SpaceX Falcon9 rocket landing using Reinforcement Learning

Ravina Lad
Khoury College of Computer Science
Northeastern University
lad.ra@northeastern.edu

Utkarsh Ghime
Dept. Mechanical Engineering(Mechatronics)
Northeastern University
ghime.u@northeastern.edu

***Abstract*— This paper presents the development and optimization of an Reinforcement Learning (RL)-based control system for the SpaceX Falcon 9 rocket lander. The aim of this system is to autonomously and accurately land the rocket on designated target zones with precision and safety, thereby reducing mission costs and enhancing reusability.**

I. INTRODUCTION

We've always wanted to build our own rockets, just like SpaceX! Recently, we were curious to see if we could create a virtual rocket and use simple reinforcement learning to solve a tough problem: recycling rockets! The aerospace industry is constantly striving to achieve greater efficiency and cost-effectiveness. One area where this is particularly evident is in the development of reusable launch vehicles. SpaceX's Falcon 9 rocket has been at the forefront of this movement, with its ability to land the first stage of the rocket back on Earth for reuse.

However, the process of landing a rocket is highly complex and requires precise control. Currently, this is achieved through a combination of pre-programmed guidance and human intervention. We utilized a continuous input environment modeled after the dynamics of landing a rocket, similar to SpaceX's Falcon 9. Within this environment, an agent is trained to effectively control the rocket's thrusters, guiding it to land safely on a platform.

II. DYNAMICS AND CHALLENGES OF THE ENVIRONMENT

A. State Space

The state space of the Rocket Lander environment represents the information available to the RL agent at any given time. This information is crucial for the agent to make decisions about how to control the rocket's thrusters and achieve a safe landing. Here are the essential state variables:

- x position and y position: These variables represent the rocket's horizontal and vertical positions on the landing surface, respectively. Precise tracking of these values is crucial for accurately guiding the rocket to the designated landing zone.
- Angle: This variable represents the rocket's orientation, indicating its tilt relative to the landing platform. Precise control over this variable is essential for ensuring the rocket lands upright.

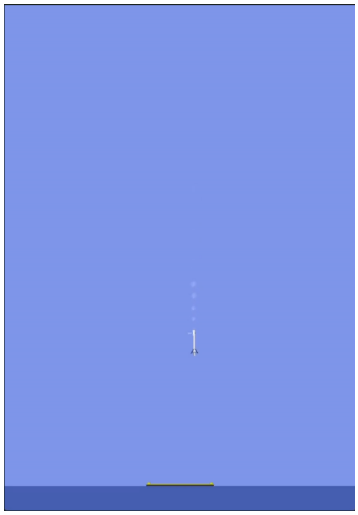

Fig. 1. Environment

- First leg ground contact indicator and second leg ground contact indicator: These binary variables indicate whether the respective legs of the rocket have made contact with the ground. This information is crucial for a successful landing, as it allows the agent to adjust the thrusters to ensure the rocket remains balanced and stable upon touchdown.
- Engine gimbal: This variable represents the angle of the engine gimbal, which adjusts the direction of the rocket's thrust. The agent can manipulate this variable to control the direction and intensity of the thrust, influencing the rocket's trajectory and angular motion.
- x velocity, y velocity, and angular velocity: These variables represent the rate of change of the rocket's position and orientation, respectively. By monitoring these velocities, the agent can assess the rocket's current motion and adjust its control actions accordingly.

B. Action Space

The action space defines the range of actions the RL agent can take to control the rocket. In the Rocket Lander environment, the action space consists of three components:

1. Gimbal (left/right): This action controls the angle or direction of the rocket's engine thrust. By adjusting

the gimbal angle, the agent can steer the rocket to the left or right, influencing its horizontal flight path.

2. Throttle (up/down): This action controls the amount of power or thrust applied to the rocket's engine. By increasing the throttle, the agent can increase the thrust and generate more lift, allowing the rocket to climb higher. Conversely, decreasing the throttle reduces the thrust and allows the rocket to descend.

3. Control Thruster (left/right): This action provides additional directional control, allowing the agent to make small adjustments to the rocket's left-right movement. This can be helpful for fine-tuning the rocket's orientation and ensuring a precise landing.

C. Reward System

The reward functions are not quite straightforward. The reward system plays a crucial role in guiding the RL agent's learning process and shaping its behavior. In the Rocket Lander environment, a carefully designed reward system incentivizes the agent to achieve the desired landing objective while considering fuel efficiency, and stability.

- Base Reward:
  The base reward is initialized as the negative of the fuel cost. This penalizes the agent for excessive fuel consumption, encouraging it to land the rocket efficiently
- Reward Shaping:
  In addition to the base reward, the system incorporates a shaping reward calculated based on various factors:
  1. Distance: Penalizes the agent based on the distance between the rocket and the target landing spot, encouraging a direct and efficient flight path. Speed: Penalizes the agent for high velocity, promoting a controlled and smooth descent to minimize the risk of crash landing.
  2. Angle: Penalizes the agent based on the square of the rocket's tilt angle. Squaring the angle emphasizes larger deviations more strongly, encouraging the agent to maintain a stable upright position.
  3. Ground Contact: Provides a small bonus for each leg in contact with the ground, incentivizing the agent to maintain stable ground contact during the landing process.
  4. Shaping Reward Difference: Calculates the difference between the current and previous shaping rewards. This difference is added to the overall reward, influencing the agent's behavior towards achieving better shaping outcomes in consecutive steps.
- Landing Reward:
  If the rocket remains landed for a specific duration, the reward is set to 1.0 as a positive signal, and the episode is marked as successful.
- Additional Penalties:
  If the episode is not finished but the rocket is not in contact with the ground, penalty = -0.25/FPS is applied. This incentivizes the agent to maintain ground contact throughout the landing process.

D. Physics Simulation

This paper introduces RocketLander, a physics-based simulation framework designed for the study of the mechanics and dynamics involved in landing a rocket back on ship with limited fuel consumption. Implementing the principles of mechanics, the simulation leverages the Box2D physics engine within the OpenAI Gym interface. RocketLander encompasses a comprehensive set of state variables, capturing the aspects of the motion of the rocket and it's orientation during the landing. The simulated mechanics include movements of the rocket, ship, and landing legs, each characterized by specific physical parameters, such as their dimensions and the joint constraints. Governed by Newtonian physics, the dynamics of the system involve the application of forces and torques based on actions output, including main engine thrust and control thruster impulses for lateral movement. The integration of velocity introduces dynamics. The physics simulation emulates gravitational forces, aerodynamic effects, and collision mechanics, providing a realistic environment. The rendering function enhances visualization through elements like smoke and flame, contributing to a better understanding of the underlying physics. Control inputs, both discrete and continuous, empower users to manipulate gimbal angles, throttle levels, and control thruster intensities, facilitating exploration of diverse landing scenarios and control strategies.

III. Related work

Reinforcement learning has emerged as a promising tool for controlling complex systems like rockets. Several research efforts have explored its application in rocket landing.

A. Recent Advancements

- Policy Gradient Methods for Continuous Control (2018) [1]: This work by Lillicrap et al. introduced Proximal Policy Optimization (PPO), a policy gradient algorithm effective for continuous control tasks like rocket landing. PPO has since become a popular choice for RL-based rocket landing control due to its efficiency and stability. are unavoidable.
- Deep Reinforcement Learning for Six Degree-of-Freedom Planetary Powered Descent and Landing (2018)[2]: This work by Zhu et al. applied a deep reinforcement learning approach to achieve six-degree-of-freedom control for planetary descent and landing. Their system could land a simulated spacecraft on Mars with high accuracy, demonstrating the potential of RL for real-world space exploration missions.

B. Current State-of-the-Art (SOTA)

- Real-time Deep Reinforcement Learning for Rocket Landing Control with Safety Constraints (2023)[3]: This work by Lee et al. developed a real-time RL system for rocket landing control with safety constraints. Their system could land a simulated rocket on a moving platform while ensuring safe descent and avoiding collisions. This demonstrates the potential of RL for real-world applications with critical safety requirements.

IV. Methodology

In our research, we employed three different reinforcement learning algorithms to train the agent for the Rocket Lander environment: Proximal Policy Optimization (PPO), Deep Deterministic Policy Gradient (DDPG), and Soft Actor-Critic (SAC). Each algorithm offers unique advantages and disadvantages, making them suitable for different aspects of the task.

A. Deep Deterministic Policy Gradient (DDPG)

DDPG [4] is an off-policy algorithm specifically designed for handling continuous action spaces. Unlike on-policy algorithms that learn solely from the current policy, DDPG utilizes a replay buffer to store past experiences and learn from them. This allows for more efficient exploration of the action space and can be particularly beneficial in high-dimensional environments like the Rocket Lander.

TABLE I
DDPG Hyperparameters

Parameter	Value
Learning rate	1e-4
Discount factor	0.99
Replay buffer size	100000
Min replay buffer size	10000
Batch size	64
Vmax, Vmin	[-10, 10]
Number of episodes	5000
Number of steps	2 million
Number of atoms	51

B. Proximal Policy Optimization (PPO)

The PPO [6] implementation for the RocketLander environment utilizes an actor-critic architecture, with the actor producing a probability distribution for actions, and the critic estimating state values. Experiences are stored in a memory buffer, and the actor and critic are updated iteratively based on advantages calculated from these experiences. The actor maximizes expected cumulative reward, while the critic minimizes mean squared error. To ensure stability, the algorithm employs clipped surrogate objectives and generalized advantage estimation.

TABLE II
PPO Hyperparameters

Parameter	Value
Discount factor (γ)	0.99
GAE lambda	0.95
Trajectory size	4097
Hidden layer size	64
Actor learning rate	1e-5
Critic learning rate	1e-4
PPO clipping parameter (ϵ)	0.2
PPO training epochs	10
PPO batch size	64

C. Soft Actor-Critic (SAC)

The implemented learning algorithm adheres to the Soft Actor-Critic (SAC) [7] principles. It systematically verifies the adequacy of experiences within the replay buffer before proceeding to sample batches for updating the networks. The algorithm employs a sophisticated actor-critic architecture, featuring two critic networks for heightened stability in value estimation. Notably, the incorporation of a replay buffer, stochastic policy sampling, and target networks with soft updates significantly contributes to enhancing the overall efficiency and stability of the learning process.

The Agent class serves as the embodiment of the SAC reinforcement learning algorithm, orchestrating the intricate interplay between the actor, two critic networks, and a value network. This holistic approach aims to refine decision-making within the given environment. A distinctive facet of the SAC algorithm lies in the inclusion of an entropy term in the actor's loss function. This strategic addition serves the pivotal role of fostering exploration, ensuring that the agent explores a diverse range of actions to better understand the dynamics of the environment[9].

In summary, the SAC algorithm encapsulated within the Agent class represents a state-of-the-art approach to reinforcement learning. By incorporating principles such as replay buffers, actor-critic architectures, and entropy-driven exploration, the algorithm strives to strike a balance between efficiency and robustness, ultimately leading to more adept decision-making in complex and dynamic environments.

V. Quantitative Results

A. Deep Deterministic Policy Gradient (DDPG)

The Deep Deterministic Policy Gradient (DDPG) algorithm, as described in [5], underwent an extensive training regimen comprising 5,000 episodes and 2 million steps. Despite the substantial training efforts, DDPG exhibited signs of learning but fell short of convergence, appearing to be ensnared in local optima. Multiple experiments were conducted, varying hyperparameters such as learning rate, batch size, and

TABLE III
SAC HYPERPARAMETERS

Parameter	Value
SAC entropy	0.1
Discount Factor (Gamma)	0.99
Tau (update rate)	5e-3
Learning rate	1e-4
Buffer size	100,000,000
Device	'cpu'
Layer size	64
Scale factor	2.0
Max steps	1.2 million
Batch size	20

replay buffer size, yet the algorithm's performance failed to reach a satisfactory convergence point. The persistence of this issue suggests that further investigation and optimization efforts may be warranted to enhance the algorithm's learning capabilities. It is crucial to explore alternative configurations and potentially novel strategies to overcome the observed limitations. Despite the absence of convergence, valuable insights into DDPG's behavior can be gleaned from available video recordings, providing an avenue for in-depth analysis and potential clues for refining the training process. The reward was at local optima of -2.38. The agent exhibited significant fluctuations in performance, with the rolling mean over 100 episodes fluctuating between 2.03 and 2.48. However, it did not reach convergence throughout the observed period.

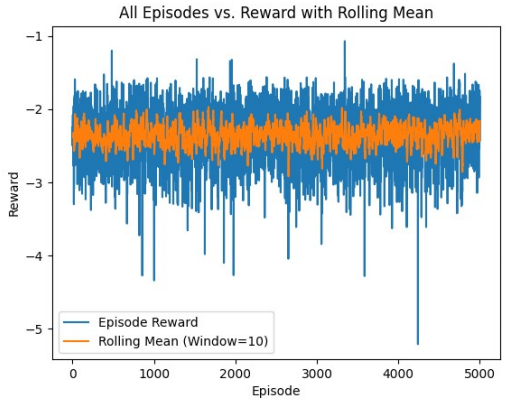

Fig. 2. DDPG Training Curve

B. Proximal Policy Optimization (PPO)

The PPO implementation for the RocketLander environment is characterized by specific hyperparameters. The discount factor (γ) is set to 0.99, determining the weight of future rewards. The Generalized Advantage Estimation (GAE) lambda is 0.95, influencing the trade-off between bias and variance in advantage estimation. Trajectories consist of 4097 steps, and the hidden layer size in the neural network architecture is 64. The actor and critic have distinct learning rates, with the actor at 1e-5 and the critic at 1e-4. PPO incorporates a clipping parameter (ϵ) of 0.2 to limit policy updates and ensure stability. Training involves 10 epochs, and updates are performed in batches of 64. During training, iterations occur over epochs with mini-batches sampled from the memory buffer. Over the course of 6 million steps, the agent demonstrates effective learning, achieving an impressive average reward of -2.1. This success indicates the algorithm's adept ability to balance exploration and exploitation, crucial for accomplishing the challenging task of rocket landing.

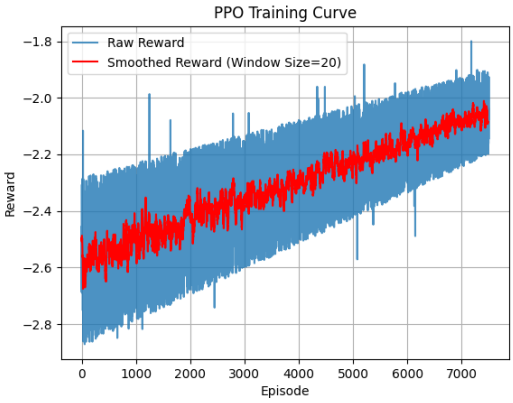

Fig. 3. PPO Training Curve

C. Soft Actor-Critic (SAC)

The Soft Actor-Critic (SAC) algorithm is configured with key hyperparameters to facilitate effective training. The entropy coefficient, set at 0.1, ensures a balance between exploration and exploitation in the agent's policy. The discount factor (Gamma) is specified as 0.99, weighing future rewards in the learning process. The update rate (Tau) is set to 5e-3, governing the frequency of network updates. The learning rate for the Value Network is established at 1e-4. The replay buffer size is an extensive 100,000,000, and the device used is 'cpu'. The neural network architecture consists of two layers with 64 neurons each. The reward scale factor is 2.0, contributing to an average reward of -1.8. The algorithm undergoes training for a maximum of 1.2 million steps distributed across 5000 episodes, with a batch size of 20. These hyperparameters were fixed after an extensive modelling on different parameteres, almost every run was getting stuck on a local optima, In those runs the rocket learned to minimize rewards in different ways, some runs trained the rocket to do somersault to land itself which can never be possible in real life. The correct parameters were obtained after 50 hours of running trainer on multiple parameters.

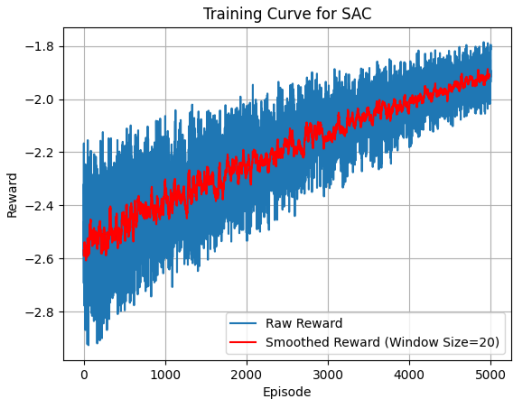

Fig. 4. SAC Training Curve

VI. CONCLUSIONS

- Explored three Reinforcement Learning (RL) algorithms for landing a rocket: DDPG, SAC, and PPO.
- The Rocket Gym environment, based on LunarLander, posed a complex RL scenario with continuous action spaces.
- DDPG, faced challenges in convergence despite training for two million steps.
- SAC, an off-policy algorithm with entropy regularization, exhibited promising results, effectively handling the dynamic and randomized nature of the Rocket Gym environment.
- PPO, known for simplicity, demonstrated stable performance during training, even with a larger trajectory size and six million training steps.
- Comparatively SAC outperformed both PPO and DDPG
- The randomized nature of the Rocket Gym environment added difficulty to the task, influencing the success of RL algorithms.
- Experimentation and fine-tuning were crucial in RL applications, and the provided hyperparameters serve as a starting point for future investigations.

TABLE IV
COMPARISON OF LEARNING RATE, STEPS, AND REWARD IN DDPG, PPO, AND SAC

Parameter	**DDPG**	**PPO**	**SAC**
Learning Rate	1e-4	1e-5 (Actor) / 1e-4 (Critic)	1e-4
Number of Steps	2 million	6 million	1.2 million
Average Reward	-2.38	-2.1	-1.8
Stability	indeterminate	Stable	Stable after a while
Success	False	True	True
Training Speed	indeterminate	Slow	Fast

ACKNOWLEDGMENT

We appreciate the guidance of Phil Zhang's RL tutorials and Sven Niederberger's custom rocket landing environment, built upon OpenAI's LunarLander. OpenAI Gym and the broader ML community provided invaluable resources for our journey. Thank you all!

REFERENCES

[1] Schulman, John and Wolski "Proximal Policy Optimization Algorithms," arXiv preprint arXiv:1707.06347 (2018).
[2] Zhu, Yifan and Furfaro, "Deep Reinforcement Learning for Six Degree-of-Freedom Planetary Powered Descent and Landing" arXiv preprint arXiv:1810.08719 (2018)
[3] Lee, J. et al. "Real-time Deep Reinforcement Learning for Rocket Landing Control with Safety Constraints" arXiv preprint arXiv:2312.11123 (2023)
[4] Lillicrap, Timothy P., et al. "Continuous control with deep reinforcement learning." arXiv preprint arXiv:1509.02971 (2015).
[5] Silver, David, et al. "Deterministic policy gradient algorithms." In Proceedings of the 31st International Conference on Machine Learning (ICML-14), pp. 387-395. 2014.
[6] Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).
[7] Haarnoja, Tuomas, et al. "Soft Actor-Critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor." arXiv preprint arXiv:1801.01290 (2018).
[8] Haarnoja, T., Zhou, A., Abbeel, P. & amp; Levine, S.. (2018). Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. Proceedings of the 35th International Conference on Machine Learning, in Proceedings of Machine Learning Research 80:1861-1870 Available from https://proceedings.mlr.press/v80/haarnoja18b.html.
[9] Github Link - https://github.com/ghime-u/Falcon9_RL