E}\left[\sum_{k=t}^{t+N-1} \gamma^k R(s_k, a_k) + \gamma^{t+N} \max Q_{\theta^-}(s_{t+N}, a_{t+N})\right]. \tag{8}$$

As shown in Figure 1, the main difference between n-TD and our method is that, in (7) and (8), the states and rewards at moments $t+1$ to $t+N$ are obtained from real-experience trajectory. In contrast, the states and rewards of MPC-based RL are obtained from trajectory branching for N-step prediction based on the environment model. MPC-based RL not only improves the TD target but also employs a multistep approximation approach to estimate value function.

The goal of strategy in MPC is to predict a sequence in N steps of actions such that the short-term cumulative rewards are maximized in such interval. The goal of policy in RL is to maximize the long-term cumulative rewards for all future moments $\mathbb{E}_{\Gamma\sim\pi}[\sum_{k=1}^{\infty}\gamma^k r_k], r_k \sim R(s_k, a_k)$. We replace the reward $R(s_k, a_k)$ with the cumulative rewards $\sum_{i=k}^{\infty}\gamma^i r_i$ in (7) that maximizes both short- and long-term rewards, and the strategy can be described in the following form:

$$c\pi^{\mathrm{MPC-RL}}(s_t) = \arg\max_{a_{t:t+N}} \mathbb{E}\left[\sum_{k=t}^{t+N}\gamma^k\left(\sum_{i=k}^{\infty}\gamma^i r_i\right)\right]. \tag{9}$$

In (9), we perform predictive control based on a single moment t to produce a local optimal solution in the present moment. The rolling optimization is then performed to generate multiple local optimal solutions and calculate the optimal value. The use of cumulative rewards compensates for the lack of the terminal reward. We can improve sample utilization and training efficiency by accumulating the cumulative rewards in the next N steps.

The interaction results in the next N moments are required in (9), but the sample data stored in the experience replay buffer $\mathfrak{B}$ have only the interaction tuple for a single moment t. The models are unknown for most of the interaction environments, and these sample data cannot be inferred from a priori knowledge. Therefore, we need to build neural networks with states transition function P and reward function R to predict the interaction outcome of the next N moments from the input states and actions. P and R can be modeled as separate neural networks (P_θ, R_τ) or combined into one network (PR_ψ). The subscripts $\theta, \tau,$ and ψ are the weighting factors of neural networks. We use a deterministic model-based approach instead of the probabilistic ensemble.[15] It reduces the number of neural networks and improves the computational speed of the algorithm, albeit at the cost of partially ignoring model uncertainty.

In our framework shown in Figure 2, the samples drawn from the experience replay buffer are used to update the environment model and the policy. Meanwhile, the environment model has a facilitating effect on the update of the policy model. Our approach performs multistep prediction based on one sample data, resulting in faster convergence of the strategy. Similar to RL, we approximate the cumulative rewards $\sum_{t=1}^{\infty}\gamma^t r_t$ by $R(s,a)+\gamma\max Q^*(s',a')$. Instead of letting $Q(s_k, a_k)$ be close to y_k, our method performs the following multistep approximation approach:

$$\begin{cases} Q(s_k, a_k) \to y_k \\ Q(\hat{s}_{k+1}, a_{k+1}) \to y_{k+1} \\ \vdots \\ Q(\hat{s}_{k+n}, a_{k+n}) \to y_{k+n} \end{cases} \tag{10}$$

where $\hat{s}, \hat{r}$ indicate that the state and reward, respectively, are predicted based on the model we learned.

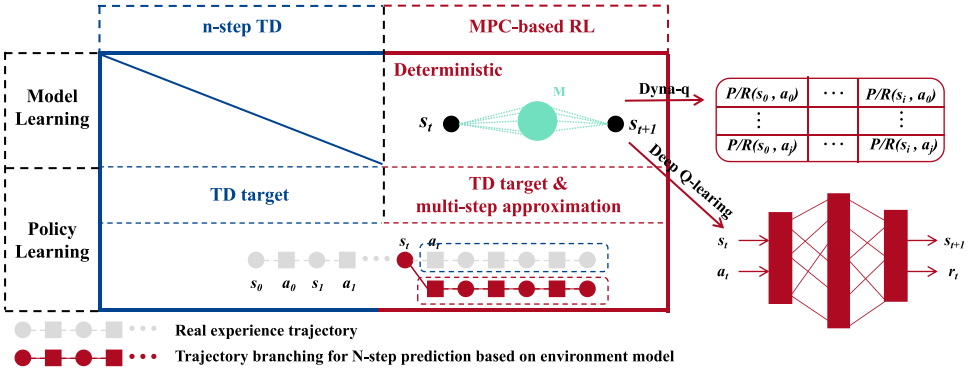

FIGURE 1. Differences between *n*-TD and our method.

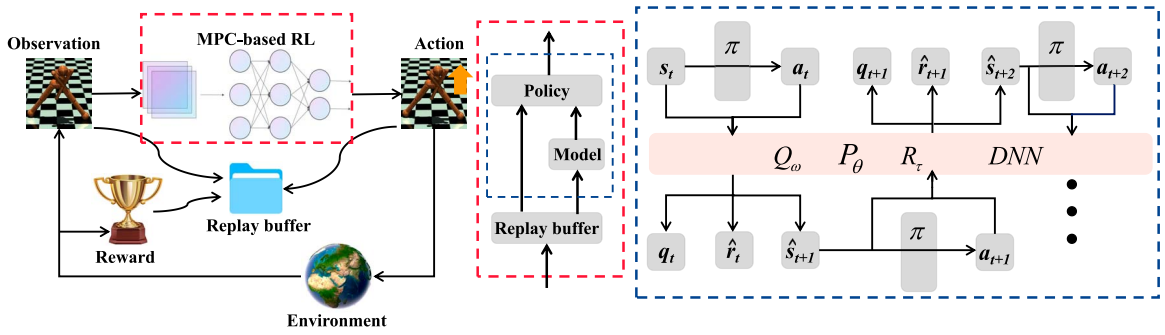

FIGURE 2. Framework of MPC-based RL. DNN: deep neural network.

In RL, we define the value estimation as (1), which is a single-step policy updating. Based on the idea that MPC performs multistep prediction in the forecast interval, we can improve the value estimation of $Q(s,a)$ by

$$Q(s_{k+n},a_{k+n}) \leftarrow Q(s_{k+n},a_{k+n}) + \alpha[\hat{r}_{k+n} + \gamma Q(\hat{s}_{k+n+1},a_{k+n+1}) - Q(s_{k+n},a_{k+n})] \quad (11)$$

where $n = 0,1,2,...,N-1$ is the current prediction step. We improve the loss function in (2) as follows:

$$L_\omega = \mathbb{E}_{(s_k,a_k)\sim\mathfrak{B}} \sum_{n=0}^{N-1} \gamma^n ||Q_\omega(s_{k+n},a_{k+n}) - y_{k+n}||^2 . \quad (12)$$

Algorithm 1. MPC-Based RL

Input: Environment(*Env*); action space(*Act*); discount factor(γ); learning rate(α); training episode(*E*); training step(*T*); prediction step(*N*);
Output: Policy-network ($\pi(s)$)

Initialize networks ω,θ,τ,ψ and the replay buffer $\mathfrak{B}$;
for all $e = 1 \rightarrow E$ **do**
  Get the initial state s_0;
  for all $t = 1 \rightarrow T$ **do**
    $a_t \leftarrow \pi(s)$, ϵ–greedy; ▷ on-policy action
    $r_t, s_{t+1} \leftarrow$ Env; ▷ transition in environment
    $\mathfrak{B} \leftarrow (s_t,a_t,r_t,s_{t+1})$; ▷ replay buffer update
    Randomly selected B samples (s_k,a_k,r_k,s_{k+1}) from $\mathfrak{B}$;
    $\hat{s}_{k+1},\hat{r}_k \leftarrow$ ▷ prediction in model $P_\theta(s_k,a_k), R_\tau(s_k,a_k)$ or $PR_\psi(s_k,a_k)$;
    if $L_\theta, L_\tau < \epsilon_m$ or $L_\psi < \epsilon_m$ **then**
      for all $n = 1 \rightarrow N$ **do**
        $a_{k+n} \leftarrow \pi(s_{k+n})$;
        $\hat{s}_{k+n+1},\hat{r}_{k+n} \leftarrow$ ▷ multistep prediction $P_\theta(s_{k+n},a_{k+n}), R_\tau(s_{k+n},a_{k+n})$ or $PR_\psi(s_{k+n},a_{k+n})$;
      end for
    else
      $a_{k+1} \leftarrow \pi(s_{k+1})$;
    end if
    Calculate L_ω through (12) and (13);
    Calculate L_θ, L_τ through (18);
    Refresh networks ω,θ,τ; ▷ gradient-descent
    Refresh the strategy $\pi(s)$;
  end for
end for

According to (8), we modify the *Q-target* to the following form:

$$y_{k+n} = \sum_{i=n}^{N-1} \gamma^{i-n}\, \hat{r}_{k+i} + \gamma^{N-n} \max Q_{\omega^-}(s_{k+N},a_{k+N}). \quad (13)$$

MPC-based RL can lead the policy to converge to the local optimal value faster. However, learning speed and accuracy of the environment model can limit the speed and optimality with which the policy finds the local optimal value.[4]

Meanwhile, we need to update the neural networks of the environment model in real time. The loss functions of P_θ and R_τ are

$$L_\theta = \mathbb{E}_{(s_k,a_k)\sim\mathfrak{B}} ||(\hat{s}_{k+1} - s_{k+1})||^2$$
$$L_\tau = \mathbb{E}_{(s_k,a_k)\sim\mathfrak{B}} ||(\hat{r}_k - r_k)||^2. \quad (14)$$

The loss function of PR_ψ is

$$L_\theta = \mathbb{E}_{(s_k,a_k)\sim\mathfrak{B}} [||(\hat{s}_{k+1} - s_{k+1})||^2 + \lambda ||(\hat{r}_k - r_k)||^2] \quad (15)$$

where $\lambda > 0$ is a hyperparameter, which defines the relative weighting of states and rewards. Equations (14) and (15) indicate the degree of deviation of the constructed environment model from the real environment. By minimizing the values of these functions, the environment model can increasingly approximate the real environment.

We show that the environment model we learned approximates the real environment well in low-dimensional state and action spaces in simulations.

However, in high-dimensional environments, the loss functions of environment models may not converge to zero and fail to reflect the complete situation of the real environment. This model error can accumulate and lead to a suboptimal policy. To avoid the impact of inaccurate environment models on the algorithm, we enable environment models for prediction only when their loss functions are less than the certain bounds ϵ_m. The flowchart of the MPC-based RL approach is shown in Algorithm 1.

To account for the monotonic improvement of model-based methods over model-free methods, we can construct a bound of the following form:[8]

$$\eta[\pi] \geq \eta^{\text{branch}}[\pi] - C$$
$$C = 2r_{\max}\left[\frac{\gamma^{k+1}\epsilon_\pi}{(1-\gamma)^2} + \frac{(\gamma^k+2)\epsilon_\pi + N(\epsilon_m+2\epsilon_\pi)}{(1-\gamma)}\right] \quad (16)$$

where $\eta[\pi]$ and $\eta^{\text{branch}}[\pi]$ denote the returns of the policy in the MDP and under our model, respectively. ϵ_m denotes the generalization error due to sampling, and ϵ_π denotes the distribution shift due to the updated policy encountering states not seen during model training. We can guarantee that it also improves in the MDP if the strategy enhances its return under the model by at least C. MPC-based RL updates the environment model at each decision step t. Therefore, the environment model is more accurate and the strategy difference between iterations is more significant. In this case, the generalization error ϵ_m is much smaller than distribution shift ϵ_π. The optimal prediction step N^* can be expressed in the following form:

$$N^* = \arg\min_n \left\{\frac{\gamma^{k+1}\epsilon_\pi}{(1-\gamma)^2} + \frac{(\gamma^k+2N+2)\epsilon_\pi}{(1-\gamma)}\right\} > 0. \quad (17)$$

EXPERIMENTAL RESULTS

Here, we compare our method to traditional RL methods, including n-TD,[4] Dyna-q,[16] deep Q-learning (DQN),[17] and deep deterministic policy gradient (DDPG);[18] in classic simulation environments; and a designed UAV dynamic obstacle-avoidance environment. We apply Algorithm 1 to obtain Dyna-MPC, DQN-MPC, and DDPG-MPC. Our results demonstrate that our method significantly reduces the samples required for the agent to interact with the environment while achieving local optimal performance.

Classic Simulation Environment

We evaluate our proposed method and compare its performance against baselines on diverse and challenging control tasks from OpenAI Gym. Our experiments are conducted in classic simulation environments, such as cliff walking (CW), CartPole (CP), pendulum (PD), and humanoid (HO).

Implementation Details

We use deterministic components implemented using multilayer perceptrons. The settings for the algorithm parameters are presented in Table 1. We conduct these simulations on a server with a Windows 10 operating system, an Intel Core i7-11700 CPU, a 16-GB memory, and Radeon 520 GPU. All the simulation programs are developed based on Python 3.7 and PyCharm 2022.2.3 compiler. To plot experimental curves, we adopt solid curves to depict the mean of four trials and shaded regions corresponding to standard deviation among trials.

Dyna-MPC

We conduct experiments using n-TD and Dyna-q as baselines to implement CW and compare them with Dyna-MPC, as shown in Figure 3(a). For n-TD, it is equivalent to Q-learning when $N = 1$. As N increases, the curves fall more sharply in the early period but rise to convergence more rapidly. The reason is that n-TD adopts multistep interaction with the environment, and the deviation of the strategy will directly lead to the cumulative deviation of the interaction data over multiple steps. In Dyna-q, it stores $P(s,a)$ and $R(s,a)$ in

TABLE 1. Setting of the algorithm parameters.

Env	α_ω $(\times 10^{-2})$	α_β $(\times 10^{-4})$	γ	ϵ $(\times 10^{-2})$	$\mathfrak{B}$ $(\times 10^{4})$	α_θ $(\times 10^{-3})$	α_τ $(\times 10^{-3})$	α_ψ $(\times 10^{-3})$
CW	10	—	0.9	1	—	—	—	—
CP	0.2	—	0.98	1	1	2	2	2
PD	0.3	3	0.98	—	1	3	3	3
HO	0.1	1	0.99	—	100	1	1	—
UAV path planning	0.1	10	0.99		100	1	1	1

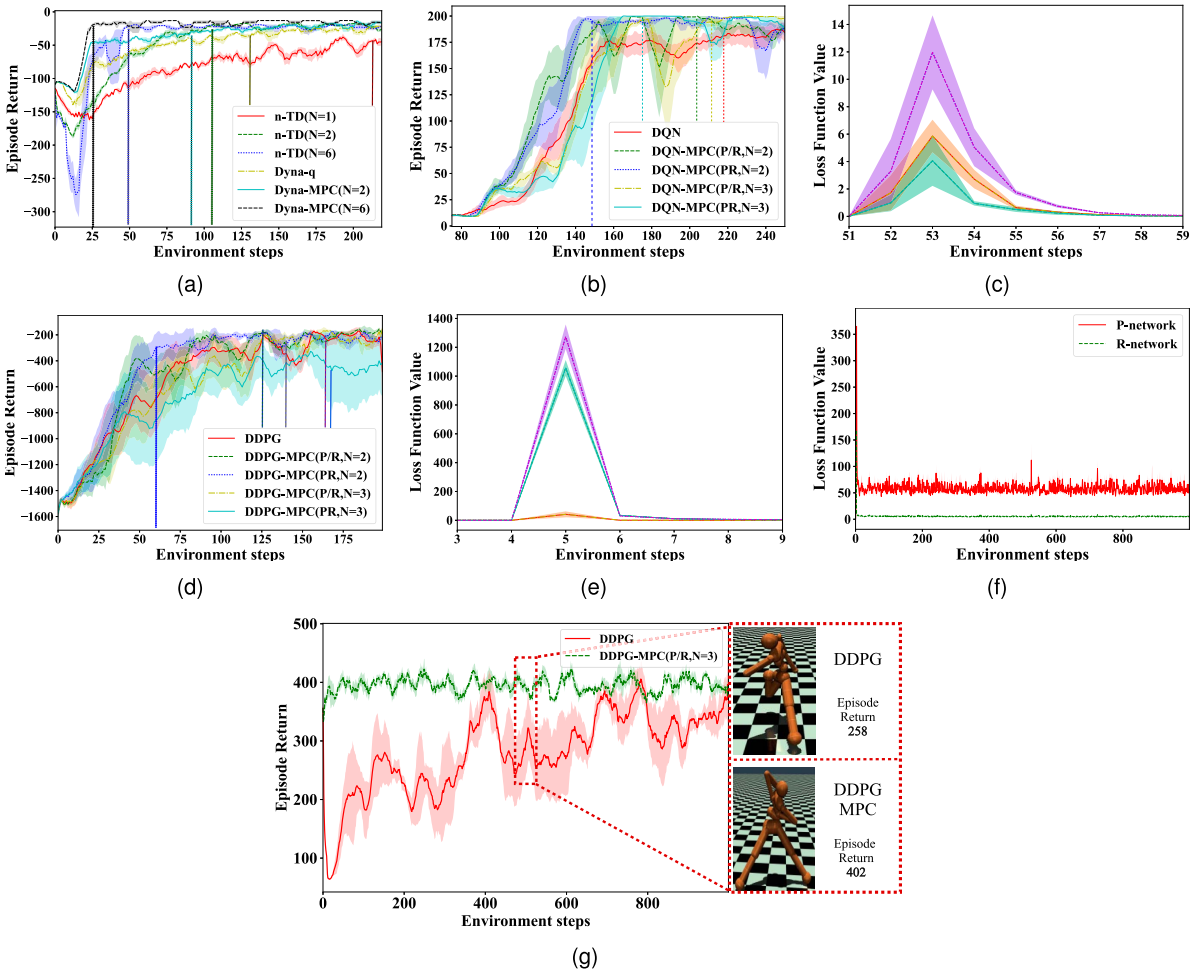

FIGURE 3. Comparison with MPC-based RL and baseline. (a) Episode return in CW. (b) Episode return in CP. (c) Loss function value in CP. (d) Episode return in PD. (e) Loss function value in PD. (f) Loss function value in HO. (g) Episode return in HO.

a table similar to $Q(s,a)$, which provides an accurate model of the environment when the environment is a deterministic process. Therefore, by the model-based approach, its convergence will be faster than Q-learning. Different from n-TD, Dyna-MPC performs multistep prediction in the value estimation by relying on deterministic environment models $P(s,a)$ and $R(s,a)$. The method avoids the cumulative deviation of interaction data and improves the training efficiency. Through comparisons, Dyna-MPC leads the strategy to quickly converge to the local optimal value based on fewer interaction data. As the prediction step N increases, fewer episodes are required for the strategy to converge to the local optimal value. However, $N=6$ provides limited performance improvement over $N=2$ and requires more computational efforts for multi-step prediction.

DQN-MPC and DDPG-MPC

DQN is adopted as the baseline to implement CP, and the performance is compared to DQN-MPC, as shown in Figure 3(b) and (c). P/R or *PR* denotes modeling P and R as separate neural networks or combining them into one network. The buffer size of DQN-MPC is half of DQN's, but it can reduce the number of sample interactions required for the strategy to converge to the local optimal value. In CP $(N^*=2)$, a better result is achieved by modeling P and R as one neural network. An additional prediction step N cannot further reduce the number of sample interactions.

We use DDPG as the baseline to implement PD and HO and compare the performance to DDPG-MPC, as shown in Figure 3(d) and (e). In PD $(N^*=2)$, the buffer size of DDPG-MPC is half of DDPG's, but it can reduce the number of sample interactions required for the

strategy to converge to the local optimal value. Better results are achieved by modeling P and R as one neural network. In HO $(N^* = 3)$, the buffer size of DDPG-MPC is one-tenth of DDPG's. However, it can significantly reduce the sample interactions required for the strategy to converge to the suboptimal value, which is not the global optimality. The right part of Figure 3(g) shows the average returns from the 480th to the 520th episode. Since the Mujoco environment has high-dimensional state and action spaces, it is difficult to approximate its state transition and reward functions. The learning speed and accuracy of the environment model limit the speed and optimality with which the policy finds the local optimal value. Through comparisons, DQN-MPC and DDPG-MPC lead the strategy to quickly converge to the local optimal value based on fewer interaction data and free up more sample capacity space required for the experience replay buffer.

Model Learning of Environment

MPC-based RL method learns the state transition and reward functions of the environment while optimizing the policy, as shown in Figure 3. In classical control environments with low dimension, the loss function values of P and R converge quickly to zero, indicating that the networks fit the environment well. However, in HO, due to its high dimensionality, the loss function values of the networks cannot converge to zero. As a result, the networks cannot accurately represent the state transition and reward of the real environment. The deviation of the learned model from the actual environment accumulates in MBRL, leading to only a suboptimal strategy.

UAV Dynamic Obstacle-Avoidance Environment

To validate MPC-based RL in the path-planning task, we establish the UAV dynamic obstacle-avoidance problem as a complete MDP shown in Figure 4, and we use the orb as a dynamic obstacle in the scene. We simulate obstacle avoidance in real scenarios through the simulation, including the dynamics of the UAV and the movement of the obstacle.

Let the position and velocity vectors in 3-D be denoted by p and v, respectively. p_u, p_o, p_e denote the positions of the UAV's center, obstacle center, and destination, respectively. v_o and ρ_o denote the obstacle's velocity and radius, respectively. ρ_u denotes the radius of the UAV. The state space can be set in the following form:

$$s = \begin{bmatrix} c(p_o - p_u) \cdot \dfrac{d_{ou} - (\rho_o + \rho_u)}{d_{ou}} \\ p_e - p_u \\ v_o \end{bmatrix} \tag{18}$$

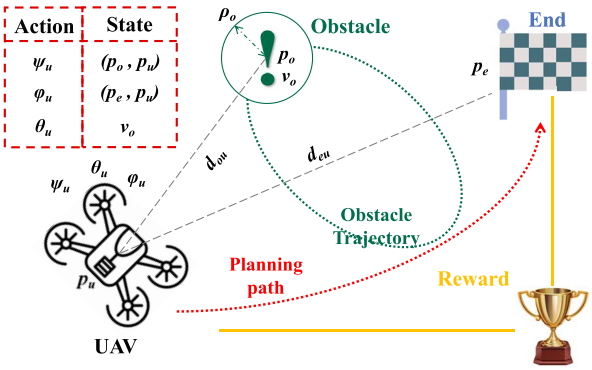

FIGURE 4. The MDP of UAV dynamic obstacle-avoidance problem.

and $d_{ou} = ||p_o - p_u||$ is the distance between the UAV and the obstacle's center, where $||\cdot||$ denotes the Euclidean norm. The action space can be set in the following form:

$$a = [\psi_u \quad \theta_u \quad \varphi_u]^T \tag{19}$$

where $\psi_u, \theta_u,$ and φ_u denote the UAV's roll, yaw, and pitch, respectively. To lead the UAV to avoid a dynamic obstacle and achieve the shortest path planning, we design the reward as follows:

$$r = \begin{cases} \dfrac{d_{ou} - (\rho_o + \rho_u)}{\rho_o + \rho_u} - r_a, & \text{if } d_{ou} < \rho_o + \rho_u \\ -\dfrac{d_{eu}}{d_{es}} + r_b + r_c, & \text{if } d_{ou} > \rho_o + \rho_u \,\&\, d_{eu} < d_{\text{com}} \\ -\dfrac{d_{eu}}{d_{es}} + r_c, & \text{otherwise} \end{cases} \tag{20}$$

where $d_{eu} = ||p_e - p_u||$ denotes the distance between the destination and UAV's center. $d_{es} = ||p_e - p_s||$ denotes the distance between the destination and the beginning. d_{com} is an arrival distance and, if $d_{eu} < d_{\text{com}}$, the task is completed. r_a is a constant reward and r_c is the additional reward for completing the task. r_b is a threat reward set to keep the UAV as far away from the obstacle as possible

$$r_b = \begin{cases} \dfrac{d_{ou} - (\rho_o + \rho_u + d_{\text{thr}})}{\rho_o + \rho_u + d_{\text{thr}}} - r_d, & \text{if } d_{ou} < \rho_o + \rho_u + d_{\text{thr}} \\ 0, & \text{otherwise} \end{cases} \tag{21}$$

where r_d is a constant reward we set. d_{thr} is a threat distance and, if $d_{ou} < \rho_o + \rho_u + d_{\text{thr}}$, we consider that the UAV is entering the area where it is about to collide

with the obstacle. With the design of the aforementioned reward function, when the path is planned with higher cumulative rewards, it can indicate that the path takes less time and has a smaller probability of collision with the obstacle. We apply (18)–(20) as the state, action, and reward spaces in the MDP, respectively. The parameters are set as follows: $\rho_o = 1.5 \ \ \mathrm{m}$, $d_{\mathrm{thr}} = 0.4 \ \ \mathrm{m}$, $r_a = 1$, $r_b = 3$, and $r_d = 0.3$. In addition, the state transition of the environment $P(s, a)$ is implemented based on the disturbed flow field algorithm and kinematic constraint functions.[19] Next, we can verify the performance of MPC-based RL based on this MDP model.

We adopt DDPG as the baseline to implement UAV path planning and compare its performance with DDPG-MPC $(N^* = 2)$, as shown in Figure 5(a) and (b). The buffer size of DDPG-MPC is one-tenth of DDPG's, but it can significantly reduce the sample interaction required for the strategy to converge to the local optimal value. By modeling P and R as one neural network, the loss function values of the network can converge quickly to zero, leading to better results. In Figure 5(c)–(f), we compare the path-planning performance of DDPG-MPC and DDPG under different episodes. DDPG-MPC achieves higher cumulative rewards for the same episode than DDPG. As higher cumulative

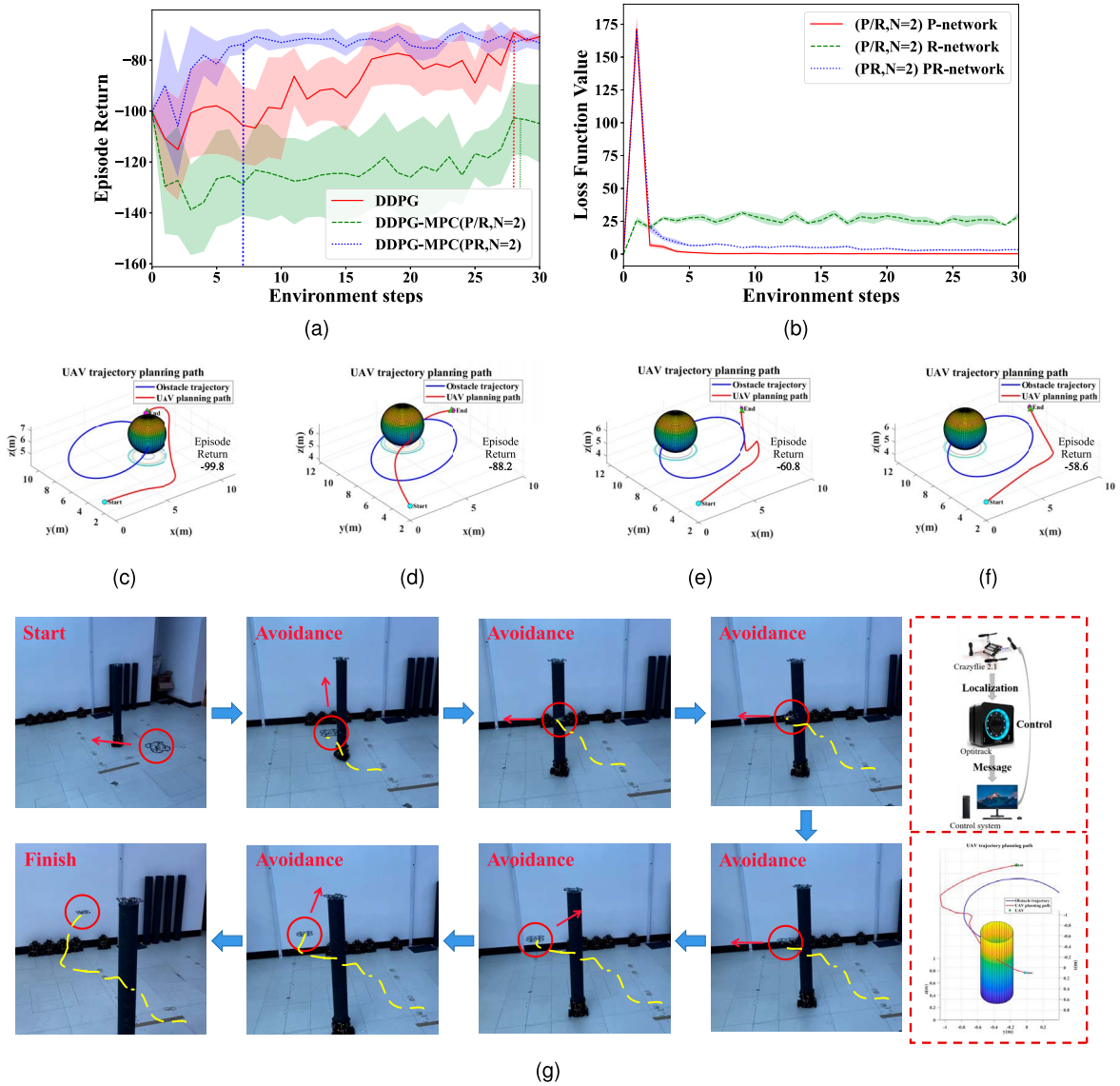

FIGURE 5. Comparison with DDPG in UAV path planning. (a) Episode return. (b) Loss function value. (c) DDPG 7th episode. (d) DDPG 23rd episode. (e) DDPG-MPC 7th episode. (f) DDPG-MPC 23rd episode. (g) Indoor flight verification.

rewards mean shorter path-spending time with less probability of obstacle collisions, DDPG-MPC plans better paths than does DDPG.

Indoor Flight Experiment

We conduct experiments with a real UAV to verify the feasibility of the designed DDPG-MPC approach. In the experiment, we adopt the method of simulation to reality.[20] In an indoor environment, as shown in Figure 5(g), we maneuver the crazyflie through the ground control center to complete experiments with motion capture from the optitrack. We use a plastic tube with a radius of 0.1 m and a height of 1 m as a cylindrical obstacle, and the UAV traverses the obstacle from one end to the other. By performing DDPG-MPC training in the simulation, we can get the policy network of the UAV in the dynamic obstacle-avoidance scenario. Based on the policy network noted earlier, the UAV can avoid the obstacle well in the experiment and reach the target position. The experiment's success verifies that DDPG-MPC can accomplish the UAV's dynamic obstacle-avoidance task by offline training and online decision making, and this RL method is improved based on MPC.

CONCLUSION

This article presented a novel MPC-based value estimation approach, which improves the training efficiency and sample utilization of the agent in RL. The proposed method has been effective in classical simulation environments and UAV path-planning environments. The method has successfully approximated the model of the environment, which has been well adapted to low-dimensional space. We plan to investigate applying probabilistic ensemble models to learn the environment in high-dimensional space.

ACKNOWLEDGMENTS

This work was supported in part by the National Key R&D Program of China under Grant 2022YFC3300703, and in part by the National Science Foundation of China under Grant 62088101 and Grant 62003015.

REFERENCES

1. P. R. Wurman, P. Stone, and M. Spranger, "Challenges and opportunities of applying reinforcement learning to autonomous racing," *IEEE Intell. Syst.*, vol. 37, no. 3, pp. 20–23, May/Jun. 2022, doi: 10.1109/MIS.2022.3184427.
2. L. Gao, J. Schulman, and J. Hilton, "Scaling laws for reward model overoptimization," in *Proc. 40th Int. Conf. Mach. Learn. (ICML)*, 2023, pp. 10,835–10,866.
3. J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *Int. J. Robot. Res.*, vol. 32, no. 11, pp. 1238–1274, 2013, doi: 10.1177/0278364913495721.
4. R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.
5. J. Yu, X. Dong, Q. Li, J. Lv, and Z. Ren, "Distributed adaptive cooperative time-varying formation tracking guidance for multiple aerial vehicles system," *Aerosp. Sci. Technol.*, vol. 117, Oct. 2021, Art. no. 106925, doi: 10.1016/j.ast.2021.106925.
6. T. Liu, X. Hu, W. Hu, and Y. Zou, "A heuristic planning reinforcement learning-based energy management for power-split plug-in hybrid electric vehicles," *IEEE Trans. Ind. Informat.*, vol. 15, no. 12, pp. 6436–6445, Dec. 2019, doi: 10.1109/TII.2019.2903098.
7. I. Kostrikov, D. Yarats, and R. Fergus, "Image augmentation is all you need: Regularizing deep reinforcement learning from pixels," in *Proc. Int. Conf. Learn. Representations (ICLR)*, 2021.
8. M. Janner, J. Fu, M. Zhang, and S. Levine, "When to trust your model: Model-based policy optimization," in *Proc. 32nd Adv. Neural Inf. Process. Syst. (NeurIPS)*, 2019, pp. 12519–12530.
9. F. Y. Pan, J. He, D. D. Tu, and Q. He, "Trust the model when it is confident: Masked model-based actor-critic," in *Proc. Adv. Neural Inf. Process. Syst.*, 2020, vol. 33, pp. 10,537–10,546.
10. S. Gros and M. Zanon, "Data-driven economic NMPC using reinforcement learning," *IEEE Trans. Autom. Control*, vol. 65, no. 2, pp. 636–648, Feb. 2020, doi: 10.1109/TAC.2019.2913768.
11. J. Buckman, D. Hafner, G. Tucker, E. Brevdo, and H. Lee, "Sample-efficient reinforcement learning with stochastic ensemble value expansion," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, vol. 31, pp. 8234–8244.
12. N. Hansen, X. Wang, and H. Su, "Temporal difference learning for model predictive control," in *Proc. Int. Conf. Mach. Learn. (ICML)*, 2022, pp. 8387–8406.
13. Y. Cui, L. Peng, and H. Li, "Filtered probabilistic model predictive control-based reinforcement learning for unmanned surface vehicles," *IEEE Trans. Ind. Informat.*, vol. 18, no. 10, pp. 6950–6961, Oct. 2022, doi: 10.1109/TII.2022.3142323.
14. C. E. Garcia, D. M. Prett, and M. Morari, "Model predictive control: Theory and practice—A survey," *Automatica*, vol. 25, no. 3, pp. 335–348, May 1989, doi: 10.1016/0005-1098(89)90002-2.
15. K. Chua, R. Calandra, R. McAllister, and S. Levine, "Deep reinforcement learning in a handful of trials using probabilistic dynamics models," in *Proc. Adv. Neural Inf. Process. Syst.*, 2018, vol. 31, pp. 4759–4770.

16. R. S. Sutton, "Integrated architectures for learning, planning, and reacting based on approximating dynamic programming," in *Proc. Mach. Learn.*, 1990, pp. 216–224, doi: 10.1016/B978-1-55860-141-3.50030-4.
17. V. Mnih et al., "Playing Atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.
18. T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.
19. Y.-b. Chen, G.-c. Luo, Y.-s. Mei, J.-q. Yu, and X.-l. Su, "UAV path planning using artificial potential field method updated by optimal control theory," *Int. J. Syst. Sci.*, vol. 47, no. 6, pp. 1407–1420, 2016, doi: 10.1080/00207721.2014.929191.
20. W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: A survey," in *Proc. Symp. Ser. Comput. Intell. (SSCI)*, 2020, pp. 737–744.

QIZHEN WU is pursuing his master's degree with the School of Automation Science and Electrical Engineering, Beihang University, Beijing, 100191, China. His research interests include reinforcement learning, robotic control, and task scheduling. Wu received his B.S. degree from Sun Yat-sen University. Contact him at wuqzh7@buaa.edu.cn.

KEXIN LIU is an associate professor with the School of Automation Science and Electrical Engineering, Beihang University, Beijing, 100191, China. His research interests include multiagent systems and complex networks. Liu received his Ph.D. degree in system theory from the Chinese Academy of Science. Contact him at skxliu@163.com.

LEI CHEN is an associate research fellow with the Advanced Research Institute of Multidisciplinary Science, Beijing Institute of Technology, Beijing, 100081, China. His research interests include complex networks, characteristic model, and network control. Chen received his Ph.D. degree in control theory and engineering from Southeast University. He is the corresponding author. Contact him at bit_chen@bit.edu.cn.