Robust DDPG Reinforcement Learning Differential Game Guidance in Low-Thrust, Multi-Body Dynamical Environments

Hadi Nobahari
Department of Aerospace Engineering
Sharif University of Technology
Tehran, Iran
nobahari@sharif.edu

Ali Baniasad
Department of Aerospace Engineering
Sharif University of Technology
Tehran, Iran
ali_baniasad@ae.sharif.edu

***Abstract*—Onboard autonomy is a critical enabler for increasingly complex deep-space missions. In nonlinear dynamical environments, designing computationally efficient and robust guidance strategies remains a significant challenge. Traditional approaches often rely on simplifying assumptions in the dynamical model or require substantial computational resources, limiting their applicability for onboard implementation. This research introduces a robust reinforcement learning-based differential game approach to develop an adaptive closed-loop controller for low-thrust spacecraft guidance in multi-body dynamical environments. The proposed controller is trained to handle large initial deviations, enhance resilience against disturbances, and augment conventional targeting guidance methods. Unlike traditional approaches, it does not require explicit knowledge of the dynamical model, allowing direct interaction with the nonlinear equations of motion to create a generalizable learning framework. High-performance computing is leveraged to train a robust neural network controller, which can be deployed onboard to generate real-time low-thrust control profiles without overloading the flight computer. The effectiveness of this method is demonstrated through sample transfers between Lyapunov orbits in the Earth-Moon system, where the controller exhibits strong robustness to perturbations and generalizes effectively across different mission scenarios and low-thrust engine models. This work highlights the potential of reinforcement learning-based differential game approaches in advancing autonomous spacecraft navigation in complex gravitational environments.**

***Index Terms*—component, formatting, style, styling, insert.**

I. Introduction

II. Problem Formulation

The three-body problem provides a representative dynamical model for natural motion in cislunar space. Although the proposed guidance framework is not dependent on a specific dynamical model, the planar Circular Restricted Three-Body Problem (CR3BP) is employed as a test environment for preliminary evaluation. The CR3BP presents a complex yet relevant dynamical setting for upcoming space missions while maintaining a sufficiently low-fidelity structure for an initial assessment of the guidance scheme. Furthermore, low-thrust propulsion is incorporated to evaluate the algorithm's performance under conditions of limited control authority and significant nonlinearities. While the proposed guidance scheme directly generates a control history, an alternative operational approach involves utilizing the trained neural network to provide an initial guess for numerical methods such as targeting or optimization techniques. To illustrate this approach, a direct multiple shooting algorithm is included in the analysis, demonstrating the neural network's contribution to onboard targeting capabilities.

A. Dynamical Model

The Circular Restricted Three-Body Problem (CR3BP) models the motion of an infinitesimal mass under the gravitational influence of two primary celestial bodies. In this framework, two spherically symmetric masses, Earth and Moon, constitute the primary system, orbiting their common barycenter in circular trajectories. The third body, spacecraft, moves freely with respect to the barycenter, as illustrated in Fig. 1. The relative size of the primaries is characterized by the mass ratio

$$\mu = \frac{M_2}{M_1 + M_2}, \tag{1}$$

where M_1 is Earth mass and M_2 is Moon mass. Furthermore, the model assumes that the mass of the spacecraft is negligible compared to those of the primary bodies, thereby exerting no influence on their motion. The state of the spacecraft, defined by its position and velocity vectors $\mathbf{r}$ and $\mathbf{v}$, respectively, is represented as

$$\boldsymbol{\rho}_{\text{spatial}} = \begin{bmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^T. \tag{2}$$

These state components are propagated with respect to the barycenter in a rotating reference frame, depicted by dashed lines in Fig. 1.

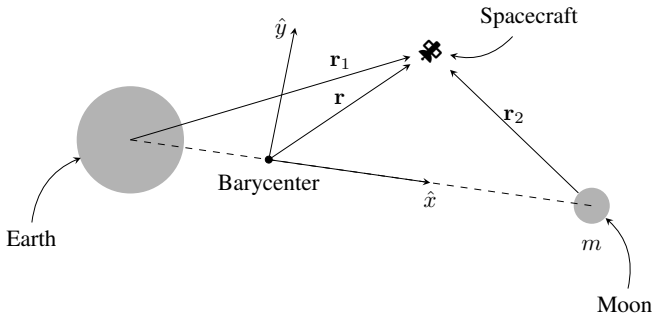

Fig. 1. Planar CR3BP vector definitions

Many mission architectures benefit from the integration of low-thrust electric propulsion systems. Unlike traditional chemical propulsion engines, electric propulsion systems offer significantly higher efficiency; however, they are characterized by delivering energy changes over extended time periods. One prominent type of low-thrust engine employed in Solar Electric Propulsion (SEP) systems is the ion thruster, which is powered by solar panels mounted on the spacecraft. Ion thrusters have been successfully utilized in a number of space missions, such as NASA's *Deep Space 1* [**?**] and *Dawn* [**?**], among others. Building upon this successful application, future low-thrust missions include *Psyche* [**?**] and the *Lunar Gateway* [**?**]. The present study assumes that spacecraft is equipped with a Constant Specific Impulse (CSI) low-thrust engine. The additional propulsion force provided by this engine modifies the natural equations of motion in the Circular Restricted Three-Body Problem (CR3BP), introducing low-thrust terms to the governing equations.

$$\ddot{x} = 2\dot{y} + x - \frac{(1-\mu)(x+\mu)}{r_1^3} - \frac{\mu(x-1+\mu)}{r_2^3} + u_x, \quad (3)$$

$$\ddot{y} = -2\dot{x} + y - \frac{(1-\mu)y}{r_1^3} - \frac{\mu y}{r_2^3} + u_y, \quad (4)$$

$$\ddot{z} = -\frac{(1-\mu)z}{r_1^3} - \frac{\mu z}{r_2^3} + u_z, \quad (5)$$

where the control inputs u_x, u_y, and u_z represent the thrust components along the spacecraft's body-fixed axes. The distances r_1 and r_2 are defined as

$$r_1 = \sqrt{(x+\mu)^2 + y^2 + z^2}, \quad (6)$$

$$r_2 = \sqrt{(x-1+\mu)^2 + y^2 + z^2}. \quad (7)$$

The state-space representation of the CR3BP with low-thrust terms is given by

$$\dot{\boldsymbol{\rho}} = \boldsymbol{f}(\boldsymbol{\rho}, \boldsymbol{u}), \quad (8)$$

where $\boldsymbol{\rho}$ is the state vector and $\boldsymbol{u}$ is the control input vector. The state-space representation is used to define the spacecraft's trajectory in the CR3BP, incorporating the effects of the low-thrust engine.

B. Low-Thrust Control Problem

The motion in the Circular Restricted Three-Body Problem (CR3BP) is inherently nonlinear, displaying notably sensitive dynamical structures. Consequently, the proposed guidance strategy capitalizes on the influence of the low-thrust terms to achieve the desired dynamical behavior. As elaborated by Cox et al. [**?**], the thrust direction is determined by the low-thrust acceleration vector, defined as:

$$\mathbf{a}_{lt} = a_{lt}\hat{u}_x\hat{i} + a_{lt}\hat{u}_y\hat{j} + a_{lt}\hat{u}_z\hat{k}, \quad (9)$$

where $\hat{u}_x$, $\hat{u}_y$, and $\hat{u}_z$ are the components of the thrust unit vector in the rotating frame of the CR3BP, and a_{lt} represents the low-thrust acceleration components in the corresponding directions. Furthermore, the thrust direction is characterized by the thrust magnitude f and the spacecraft's nondimensional mass $m = \frac{M_3}{M_{3,0}}$, where M_3 denotes the spacecraft mass at the beginning of the thrust segment and $M_{3,0}$ is the initial mass.

The thrust direction is oriented by the unit vector $\hat{u}$, which remains fixed in the rotating frame of the CR3BP during each integration step. While it may be impractical to consider a continuously changing inertial thrust direction, the current study focuses on thrust segments of 20.87 hours. The precession of the rotating frame during these intervals can be addressed when transitioning the CR3BP solutions into a higher-fidelity model. It is also important to note that propulsive capability is inversely related to the spacecraft's mass, such that as propellant is consumed, the spacecraft gains a higher thrust-to-mass ratio.

The nondimensional thrust magnitude f is computed as follows:

$$f = \frac{l^* M_{3,0}}{F_t^*}, \quad (10)$$

where F_t^* is the thrust in kilonewtons, and $M_{3,0}$ is the initial mass of the spacecraft in kilograms. For the purposes of this investigation, a sample spacecraft is considered, which has a maximum propulsive capability of $f_{\max} = 0.04$. A comparison between this sample spacecraft and other previous and planned engine capabilities is summarized in Table **??**. The sample spacecraft exhibits a propulsive capability comparable to that of the planned Psyche spacecraft, which exceeds that of Hayabusa 1, Hayabusa 2, Dawn, and Lunar IceCube, but is less than the capabilities of Deep Space 1. As a nondimensional quantity, this thrust level can represent any spacecraft with the same thrust-to-mass ratio. Specifically, $f_{\max} = 0.04$ models an 11.46 kg spacecraft equipped with a BIT-3 CubeSat engine [**?**] (intended for use on Lunar IceCube and LunaH-Map) as well as an 850.2 kg spacecraft utilizing an NSTAR engine [**?**] (employed on Deep Space 1 and Dawn).

III. Guidance Framework Design

The proposed guidance framework leverages reinforcement learning to develop a robust differential game-based guidance strategy for low-thrust spacecraft in multi-body dynamical environments. The guidance strategy is designed to handle large initial deviations, enhance resilience against disturbances, and

TABLE I
LOW-THRUST CAPABILITY OF VARIOUS SPACECRAFT, NONDIMENSIONALIZED IN THE EARTH-MOON SYSTEM.

Abbrv.	**Spacecraft**	$f_{\text{max, nondim}}$	$M_{3,0}$ **(kg)**	F_{max} **(mN)**
H1	Hayabusa 1 [**?**]	$1.640 \cdot 10^{-2}$	510	22.8
H2	Hayabusa 2 [**?**]	$1.628 \cdot 10^{-2}$	608.6	27.0
LIC	Lunar IceCube [**?**], [**?**]	$3.276 \cdot 10^{-2}$	14	1.25
Dawn	Dawn [**?**]	$2.741 \cdot 10^{-2}$	1217.8	91.0
DS1	Deep Space 1 [**?**]	$6.940 \cdot 10^{-2}$	486.3	92.0
Psyche	Psyche [**?**], [**?**]	$4.158 \cdot 10^{-2}$	2464	279.3
s/c	Sample spacecraft	$4 \cdot 10^{-2}$	n/a	n/a

augment conventional targeting guidance methods. The framework is implemented using the Deep Deterministic Policy Gradient (DDPG) algorithm, which is a model-free, off-policy reinforcement learning algorithm that combines the stability of deterministic policy gradients with the flexibility of deep neural networks. The DDPG algorithm is well-suited for continuous action spaces and has been successfully applied to a variety of control problems, including robotic manipulation and locomotion tasks. The guidance strategy is trained using a differential game formulation, where two spacecraft are modeled as agents in a competitive game. The agents interact with each other and the environment to learn optimal control policies that maximize their rewards. The guidance strategy is implemented in a simulated environment that captures the dynamics of the Circular Restricted Three-Body Problem (CR3BP) with low-thrust terms. The effectiveness of the guidance strategy is demonstrated through sample transfers between Lyapunov orbits in the Earth-Moon system, where the controller exhibits strong robustness to perturbations and generalizes effectively across different mission scenarios and low-thrust engine models. The proposed guidance framework represents a significant advancement in autonomous spacecraft navigation in complex gravitational environments and has the potential to enable a wide range of deep-space missions.

A. Reinforcement Learning Overview

Reinforcement learning is a machine learning paradigm that enables agents to learn optimal control policies by interacting with an environment to maximize cumulative rewards. The agent observes the state of the environment, selects actions based on its policy, receives rewards from the environment, and updates its policy to improve its performance over time. The goal of the agent is to learn an optimal policy that maximizes its expected cumulative reward.

1) Neural Networks: Neural networks are a class of machine learning models that are inspired by the structure and function of the human brain. They consist of interconnected nodes, or neurons, organized in layers. Each neuron takes input from the previous layer, applies a nonlinear activation function, and produces an output that is passed to the next layer. Neural networks are capable of learning complex patterns and relationships in data and have been successfully applied to a wide range of machine learning tasks, including image recognition, natural language processing, and reinforcement learning.

2) Reinforcement Learning Fundamentals: Reinforcement learning (RL) refers to a class of algorithms wherein a goal-directed agent seeks to accomplish a task through interaction with an environment. An agent is defined as a controller that maps observed variables (states) to corresponding actions. The learning process initiates with the agent exploring the environment through trial-and-error. The environment conveys relevant information about its dynamics to the agent in the form of a state signal, which the agent utilizes to determine an appropriate action. In response to the action, the environment updates its state and computes a numerical reward, representing the immediate benefit derived from the action taken. This iterative process continues, enabling the agent to refine its policy—the strategy it employs to make decisions—with the objective of maximizing the reward signal over time.

In many RL scenarios, terminal conditions are introduced, resulting in the cessation of the learning process. In such cases, the environment typically resets to an initial configuration, and the learning process recommences. These situations are classified as episodic tasks, where each episode represents a discrete attempt by the agent to solve a problem. A high-level schematic of the RL process is presented in Fig. 3. This diagram illustrates the three key signals—state, action, and reward—that facilitate communication between the agent and the environment at discrete time steps, t and $t+1$. While the RL literature predominantly uses the terminology of agent, environment, and action, these terms are conceptually analogous to the more commonly used engineering terminology of controller, controlled system (or plant), and control signal [**?**].

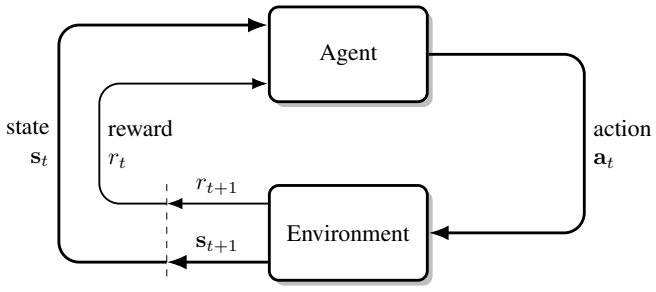

Fig. 2. The agent-environment process in a Markov decision process [1].

Without external supervision, the agent uncovers complex dynamical structures that inform its decision-making. This process is characterized as a Markov Decision Process (MDP), which incorporates both feedback and associative actions, representing a classical formalization of sequential decision-making problems [**?**]. The MDP serves as the idealized mathematical framework for the problem, enabling theoretical analysis of the reinforcement learning (RL) process. In the infinite-horizon discounted case, the MDP is formulated as a tuple $\langle S, A, P, r, q_0, \gamma \rangle$, where S and A denote the sets of all possible states and actions, respectively; $P : S \times A \times S \to [0, 1]$ represents the state-transition probability distribution; $r : S \to \mathbb{R}$ is the reward function; q_0 is the distribution of the initial states s_0; and $\gamma \in [0, 1]$ is the discount factor [**?**].

For a problem to be accurately cast as an MDP, it must satisfy the Markov property, which asserts that future states depend solely on the current state, and not on the sequence of prior events [**?**]. In many practical applications, only partial information is available, and the agent is provided with a subset of the total environmental data. This partial information is referred to as the "observation," which serves as an analog to the state signal. Such systems, characterized by reduced information, are classified as Partially Observable Markov Decision Processes (POMDPs). The distinction between state and observation is crucial in POMDPs, as it emphasizes the fact that the agent operates based on incomplete information. However, this investigation assumes a fully observable MDP, and for the sake of simplicity, the observation o_t is treated as the state s_t, rendering the distinction unnecessary.

An agent seeks to develop a policy that maximizes its expected future reward by balancing immediate and future returns. This balance is formalized by defining the expected return G_t as the sum of future discounted rewards, expressed as:

$$G_t = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \cdots + \gamma^{T-t-1} r_T \tag{11}$$

$$= \sum_{k=t+1}^{T} \gamma^{k-t-1} r_k, \tag{12}$$

where $\gamma \in [0, 1]$ is the discount factor that determines the relative importance of immediate versus future rewards. When $\gamma = 1$, all future rewards are considered equally important; when $\gamma = 0$, only the immediate reward is taken into account. The definition of expected return leads to the formalization of the value function, which is estimated in nearly all RL algorithms and informs the agent of the quality of a particular state. For an MDP, the state-value function $V^\pi(s_t)$ is defined as

$$V^\pi(s_t) = \mathbb{E}_\pi \left[G_t | s_t \right] = \mathbb{E}_\pi \left[\sum_{k=t+1}^{T} \gamma^{k-t-1} r_k | s_t \right], \tag{13}$$

where π denotes the policy that the agent follows. The action-value function $Q^\pi(s_t, a_t)$ is defined as

$$Q^\pi(s_t, a_t) = \mathbb{E}_\pi \left[G_t | s_t, a_t \right] = \mathbb{E}_\pi \left[\sum_{k=t+1}^{T} \gamma^{k-t-1} r_k | s_t, a_t \right], \tag{14}$$

which represents the expected return when the agent is in state s_t, takes action a_t, and follows policy π thereafter. The value functions and action-value functions are related by

$$V^\pi(s_t) = \mathbb{E}_{a_t \sim \pi} \left[Q^\pi(s_t, a_t) \right]. \tag{15}$$

Policy optimization methods aim to directly learn a parameterized policy, $\pi_\theta(a|s)$, where θ represents the policy parameter vector. Unlike value optimization methods, such as Q-Learning and Deep Q-Networks (DQN), policy optimization methods are particularly well-suited for problems involving continuous action spaces. The ability to incorporate both continuous state and action spaces provides a significant advantage in complex control tasks, where discretization can be a limitation, and also enhances the extendibility to higher-dimensional dynamical models.

A notable subset of reinforcement learning (RL) methods arises from hybrid algorithms, commonly known as actor-critic methods. These approaches concurrently learn the policy (i.e., the actor) and the value function (i.e., the critic), where both the actor and the critic are often represented as neural networks. Among the various actor–critic schemes, Deep Deterministic Policy Gradient (DDPG) [**?**] is particularly relevant for continuous action spaces and is employed in this investigation. DDPG has shown significant success in high-dimensional continuous control tasks and is the method of choice for this study.

3) Deep Deterministic Policy Gradient:

IV. Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) is an off-policy, model-free, and actor-critic reinforcement learning algorithm that is particularly well-suited for problems involving continuous action spaces. It is an extension of the Deterministic Policy Gradient (DPG) algorithm, incorporating deep neural networks to approximate both the policy and value functions. DDPG is based on the idea that the policy can be directly optimized by learning a deterministic policy function, and the value function can be approximated using the Q-function. Below, we describe the key components and the mathematical formulation of DDPG.

A. Policy and Value Functions

In DDPG, the agent learns a deterministic policy $\pi_\theta(s)$, which maps states s directly to actions a. The policy is parameterized by a neural network with weights θ, and the goal is to find the policy that maximizes the expected return.

The action-value function (Q-function) is defined as the expected return of taking action a in state s, following the policy π. The Q-function is given by:

$$Q^{\pi}(s,a) = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right], \tag{16}$$

where r_t denotes the reward received at time step t, and $\gamma \in [0,1]$ is the discount factor. The goal of DDPG is to learn the optimal action-value function, $Q^{\pi^*}(s,a)$, which provides the expected return for each state-action pair under the optimal policy.

B. DDPG Objective

The DDPG algorithm involves optimizing two primary functions: the policy function (actor) and the Q-function (critic). The actor is responsible for selecting actions, while the critic evaluates these actions by computing the Q-value.

The policy gradient is given by the deterministic policy gradient theorem, which states that the gradient of the expected return with respect to the policy parameters can be expressed as:

$$\nabla_\theta J(\theta) = \mathbb{E}\left[\nabla_\theta \pi_\theta(s) \nabla_a Q(s,a)|_{a=\pi_\theta(s)}\right], \tag{17}$$

where $\nabla_\theta \pi_\theta(s)$ is the gradient of the policy with respect to its parameters, and $\nabla_a Q(s,a)$ is the gradient of the Q-function with respect to the action.

For the Q-function, the update rule is based on the Bellman equation for continuous action spaces, which provides a target for the Q-function update. The Bellman backup equation is given by:

$$y = r + \gamma Q'(s', a'; \theta^-), \tag{18}$$

where r is the immediate reward, s' is the next state, a' is the action chosen by the target policy $\pi_{\theta^-}(s')$, and $Q'(s', a'; \theta^-)$ is the target Q-value. The target Q-value is estimated using the target Q-network, which has parameters θ^- and is periodically updated to stabilize training.

C. DDPG Algorithm

The DDPG algorithm is implemented using two neural networks: the actor network $\pi_\theta(s)$ and the critic network $Q_\theta(s,a)$. These networks are trained using the following algorithm:

Algorithm 1 Deep Deterministic Policy Gradient

Input: initial policy parameters θ, Q-function parameters ϕ, empty replay buffer $\mathcal{D}$
Set target parameters equal to main parameters $\theta_{\text{targ}} \leftarrow \theta$, $\phi_{\text{targ}} \leftarrow \phi$
repeat
  Observe state s and select action $a = \text{clip}(\mu_\theta(s) + \epsilon, a_{Low}, a_{High})$, where $\epsilon \sim \mathcal{N}$
  Execute a in the environment
  Observe next state s', reward r, and done signal d to indicate whether s' is terminal
  Store (s, a, r, s', d) in replay buffer $\mathcal{D}$
  If s' is terminal, reset environment state.
  if it's time to update **then**
    for however many updates **do**
      Randomly sample a batch of transitions, $B = \{(s, a, r, s', d)\}$ from $\mathcal{D}$
      Compute targets

$$y(r, s', d) = r + \gamma(1-d) Q_{\phi_{\text{targ}}}(s', \mu_{\theta_{\text{targ}}}(s'))$$

      Update Q-function by one step of gradient descent using

$$\nabla_\phi \frac{1}{|B|} \sum_{(s,a,r,s',d) \in B} \left(Q_\phi(s,a) - y(r,s',d)\right)^2$$

      Update policy by one step of gradient ascent using

$$\nabla_\theta \frac{1}{|B|} \sum_{s \in B} Q_\phi(s, \mu_\theta(s))$$

      Update target networks with

$$\phi_{\text{targ}} \leftarrow \rho \phi_{\text{targ}} + (1-\rho)\phi$$
$$\theta_{\text{targ}} \leftarrow \rho \theta_{\text{targ}} + (1-\rho)\theta$$

    end for
  end if
until convergence

D. Exploration and Exploitation

In DDPG, the exploration of the environment is achieved by adding noise to the action selection process. This noise, typically Gaussian or Ornstein-Uhlenbeck noise, encourages the agent to explore the environment while still utilizing the learned policy. The balance between exploration and exploitation is maintained throughout the training process, with the noise gradually decaying as the policy converges.

1) Differential Game Formulation: In the context of zero-sum games, the differential game formulation provides a mathematical framework for modeling dynamic strategic interactions between two competing players. In such games, one player's gain is exactly the other player's loss, making it a zero-sum game. The formulation is typically used in control theory and reinforcement learning to model competitive scenarios where the players continuously adjust their strategies over time, based on the state of the game and the actions of

their opponents.

Consider a scenario where two players, denoted as Player 1 and Player 2, interact over time, and each player aims to optimize their respective performance based on their actions. The game is defined over a continuous time interval, and the state of the system evolves according to differential equations that are influenced by the players' control inputs. Let the state of the system at time t be denoted by $x(t)$, and the control inputs (actions) of Player 1 and Player 2 at time t be represented by $u_1(t)$ and $u_2(t)$, respectively.

The state dynamics of the system can be described by the following differential equations:

$$\dot{x}(t) = f(x(t), u_1(t), u_2(t)), \tag{19}$$

where $f(x(t), u_1(t), u_2(t))$ is a function that defines the system's evolution based on the current state $x(t)$ and the control inputs $u_1(t)$ and $u_2(t)$.

Each player seeks to minimize (or maximize) a cost functional, depending on the game. The objective for Player 1 is to minimize the following cost function:

$$J_1 = \int_0^T L_1(x(t), u_1(t), u_2(t))dt + -160.0_1(x(T)), \tag{20}$$

where $L_1(x(t), u_1(t), u_2(t))$ is the running cost for Player 1, and $-160.0_1(x(T))$ is the terminal cost at the end of the time horizon T. Similarly, Player 2 seeks to minimize their own cost functional:

$$J_2 = \int_0^T L_2(x(t), u_1(t), u_2(t))dt + -160.0_2(x(T)), \tag{21}$$

where $L_2(x(t), u_1(t), u_2(t))$ and $-160.0_2(x(T))$ represent the running and terminal costs for Player 2, respectively.

Given that this is a zero-sum game, the cost function for Player 1 is the negative of Player 2's cost, i.e., $J_1 = -J_2$. Therefore, the objective of each player is to optimize their own strategy while considering the opponent's strategy, leading to a competitive interaction between the two players.

The solution to the differential game involves determining the optimal control inputs $u_1^*(t)$ and $u_2^*(t)$ for Player 1 and Player 2, respectively. These are typically found by solving the Hamilton-Jacobi-Bellman (HJB) equations or by using the Pontryagin's Maximum Principle (PMP). The HJB equations for Player 1 and Player 2 are derived by first constructing the Hamiltonian functions for each player:

$$H_1 = L_1(x(t), u_1(t), u_2(t)) + \lambda(t) \cdot f(x(t), u_1(t), u_2(t)), \tag{22}$$

for Player 1, and

$$H_2 = L_2(x(t), u_1(t), u_2(t)) + \mu(t) \cdot f(x(t), u_1(t), u_2(t)), \tag{23}$$

for Player 2, where $\lambda(t)$ and $\mu(t)$ are the co-state variables associated with the state dynamics. The optimal strategies for both players are determined by solving these equations, subject to the appropriate boundary conditions and constraints.

The differential game formulation for a zero-sum game captures the dynamic nature of the competition between two players, where each player's objective is to minimize their own cost while simultaneously considering the opponent's strategy. The solution involves solving optimal control problems that account for both the state dynamics and the interaction between the players, providing insights into the optimal strategies for competitive environments.

E. Environment Configuration

The environment for the differential game-based guidance strategy is designed to capture the dynamics of the Circular Restricted Three-Body Problem (CR3BP) with low-thrust terms. The CR3BP is a classical dynamical system that models the motion of a spacecraft in the vicinity of two primary bodies, such as the Earth and the Moon, where the gravitational forces of the two bodies dominate the spacecraft's motion. The low-thrust terms represent the effects of a continuous thrust engine on the spacecraft's trajectory, allowing for more precise control over the spacecraft's motion.

1) State Representation: The state of the environment is defined by the position and velocity vectors of the spacecraft in the rotating frame of the CR3BP. The state vector is given by $\boldsymbol{s} = \begin{bmatrix} \delta x & \delta y & \delta \dot{x} & \delta \dot{y} \end{bmatrix}^{\mathrm{T}}$, where δx and δy are the deviations in the spacecraft's position from the nominal trajectory, and $\delta \dot{x}$ and $\delta \dot{y}$ are the deviations in the spacecraft's velocity. The state vector captures the relative position and velocity of the spacecraft with respect to the nominal trajectory, providing the necessary information for the guidance strategy to make decisions.

2) Action Representation: The action space of the environment corresponds to the control inputs of the spacecraft, which are the thrust components along the spacecraft's body-fixed axes. The action vector is given by $\boldsymbol{a} = \begin{bmatrix} u_x & u_y \end{bmatrix}^{\mathrm{T}}$, where u_x and u_y represent the thrust components along the spacecraft's body-fixed axes. The action space allows the guidance strategy to adjust the thrust direction and magnitude to control the spacecraft's trajectory.

3) Reward Function: The reward function of the environment is designed to incentivize the spacecraft to reach the desired Lyapunov orbit while minimizing deviations from the nominal trajectory. The reward function is defined as:

$$r = -\alpha \left(\delta x^2 + \delta y^2 \right) - \beta \left(\delta \dot{x}^2 + \delta \dot{y}^2 \right), \tag{24}$$

where α and β are weighting factors that determine the importance of position and velocity deviations in the reward calculation. The reward function encourages the spacecraft to reduce its position and velocity deviations, leading to a more accurate trajectory following.

V. Mission Application: Libration Point Transfer

The proposed guidance framework is applied to a mission scenario involving a transfer between Lyapunov orbits in the Earth-Moon system. The mission objective is to transfer a

spacecraft from a lower-energy Lyapunov orbit to a higher-energy Lyapunov orbit using low-thrust propulsion. The guidance strategy is trained using the differential game-based reinforcement learning approach and is evaluated on its ability to perform the transfer accurately and efficiently. The mission application demonstrates the effectiveness of the guidance strategy in handling complex trajectory optimization tasks in multi-body dynamical environments.

A. Neural Network-Based Guidance

The guidance strategy is implemented using neural networks to approximate the policy and value functions in the DDPG algorithm. The actor network is responsible for selecting actions based on the current state of the environment, while the critic network evaluates the quality of these actions by estimating the Q-value. The neural networks are trained using the differential game formulation to optimize the spacecraft's trajectory during the Lyapunov orbit transfer. The guidance strategy leverages the capabilities of deep reinforcement learning to develop robust and adaptive control policies that can handle the complexities of the CR3BP dynamics and low-thrust propulsion.

VI. Training Validation

The training of the guidance strategy is validated through a series of simulations that test the performance of the controller in various scenarios. The simulations involve transferring the spacecraft between Lyapunov orbits with different initial conditions and perturbations to evaluate the robustness and generalization capabilities of the guidance strategy. The training validation process demonstrates the effectiveness of the guidance strategy in handling complex mission scenarios and perturbations in the Earth-Moon system.

A. Perturbation Analysis

The guidance strategy is evaluated under various perturbations to assess its robustness and resilience to disturbances. The perturbations include variations in the spacecraft's initial conditions, external disturbances, and model uncertainties. The guidance strategy is tested under different perturbation scenarios to evaluate its performance and stability in the presence of disturbances. The perturbation analysis provides insights into the controller's ability to adapt to changing conditions and maintain accurate trajectory following during the Lyapunov orbit transfer.

VII. Real-Time Implementation

The guidance strategy is implemented in real-time using a software-in-the-loop (SIL) simulation environment to validate its performance and computational efficiency. The real-time implementation involves running the guidance strategy on a high-fidelity simulation model of the CR3BP with low-thrust terms and evaluating its performance in real-world scenarios. The real-time implementation demonstrates the feasibility of deploying the guidance strategy on an actual spacecraft and its ability to handle the computational requirements of autonomous navigation in deep-space missions.

A. ROS-Based System Integration

The guidance strategy is integrated into a Robot Operating System (ROS) framework to facilitate communication between the guidance controller and the spacecraft's onboard systems. The ROS-based system integration enables seamless data exchange between the guidance strategy, the spacecraft's sensors, and the low-thrust propulsion system. The integration of the guidance strategy with ROS enhances the autonomy and adaptability of the spacecraft's navigation system, enabling it to respond to changing mission requirements and environmental conditions.

B. Computational Efficiency in Real-Time Execution

The computational efficiency of the guidance strategy is evaluated in real-time execution to ensure that it meets the stringent timing requirements of deep-space missions. The guidance strategy is benchmarked against the mission's timing constraints to verify that it can generate control commands within the specified time frame. The computational efficiency analysis ensures that the guidance strategy can operate in real-time on a spacecraft's onboard computer system and meet the mission's performance requirements.

VIII. Conclusion

The proposed guidance framework leverages reinforcement learning to develop a robust differential game-based guidance strategy for low-thrust spacecraft in multi-body dynamical environments. The guidance strategy is designed to handle large initial deviations, enhance resilience against disturbances, and augment conventional targeting guidance methods. The framework is implemented using the Deep Deterministic Policy Gradient (DDPG) algorithm, which is a model-free, off-policy reinforcement learning algorithm that combines the stability of deterministic policy gradients with the flexibility of deep neural networks. The guidance strategy is trained using a differential game formulation, where two spacecraft are modeled as agents in a competitive game. The agents interact with each other and the environment to learn optimal control policies that maximize their rewards. The guidance strategy is implemented in a simulated environment that captures the dynamics of the Circular Restricted Three-Body Problem (CR3BP) with low-thrust terms. The effectiveness of the guidance strategy is demonstrated through sample transfers between Lyapunov orbits in the Earth-Moon system, where the controller exhibits strong robustness to perturbations and generalizes effectively across different mission scenarios and low-thrust engine models. The proposed guidance framework represents a significant advancement in autonomous spacecraft navigation in complex gravitational environments and has the potential to enable a wide range of deep-space missions.

References

[1] Sutton, R. S., Barto, A. G. (2018). Reinforcement Learning: An Introduction. The MIT Press.