JOURNAL OF AEROSPACE INFORMATION SYSTEMS
Vol. 18, No. 8, August 2021

Computational Missile Guidance: A Deep Reinforcement Learning Approach

Shaoming He*
Beijing Institute of Technology, 100081 Beijing, People's Republic of China
and
Hyo-Sang Shin† and Antonios Tsourdos‡
Cranfield University, Cranfield, England MK43 0AL, United Kingdom

https://doi.org/10.2514/1.I010970

This paper aims to examine the potential of using the emerging deep reinforcement learning techniques in missile guidance applications. To this end, a Markovian decision process that enables the application of reinforcement learning theory to solve the guidance problem is formulated. A heuristic way is used to shape a proper reward function that has tradeoff between guidance accuracy, energy consumption, and interception time. The state-of-the-art deep deterministic policy gradient algorithm is used to learn an action policy that maps the observed engagements states to a guidance command. Extensive empirical numerical simulations are performed to validate the proposed computational guidance algorithm.

I. Introduction

PROPORTIONAL navigation guidance (PNG) law and its variants have been widely used in missile guidance systems due to their effectiveness and simple implementation [1]. The basic idea of PNG is to generate a lateral guidance command to nullify the zero-effort-miss (ZEM) distance in finite time [2]. The PNG with navigation gain three is theoretically proved to be energy optimal for constant moving vehicles [1] and also optimal in terms of terminal velocity maximization [3]. With the increasing complexity of application scenarios, however, real-world guidance problems in autonomous aerospace systems will be characterized by numerous practical constraints and highly time-varying, nonlinear dynamics. Even though PNG and its variants can be used to control the impact angle and impact time [4–6], the guidance command is derived based on linearized kinematics by ignoring the aerodynamic forces. Therefore, classical closed-form guidance laws, such as PNG, that rely on approximated models with linearization and idealistic assumptions, are no longer appealing to solve future real-world guidance problems.

Thanks to the rapid development on embedded computational capability, there has been an increasing attention on the development of computational guidance algorithms in recent years [7,8]. Unlike classical optimal guidance laws, computational guidance algorithms generate the guidance command, relying extensively on onboard computation, and therefore does not require analytic solution of specific guidance laws. Generally, computational guidance can be classified into two main categories: 1) model based and 2) data based. The authors in [9] proposed a model-based three-dimensional computational guidance algorithm with terminal flight path angle constraints using model predictive static programming (MPSP) [10]. This basic idea behind MPSP is that it converts a dynamic programming problem into a static programming problem and therefore is computationally efficient. Because of this property, MPSP algorithm was later used in many practical guidance problems, e.g., impact-angle control guidance [11] and reentry guidance [12]. However, the major limitation of MPSP-based computational guidance algorithms is that they require a good initial solution guess to guarantee the convergence [13].

Notice that model-based guidance algorithms are generally designed under the assumption that the model information is fully known. It is clear that the performance of model-based optimization approaches highly relies on the accuracy of the model used. In our previous work [14], we demonstrated that data-based learning algorithm could provide performance improvement to the flight controller in the presence of model uncertainties, compared with model-based approaches. However, the potential of this concept for guidance algorithm development has not been examined. To this end, it would be more beneficial to develop data-based guidance algorithms for guidance problems that suffer from uncertainties, e.g., target movement and aerodynamic force. Considering the properties of the guidance problem, leveraging the reinforcement learning (RL) concept might be most appropriate for developing a data-based guidance algorithm [15,16]. Previous works using RL to solve control problems mainly focused on the applications of robotics, with few works addressing aerospace guidance problems. The authors in [17] developed an RL guidance law, and this work seems to be the first paradigm in this domain. However, only one single fixed scenario is considered in this paper. Very recently, the emerging deep RL techniques have been applied to the Mars powered descent guidance [18,19]. As learning from scratch is generally time-consuming, the authors in [20] used RL to learn the gains of classical impact angle guidance law and applied this algorithm in relative motion guidance in near-rectilinear orbit.

The main objective of this paper is to examine the potential of using the emerging deep RL techniques in missile guidance applications. To this end, we formulate the guidance problem in an RL framework by considering the engagement kinematics as the environment and the acceleration command as the agent action. Notice that the reward function determines the convergence of the learning process and the performance of the trained agent. A heuristic reward function that provides tradeoff between guidance accuracy, energy consumption, and interception time is proposed. The state-of-the-art policy gradient algorithm, i.e., deep deterministic policy gradient (DDPG), is used to learn a deterministic action function that maps the observed engagement states to a guidance command. In this paper, we examined two different categories of learning agents:

1) Learning from scratch: Directly learn the guidance command from data.

2) Learning with prior knowledge: Fix the guidance command as a feedback form and then learn the guidance gain.

Extensive numerical analysis reveals that the proposed DDPG guidance algorithms guarantee high interception accuracy with

Received 29 January 2021; revision received 29 April 2021; accepted for publication 3 May 2021; published online Open Access 28 June 2021. Copyright © 2021 by Hyo-Sang Shin. Published by the American Institute of Aeronautics and Astronautics, Inc., with permission. All requests for copying and permission to reprint should be submitted to CCC at www.copyright.com; employ the eISSN 2327-3097 to initiate your request. See also AIAA Rights and Permissions www.aiaa.org/randp.

*Associate Professor, School of Aerospace Engineering, 5 South Zhongguancun Road; shaoming.he@bit.edu.cn. Member AIAA.

†Professor, School of Aerospace, Transport and Manufacturing, College Road; h.shin@cranfield.ac.uk. Member AIAA.

‡Professor, School of Aerospace, Transport and Manufacturing, College Road; a.tsourdos@cranfield.ac.uk. Senior Member AIAA.

acceptable performance under various scenarios. Also, the numerical analysis shows that learning with prior knowledge is helpful in improving the learning efficacy and is demonstrated to provide better performance, compared with learning from scratch.

II. Reinforcement Learning

In the RL framework, an agent learns an action policy through episodic interaction with an unknown environment. The RL problem is often formalized as a Markov decision process (MDP) or a partially observable MDP (POMDP). An MDP is described by a five-tuple $(\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{P}, \mathcal{R})$, where $\mathcal{S}$ refers to the set of states, $\mathcal{O}$ the set of observations, $\mathcal{A}$ the set of actions, $\mathcal{P}$ the state transition probability, and $\mathcal{R}$ the reward function. If the process is fully observable, we have $\mathcal{S} = \mathcal{O}$. Otherwise, $\mathcal{S} \neq \mathcal{O}$.

At each time step t, an observation $o_t \in \mathcal{O}$ is generated from the internal state $s_t \in \mathcal{S}$ and given to the agent. The agent uses this observation to generate an action $a_t \in \mathcal{A}$ that is sent to the environment, based on specific action policy π. The action policy is a function that maps observations to a probability distribution over the actions. The environment then leverages the action and the current state to generate the next state s_{t+1} with conditional probability $\mathcal{P}(s_{t+1}|s_t, a_t)$ and a scalar reward signal $r_t \sim \mathcal{R}(s_t, a_t)$. For any trajectory in the state-action space, the state transition in RL is assumed to follow a stationary transition dynamics distribution with conditional probability satisfying the Markov property, i.e.,

$$\mathcal{P}(s_{t+1}|s_1, a_1, \ldots, s_t, a_t) = \mathcal{P}(s_{t+1}|s_t, a_t) \tag{1}$$

The goal of RL is to seek a policy for an agent to interact with an unknown environment while maximizing the expected total reward it received over a sequence of time steps. The total reward in RL is defined as the summation of discounted reward to facilitate temporal credit assignment as

$$R_t = \sum_{i=t}^{N} \gamma^{i-t} r_i \tag{2}$$

where $\gamma \in (0, 1]$ denotes the discount factor. The expected total reward is then given by

$$J(\pi) = \mathbb{E}_\pi[R_t|s_t] \tag{3}$$

Given current state s_t, the expected total reward is also known as the value function $V^\pi(s_t) = \mathbb{E}_\pi[R_t|s_t]$. According to Bellman equation, the value function satisfies the following recursion:

$$V^\pi(s_t) = \mathbb{E}_\pi[\mathcal{R}(s_t, a_t)] + \gamma V^\pi(s_{t+1}) \tag{4}$$

The optimal policy can be obtained by maximizing the value function as

$$\pi^* = \arg\max_\pi V^\pi(s_t) \tag{5}$$

Many approaches in RL also make use of the action-value function

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi[R_t|s_t, a_t] \tag{6}$$

According to Eq. (4), the action-value function also satisfies a recursive form as

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi[\mathcal{R}(s_t, a_t) + \gamma \mathbb{E}_\pi[Q^\pi(s_{t+1}, a_{t+1})]] \tag{7}$$

Therefore, the optimal policy can also be obtained by optimizing the action-value function. However, directly optimizing value function or action-value function requires accurate model information and therefore is difficult to implement with model uncertainties. Model-free RL algorithms relax the requirement on accurate model information and hence can be used even with high model uncertainties.

Generally, model-free RL algorithms can be categorized into two classes: value function methods and policy gradient approaches. Value function approaches leverage a nonlinear mapping, e.g., neural network, to approximate the value function and greedily finds the action by iteratively evaluating the value function based on Bellman optimality condition. These approaches randomly explore the action space and consider all possible actions during each iteration. Therefore, value function algorithms only work with discrete action spaces, and the well-known deep Q learning [21] belongs to this category. As a comparison, the policy gradient algorithms learn a deterministic function that directly maps the states to the actions, rather than taking the action that globally maximizes the value function. The action function is updated by following the gradient direction of the value function with respect to the action, thus termed as "policy gradient." Thanks to this property, the policy gradient algorithms are applicable to continuous control problems. The DDPG algorithm, proposed by Google Deepmind [22], is one of the state-of-the-art solutions that belong to the policy gradient approach.

A. Deep Deterministic Policy Gradient

For the autopilot problem, the main goal is to find a deterministic and continuous actuator command that could drive the air vehicle to track the target lateral acceleration command in a rapid and stable manner. For this problem, we use the DDPG algorithm to develop a computational lateral acceleration autopilot for an air vehicle. DDPG is an actor–critic method that consists of two main function blocks: 1) critic evaluates the given policy based on current states to calculate the action-value function; 2) actor generates policy based on the evaluation of critic. DDPG uses two different deep neural networks, i.e., actor network and critic network, to approximate the action function and the action-value function. The basic concept of DDPG is shown in Fig. 1.

Denote $A^\mu(s_t)$ as the deterministic policy, which is a function that directly maps the states to the actions, i.e., $a_t = A^\mu(s_t)$. Here, we assume that the action network $A^\mu(s_t)$ is parameterized by μ. In DDPG, the actor function is optimized by adjusting the parameter μ toward the gradient of the expected total reward as [22]

$$\begin{aligned} \nabla_\mu J(A^\mu) &= \nabla_\mu Q^w(s_t, A^\mu(s_t)) \\ &= \nabla_\mu A^\mu(s_t) \nabla_{a_t} Q^w(s_t, a_t) \end{aligned} \tag{8}$$

where $Q^w(s_t, a_t)$ stands for the action-value function, which is parameterized by w.

The parameter μ is then updated by moving the policy in the direction of the gradient of Q^w in a recursive way as

$$\mu_{t+1} = \mu_t + \alpha_\mu \nabla_\mu A^\mu(s_t) \nabla_{a_t} Q^w(s_t, a_t) \tag{9}$$

where α_μ refers to the learning rate of the actor network.

Similar to Q learning, DDPG also uses the temporal-difference (TD) error δ_t in approximating the error of action-value function as

$$\delta_t = r_t + \gamma Q^w(s_{t+1}, A^\mu(s_{t+1})) - Q^w(s_t, a_t) \tag{10}$$

DDPG uses the square of TD error as the loss function $\mathcal{L}(w)$ in updating the critic network, i.e.,

$$\mathcal{L}(w) = \delta_t^2 \tag{11}$$

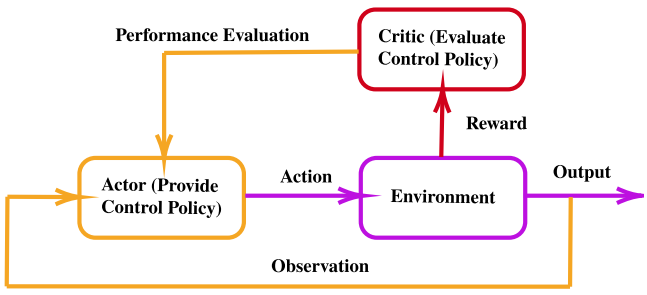


Fig. 1 Basic concept of DDPG.

Taking the partial derivative of $\mathcal{L}(w)$ respect to w gives

$$\nabla_w \mathcal{L}(w) = -2\delta_i \nabla_w Q^w(s_t, a_t) \tag{12}$$

The parameter w is then updated using gradient descent by following the negative gradient of $\mathcal{L}(w)$ as

$$w_{t+1} = w_t + \alpha_w \delta_t \nabla_w Q^w(s_t, a_t) \tag{13}$$

where α_w stands for the learning rate of the critic network.

One major issue of using deep neural networks in RL is that most neural network optimization algorithms assume that the samples for training are independently and identically distributed. However, this assumption is violated if the training samples are directly generated by sequentially exploring the environment. To resolve this issue, DDPG leverages a mini batch buffer that stores the training samples using the experience replay technique. Denote $e_t = (s_t, a_t, r_t, s_{t+1})$ as the transition experience of the tth step. DDPG uses a buffer $\mathcal{D}$ with its size being $|\mathcal{D}|$ to store transition experiences. DDPG stores the current transition experience in the buffer and deletes the oldest one if the number of the transition experience reaches the maximum value $|\mathcal{D}|$. At each time step during training, DDPG uniformly draws N transition experience samples from the buffer $\mathcal{D}$ and uses these random samples to train actor and critic networks. By using the experience buffer, the critic network is updated as

$$\nabla_\mu J(A^\mu) = \frac{1}{N}\sum_{i=1}^{N} \nabla_\mu A^\mu(s_i) \nabla_{a_t} Q^w(s_i, a_i)$$
$$\mu_{t+1} = \mu_t + \alpha_\mu \nabla_\mu J(A^\mu) \tag{14}$$

With N transition experience samples, the loss function in updating the critic network now becomes

$$\mathcal{L}(w) = \frac{1}{N}\sum_{i=1}^{N} \delta_i^2 \tag{15}$$

The parameter of the critic network is updated by gradient descent as

$$\nabla_w \mathcal{L}(w) = \frac{1}{N}\sum_{i=1}^{N} \delta_i \nabla_w Q^w(s_i, a_i)$$
$$w_{t+1} = w_t + \alpha_w \nabla_w \mathcal{L}(w) \tag{16}$$

Notice that the update of the action-value function is also used as the target value as shown in Eq. (10), which might cause the divergence of critic network training [22]. To address this problem, DDPG creates one target actor network and one target critic network. Suppose that the additional actor and critic networks are parameterized by μ' and w', respectively. These two target networks use soft update, rather than directly copying the parameters from the original actor and critic networks, as

$$\mu' = \tau\mu + (1-\tau)\mu' \quad w' = \tau w + (1-\tau)w' \tag{17}$$

where $\tau \ll 1$ is a small update rate. This soft update law shares similar concept as low-frequency learning in model reference adaptive control to improve the robustness of the adaptive process [23,24].

The soft-updated two target networks are then used in calculating the TD error as

$$\delta_t = r_t + \gamma Q^{w'}(s_{t+1}, A^{\mu'}(s_{t+1})) - Q^w(s_t, a_t) \tag{18}$$

With very small update rate, the stability of critic network training greatly improves at the expense of slow training process. Therefore, the update rate is a tradeoff between training stability and convergence speed.

B. Training a DDPG Agent

DDPG is an off-policy learning algorithm and is trained in an episodic style. The environment initializes an episode by randomly generating internal states and mapping the internal states to observations. This random initialization allows the agent to explore the diversity of the state space. At the beginning of each episode, both actor and critic networks are initialized with random weights. The target actor and target critic networks directly copy the random weights from the original networks. During each episode, the actor and critic networks are updated using gradient descent according to Eqs. (14) and (16), and the target networks are trained by soft update as Eq. (17). The episode is terminated if the number of steps reaches the maximum value or the agent completes the task.

A major issue of learning in a continuous space is how to explore the state space to escape from the local minima of the total reward function. DDPG addresses this problem by adding a random noise v_t to the action generated by the actor network

$$a_t' = a_t + v_t \tag{19}$$

and using this new noise-corrupted action for system propagation. In DDPG, the random noise v_t is updated recursively using an Ornstein–Uhlenbeck process, which is defined as

$$v_t = v_{t-1} + \beta_{\text{attract}}(\mu_v - v_{t-1})T_s + \mathcal{N}(0, \Sigma_t)\sqrt{T_s} \tag{20}$$

where μ_v represents the mean of the noise; $\mathcal{N}(0, \Sigma_t)$ denotes the Gaussian distribution with zero mean and variance Σ_t; β_{attract} is the mean attraction constant that quantifies how quickly the noise is attracted to the mean; and T_s stands for the sampling time. With more experience gained during the training, the exploration variance Σ_t exponentially decays with rate ϵ as

$$\Sigma_t = \Sigma_{t-1}(1-\epsilon) \tag{21}$$

The advantage of the Ornstein–Uhlenbeck process is that it can generate temporally correlated explorations and thus provides smooth transitions. The detailed pseudocode of DDPG is summarized in Algorithm 1.

Algorithm 1: Deep deterministic policy gradient

1: Initialize the actor and critic networks with random weights μ and w
2: Initialize the target actor and critic networks with weights $\mu' \leftarrow \mu$ and $w' \leftarrow w$
3: Initialize the experience buffer $\mathcal{D}$
4: **for** episode = 1: MaxEpisode **do**
5: **for** t = 1: MaxStep **do**
6: Generate an action from the actor network based on current state $a_t = A^\mu(s_t)$
7: Add a random noise v_t to the action for exploration $a_{t'} = a_t + v_t$
8: Execute the action $a_{t'}$ and receive new state s_{t+1} and reward r_t
9: Store the transition experience $e_t = (s_t, a_t, r_t, s_{t+1})$ in the experience buffer $\mathcal{D}$
10: Uniformly draw N random samples e_i from the experience buffer $\mathcal{D}$
11: Calculate the TD error δ_i
$$\delta_i = r_i + \gamma Q^{w'}(s_{i+1}, A^{\mu'}(s_{i+1})) - Q^w(s_i, a_i)$$
12: Calculate the loss function $\mathcal{L}(w)$
$$\mathcal{L}(w) = \tfrac{1}{N}\sum_{i=1}^{N} \delta_i^2$$
13: Update the critic network using gradient descent as
$$\nabla_w \mathcal{L}(w) = \tfrac{1}{N}\textstyle\sum_{i=1}^{N} \delta_i \nabla_w Q^w(s_i, a_i)$$
$$w_{t+1} = w_t + \alpha_w \nabla_w \mathcal{L}(w)$$

Algorithm 1: (*Continued.*)

14: Update the actor network using policy gradient as

$$\nabla_\mu J(A^\mu) = \frac{1}{N}\sum_{i=1}^{N} \nabla_\mu A^\mu(s_i) \nabla_{a_t} Q^w(s_i, a_i)$$

$$\mu_{t+1} = \mu_t + \alpha_\mu \nabla_\mu J(A^\mu)$$

15: Update the target networks as

$$\mu' = \tau\mu + (1-\tau)\mu'$$

$$w' = \tau w + (1-\tau)w'$$

16: **if** the task is accomplished **then**
17: Terminate the current episode
18: **end if**
19: **end for**
20: **end for**

III. Reinforcement Learning Formulation of Guidance Problem

This section formulates the guidance problem in an RL framework. Before introducing the system kinematics, we make three basic assumptions as follows:

Assumption 1: Both the interceptor and the target are assumed as point-mass models.

Assumption 2: The engagement occurs in a two-dimensional (2-D) vertical plane.

Assumption 3: Both the missile and the target are flying with constant velocity.

Note that these assumptions are widely accepted in guidance law design for tactical missiles: (Assumption 1) Typical philosophy treats the guidance and control loops separately by placing the kinematic guidance system in an outer loop, generating guidance commands tracked by an inner dynamic control loop, also known as autopilot. (Assumption 2) Homing engagement can be treated as a 2-D problem for roll-stabilized airframes. (Assumption 3) The vehicle's velocity is generally slowly varying and hence can be assumed as piecewise constant.

A. Relative Kinematics

This paper considers a 2-D planar homing engagement geometry shown in Fig. 2. As presented in the geometry, the inertial reference frame is denoted as (X, Y). Variables with subscripts of M and T denote those of the missile and target, respectively. The notations of λ and r are the line-of-sight (LOS) angle and the missile–target relative range. γ denotes the flight path angle defined in the inertial reference frame. The velocity and lateral acceleration are represented by V and a, respectively.

The corresponding equations describing the missile–target relative motion kinematics can be formulated as

$$\dot{r} = V_T \cos(\gamma_T - \lambda) - V_M \cos(\gamma_M - \lambda) \tag{22}$$

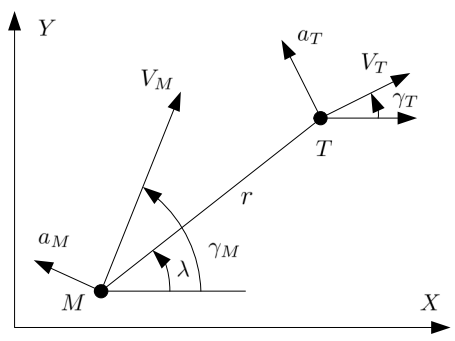


Fig. 2 The homing engagement geometry and parameter definitions.

$$r\dot{\lambda} = V_T \sin(\gamma_T - \lambda) - V_M \sin(\gamma_M - \lambda) \tag{23}$$

Let $V_r\Delta = \dot{r}$, $V_\lambda\Delta = r\dot{\lambda}$ be the relative velocities along and perpendicular to the LOS, respectively. Then, differentiating Eqs. (22) and (23) with respect to time yields

$$\dot{V}_r = \frac{V_\lambda^2}{r} + a_{Tr} - a_M \sin(\lambda - \gamma_M) \tag{24}$$

$$\dot{V}_\lambda = -\frac{V_r V_\lambda}{r} + a_{T\lambda} - a_M \cos(\lambda - \gamma_M) \tag{25}$$

where $a_{Tr}\Delta = a_T \sin(\lambda - \gamma_T)$ and $a_{T\lambda}\Delta = a_T \cos(\lambda - \gamma_T)$ denote the target acceleration along and normal to the LOS, respectively.

The complementary equations defining the relationship between the flight path angle and lateral acceleration are

$$\dot{\gamma}_M = \frac{a_M}{V_M} \tag{26}$$

$$\dot{\gamma}_T = \frac{a_T}{V_T} \tag{27}$$

Because the target maneuver is difficult to measure or obtain in practice, we assume that the target is flying with a constant course, i.e., $a_t = 0$. During real flight, the lateral acceleration is generated by an onboard autopilot, which has inevitable time delays. To account for such time delay, we assume that the missile autopilot is modeled by a first-order time lag system as

$$\dot{a}_M = -\frac{1}{\tau_a} a_M + \frac{1}{\tau_a} a_c \tag{28}$$

where τ_a denotes the autopilot time constant and a_c represents the guidance command.

B. Reinforcement Learning Problem Formulation

To solve the guidance problem using DDPG, we need to formulate the problem in the RL framework by constructing an MDP with a proper reward function.

To apply the DDPG algorithm to the guidance problem, we need to define a proper MDP. The relative kinematics, shown in Eqs. (22–27), constitutes the environment, which is fully characterized by the engagement state

$$s_t = (r, \lambda, \dot{r}, \dot{\lambda}) \tag{29}$$

For guidance law design, the agent action is naturally defined as the guidance command a_c. We assume that the missile is equipped with an active radar seeker that can measure the relative distance r, LOS angle λ, and their rates. This means that the agent observation is given by

$$o_t = (r, \lambda, \dot{r}, \dot{\lambda}) \tag{30}$$

which gives a fully observable MDP.

The relative kinematics (22–25), engagement state (29), agent observation (30), and a suitable agent action, together with a proper reward function, constitute a complete MDP formulation of the guidance problem. The conceptual flowchart of the proposed guidance RL framework is shown in Fig. 3.

C. Reward Function Shaping

The most challenging part of solving the guidance problem using DDPG is the development of a proper reward function. Notice that the primary objective of a guidance law is to drive the missile to intercept a target in a stable manner with acceptable miss distance. A naive selection of the reward function is that we give a bonus to the agent

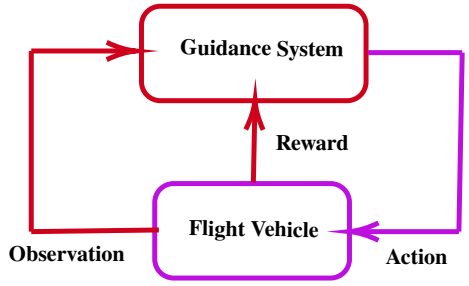


Fig. 3 Conceptual flowchart of the proposed guidance RL framework.

once the missile successfully intercepts the target and the agent is penalized with a negative reward otherwise. However, this simple reward function is demonstrated to be ineffective during our test, and the agent would never see a positive reward within a realistic number of episodes because the probability of successful interception with random initial guesses is extremely low.

It is known that nullifying the ZEM results in perfect interception with zero miss distance [1]. For this reason, we will shape the reward function using ZEM. At any time instant t, the ZEM is defined as the closest distance between the missile and the target if, from the time instant t onward, both the missile and the target do not maneuver. The ZEM, denoted as z, is formulated as [25,26]

$$z = \frac{rV_\lambda}{\sqrt{V_r^2 + V_\lambda^2}} \tag{31}$$

For aerodynamically controlled airframes, the quadratic energy consumption is also an important factor as it directly quantifies the velocity loss due to induced drag. Considering this fact, the control effort is also taken into account in the reward function. Notice that the homing phase for intercepting a ballistic target is usually very short. This motivates us to shape the reward function in a way that enables rapid interception of the target. In summary, the reward function is shaped using a heuristic way as

$$r_t = r_a + r_z + r_{V_r} + r_r \tag{32}$$

where

$$r_a = k_a\left(\frac{a_M}{|a_{\max}|}\right)^2 \tag{33}$$

$$r_z = k_z\left(\frac{z}{|z_0|}\right)^2 \tag{34}$$

$$r_{V_r} = \begin{cases} k_{V_r}, & V_r > 0 \\ 0, & V_r \le 0 \end{cases} \tag{35}$$

$$r_r = \begin{cases} k_r, & r \le r_d \\ 0, & r > r_d \end{cases} \tag{36}$$

with k_a, k_z, k_{V_r}, and k_r are four constants to shape the reward function. The notation $a_{\max}$ denotes the maximum permissible lateral acceleration of the missile due to physical constants and z_0 represents the initial value of ZEM.

From Eq. (32), it is clear that the first two terms in the reward function penalize the normalized control effort and ZEM, respectively. The reason of using normalization in the reward function is that the lateral acceleration and ZEM have different units and scales. This means that it is difficult to directly compare these two metrics in an integrated manner without normalization. The third term r_{V_r} encourages the missile to continuously reduce the relative distance, yielding fast interception. Without this term, the missile's flying trajectory might become longer and therefore requires longer engagement time. This term also penalizes the mission failure: if the missile misses the target when the time-to-go becomes zero, the relative distance between the missile and the target will increase. The fourth term r_r gives bonus to the agent if the relative distance is smaller than a positive constant r_d, thus encouraging the missile to intercept the target. Note that this term is only active when the missile is close to the target and therefore provides similar role as the terminal constraint cost in typical optimal control problems for shaping the terminal state. In a nutshell, the proposed reward function allows the agent to tradeoff between interception accuracy, energy consumption, and interception time. The hyperparameters in shaping the reward function are summarized in Table 1.

Table 1 Hyperparameters in shaping the reward function

k_a	k_z	k_{V_r}	k_r	$a_{\max}$	r_d
−0.2	−1	−2	10	100	20

IV. Training a DDPG Guidance Agent from Scratch

In this section, we will propose a DDPG guidance agent that directly learns the guidance command a_c during the training process. In other words, this section aims to use DDPG to provide a direct mapping from engagement states to guidance command, i.e.,

$$a_c = f_s(r, \lambda, \dot{r}, \dot{\lambda}) \tag{37}$$

where f_s is a nonlinear function.

Generally, training a DDPG agent involves three main steps: 1) obtaining training scenarios; 2) building the actor and critic networks; and 3) tuning the hyperparameters.

A. Training Scenarios

In this paper, we consider a head-on engagement scenario. For better using the learned experience, we select several characteristic initial engagement conditions, e.g., states with their maximum, minimum, and medium values, as shown in Table 2, and train the DDPG agent based on these initial conditions sequentially. More specifically, the DDPG agent is trained using one fixed characteristic initial condition and switches to another initial condition after convergence. Through larger empirical tests, we found that the proposed DDPG agent converges within less than 100 episodes for one single initial condition. After the training process using characteristic initial conditions is finished, we randomly initialize the engagement states with values uniformly distributed between the minimum and the maximum values at the beginning of each episode. This heuristic training process is shown to perform much faster than starting training based on random initial conditions only.

B. Network Construction

Inspired by the original DDPG algorithm [22], the actor and critic are represented by four-layer fully connected neural networks. Note that this four-layer network architecture is commonly used in deep RL applications [27]. The layer sizes of these two networks are summarized in Table 3. Except for the actor output layer, each neuron in other layers is activated by a rectified linear units (Relu) function, which is defined as

Table 2 Initial conditions in training

Parameter	Minimum value	Maximum value
Relative range, r	4000 m	6000 m
LOS angle, λ	−10°	10°
Missile's flight path angle, γ_M	0°	20°
Target's flight path angle, γ_T	140°	160°
Autopilot time constant, τ_a	0.1 s	0.3 s

Table 3 Network layer size

Layer	Actor network	Critic network
Input layer	4 (size of states)	5 (size of states + size of action)
Hidden layer 1	30	50
Hidden layer 2	20	40
Output layer	1 (size of action)	1 (size of action-value function)

$$g(z) = \begin{cases} z, & \text{if } z \geq 0 \\ 0, & \text{if } z < 0 \end{cases} \tag{38}$$

which provides faster processing speed than other nonlinear activation functions due to the linear relationship property.

The output layer of the actor network is activated by the tanh function, which is given by

$$g(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}} \tag{39}$$

The benefit of the use of tanh activation function in actor network is that it can prevent the control input from saturation as the actor output is constrained by $(-1, 1)$. The output layer of the actor network is scaled by a constant $a_{\max}$.

As different states have different scales and units, we normalize the engagement states and action at the input layers of the networks, thus providing unitless observations and action that belong to approximately the same scale. This normalization procedure is shown to be of paramount importance for our problem and helps to increase the training efficiency. Without normalization, the average reward function cannot converge and even shows divergent patterns after 10^3 episodes. Denote $(\cdot)$ as the normalized version of variable $(\cdot)$. The normalization of states and action are defined as

$$\bar{r} = \frac{r}{r_0}, \quad \bar{\lambda} = \frac{\lambda}{\lambda_0}, \quad \bar{\dot{r}} = \frac{\dot{r}}{\dot{r}_0}, \quad \bar{\dot{\lambda}} = \frac{\dot{\lambda}}{\dot{\lambda}_0}, \quad \bar{a}_c = \frac{a_c}{a_{\max}} \tag{40}$$

where $(\cdot)_0$ stands for the initial value of variable $(\cdot)$.

Both actor and critic networks are trained using Adam optimizer with $\mathcal{L}_2$ regularization to address the overfitting problem for stabilizing the learning process. With $\mathcal{L}_2$ regularization, the updates of actor and critic are modified as

$$\mathcal{L}_{\text{actor}} = J(A^\mu) + \lambda_2 \mathcal{L}_2^A \quad \mu_{t+1} = \mu_t + \alpha_\mu \nabla_\mu \mathcal{L}_{\text{actor}} \tag{41}$$

$$\mathcal{L}_{\text{critic}} = \mathcal{L}(w) + \lambda_2 \mathcal{L}_2^C \quad w_{t+1} = w_t + \alpha_w \nabla_w \mathcal{L}_{\text{critic}} \tag{42}$$

where $\mathcal{L}_2^A$ and $\mathcal{L}_2^C$ denote the $\mathcal{L}_2$ regularization losses on the weights of the actor and the critic, respectively; λ_2 is the regularization constant.

To increase the stability of the network training process, we use the gradient clip technique to constrain the update of both actor and critic networks. More specifically, if the norm of the gradient exceeds a given upper bound ρ, the gradient is scaled to equal with ρ. This helps to prevent a numerical overflow or underflow during the training process.

C. Hyperparameter Tuning

Each episode during training is terminated when the relative distance between the missile and the target is lower than a threshold $r_m = 2$ m or the number of time steps exceeds the maximum permissible value. All hyperparameters that are used in DDPG training for our problem are summarized in Table 4. Notice that the tuning of hyperparameters imposes great effects on the performance of DDPG, and this tuning process is not consistent across different ranges of applications [27,28], i.e., different works used different set of hyperparameters for their own problems. For this reason, we tune these hyperparameters for our guidance problem based on several trial-and-error tests.

Table 4 Hyperparameter settings

Parameter	Value
Maximum permissible steps	500
Maximum permissible episodes	1000
Actor learning rate, α_μ	10^{-3}
Critic learning rate, α_w	10^{-3}
$\mathcal{L}_2$ regularization constant, λ_2	6×10^{-3}
Gradient upper bound, ρ	1
Discounting factor, γ	0.99
Size of experience buffer, $\lvert\mathcal{D}\rvert$	5×10^5
Size of mini-batch samples, N	64
Mean of exploration noise, μ_v	0
Initial variance of exploration noise, Σ_1	0.1
Variance decay rate, ϵ	10^{-6}
Mean attraction constant, β_{attract}	0.15
Target network smoother constant, τ	0.1

D. Simulation Results

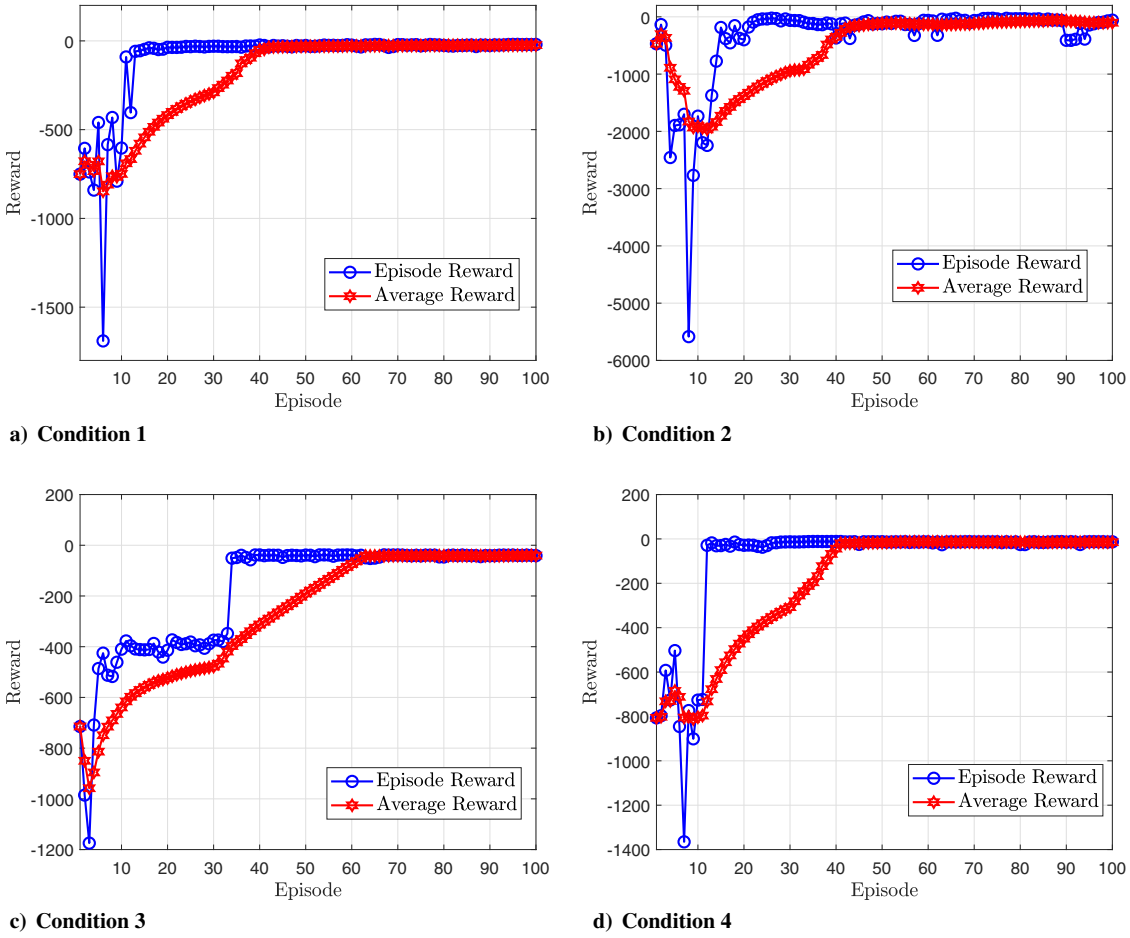
1. *Training Results*

As discussed before, we first train our DDPG guidance agent using 10 fixed characteristic initial conditions sequentially. Once the average reward of one scenario converges to a certain steady-state value, the training process switches to another initial condition. We use 100 episodes for each scenario to train our DDPG guidance agent. Figure 4 presents the learning curves with four representative fixed initial conditions. The average reward is obtained by averaging the episode reward within 30 episodes. From this figure, it can be observed that the proposed DDPG guidance agent guarantees convergence within 100 episodes for all representative scenarios. Once the training process using characteristic scenarios is finished, each episode is then initialized with random engagement states with values uniformly distributed between the minimum and the maximum values. The learning curves of the training process with random initial conditions are shown in Fig. 5. It can be clearly noted from this figure that the average reward of the proposed DDPG guidance agent converges to its steady-state value within 100 episodes, even with random initial conditions. The reason is that the agent has already gained some experience during training using representative scenarios. In our numerical tests, we found that the proposed heuristic training process provides much faster convergence rate than starting training based on random initial conditions only.

To show the importance of observation and action normalization in training DDPG guidance agent, Fig. 6 presents the comparison results of average reward function for conditions 1 and 2. From this figure, it is clear that using normalization provides fast convergence rate of the learning process and higher steady-state value of the average reward function. This means that leveraging observation and action normalization helps to achieve more efficient and effective training process. The reason can be attributed to the fact that normalization imposes equally importance on each element of the observation vector. Without normalization, the scale difference between the elements varies in a great deal, e.g., the magnitude of the relative range is much larger than that of the LOS rate and therefore prohibits effective training of the actor and critic networks.

2. *Test Results*

To test the proposed DDPG guidance agent under various conditions, the trained agent is applied to some random scenarios and the results are presented in Fig. 7. From Figs. 7a and 7b, one can note that the proposed computational guidance algorithm successfully drives the missile to intercept the target with different initial conditions. The miss distances for all tested cases are smaller than 2 m recorded in our simulations. The achieved accelerations produced by the onboard autopilot are given in Fig. 7c, which reveals that the proposed computational guidance algorithm provides smooth guidance command with



a) Condition 1 b) Condition 2

c) Condition 3 d) Condition 4

Fig. 4 Learning curves with some fixed characteristic initial conditions.

acceptable energy consumption. As shown in Fig. 7c, the guidance commands of some scenarios converge to approximate zero, but the lateral acceleration commands of some scenarios are also close to saturation when the missile is close to the target.

Notice that the proposed DDPG guidance agent is trained by assuming that the target performs no evasive maneuvers. For this reason, we also perform numerical simulations to evaluate the robustness against random target maneuvers. At each round of test, the target lateral acceleration a_T is randomly sampled from a uniform distribution, with minimum value being -20 m/s^2 and maximum value being 20 m/s^2. The simulation results, including flight trajectory, relative range, and acceleration response, are shown in Fig. 8. From this figure, it can be observed that the proposed DDPG guidance algorithm also guarantees target interception with satisfactory performance. However, the responded acceleration exhibits undesired oscillations for some scenarios. Further investigation and parameter tuning are required to alleviate this issue.

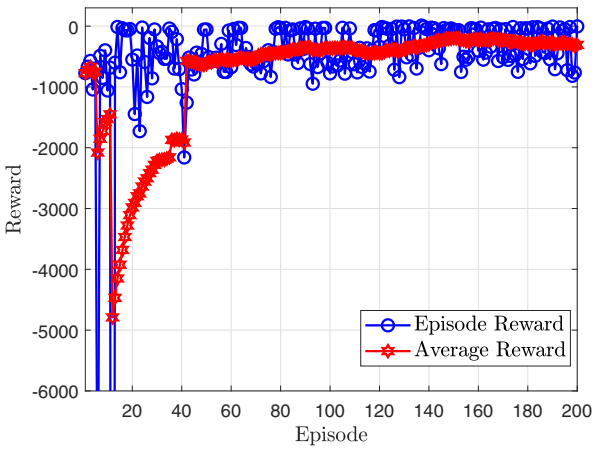
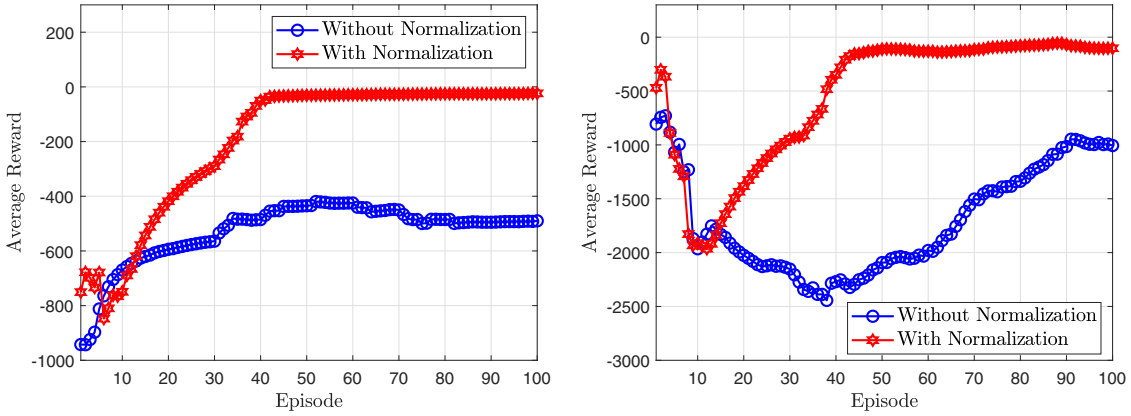


Fig. 5 Learning curves with random initial conditions.

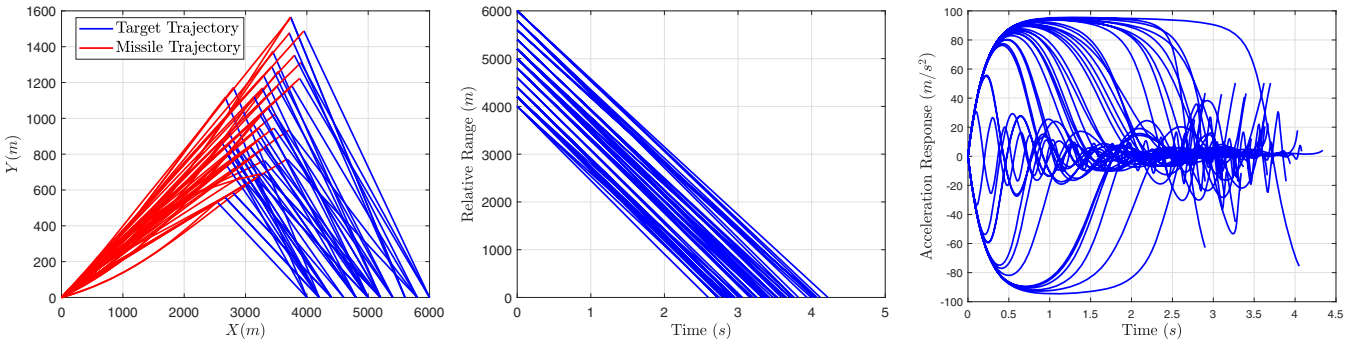
3. *Comparison with Proportional Navigation Guidance*

To further show the advantage of the proposed DDPG guidance algorithm, we compare the proposed approach with the benchmark PNG. As stated in [1], the navigation gain of PNG in real applications usually belongs to [3,5]. For this reason, both PNGs with navigation as three and five are performed in the simulations for the purpose of comparison. The comparison results for two representative scenarios are presented in Figs. 9 and 10, respectively. From this figure, it can be observed that all guidance laws successfully drive the missile to intercept the target, but the proposed DDPG guidance algorithm provides slightly reduced energy consumption, compared with other two approaches. As discussed before, one benefit of the proposed computational guidance algorithm is that it can avoid command saturation due to the nature of tanh activation function, as confirmed by Fig. 9b. For the second scenario, it can be noted from Fig. 10e that the proposed approach provides converged acceleration response. Hence, the proposed computational guidance algorithm is helpful in providing more operational margins to cope with undesired disturbances, e.g., wind, when the missile approaches the target. Another potential benefit of using RL computational guidance is that we can shape the guidance command as desired by properly tuning the reward function.



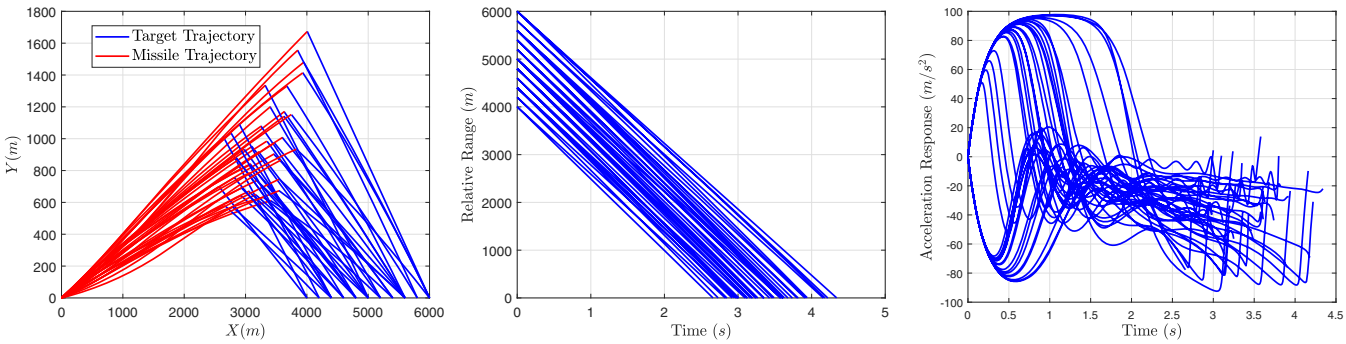
a) Condition 1 b) Condition 2

Fig. 6 Learning process comparison with respect to normalization.



a) Flight trajectory b) Relative range c) Commanded response

Fig. 7 Test results with random initial conditions for nonmaneuvering target.



a) Flight trajectory b) Relative range c) Commanded response

Fig. 8 Test results with random initial conditions for maneuvering target.

V. Training a DDPG Guidance Agent with Prior Knowledge

Even though directly learning the guidance command provides satisfactory performance for some scenarios, our tests reveal that this learning strategy might not guarantee target interception for some specific initial conditions. Therefore, instead of directly learning the guidance command from scratch, this section suggests another way to train a DDPG guidance agent. Over the past several decades, classical PNG is widely used in many missile guidance systems due to easy implementation and efficacy. The main idea behind PNG is to generate a lateral acceleration to nullify the LOS rate so as to force the missile to converge to the collision triangle. It has also been proved that the ZEM under PNG converges to zero at the time of impact against nonmaneuvering targets [2]. Analogous to PNG, we fix the guidance command using classical proportional control concept as

$$a_c = N\frac{z}{t_{\text{go}}^2} \tag{43}$$

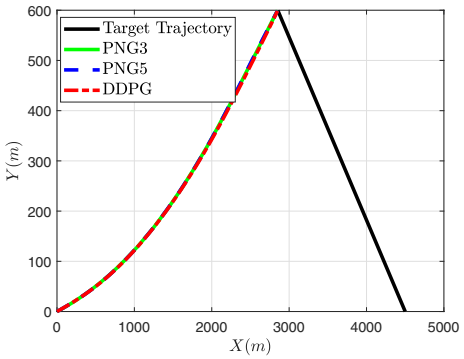
where N is a time-varying guidance gain and t_{go} represents the remaining flight time, which is approximated as

$$t_{\text{go}} = -\frac{r}{\dot{r}} \tag{44}$$

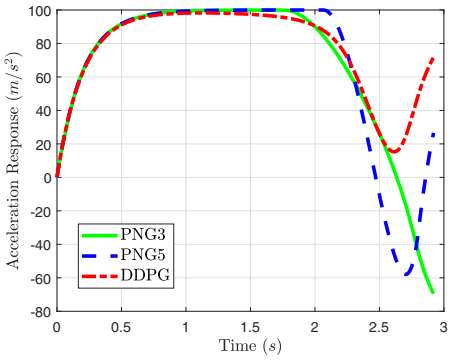
Then, DDPG is used to provide a direct mapping from engagement states to guidance gain, i.e.,

$$N = f_n(r, \lambda, \dot{r}, \dot{\lambda}) \tag{45}$$

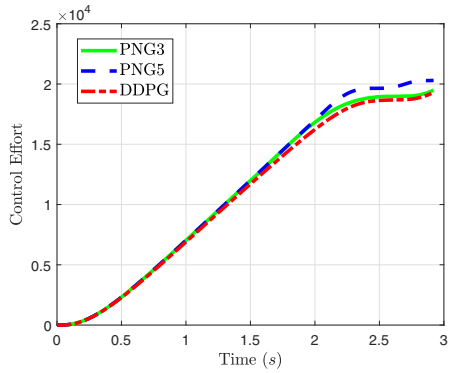
where f_n is a nonlinear function.



a) Flight trajectory

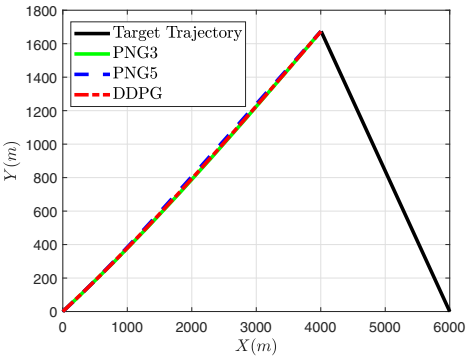


b) Acceleration response

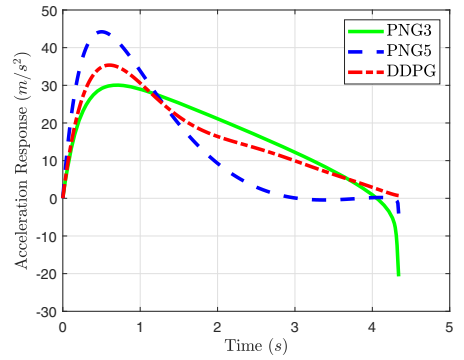


c) Control effort

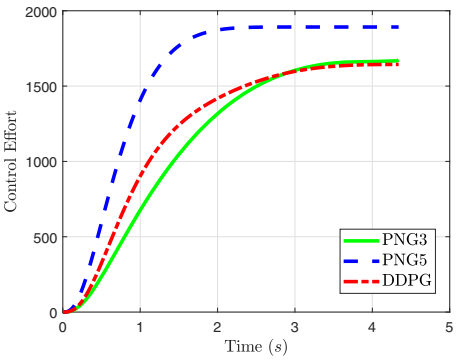
Fig. 9 Comparison results of scenario 1.



a) Flight trajectory



b) Acceleration response



c) Control effort

Fig. 10 Comparison results of scenario 2.

The reward function, network structure, and hyperparameters are the same as previous section except that the out layer of the actor is scaled by a constant $N_{\max} = 10$.

A. Simulation Results

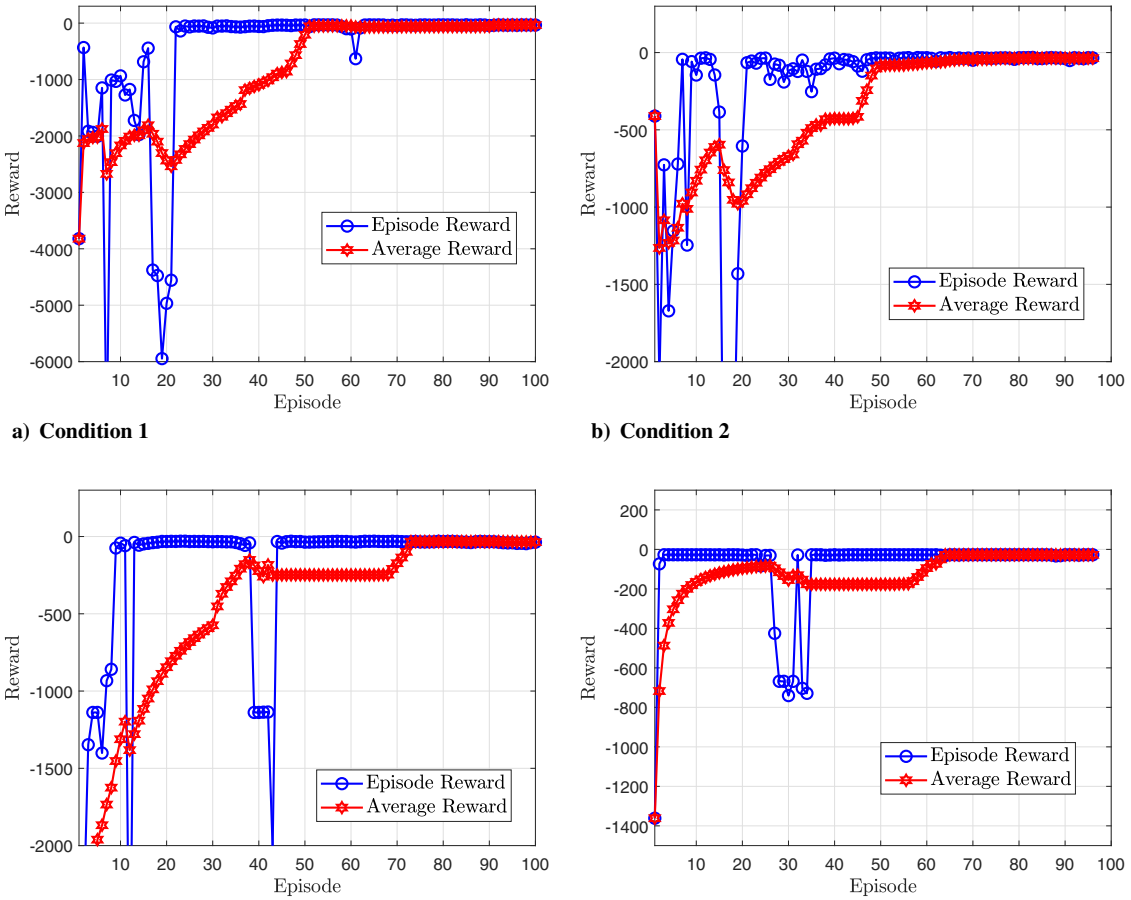
1. *Training Results*

Similar to previous section, we first train our DDPG guidance agent using 10 fixed characteristic initial conditions sequentially. Once the average reward of one scenario converges to a certain steady-state value, the training process switches to another initial condition. We use 100 episodes for each scenario to train our DDPG guidance agent. Figure 11 presents the learning curves with four representative fixed initial conditions. The average reward is obtained by averaging the episode reward within 30 episodes. From this figure, it can be observed that the proposed DDPG guidance agent guarantees convergence within 100 episodes for all representative scenarios. Once the training process using characteristic scenarios is finished, each episode is then initialized with random engagement states with values uniformly distributed between the minimum and the maximum values. The learning curves of the training process with random initial conditions are shown in Fig. 12. It can be clearly noted from this figure that the average reward of the proposed DDPG guidance agent converges to its steady-state value within 100 episodes, even with random initial conditions.

2. *Test Results*

Similar to previous section, we test our DDPG guidance agent using random scenarios with and without target maneuver. The results for scenarios without target maneuver are provided in Fig. 13, and the results for scenarios with random target maneuvers are given in Fig. 14. From these figures, it can be observed that the proposed DDPG guidance agent provides converged guidance command for all scenarios. Similar to classical PNG, the guidance algorithm developed guarantees near zero final guidance command against nonmaneuvering targets if the interceptor has enough maneuverability. Compared with directly learning guidance commands, learning guidance gain provides smoother guidance commands with less command oscillations and therefore is more desired for practical applications.

To better show the advantages of learning with prior knowledge, Monte Carlo simulations are performed to compare the two learning



a) Condition 1 b) Condition 2

c) Condition 3 d) Condition 4

Fig. 11 Learning curves with some fixed characteristic initial conditions.

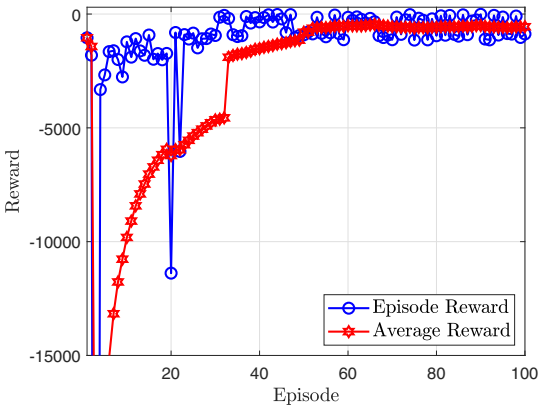
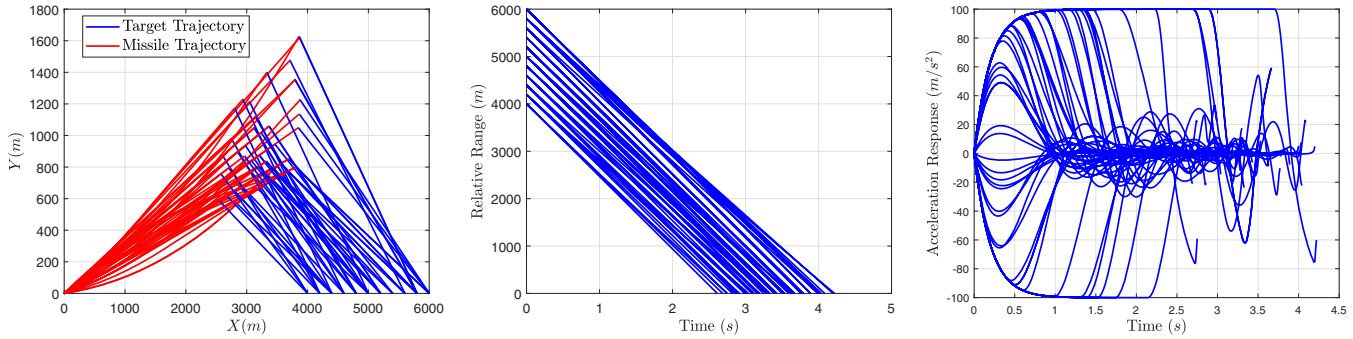


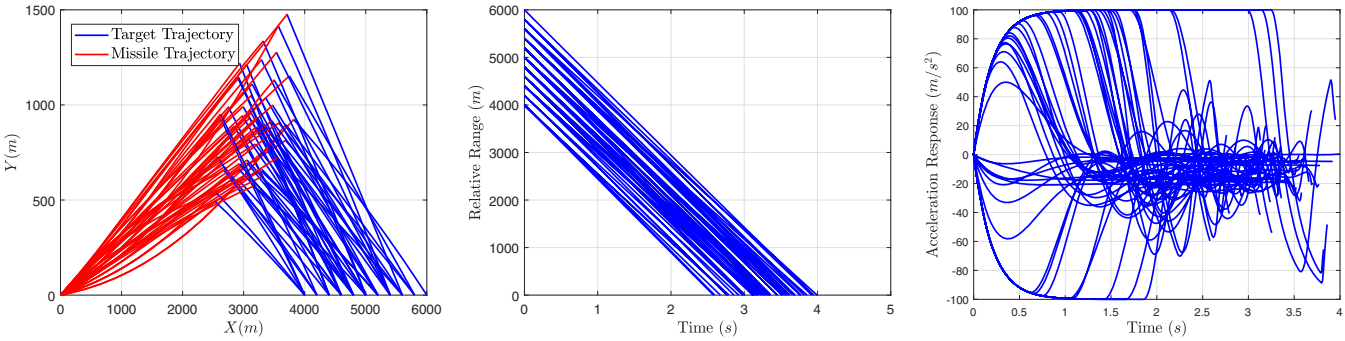
Fig. 12 Learning curves with random initial conditions.

strategies that have been developed in this paper. To test the robustness of the proposed formulations, the simulations are carried out in noise-corrupted environment, and the statistical characteristics of the considered measurement noise are summarized in Table 5. The detailed statistical comparisons of two different learning strategies are summarized in Table 6. The successful rate is defined by the number of scenarios with miss distance less than 2 m over the total number of scenarios. Because the actions of these two different learning strategies are given in a nonlinear feedback form of the engagement states, these two computational guidance algorithms show strong robustness against the undesired noise; e.g., the performance variation is small. However, it is clear from Table 5 that leveraging prior knowledge significantly outperforms learning from scratch: using prior knowledge during learning increases the expected total reward and thus provides higher successful interception rate.



a) Flight trajectory b) Relative range c) Commanded response

Fig. 13 Test results with random initial conditions for nonmaneuvering target.



a) Flight trajectory b) Relative range c) Commanded response

Fig. 14 Test results with random initial conditions for maneuvering target.

Table 5 Statistical characteristics of measurement noise

Parameter	Mean	Standard deviation
Range, r	0	20 m
Range rate, $\dot{r}$	0	5 m/s
LOS angle, λ	0	0.2 rad
LOS rate, $\dot{\lambda}$	0	0.2 (m · rad)/s

Table 6 Statistical comparisons of two different learning strategies

			Total reward			
Strategy	Target maneuver	Successful rate, %	Mean	Standard deviation	Maximum	Minimum
From scratch	No	61.29	−9.1103	22.4912	−0.7997	−146.6742
	Random	58.07	−6.3011	3.9781	−2.2097	−16.3913
With prior knowledge	No	77.93	−4.7860	5.0917	−0.4975	−20.2109
	Random	74.01	−4.5001	4.4909	−0.9003	−25.3378

VI. Conclusions

A computational guidance algorithm has been developed for missile–target interception using deep RL techniques. A heuristic reward function is proposed to encourage the missile to intercept the target in a rapid and stable manner with acceptable energy consumption. The state-of-the-art DDPG approach is leveraged to train an RL agent with a deterministic action policy that maximizes the expected total reward. Extensive numerical simulations validate the effectiveness of the proposed approach. Future work includes solving the deep RL guidance problem using a more realistic model that contains aerodynamic forces, thrust, and target maneuver. Validating the proposed computational guidance algorithm under uncertain environment is also an important issue and requires further explorations.

References

[1] Zarchan, P., *Tactical and Strategic Missile Guidance*, AIAA, Reston, VA, 2012, Chap. 2.

[2] He, S., and Lee, C.-H., "Optimality of Error Dynamics in Missile Guidance Problems," *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 7, 2018, pp. 1624–1633. https://doi.org/10.2514/1.G003343

[3] Jeon, I. S., Karpenko, M., and Lee, J. I., "Connections Between Proportional Navigation and Terminal Velocity Maximization Guidance," *Journal of Guidance, Control, and Dynamics*, Vol. 39, No. 8, 2019, pp. 1887–1892. https://doi.org/10.2514/1.G001681

[4] Kim, B. S., Lee, J. G., and Han, H. S., "Biased PNG Law for Impact with Angular Constraint," *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 34, No. 1, 1998, pp. 277–288. https://doi.org/10.1109/7.640285

[5] Lee, C.-H., Kim, T.-H., and Tahk, M.-J., "Interception Angle Control Guidance Using Proportional Navigation with Error Feedback," *Journal of Guidance, Control, and Dynamics*, Vol. 36, No. 5, 2013, pp. 1556–1561.

[6] Kim, T.-H., Lee, C.-H., Tahk, M.-J., and Jeon, I.-S., "Biased PNG Law for Impact-Time Control," *Transactions of the Japan Society for Aeronautical and Space Sciences*, Vol. 56, No. 4, 2013, pp. 205–214. https://doi.org/10.2322/tjsass.56.205

[7] Lu, P., "Introducing Computational Guidance and Control," *Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 2, 2017, pp. 193–193. https://doi.org/10.2514/1.G002745

[8] Shukla, D., Lal, R., Hauptman, D., Keshmiri, S. S., Prabhakar, P., and Beckage, N., "Flight Test Validation of a Safety-Critical Neural Network Based Longitudinal Controller for a Fixed-Wing UAS," *AIAA Aviation 2020 Forum*, AIAA Paper 2020-3093, 2020.

[9] Dwivedi, P. N., Bhattacharya, A., and Padhi, R., "Suboptimal Midcourse Guidance of Interceptors for High-Speed Targets with Alignment Angle Constraint," *Journal of Guidance, Control, and Dynamics*, Vol. 34, No. 3, 2011, pp. 860–877. https://doi.org/10.2514/1.50821

[10] Padhi, R., and Kothari, M., "Model Predictive Static Programming: A Computationally Efficient Technique for Suboptimal Control Design," *International Journal of Innovative Computing, Information and Control*, Vol. 5, No. 2, 2009, pp. 399–411.

[11] Oza, H. B., and Padhi, R., "Impact-Angle-Constrained Suboptimal Model Predictive Static Programming Guidance of Air-to-Ground Missiles," *Journal of Guidance, Control, and Dynamics*, Vol. 35, No. 1, 2012, pp. 153–164. https://doi.org/10.2514/1.53647

[12] Halbe, O., Raja, R. G., and Padhi, R., "Robust Reentry Guidance of a Reusable Launch Vehicle Using Model Predictive Static Programming," *Journal of Guidance, Control, and Dynamics*, Vol. 37, No. 1, 2014, pp. 134–148. https://doi.org/10.2514/1.61615

[13] Pan, B., Ma, Y., and Yan, R., "Newton-Type Methods in Computational Guidance," *Journal of Guidance, Control, and Dynamics*, Vol. 42, No. 2, 2019, pp. 377–383. https://doi.org/10.2514/1.G003931

[14] Shin, H.-S., He, S., and Tsourdos, A., "Computational Flight Control: A Domain-Knowledge-Aided Deep Reinforcement Learning Approach," arXiv preprint arXiv:1908.06884, 2019, https://arxiv.org/abs/1908.06884.

[15] Zhao, X., Wang, Z., and Zheng, G., "Two-Phase Neural Combinatorial Optimization with Reinforcement Learning for Agile Satellite Scheduling," *Journal of Aerospace Information Systems*, Vol. 17, No. 7, 2020, pp. 346–357. https://doi.org/10.2514/1.I010754

[16] Bloem, M., and Bambos, N., "Ground Delay Program Analytics with Behavioral Cloning and Inverse Reinforcement Learning," *Journal of Aerospace Information Systems*, Vol. 12, No. 3, 2015, pp. 299–313. https://doi.org/10.2514/1.I010304

[17] Gaudet, B., and Furfaro, R., "Missile Homing-Phase Guidance Law Design Using Reinforcement Learning," *AIAA Guidance, Navigation, and Control Conference*, AIAA Paper 2012-4470, 2012.

[18] Gaudet, B., Linares, R., and Furfaro, R., "Deep Reinforcement Learning for Six Degree-of-Freedom Planetary Powered Descent and Landing," *Advances in Space Research*, Vol. 65, No. 7, 2020, pp. 1723–1741.

[19] Gaudet, B., and Linares, R., "Adaptive Guidance with Reinforcement Meta-Learning," arXiv preprint arXiv:1901.04473, 2019, https://arxiv.org/abs/1901.04473.

[20] Scorsoglio, A., Furfaro, R., Linares, R., and Massari, M., "Actor-Critic Reinforcement Learning Approach to Relative Motion Guidance in Near-Rectilinear Orbit," *29th AAS/AIAA Space Flight Mechanics Meeting*, American Astronautical Soc., San Diego, CA, 2019, pp. 1–20.

[21] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., and Petersen, S., "Human-Level Control Through Deep Reinforcement Learning," *Nature*, Vol. 518, No. 7540, 2015, p. 529. https://doi.org/10.1038/nature14236

[22] Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., "Continuous Control with Deep Reinforcement Learning," arXiv preprint arXiv:1509.02971, 2015, https://arxiv.org/abs/1509.02971.

[23] Yucelen, T., and Haddad, W. M., "Low-Frequency Learning and Fast Adaptation in Model Reference Adaptive Control," *IEEE Transactions on Automatic Control*, Vol. 58, No. 4, 2013, pp. 1080–1085. https://doi.org/10.1109/TAC.2012.2218667

[24] Gaudio, J. E., Gibson, T. E., Annaswamy, A. M., Bolender, M. A., and Lavretsky, E., "Connections Between Adaptive Control and Optimization in Machine Learning," *2019 IEEE 58th Conference on Decision and Control (CDC)*, Inst. of Electrical and Electronics Engineers, New York, 2019, pp. 4563–4568.

[25] Rawling, A., "On Nonzero Miss Distance," *Journal of Spacecraft and Rockets*, Vol. 6, No. 1, 1969, pp. 81–83. https://doi.org/10.2514/3.29539

[26] He, S., Song, T., and Lin, D., "Impact Angle Constrained Integrated Guidance and Control for Maneuvering Target Interception," *Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 10, 2017, pp. 2653–2661. https://doi.org/10.2514/1.G002201

[27] Henderson, P., Islam, R., Bachman, P., Pineau, J., Precup, D., and Meger, D., "Deep Reinforcement Learning that Matters," *32nd AAAI Conference on Artificial Intelligence*, New Orleans, Louisiana, 2018, Paper 1169.

[28] Islam, R., Henderson, P., Gomrokchi, M., and Precup, D., "Reproducibility of Benchmarked Deep Reinforcement Learning Tasks for Continuous Control," *Proceedings of the Reproducibility in Machine Learning Workshop at the 34th International Conference on Machine Learning*, Sydney, Australia, 2017, Paper 7.

E. Atkins
Associate Editor

https://dspace.org/

Computational missile guidance: a deep reinforcement learning approach

He, Shaoming; Shin, Hyo-Sang; Tsourdos, Antonios

2021-06-28
American Institute of Aeronautics and Astronautics (AIAA)
https://doi.org/10.2514/1.I010970
He S, Shin H-S, Tsourdos A. (2021) Computational missile guidance: a deep reinforcement learning approach. Journal of Aerospace Information Systems, Volume 18, Number 8, August 2021, pp. 571-582

Downloaded from a DSpace repository. https://dspace.org/