Development of a Lunar Rover Simulator with an Interface for Reinforcement Learning

Assawayut Khunmaturod
School of Electrical Engineering
Korea Advanced Institute of Science and Technology
Daejeon, Republic of Korea
assawayut@kaist.ac.kr

Dong Eui Chang
School of Electrical Engineering
Korea Advanced Institute of Science and Technology
Daejeon, Republic of Korea
dechang@kaist.ac.kr

2024 9th International Conference on Automation, Control and Robotics Engineering (CACRE) | 979-8-3503-5030-2/24/$31.00 ©2024 IEEE | DOI: 10.1109/CACRE62362.2024.10634867

***Abstract*—In this paper, we propose an open-source lunar rover simulator integrated with a reinforcement learning framework. We incorporate lunar environmental effects in our simulator to enhance the rover locomotion realism while employing PyBullet to simulate the dynamics of a multi-degree-of-freedom planetary rover. We also extend our lunar rover simulator to interface with a reinforcement learning framework using OpenAI Gym environment and a compatible training workflow. We demonstrate the success of this integration through an autonomous rover navigation task on the lunar environment in our simulator using two reinforcement learning algorithms, DDPG and TD3. The lunar rover simulator is available at** `https://github.com/assawayut/LunarRoverSim` **and an accompanying video can be accessed at** `https://www.youtube.com/watch?v=8HMqAkDbpqI`**.**

***Index Terms*—Simulation and Animation, Space Robotics and Automation, Reinforcement Learning**

I. Introduction

In recent years, lunar exploration missions have garnered significant attention from numerous space agencies and research organizations. These missions typically employ unmanned ground vehicles (UGVs) or autonomous planetary rovers to conduct exploration tasks on the lunar surface. However, due to the significant costs and risks associated with physical hardware testing, there is a growing need to assess and validate rovers in a simulated environment. The planetary rover simulator, in this context, plays a pivotal role in lunar exploration by providing a preliminary evaluation of rover performance.

One of the fundamental factors in the rover simulator is to create a realistic simulated planetary environment that encompasses physical and visual aspects. Lunar environments, unlike the Earth environment, often exhibit extreme conditions that can impact rover locomotion, so they must be addressed in the simulation. Since rovers must operate and traverse the lunar surface, their interaction with the terrain, particularly wheel-terrain interaction, must be accurately modeled and integrated into the simulator.

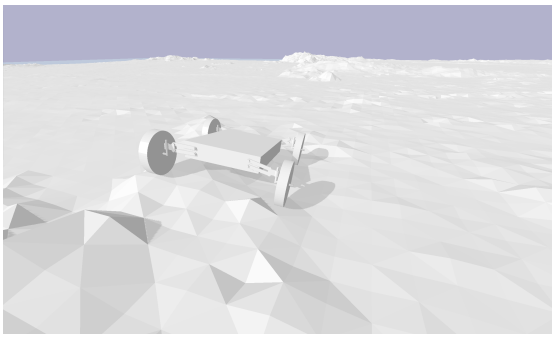

Fig. 1: Display of planetary rover simulator

Another challenge concerning the rover simulators is the limited accessibility of well-developed simulators. While several comprehensive rover simulators exist, most of them are not publicly accessible, hindering progress in the field of rover development. Thus, there is a growing demand for open-source rover simulators.

Moreover, with the emergence of deep learning (DL), there is a heightened requirement for simulators that can integrate with data-driven algorithms. However, generating large-scale data for DL training involves a trade-off between runtime and simulation realism. In this regard, most existing rover simulators, such as [1]–[5], lack a readily available interface for DL applications. Hence, developing a rover simulator capable of integrating and being compatible with DL becomes imperative.

Motivated by the aforementioned facts, we propose an open-source lunar rover simulator seamlessly integrated with a reinforcement learning (RL) interface. We develop the simulator on top of the PyBullet physics simulator, offering high fidelity in multi-DOF rover dynamics and the contact solver. We enhance realism by incorporating environmental effects. Moreover, we establish integration between our simulator and the RL framework and showcase this integration through an autonomous rover navigation task using our simulator and open-source RL algorithm library. Our main contributions are

This work was in part supported by the CARAI grant funded by DAPA and ADD (No. UD190031RD), by IITP (No. 2022-0-00469, No. 20210005900012003), by NRF (No. 2021R1A2C2010585) and by BK21 Program.

as follows:

1) We develop an open-source lunar rover simulator for lunar missions, where we additionally incorporate and model the environmental influence on rover locomotion into our simulator to enhance the degree of realism.
2) We establish the interface between our lunar rover simulator and the RL framework via OpenAI Gym environment and an open-source RL workflow. We demonstrate an example of using RL for an autonomous rover navigation task in our lunar rover simulator.

The paper is organized as follows. Section II reviews the related work on existing planetary rover simulators and their integration with RL. Section III explains the methodology of the simulators, especially the rover and lunar environment. Section IV describes simulator and RL interface. Section V shows an example of using RL for a navigation task in our simulator. Section VI concludes the paper.

II. RELATED WORK

Designed to support planetary missions, various planetary rover simulators have been developed by space organizations. Notably, Rover Analysis, Modeling and Simulation (ROAMS) [1] was created by Mars Technology Program under NASA's supervision, serving as a testbed for Mars rovers. The simulator incorporates mechanical and electrical models of the rover while also accounting for terrain and the dynamics of wheel-soil interaction. Another significant advancement is ARTEMIS simulator [2], designed to evaluate the locomotion of rovers across diverse soil types, with the interaction model derived from wheel-soil experiments. Furthermore, to support lunar missions, NASA developed Lunar Rover Driving Simulator [3]. Based on open-source Gazebo and ROS (Robotic Operating System), this simulator provides a high-fidelity visual simulation of lunar terrain and middleware communication for off-board flight software testing.

MMX Rover simulation [4], developed by DLR, aims to provide a unified rover simulator to support various planetary rover experiments, including driving, spacecraft separation, and up-righting from landing. ESA has developed 3DRover [5], an end-to-end rover mission and onboard algorithm testing simulator. However, despite the accuracy of contact interaction or the high-realistic visual scenarios, these rover simulators are neither available in public nor accessible free of charge. MarsSim [6], which aspires to provide high-fidelity physical and visual aspects to the planetary rover simulator, is intended to be open-source but has yet to be released.

In terms of RL and rover simulator, OpenAI Gym [7] was developed to standardize the API interface of RL framework with the physics simulator. As a result, many rover simulators have been developed on top of an open-source physics simulator to facilitate the implementation of RL algorithms. Works such as [8], [9], [10], and [11] utilize Gazebo as a physics engine simulator, interfacing with OpenAI Gym in order to train the rover for navigation and grasping tasks. In addition, the work presented in [12] combines CoppeliaSim and OpenAI to build an open-source RL rover simulator platform. Since Gazebo was not primarily developed to interface with OpenAI Gym, it requires ROS as communication middleware, and CoppeliaSim currently requires a license. All of these highlight the need for unified open-source codes and RL integration of the planetary rover simulator.

III. SIMULATOR METHODOLOGY

This section outlines the methodology employed in constructing our lunar rover simulator utilizing PyBullet physics engine. The GUI interface of the simulator is depicted in Figure 1, and the overall workflow is depicted in Figure 2.

A. PyBullet

We use PyBullet [13] for the rover simulator based on several key considerations. Firstly, the open-source nature of PyBullet facilitates broader accessibility within the research and development community. Secondly, accurate modeling of rover-terrain interaction, including precise contact dynamics between rigid and deformable bodies, is essential for the simulation, which is an aspect where PyBullet outperforms other considered simulators [14]. Furthermore, PyBullet supports various robot model formats, such as URDF, SDF, and MJCF, enhancing flexibility for incorporating diverse models.

B. Rover Model

Our rover models are based on two well-established lunar rovers, LUVMI-X [15] and VIPER [16], as depicted in Figure 4a and 4b respectively. Both share an identical mechanical structure comprising twelve actuated joints, including four actuators mounted on each wheel for speed control, four revolute steering joints for steering control, and four revolute suspension joints for active suspension control. To incorporate these models into the simulator, we first design these models on CAD drawing software. Then, we export URDF files, upload them to the simulator, and fine-tune the joint parameters to achieve rover stability.

It should be noted, however, that the VIPER rover model encompasses a closed-loop mechanism within its suspension link, following a four-bar linkage structure. The standard URDF format does not inherently support this structure. Consequently, one additional constraint in the suspension link is imposed in the simulator.

C. The Moon Environment

To enhance the realism and fidelity of rover operations in our simulator, we model various environmental factors on the moon. These factors include gravity, temperature, terrain, and wheel-soil interaction.

1) Gravity: The gravitational acceleration on the lunar surface is approximately 1.625m/s^2. The value of gravity can be adjusted directly through PyBullet API.

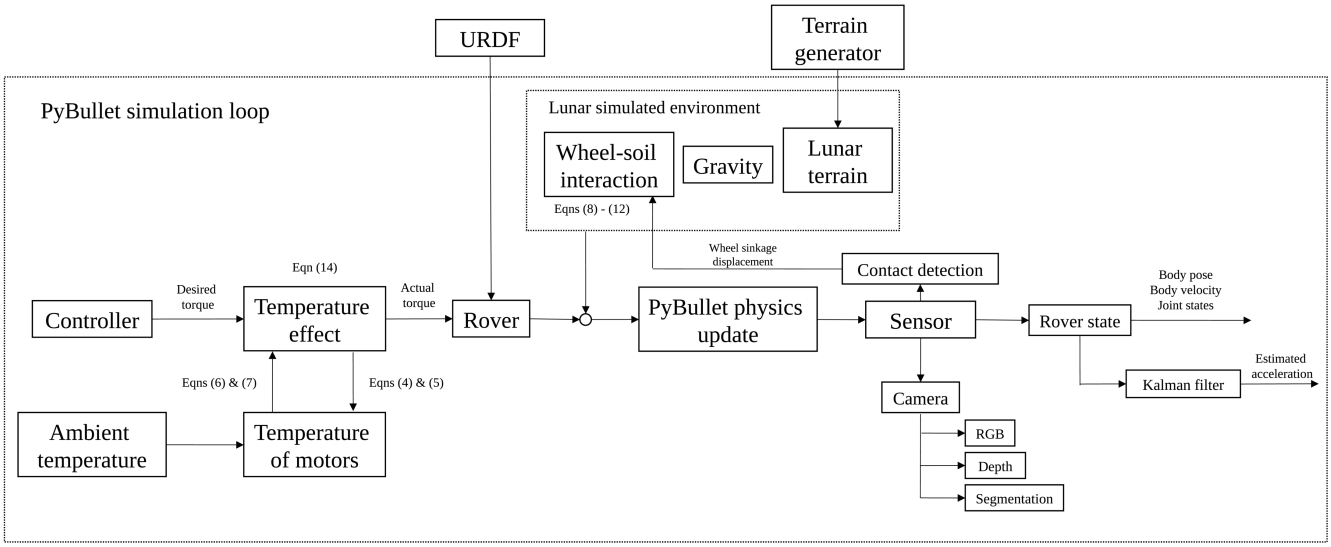

Fig. 2: Overall workflow of lunar rover simulator

2) Temperature Effect: Due to the absence of an atmosphere, the lunar surface experiences a wide range of temperature, oscillating between values of -173°C to 127°C. These ambient temperature variations significantly impact the physical characteristics of the motor, particularly the winding resistance and torque constant. As a result, these parameters have a direct impact on the generated motor torque, which in turn affects the overall locomotion of the rover.

To incorporate the temperature effect, we assume that the electric heat loss dominates over the frictional loss. Then, we divide the motor into the rotor and stator. The thermal equilibrium equation of the rotor is

$$P_v \mathrm{d}t - \frac{T_R - T_S}{R_{th1}} \mathrm{d}t - C_R \mathrm{d}T_R = 0, \tag{1}$$

where $P_v = I^2 R$ is the internal loss with I denoting the applied current to the motor and R the internal electrical resistance of rotor (winding resistance), T_R and T_S are the temperature of rotor and stator respectively, R_{th1} is the thermal resistance of the rotor, and C_R is the thermal capacity of rotor. Rearranging (1), we obtain the dynamics of rotor temperature as

$$\dot{T}_R = \frac{1}{C_R} \left(P_v - \frac{T_R - T_S}{R_{th1}} \right). \tag{2}$$

Similarly, for the stator temperature, we have

$$\dot{T}_S = \frac{1}{C_S} \left(\frac{T_A - T_S}{R_{th2}} \right), \tag{3}$$

where T_A is the ambient temperature, which is a parameter to be defined in the simulator, R_{th2} is the thermal resistance of the stator, and C_S is the thermal capacity of stator.

In the simulation loop, the rotor temperature and stator temperature are updated through Euler integration with their respective dynamics in (2) and (3) as

$$T_R(k) = T_R(k-1) + \dot{T}_R(k) \mathrm{d}t, \tag{4}$$

$$T_S(k) = T_S(k-1) + \dot{T}_S(k) \mathrm{d}t, \tag{5}$$

where $k-1$ and k denote the state at the previous and current time step respectively, and $\mathrm{d}t$ is the simulation time step. Following the temperature updates in (4) and (5) for each motor, we calculate the changes in winding resistance R and torque constant for each motor K as

$$R = R_0 \left(1 + \alpha \left(T_R - T_{R0}\right)\right), \tag{6}$$

$$K = K_0 \left(1 + \beta \left(T_R - T_{R0}\right)\right), \tag{7}$$

where $T_{R0} = 25$°C, R_0 is the winding resistance at T_{R0}, α is temperature coefficient of rotor materials, K_0 is the torque constant at T_{R0}, and β is the temperature coefficient of permanent magnet used.

The updated torque constant K in (7) is used to establish the new relationship between applied current and produced torque from the motor, which will be explained in Section III-D3.

We demonstrate the effect of motor temperature on the actual applied torque by constantly applying a 15-Nm desired torque to all four driving motors, enabling the rover to traverse in a straight path on the terrain. Figure 3 shows the increase in motor temperature and the difference between the desired torque and the actual applied torque to the motor, where the actual torque decreases over time. Moreover, after the temperature reaches 150°C, which is set as the maximum operating temperature threshold, the actual applied torque instantly drops to zero.

3) Terrain: We aim to incorporate actual lunar terrain surfaces into our rover simulator. While PyBullet supports 3D

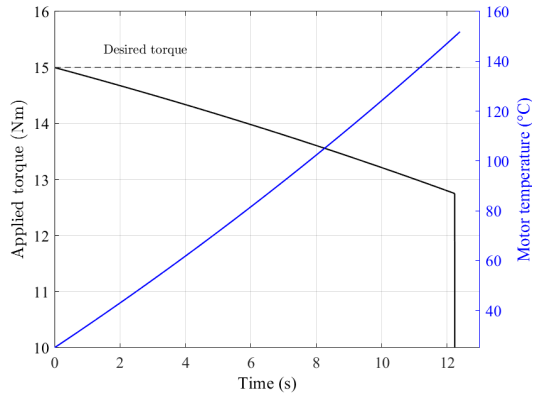

Fig. 3: Time evolution of the motor temperature (blue line), the actual torque applied to the motor (black line), and the desired torque (dash line)

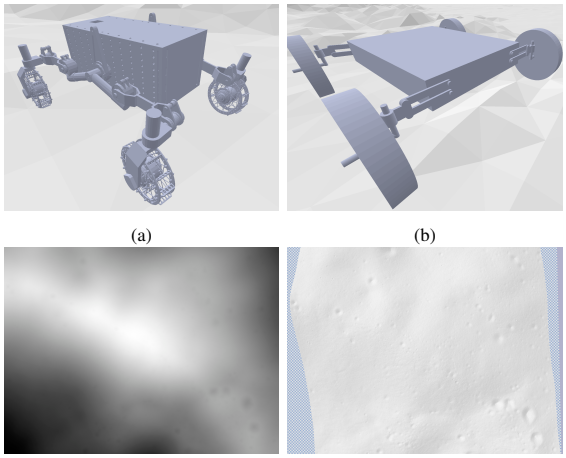

Fig. 4: (Top) Rovers in the simulator with (a) LUVMI-X model and (b) VIPER model. (Bottom) (c) 2D elevation map of de Gerlache rim and (d) generated 3D mesh with 1/100 scale

meshes as multibodies, obtaining 3D meshes of actual lunar terrain poses challenges due to their limited availability and the significant time required for manual creation using commercial 3D modeling software, which could lead to accuracy issues. To address these challenges, we instead acquire the free resources of 2D elevation maps of lunar surface provided by NASA, which are accessible with high resolution. Employing the technique and open-source software in [17], [18], [19], we generate 3D meshes directly from these 2D elevation maps.

As illustrated in Figure 4, we present an instance of 3D mesh in the simulator representing de Gerlcahe rim (Site 11), located at the south pole of the moon. In fact, a 2D elevation map (Figure 4c) was obtained from [20], and its corresponding 3D mesh (Figure 4d) was generated with [19].

4) Wheel-Soil Interaction: As the rover traverses the terrain, comprising hard rock and soft soil, it encounters complex terramechanics interactions at its wheels, which significantly impact its locomotion. Consequently, the integration of a terramechanics model into the simulator becomes essential. Following the approach presented in [21], we categorize the terrain into the two distinct types; hard rock and soft soil.

In the case of wheel-rock interaction, the dominant terramechanics model involves friction at each wheel, which is governed by

$$f = \mu F_N,$$

where μ is the friction coefficient, and F_N is the normal force. PyBullet API enables direct specification of the friction coefficient for the terrain.

In the case of the wheel-soft soil interaction, we simplify the calculation of normal force at each wheel by employing the compliance system [22] expressed as

$$F_N = k_N z_w + c_N \dot{z}_w,$$

where k_N is the contact stiffness of terrain, c_N is the contact damping of terrain, and z_w and $\dot{z}_w$ are the sinkage displacement and sinkage velocity of wheel respectively. It is noteworthy that the contact stiffness and contact damping can be adjusted through terrain parameters in PyBullet API, and the corresponding wheel sinkage displacement z_w will be calculated. Then, we calculate the slip ratio s and the slip angle β of each wheel by

$$s = \begin{cases} (r\omega - v_x)/(r\omega), & r\omega > v_x \\ (r\omega - v_x)/v_x, & r\omega < v_x \end{cases}, \tag{8}$$

$$\beta = \text{sign}(v_y) \arccos\left(v_x / \sqrt{v_x^2 + v_y^2}\right), \tag{9}$$

where r is the wheel radius, v_x and v_y are the traversing velocity and lateral velocity of the wheel respectively, and ω is the angular velocity of the wheel. Subsequently, under the assumption of small exit angle, the entrance angle θ_1 and the angle of maximum stress θ_m can be defined as

$$\theta_1 = \arccos\left(\frac{r - z_w}{r}\right), \qquad \theta_m = 0.5\theta_1.$$

Additionally, the maximum normal stress σ_m and the maximum shearing stress τ_m are defined as

$$\sigma_m = (k_c/b + k_\phi)\left[r(\cos\theta_m - \cos\theta_1)\right]^n,$$
$$\tau_m = (c + \sigma_m \tan\phi)\left(1 - e^{-j/K_s}\right),$$

where k_c and k_ϕ are the pressure sinkage moduli for cohesion stress and internal friction angle respectively, c is the soil cohesion, ϕ is the internal friction angle, K_s is the shearing deformation modulus, n is exponent sinkage, and j is shearing displacement defined as $j = r_s(\theta_1 - \theta_m) - (1 - s)(\sin\theta_1 - \sin\theta_m)$, where r_s is the shearing radius of wheel assumed as $r_s \approx r$ in this work, and s is defined in (8)

Finally, as proposed in [21], the analytical models of drawbar pull force F_{DP}, lateral force F_S, resistance moment M_R,

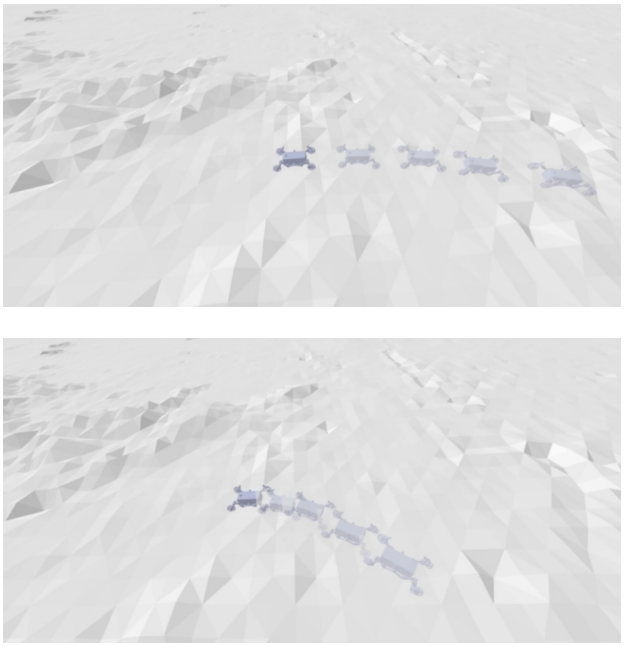

Fig. 5: (Top) The trajectory of rover on the hard soil (Bottom) The trajectory of rover on the soft soil

overturn moment M_O, and steering resistance moment M_S can be expressed as

$$F_{DP} = rb\tau_m A - r_s b\sigma_m B, \tag{10}$$

$$F_S = r_s b\tau_m \theta_m (1 - e^{-r(1-s)(\theta_m)\tan\beta/K_y}), \tag{11}$$

$$M_R = r_s^2 b\tau_m \theta_m, \tag{12}$$

$$M_O = F_S r, \tag{13}$$

$$M_S = F_S \sin\theta_m r + F_{DP} b\theta_s/8\pi, \tag{14}$$

where

$$A = \frac{2\cos\theta_m - \cos 2\theta_m - 1}{\theta_m},\ B = \frac{2\sin\theta_m - \sin 2\theta_m}{\theta_m},$$

b is the width of wheel, β is defined in (9), and K_y is the lateral shear deformation modulus.

During simulation steps, the derived interaction forces and torques given in (10) - (14) are applied to the center of each wheel on the rover.

We demonstrate the effect of wheel-soil interaction by applying a constant 15-Nm torque to all driving motors and having the rover traverse two soil types: hard soil and soft soil. Figure 5 shows the trajectory of the rover in both cases. While the rover can traverse in a straight line on hard soil, its locomotion changes in the case of soft soil due to the interaction forces and torques applied to the wheels of the rover.

D. Control Input

The rover can be controlled in the simulator by passing the desired control inputs to the joints. PyBullet supports control values in the form of position, velocity, and torque. Considering the effects of lunar environment on the motor, which may modify the current-torque relationship, we introduce the electric current as an additional virtual control input. The control input takes the form

$$U_i = (u_{sus,i}, u_{steer,i}, u_{drive,i}),$$

where sus represents the suspension joint, $steer$ represents the steering joint, $drive$ represents the driving joint, i represents the side of the rover (left front, right front, left rear, right rear), and u represents types of control input.

1) Position and Velocity Input: In the simulator, the position and velocity control mode aims to minimize the constraint error between the desired and actual values of the position and velocity of rover. This control mode employs proportional-derivative (PD) control for position and P control for velocity.

2) Torque Input: The torque input mode directly applies the desired torque to the joint motors. This approach allows for more direct manipulation of the locomotion of rover, and the torque dynamics will be tailored based on the specific joint dynamics parameters obtained from URDF.

3) Electric Current Input: For a realistic rover locomotion behavior in the lunar environment, it is essential to incorporate the electric current as virtual control input. Through motor system identification, the relationship between applied electric current and resulting torque can be determined as

$$\tau = KI, \tag{15}$$

where I is the applied current, K is the torque constant of motor given in (7), and τ is the corresponding torque. Given the values of applied electric current and torque constant of the motor, the torque applied to the joints of rover is derived from (15). Moreover, if the temperature effect discussed in Section III-C2 is considered, the resulting applied torque can be calculated as

$$\tau^{'} = \frac{K^{'}}{K}\tau, \tag{16}$$

where $K^{'}$ is the new torque constant calculated from (7) and $\tau^{'}$ is the new torque input.

E. State

After control inputs are applied, the simulator physics is updated based on Bullet engine. PyBullet API returns the states of the rover, which includes the position vector $\mathbf{p} = (p_x, p_y, p_z)$, the quaternion orientation vector $\mathbf{q} = (q_w, q_x, q_y, q_z)$ that can be transformed into the Euler angles of orientation $\eta = (\Phi, \Theta, \Psi)$, the linear velocity vector $\mathbf{v} = (v_x, v_y, v_z)$, and the angular velocity vector $\omega = (\omega_x, \omega_y, \omega_z)$ of the rover. All of these are expressed in the fixed inertial frame. Furthermore, to allow the low-level control of each motor joint, it is necessary to include the angular position and velocity of each actuated joint i.e. p_i and ω_i for $i = 1, 2, ..., 12$ in the state vector.

In addition to the state of rover, it is possible to obtain the rendered images representing diverse simulation scenarios from arbitrary camera viewpoints. The camera configuration is governed by both the view matrix and the projection matrix.

In the rover simulator, these matrices are computed to emulate image capture from a fixed front-mounted camera on the rover. Furthermore, three distinct image matrices are derived in this context: RGB image matrix $C_{RGB} \in \mathbb{R}^{320\times240\times4}$; depth image matrix $C_{depth} \in \mathbb{R}^{320\times240}$; and segmentation image matrix $C_{segment} \in \mathbb{R}^{320\times240}$. It is worth noting that, since the computation of these matrices are expensive, the camera sensors are designed to operate at a lower rate compared to the simulation frequency.

IV. Reinforcement Learning Interface

One advantage of PyBullet over other simulation platforms is its ability to perform all actions within a single dynamics step and to subsequently provide the corresponding states [23], making it compatible with RL integration. In our rover simulator, we establish an interface to align the simulator with OpenAI Gym environment class and combine it with workflows in open-source RL libraries that are compatible with Gym class.

OpenAI Gym is an open-source Python library designed to support and standardize the development of RL methodologies. Following the classic agent-environment paradigm, OpenAI Gym provides an API that bridges learning algorithms and the environment spaces.

In the context of our rover simulator, the control inputs (action), defined in Section III-D, are passed to the joints of the rover, defined in Section III-B. Subsequently, PyBullet resolves the forward and collision dynamics of the rover and other multibodies in the simulation environment, defined in Section III-C, and returns the resulting states of the rover (observation), defined in Section III-E, along with the corresponding reward value. This agent-environment loop is wrapped as a Gym class.

We rely on Stable Baselines3 (SB3) library [24], an open-source repository of RL algorithms based on PyTorch, for the training workflow due to its compatibility with Gym's wrapped environment. Notably, SB3 contains the implementable RL algorithms, including PPO, DDPG, SAC, and TD3, and also a comprehensive RL training workflow.

V. EXAMPLE DEMONSTRATION

This section presents our approach to training the rover for autonomous navigation in the lunar environment using RL algorithms. The objective for the rover is to traverse across the lunar terrain to reach a predefined destination.

For the observation space, assuming that the goal position $\mathbf{p}_g = (p_{g,x}, p_{g,y})$ is observable to the rover, we augment the observation space with the goal position. Additionally, we want to reduce the vibration effect during traversing, which implies a reduction in the vertical linear acceleration of the rover. However, PyBullet, by default, does not provide linear acceleration as direct feedback. To overcome this, we use Kalman filters to estimate the linear acceleration. Following the standard Kalman filter algorithm with the state vector $\mathbf{X} = (\mathbf{p}, \mathbf{v}, \hat{\mathbf{a}}) \in \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3$, where $\hat{\mathbf{a}}$ is the estimated linear acceleration, and measurement vector $\mathbf{Y} = H\mathbf{X}$ with measurement matrix $H = [\mathbf{I}_{6\times6}\ \mathbf{0}_{6\times3}]$, the estimated linear acceleration of rover can be calculated. Thus, the observation space can be denoted as

$$A = (\mathbf{p}, \eta, \mathbf{v}, \omega, \hat{\mathbf{a}}, \mathbf{p}_g) \in \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^2.$$

For the action space, to simplify the control method, we model the motion of rover as that of a differential drive robot, treating 4 driving motors as a single actuator for linear motion and 2 front steering motors as actuators for angular motion. The control policies for 4 suspension motors are also learned independently to enable adaptive response to rough terrain. As a result, the action space can be denoted as

$$U = (\tau_{drive}, \tau_{steer}, \tau_{sus,1}, \tau_{sus,2}, \tau_{sus,3}, \tau_{sus,4}) \in \mathbb{R}^6.$$

To accomplish the navigation task, we define a reward function as

$$r = 20r_{prog} + 0.1r_{head} + 0.05r_{tilt} + 0.1r_{excite} + G + V,$$

where the components of reward function are described as follows:

- $r_{prog} = \|\mathbf{P}_{k,g} - \mathbf{P}_{k-1,g}\|$ represents the progress made by the rover towards the goal position, where $\mathbf{P}_{k,g}$ and $\mathbf{P}_{k-1,g}$ indicate the vectors from rover to goal position at the current and previous time steps respectively.
- $r_{head} = \frac{\mathbf{P}_{k,g}}{\|\mathbf{P}_{k,g}\|} \cdot (\cos(\Psi), \sin(\Psi), 0)$, where Ψ denotes yaw angle of rover and $a \cdot b$ denotes the dot product between vectors a and b, encourages the rover to align its heading with the goal position.
- $r_{tilt} = -|\Phi| - |\Theta|$ penalizes excessive roll Φ and pitch Θ of the rover to promote stable navigation.
- $r_{excite} = -|\hat{a}_z|$ encourages the rover to maintain a smooth ride by penalizing excessive vertical acceleration $\hat{a}_z$.
- $G = 500$ is the reward for the rover successfully reaching the goal position.
- $V = -60$ is the penalty imposed when undesirable terminal conditions are met, such as the rover getting stuck in the pothole or flipping over.

For training, we employ deep deterministic policy gradients (DDPG) and twin delayed DDPG (TD3) algorithms from SB3, with action noise drawn from Gaussian distribution $N(0, 0.3)$ added to actions. We utilize the default settings of multi-layer perceptron model in both algorithms for the training process without further tuning any hyperparameters. The training results of both algorithms are depicted in Figure 6, showing the average return per episode. The convergence of return after 400 episodes validate the success of achieving the navigation task using the integration of lunar rover simulator and the RL framework. The RL simulation result can be seen in the video available at `https://www.youtube.com/watch?v=8HMqAkDbpqI`.

VI. CONCLUSION

In this paper, we propose an open-source lunar rover simulator integrated with reinforcement learning framework. The

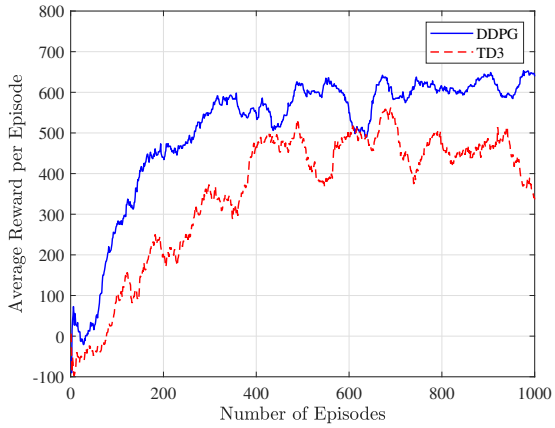

Fig. 6: Average return per episode of DDPG (solid line) and TD3 (dashed line) algorithms

simulator incorporates into the lunar environment various factors that affect the rover locomotion. PyBullet is employed to integrate the dynamical equations of the rover and provide the corresponding states of rover. Importantly, we make the simulator publicly accessible, offering flexibility in various rover simulation applications. Furthermore, we show the seamless integration of our rover simulator with reinforcement learning framework, which is demonstrated through autonomous rover navigation task on lunar terrain. Beyond lunar missions, we anticipate that this open-source simulator software will serve as a tool for upcoming planetary rover missions.

ACKNOWLEDGMENT

The authors thank Kraiphum Kerdthip and Whimin Kim for the assistance in rover design.

REFERENCES

[1] A. Jain, J. Guineau, C. Lim, W. Lincoln, M. Pomerantz, G. Sohl, and R. Steele, "ROAMS: Planetary surface rover simulation environment," Nara, May 2003, nTRS Author Affiliations: NTRS Document ID: 20060028642 NTRS Research Center: Jet Propulsion Laboratory (JPL).

[2] B. P. Trease, R. A. Lindeman, R. E. Arvidson, K. Bennett, L. P. VanDyke, F. Zhou, K. Iagnemma, and C. Senatore, "Adams-based rover terramechanics and mobility simulator - ARTEMIS," Tech. Rep. NPO-47781, Apr. 2013, nTRS Author Affiliations: California Inst. of Tech., Washington Univ., Massachusetts Inst. of Tech. NTRS Document ID: 20130012671 NTRS Research Center: Jet Propulsion Laboratory (JPL).

[3] M. Allan, U. Wong, P. M. Furlong, A. Rogg, S. McMichael, T. Welsh, I. Chen, S. Peters, B. Gerkey, M. Quigley, M. Shirley, M. Deans, H. Cannon, and T. Fong, "Planetary rover simulation for lunar exploration missions," in *2019 IEEE Aerospace Conference (AERO)*, Mar. 2019, pp. 1–19.

[4] F. Buse, A. Pignède, J. Bertrand, S. Goulet, and S. Lagabarre, "MMX rover simulation - robotic simulations for phobos operations," in *2022 IEEE Aerospace Conference (AERO)*, Mar. 2022, pp. 1–14.

[5] P. Poulakis, L. Joudrier, K. Kapellos, and R. Section, "3DROV: A planetary rover system design, simulation and verification tool," Feb. 2008.

[6] R. Zhou, W. Feng, L. Ding, H. Yang, H. Gao, G. Liu, and Z. Deng, "MarsSim: A high-fidelity physical and visual simulation for Mars rovers," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 59, no. 2, pp. 1879–1892, Apr. 2023.

[7] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "OpenAI Gym," Jun. 2016, arXiv:1606.01540 [cs].

[8] K. Zhang, F. Niroui, M. Ficocelli, and G. Nejat, "Robot navigation of environments with unknown rough terrain using deep reinforcement learning," in *2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Aug. 2018, pp. 1–7.

[9] B.-J. Park and H.-J. Chung, "Deep reinforcement learning-based failure-safe motion planning for a 4-wheeled 2-steering lunar rover," *Aerospace*, vol. 10, p. 219, Feb. 2023.

[10] W. Feng, L. Ding, R. Zhou, C. Xu, H. Yang, H. Gao, G. Liu, and Z. Deng, "Learning-based end-to-end navigation for planetary rovers considering non-geometric hazards," *IEEE Robotics and Automation Letters*, vol. 8, no. 7, pp. 4084–4091, Jul. 2023.

[11] A. Orsula, S. Bøgh, M. Olivares-Mendez, and C. Martinez, "Learning to grasp on the moon from 3D octree observations with deep reinforcement learning," in *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Oct. 2022, pp. 4112–4119.

[12] T. Blum, G. Paillet, M. Laine, and K. Yoshida, "RL STaR platform: Reinforcement learning for simulation based training of robots," Sep. 2020, arXiv:2009.09595 [cs].

[13] E. Coumans and Y. Bai, "PyBullet, a python module for physics simulation for games, robotics and machine learning," 2016. [Online]. Available: http://pybullet.org

[14] S.-J. Chung and N. Pollard, "Predictable behavior during contact simulation: a comparison of selected physics engines," *Computer Animation and Virtual Worlds*, vol. 27, no. 3-4, pp. 262–270, May 2016.

[15] J. Gancet, D. Urbina, J. Biswas, M. Losekamm, S. Sheridan, A. Evagora, L. Richter, S. Schroeder, T. Chupin, D. Fodorcan, H.-K. Madakashira, P. Reiss, S. Barber, N. Murray, G. Fau, K. Kullack, R. Aked, M. Reganaz, S. Kubitza, J. Neumann, and P. Wessels, "LUVMI and LUVMI-X: Lunar volatiles mobile instrumentation concept and extension," in *ASTRA 2019 – 15th Symposium on Advanced Space Technologies in Robotics and Automation*, Jan. 2019.

[16] R. Chen, "VIPER Mission Overview," Feb. 2020. [Online]. Available: http://www.nasa.gov/viper/overview

[17] M. G. Müller, M. Durner, A. Gawel, W. Stürzl, R. Triebel, and R. Siegwart, "A photorealistic terrain simulation pipeline for unstructured outdoor environments," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2021, pp. 9765–9772.

[18] M. Fogleman, "hmm," Jul. 2023, original-date: 2019-08-22T02:42:36Z. [Online]. Available: https://github.com/fogleman/hmm

[19] H. Technologies, "TIN Terrain," Eindhoven, the Netherlands, Jul. 2023, original-date: 2018-09-11T11:15:17Z. [Online]. Available: https://github.com/heremaps/tin-terrain

[20] M. K. Barker, E. Mazarico, G. A. Neumann, D. E. Smith, M. T. Zuber, and J. W. Head, "Improved LOLA elevation maps for south pole landing sites: Error estimates and their impact on illumination conditions," *Planetary and Space Science*, vol. 203, p. 105119, Sep. 2021.

[21] H. Yang, L. Ding, H. Gao, Z. Wang, Q. Lan, G. Liu, Z. Liu, W. Li, and Z. Deng, "High-fidelity dynamic modeling and simulation of planetary rovers using single-input-multi-output joints with terrain property mapping," *IEEE Transactions on Robotics*, vol. 38, no. 5, pp. 3238–3258, Oct. 2022.

[22] G. Sohl and A. Jain, "Wheel-Terrain Contact Modeling in the ROAMS Planetary Rover Simulation." American Society of Mechanical Engineers Digital Collection, Jun. 2008, pp. 89–97.

[23] J. Panerati, H. Zheng, S. Zhou, J. Xu, A. Prorok, and A. P. Schoellig, "Learning to Fly—a gym environment with PyBullet physics for reinforcement learning of multi-agent quadcopter control," in *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Sep. 2021, pp. 7512–7519.

[24] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-Baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021.