sensors

MDPI

Article

Deep Reinforcement Learning-Based Accurate Control of Planetary Soft Landing

Xibao Xu [1,2,†], Yushen Chen [1,†] and Chengchao Bai [1,*]

[1] School of Astronautics, Harbin Institute of Technology, Harbin 150001, China; xuxibaogz@163.com (X.X.); 20S018108@stu.hit.edu.cn (Y.C.)
[2] Beijing Institute of Astronautical Systems Engineering, Beijing 100076, China
* Correspondence: baichengchao@hit.edu.cn
† These authors contributed equally to this work.

Abstract: Planetary soft landing has been studied extensively due to its promising application prospects. In this paper, a soft landing control algorithm based on deep reinforcement learning (DRL) with good convergence property is proposed. First, the soft landing problem of the powered descent phase is formulated and the theoretical basis of Reinforcement Learning (RL) used in this paper is introduced. Second, to make it easier to converge, a reward function is designed to include process rewards like velocity tracking reward, solving the problem of sparse reward. Then, by including the fuel consumption penalty and constraints violation penalty, the lander can learn to achieve velocity tracking goal while saving fuel and keeping attitude angle within safe ranges. Then, simulations of training are carried out under the frameworks of Deep deterministic policy gradient (DDPG), Twin Delayed DDPG (TD3), and Soft Actor Critic (SAC), respectively, which are of the classical RL frameworks, and all converged. Finally, the trained policy is deployed into velocity tracking and soft landing experiments, results of which demonstrate the validity of the algorithm proposed.

Keywords: soft landing; velocity tracking; deep reinforcement learning (DRL)

Citation: Xu, X.; Chen, Y.; Bai, C. Deep Reinforcement Learning-Based Accurate Control of Planetary Soft Landing. *Sensors* **2021**, *21*, 8161. https://doi.org/10.3390/s21238161

Academic Editor: Hyun Myung

Received: 5 October 2021
Accepted: 29 November 2021
Published: 6 December 2021

1. Introduction

With the development of space technology, the scope of space exploration is constantly expanding. To further explore and study planets such as the Moon and Mars, a large number of planetary surface exploration missions have been carried out and many are in planning [1,2]. In planetary surface exploration missions, the lander faces many challenges. On the one hand, to avoid damaging onboard equipments, the landing velocity relative to the planet's surface must be kept under a threshold. In addition, the maneuvering ability of the landing probe is limited, so the lander needs to have a high landing accuracy to explore a specific area. Therefore, precise soft landing guidance technology is always one of the key technologies of planetary exploration, which has been widely studied and achieved many achievements [3–5].

At present, soft landing algorithms can be roughly divided into five categories: establishing lunar vertical line, gravity turn guidance, nominal trajectory guidance, explicit guidance, and learning-based method.

The establishment of lunar vertical line is an open-loop guidance method, which requires high accuracy of orbit entry and mid-course correction [6]. Gravity turn guidance is a semi-open loop and semi-closed loop guidance method. In the main braking period, the main goal is to reduce the speed of the lander. While the distance between the lander and the lunar surface is shortened to a certain range, the closed-loop guidance works are based on the feedback information of the sensor to improve the landing accuracy and stability [7]. These two methods were generally only used in early lunar landings [8]. Nominal trajectory guidance consists of open-loop offline trajectory planning and closed-loop online trajectory tracking. Before landing, the lander needs to plan an optimized

landing trajectory. During the landing process, the deviation between the lander position and velocity and the planned trajectory is constantly measured and eliminated, through which the lander is controlled to land at the desired landing sites [9–12]. Explicit guidance method solves the closed-loop guidance problem based on the explicit function of control functional. To obtain the analytical expression of the optimal control problem of soft landing, it is necessary to simplify the soft landing model when building the mathematical model [13]. The work in [14] proposes a guidance method for the powered soft landing of a launcher with non-cluster configured engines, for which it is difficult to maintain a low thrust-to-weight ratio. The work in [15] deals with designing soft landing trajectory, from lunar parking orbit to the surface of Moon and solve the optimization problem with Differential Evolution (DE) which is superior in convergence speed.

In recent years, machine learning has made great breakthroughs and development. Some researchers have studied soft landing based on intelligent learning algorithms such as deep learning. In [16], the deep architectures' ability to drive the onboard decision-making system is investigated in detail. Under the assumption of perfect state information, deep networks are trained to approximate the optimal control action in pinpoint landing experiments and have the ability to cope with large sets of possible initial states. The work in [17] proposes an autonomous lunar landing method based on deep learning that takes raw images taken by onboard optimal cameras as input and directly outputs fuel-optimal control actions, in which the direct filters for state estimation is not necessary. Moreover, the deep networks are trained by a supervised machine learning algorithm, and the training datasets are generated by NLP solver software packages. Further, in [18], a recurrent neural network architecture is proposed to predict the fuel-optimal thrust from a sequence of states in the powered descent phase of planetary soft landing.

In future exploration missions, it is necessary to enable the lander with higher autonomy that can make real-time adjustments to landing trajectory according to the landing condition, which is difficult for offline planning methods.

As an important branch of machine learning, DRL integrates the perception ability of deep learning with the decision-making ability of RL. It is an end-to-end algorithm that directly takes the environmental information as inputs and outputs the control efforts. It is especially suitable for solving the decision-making and planning problems of complex systems and has been widely studied and applied in the fields of games [19], autonomous driving [20], and manipulator control [21].

In this paper, the problem of planetary soft landing of the powered descent phase is studied. The soft landing problem of the powered descent phase is formulated, and the theoretical basis of RL used in this paper is introduced. To make it easier for the training process to converge, a reward function including process reward is designed to solving the problem of sparse reward. In addition, the fuel consumption penalty and constraints violation penalty are included to save fuel and keep the attitude angle within constraints. The main contributions of this work are (i) a velocity tracking reward function is designed with process reward, which makes it easier for the lander to learn to achieve the goal of soft landing as well as enables it with a better generalization capability, and (ii) the goals of keeping the attitude within constraints and reducing fuel consumption are reached by including fuel consumption penalty and constraints violation penalty into the reward function.

The remainder of this paper is organized as follows. Section 2 gives the preliminaries about RL and formulates the soft landing as an RL problem. Section 3 describes the details about soft landing control method based on DRL. Simulation results and necessary discussions are given in Section 4. Finally, conclusions are presented in Section 5.

2. Preliminaries and Problem Formulation

2.1. Soft Landing Problem Formulation

The planetary surface fixed frame of reference is defined as Figure 1. The forces acting on the lander in power descent include gravity, aerodynamic force, and engine thrust [22].

As the powered descent begins at an altitude that is quite low compared to the planet's radius, and the distance between the lander and target landing sites varies slightly during this phase, it is appropriate to assume that the planet's gravity is a constant g.

When it comes to the power descent phase, the lander has already released the parachute and the speed is on the order of 100 meters per second [22]. Compared with planetary gravity, the acceleration caused by aerodynamic force was very small. Therefore, the external force on the lander was dominated by gravity, and the aerodynamic force caused by the wind field was added into the model as an environmental disturbance.

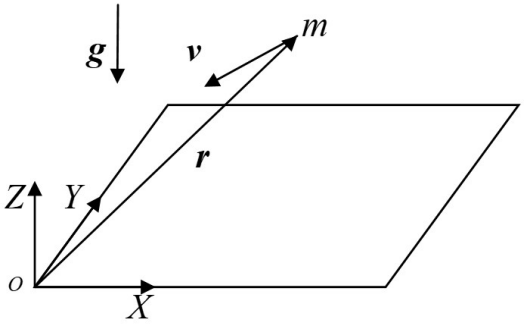


Figure 1. Navigation coordinates of planetary surface.

The lander is equipped with six thrusters deployed in body frame $O_bX_bY_bZ_b$ as Figure 2.

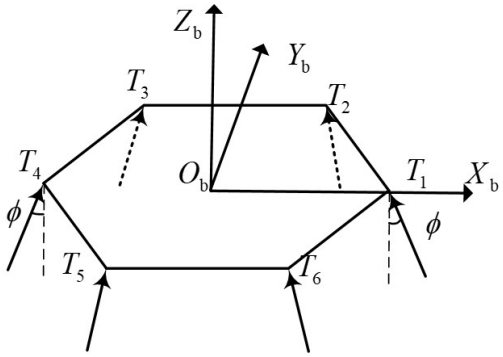


Figure 2. Lander body frame and thrusters layout. The thrusters are deployed in regular hexagon.

The thrust of each engine is T_i, and meets constraint

$$T_{\min} \leq T_i \leq T_{\max} \tag{1}$$

Let $\boldsymbol{T} = \begin{bmatrix} T_1 & T_2 & T_3 & T_4 & T_5 & T_6 \end{bmatrix}^{\mathrm{T}}$ be the vector composed of the thrusts of six thrusters, and the thrust vector in body frame can be obtained according to the geometric relation

$$\boldsymbol{T}_b = \begin{bmatrix} -\sin\phi & -\frac{1}{2}\sin\phi & \frac{1}{2}\sin\phi & \sin\phi & \frac{1}{2}\sin\phi & -\frac{1}{2}\sin\phi \\ 0 & -\frac{\sqrt{3}}{2}\sin\phi & -\frac{\sqrt{3}}{2}\sin\phi & 0 & \frac{\sqrt{3}}{2}\sin\phi & \frac{\sqrt{3}}{2}\sin\phi \\ -\cos\phi & -\cos\phi & -\cos\phi & -\cos\phi & -\cos\phi & -\cos\phi \end{bmatrix} T \tag{2}$$

Moreover, the same for the torque vector in body frame

$$\boldsymbol{M}_b = L \begin{bmatrix} 0 & -\frac{\sqrt{3}}{2}\cos\phi & -\frac{\sqrt{3}}{2}\cos\phi & 0 & \frac{\sqrt{3}}{2}\cos\phi & \frac{\sqrt{3}}{2}\cos\phi \\ \cos\phi & \frac{1}{2}\cos\phi & -\frac{1}{2}\cos\phi & -\cos\phi & -\frac{1}{2}\cos\phi & \frac{1}{2}\cos\phi \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} T \tag{3}$$

The translation dynamics are expressed as

$$\begin{aligned} \dot{\boldsymbol{v}}_{\mathrm{b}} &= \tfrac{\boldsymbol{T}_{\mathrm{b}}+\boldsymbol{C}_{\mathrm{e}}^{\mathrm{b}}\boldsymbol{g}}{m} + \boldsymbol{v}_{\mathrm{b}} \times \omega^{\mathrm{t}} \\ \dot{\boldsymbol{r}} &= \boldsymbol{C}_{\mathrm{b}}^{\mathrm{e}}\boldsymbol{v}_{\mathrm{b}} \end{aligned} \tag{4}$$

where $\boldsymbol{r} = \begin{bmatrix} x & y & z \end{bmatrix}^{\mathrm{T}}$ is the translational vector of the lander, $\boldsymbol{v}$ is the velocity vector, and $\boldsymbol{v}_b = \begin{bmatrix} u & v & w \end{bmatrix}^{\mathrm{T}}$ and $\boldsymbol{\omega}^{\mathrm{t}} = \begin{bmatrix} p & q & r \end{bmatrix}$ are the velocity and angular velocity in the body frame, respectively. $Q = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}$ is the quaternion. $\boldsymbol{I}$ and m are the inertia matrix and mass of the lander, respectively. $\boldsymbol{C}_{\mathrm{b}}^{\mathrm{e}}$ is the direction cosine matrix from the body frame to the surface fixed frame of reference.

$$\begin{aligned} \boldsymbol{C}_{\mathrm{b}}^{\mathrm{e}} &= \left(\boldsymbol{C}_{\mathrm{e}}^{\mathrm{b}}\right)^{\mathrm{T}} \\ \boldsymbol{C}_{\mathrm{e}}^{\mathrm{b}} &= \boldsymbol{C}_x(\varphi)\boldsymbol{C}_y(\theta)\boldsymbol{C}_z(\psi) \end{aligned} \tag{5}$$

Attitude dynamics are express as

$$\dot{\boldsymbol{\omega}}^{\mathrm{t}} = \boldsymbol{I}^{-1} \cdot \boldsymbol{\omega}^{\mathrm{t}} \times \left(\boldsymbol{I} \cdot \boldsymbol{\omega}^{\mathrm{t}}\right) \tag{6}$$

$$\begin{aligned} \dot{\varphi} &= p + \tan\theta(q\sin\varphi + r\cos\varphi) \\ \dot{\theta} &= q\cos\varphi - r\sin\varphi \\ \dot{\psi} &= (q\sin\varphi + r\cos\varphi)/\cos\theta \end{aligned} \tag{7}$$

During soft landing, the mass of the lander will gradually decrease with the fuel consumption, i.e.,

$$\dot{m} = -\frac{1}{I_{\mathrm{sp}}}\sum_i T_i(t) \tag{8}$$

where I_{sp} is the specific impulse of the engine, and the inertia matrix will also gradually decrease as the mass decreases. The shape of the lander is a cuboid of sides of length $a \times b \times c$ with uniform mass distribution

$$\begin{aligned} \boldsymbol{I} &= \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \\ I_{xx} &= \tfrac{m}{12}(b^2 + c^2) \\ I_{yy} &= \tfrac{m}{12}(a^2 + c^2) \\ I_{zz} &= \tfrac{m}{12}(a^2 + b^2) \end{aligned} \tag{9}$$

The soft landing problem is described as a fuel optimization problem as

$$\min_{T_i(t)} J = \int_{t_0}^{t_f} \frac{1}{I_{\mathrm{sp}}}\sum_i T_i(t)dt \tag{10}$$

2.2. RL Basis

RL is a data-driven algorithm, which is different from supervised learning. RL obtains training data(experience) through interaction with the environment, as shown in Figure 3.

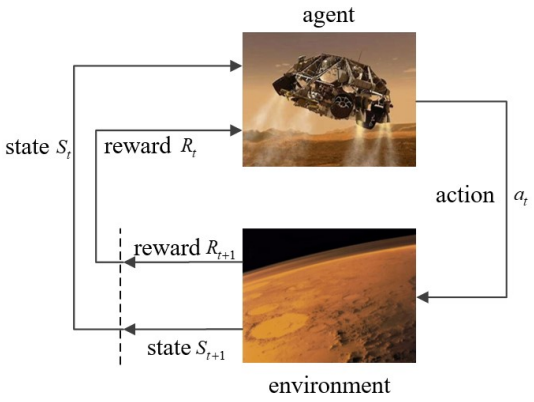


Figure 3. Interaction between agent and environment.

At time step t, the agent gets an observation $S_t = s$ from the environment, then takes an action $A_t = a$ according to the policy $\pi(A_t = a_t|S_t = s)$, the environment transfers to the next state s' based on the model $P(S_{t+1} = s'|S_t = s, A_t = a_t)$ and returns a reward R_{t+1}. Define the accumulated return of an episode as

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=1} \gamma^{k-1} R_{t+k} \tag{11}$$

where $\gamma \in [0, 1]$ is the discount factor. When $\gamma \to 0$, the agent only cares about the most recent rewards. While $\gamma \to 1$, the agent has a longer horizon and cares more about the future reward. Moreover, the target is to update policy to maximize G_t.

DDPG, TD3, and SAC are three of the most successful and popular RL algorithms, so we chose them as the framework in this paper.

(1) DDPG

DDPG is a deterministic policy RL framework that outputs a deterministic action, and it is the result of the deep Q network (DQN) extended to continuous control space. DDPG has been extensively researched and applied to the field of continuous control. It learns both a value function and a policy. First, it approximates the value function via the Bellman equation with offline experience through gradient descent. Then, the policy is updated by maximizing the approximated value function.

DDPG is one of the standard algorithms that training a deep neural network to approximate Q function. It makes use of the past experience through the trick of replay buffer. When it is time to update, it randomly samples from the buffer. In order to stabilize the training process, the size of the replay buffer should be properly chosen. If it is too small, it can only store recent experiences, which makes the policy brittle. However, if it is too large, the possibility of sampling a good experience will decrease, and it takes more episodes for the training process to converge.

As shown in Algorithm 1, the whole learning process consists of two parts: Q-learning and policy learning.

According to Bellman equation, optimal Q function under optimal policy satisfies

$$Q^*(s, a) = \underset{s' \sim P}{E}\left[r(s, a) + \gamma \max_{a'} Q^*(s', a')\right] \tag{12}$$

where P is the environment model. Given the experience (s, a, r, s', d), value function Q_{π_θ} under the policy π_θ can be represented as

$$Q_{\pi_\theta}(s, a) = \underset{(s,a,r,s',d) \sim D}{E}\left[r(s, a) + \gamma \max_{a'} Q_{\pi_\theta}(s', a')\right] \tag{13}$$

Algorithm 1: DDPG based soft landing

Initialize policy network parameters θ, value function network parameters ϕ and replay buffer D
Initialize target parameters equal to main parameters $\theta \rightarrow \theta_{\text{targ}}$,$\phi \rightarrow \phi_{\text{targ}}$
for *episode = 1,2,. . . ,* **do**
  Observe landing state s and select thrust action $a = \text{clip}\left(\mu_\theta(s) + \varepsilon, a_{\text{Low}}, a_{High}\right)$, where $\varepsilon \sim N(0, \sigma)$ is a mean-zero Gaussian noise
  Execute a in the environment and observe next state s', reward r, and done signal d that indicates whether s' is terminal Store (s, a, r, s', d) in replay buffer D
  if *the episode is terminated* **then**
    reset environment
  end
  while *it's time to update* **do**
    Sample a batch of transitions $B = \{(s, a, r, s', d)\}$ from D
    compute targets:
      $$y(r, s', d) = r + \gamma(1 - d)Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$
    update value function through gradient ascent:
      $$\nabla_{\phi_i} \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s, a) - y(r, s', d)\right)^2$$
    update policy through gradient ascent:
      $$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$
    update target parameterst:
      $$\phi_{\text{targ}} \leftarrow \rho\phi_{\text{targ}} + (1 - \rho)\phi$$
      $$\theta_{\text{targ}} \leftarrow \rho\theta_{\text{targ}} + (1 - \rho)\theta$$
  end
end

Setup a mean-squared Bellman error function as

$$L(\phi, D) = \underset{(s,a,r,s!d) \sim D}{E}\left[\left(Q_\phi(s, a) - \left(r + \gamma(1 - d)\max_{a'} Q_\phi\left(s', a'\right)\right)\right)^2\right] \tag{14}$$

where $r + \gamma(1 - d)\max_{a'} Q_\phi(s', a')$ is the value function target. In the condition of continuous action space, it is difficult to compute action a' which maximizes $Q_\phi(s', a')$. Therefore, DDPG uses target network to solve this problem

$$\max_{a'} Q_\phi(s', a') = Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s')) \tag{15}$$

where a' is obtained via target policy $\mu_{\theta_{\text{targ}}}$. With the training process going on, policy and value function will gradually converge to optimal policy and optimal value function respectively.

(2) TD3

TD3 is modified from DDPG. DDPG can perform well sometimes but it highly depends on the choice of hyperparameters. TD3 takes three critical tricks to make the training process more stable.

- ***Clipped Double-Q Learning.*** TD3 learns two value function networks at the same time. When calculating the target, and are input to the two target value function networks at the same time after obtaining. When the value function network is updated, the smaller one is selected to compute the loss function of the error of the Bellman equation.

$$y(r, s', d) = r + \gamma(1 - d)\min_{i=1,2} Q_{\phi_{\text{t arg},i}}(s', \tilde{a}'(s)) \tag{16}$$

- ***Target Policy Smoothing.*** The value function learning method of TD3 and DDPG is the same. When the value function network is updated, noise is added to the action output of the target policy network to avoid overexploitation of the value function

$$\max_{a'} Q_{\phi}(s', a') = Q_{\phi_{\text{targ}}}(s', \pi_{\theta_{\text{targ}}}(s') + \varepsilon) \tag{17}$$

  where $\varepsilon \sim N(0, \sigma)$ is a mean-zero Gaussian noise, and θ_{targ} is the parameters of the target strategy network. Adding noise to the output action of the strategy network serves as regularization, which avoids the overexploitation of value function and stabilizes the training process.
- ***Delayed Policy Updates.*** As the output of the target strategy network is used to compute the target of the value function, the agent can be brittle because of frequent strategy updates, so TD3 adopts the Delayed Policy Updates trick. When updating the strategy network, the update frequency of the strategy network is lower than that of the value function network. This helps to suppress the training fluctuation and makes the learning process more stable.

(3) SAC

SAC is an RL framework that maximizes cross-entropy. It applies the learning techniques of DDPG to the learning of random strategies and optimizes random strategies in an offline learning mode.

The agent starts from the initial state $s_0 \sim p(s_0)$, samples from policy distribution $a_t \sim \pi(\cdot \mid s_t)$, and gets an action a_t acting on the environment. Then, the environment returns a reward $r(s_t, a_t)$ and transfers to a new state $s_{t+1} \sim p(\cdot \mid s_t, a_t)$ according to the environmental model. Repeating the interacting process and the trajectory of the state, $\tau = (s_0, a_0, s_1, a_1, ...)$ can be obtained. The probability distribution of the trajectory τ regarding the strategy π is expressed as

$$\rho_{\pi}(\tau) = p(s_0) \prod_{t} \pi(a_t|s_t) p(s_{t+1}|s_t, a_t) \tag{18}$$

The maximum cross-entropy RL optimizes the cumulative return and cross-entropy of the strategy. For Markov decision processes (MDPs) with infinite loss rewards, the optimization objective can be expressed as

$$J(\pi) = \sum_{t=0}^{\infty} E_{\tau \sim \rho_x} \left[\sum_{t=0}^{\infty} \gamma^t [r(s_t, a_t) - \alpha \log \pi(a_t \mid s_t)] \right] \tag{19}$$

Moreover, the optimal policy is represented as

$$\pi^* = \arg\max_{\pi} E_{\tau \sim \rho_x} \left[\sum_{t=0}^{\infty} \gamma^t [r(s_t, a_t) - \alpha \log \pi(a_t \mid s_t)] \right] \tag{20}$$

where $\log \pi_t(a_t|s_t)$ is the cross-entropy of strategy distribution, which can be added into the optimization target to encourage agents to explore the environment in training, and to improve the robustness of training results. α is the temperature coefficient, which is used to adjust the importance of cross-entropy, and thus plays a role in regulating the randomness of the optimal strategy. A large α encourages the agent to explore the environment. Therefore, the larger the α is, the more stochastic the strategy will be. While the smaller alpha is, it is more likely that the policy falls into a local optimal point. When $\alpha \to 0$, maximizing cross-entropy RL degenerates into conventional RL that maximizes cumulative reward.

Based on the optimization objective above, the state value function is defined as

$$V^{\pi}(s) = E_{\tau\sim\rho_x}\left[\sum_{t=0}^{\infty}\gamma^t[r(s_t,a_t) - \alpha\log\pi(\cdot \mid s_t)] \mid s_0 = s\right] \tag{21}$$

Moreover, the value function

$$Q^{\pi}(s,a) = \underset{\substack{\tau\sim\rho\pi \\ s_0=s,a_0=a}}{E}\left[\sum_{t=0}^{\infty}\gamma^t r(s_t,a_t) - \alpha\sum_{t=1}^{\infty}\gamma^t\log\pi(\cdot \mid s_t)\right] \tag{22}$$

$$V^{\pi}(s) = E_{a\sim\pi}[Q^{\pi}(s,a)] - \alpha\log\pi(\cdot \mid s) \tag{23}$$

Then, according to the Bellman equation,

$$\begin{aligned} Q^{\pi}(s,a) &= \underset{s'\sim p}{E}\left[r(s,a) + \gamma V^{\pi}(s')\right] \\ &= \underset{\substack{s'\sim p \\ a'\sim\pi}}{E}\left[r(s,a) + \gamma\left(Q^{\pi}(s',a') - \alpha\log\pi(\cdot \mid s')\right)\right] \end{aligned} \tag{24}$$

SAC makes use of "Clipped Double Q-learning" and the Q-learning is similar to TD3 except for the compute of the target value function

$$y(r,s',d) = r + (1-d)\left(\min_{j=1,2} Q_{\phi_{\mathrm{ueg},j}}(s',a') - \alpha\log\pi_{\theta}(a' \mid s')\right) \tag{25}$$

According to Equation (24), then

$$\begin{aligned} &\max_{\theta} V^{\pi_\theta}(s) = \max_{\theta}\underset{a\sim\pi_\theta}{E}[Q^{\pi_\theta}(s,a) - \alpha\log(\pi_\theta(a \mid s))] \\ &= \max_{\theta}\underset{\xi\sim N}{E}[Q^{\pi_\theta}(s,\tilde{a}_\theta(s,\xi)) - \alpha\log(\pi_\theta(\tilde{a}_\theta(s,\xi) \mid s))] \end{aligned} \tag{26}$$

The updating of the policy network makes use of the re-parameterization trick

$$a = \tilde{a}_\theta(s,\xi) = \tanh(\mu_\theta(s) + \sigma_\theta(s)\odot\xi) \quad \xi\sim N(0,1) \tag{27}$$

where the action distribution is Gaussian; $\mu_\theta(s)$ and $\sigma_\theta(s)$ are the mean value and variance of the Gaussian distribution, respectively; and ξ is the standard Gaussian distribution. After sampling from the distribution, the action output is restricted to the constrained range through the Tanh activation function.

Besides, SAC makes use of the Clipped Q trick when updating its strategy

$$Q^{\pi_\theta}(s,a) = \min_{j=1,2} Q_{\phi_j}(s,\tilde{a}_\theta(s,\xi)) \tag{28}$$

The strategy optimization objective is finally represented as

$$\max_{\theta}\underset{s\sim D}{E}\left[\min_{j\sim\sim} Q_{\phi_j}(s,\tilde{a}_\theta(s,\xi)) - \alpha\log(\pi_\theta(\tilde{a}_\theta(s,\xi) \mid s))\right] \tag{29}$$

Because of the inherent stochasticity, SAC can effectively avoid the overexploitation of value function.

3. Soft Landing with DRL

Based on the dynamic model established above, in this section we will design an algorithm based on RL according to the characteristics of soft landing problems, including the selection of observation values and the design of reward function and other settings concerning how the agent interacts the environment.

3.1. Reward Setting

The reward function is an index to evaluate the behavior of agents, which is directly related to the training result.

- ***Goal achieving reward:*** When the altitude of the lander is less than 0, the speed is downward, the speed is less than the upper limit of soft landing speed, and the attitude angle and angular rate is within the limited range, the lander is considered to have achieved soft landing, and gets the reward

$$\begin{aligned} r_{goal} = \lambda(h < 0 \quad \text{and} \quad v_z < 0 \quad \text{and} \quad \|v\| < v_{\lim} \quad \text{and} \\ \phi < \phi_{\lim} \quad \text{and} \quad \theta < \theta_{\lim} \quad \text{and} \quad \psi < \psi_{\lim} \quad \text{and} \\ \|\omega\| < \omega_{\lim}) \end{aligned} \tag{30}$$

  where λ is a large positive constant serving as a soft landing bonus. $\phi_{\lim}, \theta_{\lim}, \psi_{\lim}$ are upper bounds of Euler angle of the lander, and $\omega_{\lim}$ is the upper bounds of angular rate.
- ***Velocity tracking reward:*** At the beginning of the phase of powered descent, the lander is several kilometers away from the landing zone, and the initial velocity is around 100 m/s. If the agent is rewarded only when it achieves a soft landing at the target area, the state space is so sparse that it's nearly impossible to converge. Therefore, we transfer the soft landing problem into a velocity tracking problem. The process reward is introduced in the landing process, that is, a reference velocity is given according to the real-time relative position between the lander and target landing area

$$\boldsymbol{v}_{ref} = \begin{cases} -\frac{\boldsymbol{r}-\boldsymbol{r}_1}{k_{v1}} & h \geq h_1 \\ -\frac{\boldsymbol{r}-\boldsymbol{r}_2}{k_{v2}} & 0 \leq h < h_1 \end{cases} \tag{31}$$

  where k_{v1} and k_{v2} are constant coefficients that determine the mapping relationship between position and reference velocity, and large coefficients lead to smaller reference velocity in the powered descent process. Moreover, the reward is given according to the deviation between the real velocity and the reference velocity of the lander

$$\boldsymbol{r}_{vel} = \beta \left\| v - v_{ref} \right\| \tag{32}$$

  where β is the reward coefficient of velocity tracking error.
- ***Crash penalty:*** To avoid the crash of the lander, a penalty is included in the reward. When the attitude angle or speed deviation exceeds the threshold, the episode terminates and the environment returns a large negative reward as a penalty

$$r_{crash} = \eta(\phi > \phi_{\lim} \quad or \quad \theta > \theta_{\lim} \quad or \quad \psi > \psi_{\lim}) \tag{33}$$

  where η is the penalty of attitude crash.
- ***Fuel consumption penalty***: In planetary exploration missions, the fuel carried by the lander is limited, so the fuel consumption should be minimized. A reward regarding fuel consumption is defined as

$$r_{fuel} = \alpha \frac{1}{I_{sp}} \sum_{i=1}^{6} T_i \tag{34}$$

  where α weights a term penalizing fuel consumption. The fuel consumption coefficient α and velocity tracking error coefficient β explicitly control the trade-off between fuel consumption and velocity tracking. With higher $|\alpha|$ and lower $|\beta|$, fuel consumption weights more in the reward and the lander will exchange some velocity tracking performance for less fuel consumption.
- ***Constant reward:*** Notice that the rewards $r_{vel}, r_{crash}, r_{fuel}$ are all negative. To encourage the agent to explore more, a positive constant reward needs to be introduced into the reward.

$$r_{constant} = \kappa \tag{35}$$

Therefore, the overall reward format is as follows:

$$r = r_{fuel} + r_{vel} + r_{crash} + r_{constant} + r_{goal} \tag{36}$$

3.2. Observation Space

To improve the landing performance, it is necessary to include position, velocity, attitude, and angular rate into the observation s_t. Based on the analysis of reward setting, the absolute velocity in s_t is replaced by the velocity deviation in the lander body frame.

$$\delta \boldsymbol{v}_b = \boldsymbol{C}_{\mathrm{e}}^{\mathrm{b}}(\boldsymbol{v} - \boldsymbol{v}_{ref}) \tag{37}$$

Tracking velocity deviation rather than direct position can improve the generalization ability of the trained agent.

Each attitude Angle is input into the observation vector in the form of sine value and cosine value, and the quaternion is observed simultaneously.

$$\begin{aligned} s = [&\delta v_{\mathrm{bx}}, \delta v_{\mathrm{by}}, \delta v_{\mathrm{bz}}, \sin(\phi), \cos(\phi), \sin(\theta), \cos(\theta), \\ &\sin(\psi), \cos(\psi), p, q, r, q_0, q_1, q_2, q_3] \end{aligned} \tag{38}$$

3.3. Action Space

The lander's action is the thrusts of the engines, which are bounded in a specific range. After going through the Tanh activation function, the output of the policy network is bounded to $a_i \in [-1, 1]$. Then, the actual thrust is got through a linear mapping

$$T_i = \frac{T_{\max} - T_{\min}}{2} a_i + \frac{T_{\max} + T_{\min}}{2} \tag{39}$$

where $T_{\min}$ and $T_{\max}$ are the lower and upper bounds of the thrust, respectively.

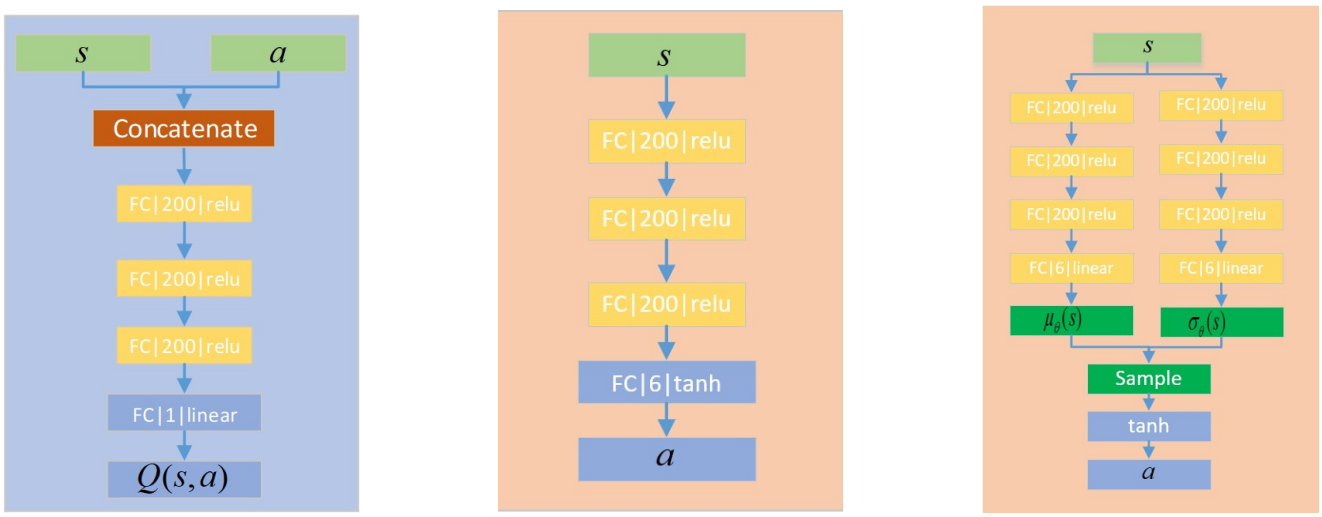
3.4. Network Architecture

We use the deep learning framework PyTorch to build the neural networks. The hyperparameters such as network learning rates and noise variance are defined Section 4.

DDPG, TD3, and SAC all contain value function networks and policy networks. In this paper, all the networks of value functions have the same structure as Figure 4a. We employ three hidden layers to process the vector concatenated of observation and action. All the hidden layers contain 200 nonlinear units and the activation function is ReLU.

The policy network structure of DDPG and TD3 is the same, as shown in Figure 4b. The network includes three hidden layers, of which the activation function is ReLU, and the activation function of the output layer is Tanh, through which the action is normalized to $[-1, 1]$.

Different from the deterministic policy of DDPG and TD3, SAC is a stochastic strategy and the structure of the policy network is shown in Figure 4c. The strategy network consists of two paths of networks. Both of them have the same structure, which outputs the mean value and variance of the Gaussian distribution, respectively. Then, the network output is obtained by sampling and activation functions in turn.



(**a**) (**b**) (**c**)

Figure 4. Network architecture. (**a**) Value function network. (**b**) Policy network of DDPG and TD3. (**c**) Policy network of SAC.

4. Simulation Results and Discussion

In this section, the simulation experiments are carried out and the results and related discussions are proposed.

4.1. Simulation Settings

We train the lander velocity controller in a 6DOF environment established in Section 2. The environmental parameters settings are shown in Table 1.

Table 1. Parameter settings of environment.

Parameters	Values
m_0	1700 kg
$a \times b \times c$	3 m × 3 m × 1 m
I_{sp}	225 s
T_{max}	2880 N
T_{min}	1080 N
L	2 m
ϕ	27°
T_s	0.2 s
T_f	50 s
k_{v1}	20
k_{v2}	20
δv_{bx0}	$[-3, 3]$ m/s
δv_{by0}	$[-3, 3]$ m/s
δv_{bz0}	$[-12, 5]$ m/s

The hyper parameter settings of DDPG, TD3, and SAC algorithms are shown in Tables 2–4, respectively. The training of DDPG is unstable and the learning rate of its value function is lower than TD3.

Table 2. Parameter settings of DDPG.

Parameters	Values
σ	0.02
$\|D\|$	10^6
ρ	0.998
Learning rate of policy networks	10^{-3}
Learning rate of value function networks	5×10^{-4}

Table 3. Parameter settings of TD3.

Parameters	Values
σ_1	0.02
σ_2	0.02
c	0.02
$\|D\|$	10^6
ρ	0.995
Learning rate of policy networks	10^{-3}
Learning rate of value function networks	10^{-3}

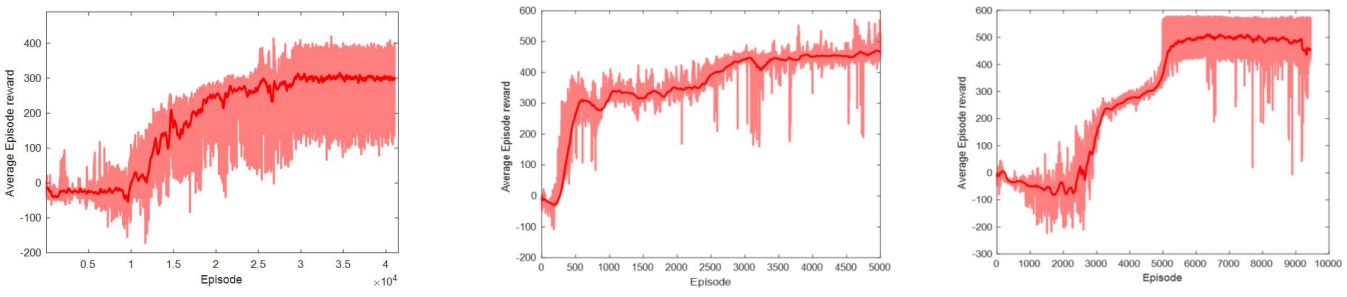
Table 4. Parameter settings of SAC.

Parameters	Values
$\|D\|$	10^6
ρ	0.995
α	0.05
Learning rate of policy networks	10^{-3}
Learning rate of value function networks	10^{-3}

4.2. Simulation Results

We deployed training algorithms according to the setting of environment and algorithm parameters listed above, and the reward change curves of DDPG, TD3, and SAC in the training process are shown in Figure 5.

The dark red curve is the average reward. From Figure 5a, the episode reward starts to rise at around episode 10,000 and continues to increase until episode 20,000. Though the curve of average reward looks stable, the episode reward fluctuates between 100 and 400, which is very unstable.



(**a**) (**b**) (**c**)

Figure 5. The curves of accumulative reward for each training episode. (**a**) DDPG. (**b**) TD3. (**c**) SAC.

It is obvious from the reward curve that the performance of the TD3 agent improves significantly after 700 episodes of training. After 5000 episodes of training, the average reward converges at 450. Compared with DDPG, the learning speed of TD3 is faster and

the performance is more stable, which is the result of the three tricks of "Clipped Double-Q Learning", "Target Policy Smoothing", and "Delayed Policy Updates".

The training of the SAC agent experienced a significant improvement at around episode 3000 and 5000, respectively, and finally stabilized and the reward converged to 500. Due to the introduction of the "Clipped Double-Q Learning" like TD3, plus the inherent smoothing characteristics of the stochastic strategy, the training process of SAC fluctuates within a very small range, and the exploration of the environment is sufficient. The accumulated rewards of some episodes are close to 600, which is higher than DDPG and SAC.

The reference velocity is set as $v_d = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$m/s, and the policy obtained from training is used to control the speed. With the initial velocity $v_b^0 = \begin{bmatrix} -2.0 & 2.0 & 10.0 \end{bmatrix}$m/s, the curves of the velocity deviation are shown in Figure 6.

All of the trained controllers can keep the velocity deviation within a certain range. But there is a continuous oscillation in the velocity control by DDPG agent. The agent trained by SAC algorithm performs the best, which gets the velocity error converges to 0 and keeps it stable.

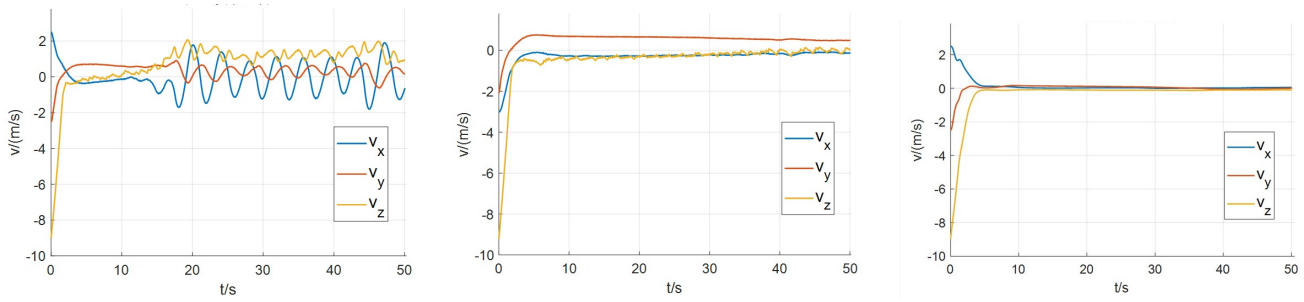


Figure 6. The curves of velocity deviation in velocity tracking experiments. (**a**) DDPG. (**b**) TD3. (**c**) SAC.

Besides velocity tracking experiments, we validate the trained policy of each algorithm by soft landing tests. Through 100 shooting experiments, landing statistics are shown in Table 5. Among the three, SAC has the highest landing success rate of 96%, while DDPG terminates 26 times due to attitude over constraint, which is caused by continuous oscillation when tracking reference velocity. In the soft landing process, the reference velocity changes dynamically, causing the oscillation more serious and leading to a higher failure rate.

Table 5. Landing success rate.

Parameters	DDPG	TD3	SAC
Number of experiments	100	100	100
Success rate of soft landing	74%	92%	96%

Taking the target landing site as the origin, the landing trajectory and landing point distribution of the three controllers are shown in Figures 7 and 8, respectively. By analyzing the distribution of landing points, DDPG and SAC have nearly the same accuracy of 100 m in successful landing cases, while some landing points of SAC distance 200 m from the origin.

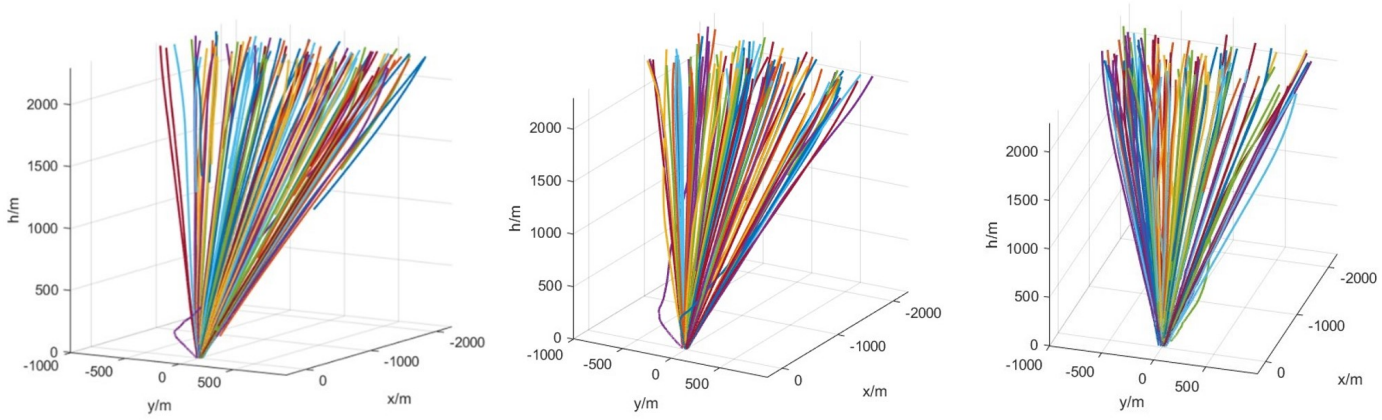


Figure 7. Landing trajectories of lander under the control of the trained velocity controller. (**a**) DDPG. (**b**) TD3. (**c**) SAC.

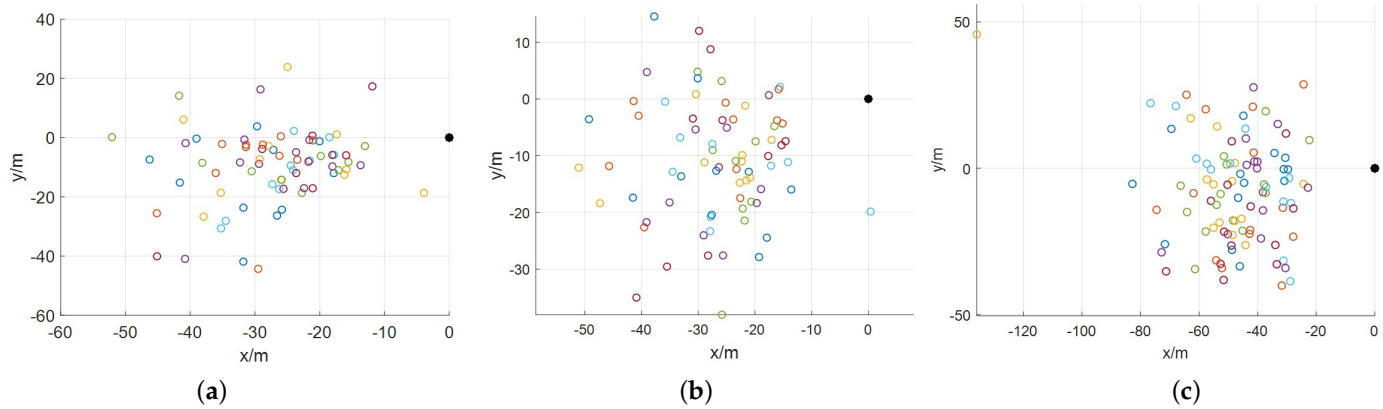


Figure 8. 2D distribution of landing points. (**a**) DDPG. (**b**) TD3. (**c**) SAC.

5. Conclusions

This paper presents an end-to-end soft landing control algorithm based on RL. First, the 6DOF soft landing dynamics model is established and the soft landing problem of the powered descent phase is formulated. The theoretical basis of RL is briefly introduced. Then, to solve the problem of sparse reward, which makes it hard for the policy to converge, the reward function including process reward is designed. Besides, the fuel consumption penalty and constraints violation penalty are included in the reward function to optimize fuel consumption and keep attitude angle within constraints. Moreover, the networks architecture of the RL algorithms used is designed. The value functions of DDPG, TD3, and SAC are approximated by deep neural networks that have the same architecture. Both DDPG and TD3 have a policy network that outputs deterministic action, while the action output of SAC is sampled according to a Gaussian distribution characterized by the output of its policy network. Finally, simulations of training are carried out to evaluate the algorithm proposed. The results show that the performance varies between different RL frameworks, and the agent trained by SAC tracks the reference velocity best. In addition, the trained policy is deployed to soft landing experiments, results of which demonstrate the validity of the algorithm proposed. Future work will focus on the stability guarantee of RL-based soft landing algorithm, which is of great importance in space exploration missions to ensure success.

Author Contributions: Conceptualization, X.X., Y.C. and C.B.; methodology, X.X. and Y.C.; software, X.X.; validation, C.B.; formal analysis, C.B.; data curation, X.X.; writing—original draft preparation, X.X. and Y.C.; writing—review and editing, C.B. and Y.C.; visualization, X.X. and Y.C.; supervision, C.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by National Natural Science Foundation of China grant number 61973101 and Aeronautical Science Foundation of China grant number 20180577005.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

DRL	Deep reinforcement learning
DDPG	Deep deterministic policy gradient
TD3	Twin Delayed DDPG
SAC	Soft Actor Critic
6DOF	6 Degree of Freedom
NLP	Nonlinear Programming
$\boldsymbol{g}$	planet gravity
T_i	thrust of each engine
$T_{\min}$	minimum thrust of each engine
$T_{\max}$	maximum thrust of each engine
ϕ	cant angle for the engines
$\boldsymbol{T}$	thrust vector composed of each engine
$\boldsymbol{T}_b$	thrust vector in lander body frame
$\boldsymbol{M}_b$	torque vector in lander body frame
$\boldsymbol{r}$	surface relative lander position vector
$\boldsymbol{v}_{\mathrm{b}}$	velocity vector
$\boldsymbol{C}_{\mathrm{b}}^{\mathrm{e}}, \boldsymbol{C}_{\mathrm{e}}^{\mathrm{b}}$	direction cosine matrix between lander body frame and planet surface frame
$\boldsymbol{\omega}^{\mathrm{t}}$	angular rate vector in lander body frame
$\boldsymbol{Q}$	attitude quaterion
m	lander mass
$\boldsymbol{I}$	inertia matrix of the lander
I_{sp}	specific impulse for thrusters
a, b, c	lander sides of length
J	optimization target
G_t	agent accumulated return in an episode
$v_{\lim}, \varphi_{\lim}, \psi_{\lim}, \theta_{\lim}, \omega_{\lim}$	limited range for velocity, attitude angle and angular rate to achieve soft landing
$\boldsymbol{v}_{ref}$	reference velocity
$\lambda, \alpha, \beta, \eta, \kappa$	reward coefficients
T_s	simulation sample time
T_f	maximum simulation time in one episode

References

1. Sanguino, T.D.J.M. 50 years of rovers for planetary exploration: A retrospective review for future directions. *Robot. Auton. Syst.* **2017**, *94*, 172–185.
2. Lu, B. Review and prospect of the development of world lunar exploration. *Space Int.* **2019**, *481*, 12–18.
3. Xu, X.; Bai, C.; Chen, Y.; Tang, H. A Survey of Guidance Technology for Moon /Mars Soft Landing. *J. Astronaut.* **2020**, *41*, 719–729.
4. Sostaric, R.R. Powered descent trajectory guidance and some considerations for human lunar landing. In Proceedings of the 30th Annual AAS Guidance and Control Conference, Breckenridge, CO, USA, 3–7 February 2007.
5. Tata, A.; Salvitti, C.; Pepi, F. From vacuum to atmospheric pressure: A review of ambient ion soft landing. *Int. J. Mass Spectrom.* **2020**, *450*, 116309. [CrossRef]
6. He, X.S.; Lin, S.Y.; Zhang, Y.F. Optimal Design of Direct Soft-Landing Trajectory of Lunar Prospector. *J. Astronaut.* **2007**, *2*, 409–413.
7. Cheng, R.K. Lunar Terminal Guidance, Lunar Missions and Exploration. In *University of California Engineering and Physical Sciences Extension Series*; Leondes, C.T., Vance, R.W., Eds.; Wiley: New York, NY, USA, 1964; pp. 308–355.

8. Citron, S.J.; Dunin, S.E.; Meissinger, H.F. A terminal guidance technique for lunar landing. *AIAA J.* **1964**, *2*, 503–509. [CrossRef]
9. Hull, D.G.; Speyer, J. Optimal reentry and plane-change trajectories. In Proceedings of the AIAA Astrodynamics Specialist Conference, Lake Tahoe, NV, USA, 3–5 August 1981.
10. Pellegrini, E.; Russell, R.P. A multiple-shooting differential dynamic programming algorithm. Part 1: Theory. *Acta Astronaut.* **2020**, *170*, 686–700. [CrossRef]
11. Bolle, A.; Circi, C.; Corrao, G. Adaptive Multiple Shooting Optimization Method for Determining Optimal Spacecraft Trajectories. U.S. Patent 9,031,818, 12 May 2015.
12. Bai, C.; Guo, J.; Zheng, H. Optimal Guidance for Planetary Landing in Hazardous Terrains. *IEEE Trans. Aerosp. Electron. Syst.* **2020**, *56*, 2896–2909. [CrossRef]
13. Chandler, D.C.; Smith, I.E. Development of the iterative guidance mode with its application to various vehicles and missions. *J. Spacecr. Rocket.* **1967**, *4*, 898–903. [CrossRef]
14. Song, Z.; Wang, C. Powered soft landing guidance method for launchers with non-cluster configured engines. *Acta Astronaut.* **2021**, *189*, 379–390. [CrossRef]
15. Amrutha, V.; Sreeja, S.; Sabarinath, A. Trajectory Optimization of Lunar Soft Landing Using Differential Evolution. In Proceedings of the 2021 IEEE Aerospace Conference (50100), Big Sky, MT, USA, 6–13 March 2021; pp. 1–9.
16. Sánchez-Sánchez, C.; Izzo, D. Real-time optimal control via deep neural networks: study on landing problems. *J. Guid. Control Dyn.* **2018**, *41*, 1122–1135. [CrossRef]
17. Furfaro, R.; Bloise, I.; Orlandelli, M.; Di Lizia, P.; Topputo, F.; Linares, R. Deep learning for autonomous lunar landing. In Proceedings of the 2018 AAS/AIAA Astrodynamics Specialist Conference, Snowbird, UT, USA, 19–28 August 2018; Volume 167, pp. 3285–3306.
18. Furfaro, R.; Bloise, I.; Orlandelli, M.; Di Lizia, P.; Topputo, F.; Linares, R. A recurrent deep architecture for quasi-optimal feedback guidance in planetary landing. In Proceedings of the IAA SciTech Forum on Space Flight Mechanics and Space Structures and Materials, Moscow, Russia, 13–15 November 2018; pp. 1–24.
19. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Graves, A.; Antonoglou, I.; Wierstra, D.; Riedmiller, M. Playing atari with deep reinforcement learning. *arXiv* **2013**, arXiv:1312.5602.
20. Kiran, B.R.; Sobh, I.; Talpaert, V.; Mannion, P.; Al Sallab, A.A.; Yogamani, S.; Pérez, P. Deep reinforcement learning for autonomous driving: A survey. *arXiv* **2020**, arXiv:2002.00444.
21. Mohammed, M.Q.; Chung, K.L.; Chyi, C.S. Review of Deep Reinforcement Learning-based Object Grasping: Techniques, Open Challenges and Recommendations. *IEEE Access* **2020**, 8, 178450–178481. [CrossRef]
22. Acikmese, B.; Ploen, S.R. Convex programming approach to powered descent guidance for mars landing. *J. Guid. Control Dyn.* **2007**, *30*, 1353–1366. [CrossRef]