IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

ROCKET LANDING USING REINFORCEMENT LEARNING IN SIMULATION

Ashwinikumar Rathod
Information and Technology
G.H Raisoni College of Engineering
Nagpur, India

Ankit Sayane
Information and Technology
G.H Raisoni College of Engineering
Nagpur, India

Chetan Galhat
Information and Technology
G.H Raisoni College of Engineering
Nagpur, India

Hardik Bopche
Information and Technology
G.H Raisoni College of Engineering
Nagpur, India

Chaitanya Shidurkar
Information and Technology
G.H Raisoni College of Engineering
Nagpur, India

Dr. Vinay Keswani
Professor
G.H Raisoni College of Engineering
Nagpur, India

***Abstract*—** The goal of this project prototype is to develop a reinforcement learning system that can successfully land an orbital rocket by observing and understanding its environment. In this project, an AI agent learns how to independently land a rocket and then selects the most efficient actions to achieve this objective. Initially, the agent performs random actions since it has no prior knowledge. However, by doing so, it gains insights into its environment and the ultimate goal of landing the rocket. Once the agent receives a reward for landing the rocket, it modifies its parameters based on the reward to enhance its output and increase the likelihood of success. The Unity ML-Agents toolkit is used in this project.

This project focuses on the multidisciplinary topic of reusable space systems, especially first stage rockets. The majority of rockets have two to three stages. The initial stages, which are largely done by boosters, carry the cargo, or spacecraft, high up in the sky. After the last stage, the rocket's main engine separates, descends back towards Earth, and either burns up due to friction in the Earth's atmosphere or crashes into the ocean, rendering future missions ineffective. The loss of the primary rocket engine is not only inefficient, but also expensive and time-consuming to develop. Reusable components dramatically reduce launch costs, decreasing the barrier to entry into space.

I. INTRODUCTION

Since the 1950s, numerous rockets, space station components, and satellites have been discarded in the Earth's oceans, leading to increased pollution and posing risks to marine life. The disposal of these objects can result in marine contamination due to potential spills of the highly hazardous rocket fuel hydrazine. Additionally, the impact on local seals, whales, bears, and walruses in areas such as the ice points off the coast of Greenland remains uncertain. It's important to note that ocean organisms and ecosystems play a vital role in our weather and climate, as the ocean absorbs approximately one-third of the carbon dioxide generated due to its thermodynamic properties.

Reinforcement learning is an area of artificial intelligence that allows autonomous entities to learn through a trial-and-error approach. In the context of rocket landing, the agent (the rocket) learns by interacting with its environment and receiving feedback based on its actions. This application of reinforcement learning is particularly intriguing as it involves training an algorithm to autonomously achieve the challenging task of rocket landing.

To achieve this objective, deep reinforcement learning techniques are applied in the project. The program utilizes the Unity ML-Agents toolbox and employs a neural network-based algorithm. The system observes its surroundings and makes decisions to facilitate a successful rocket landing. Initially, the system selects actions randomly, but as training progresses, the agent is rewarded for successfully landing the rocket. The algorithm adjusts the settings of its neural network based on the received inputs and the actions that led to success, increasing the likelihood of implementing successful actions.

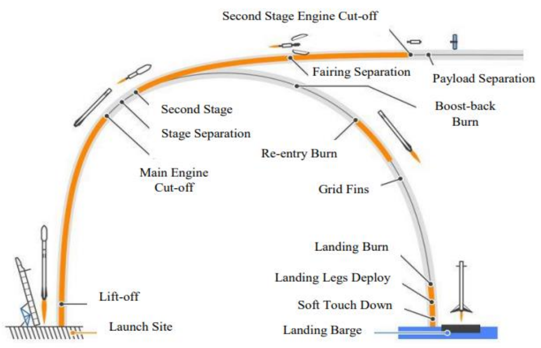

Figure.1 Brief Description of landing rocket

The privately owned company SpaceX is actively supporting the development of reusable orbital launch technologies, akin to the concept of aircraft reuse. Over the years, they have made significant progress towards their goal of achieving full and rapid reusability of space launch vehicles. Their ambitious long-term plans involve swiftly retrieving the first stage of the launch vehicle to the launch site and realigning the second stage with the launch site within 24 hours after atmospheric re-entry. SpaceX is also aiming to design two stages of their orbital launch vehicle, enabling lunar spacecraft to be reused within a few hours after completing their missions.

II. PROBLEM STATEMENT

The problem statement for this project is to develop an artificial intelligence agent that is able to land an orbital class rocket, specifically the Starship, in a simulated environment using deep reinforcement learning algorithms. The AI agent will be trained using the Unity ML-Agents toolkit and will be able to make decisions and take actions based on its observations of the simulated environment. The agent's goal is to successfully land the rocket without any errors or crashes. The agent will be rewarded for achieving successful landings, and it will utilize this feedback to optimize the parameters of its neural network, enhancing the likelihood of executing actions that result in successful landings. The agent's performance will be evaluated based on the number of successful landings it is able to achieve within a set amount of time and number of attempts.

III. LITERATURE REVIEW

In this project, we have created a 3D environment using Unity to simulate rocket landing. The rocket model is specifically designed for this purpose. We are utilizing the ML Agent toolkit, along with Machine Learning, Deep Learning, and TensorFlow algorithms, to train and simulate the rocket's behavior. Real-world parameters such as rocket speed, gravity, maximum landing speed, and the position of the landing base are taken into account. These parameters are integrated into a Neural Network for RL training. Initially, the rocket performs random operations. As training progresses, the rocket or the code is rewarded for successfully completing the landing task, such as landing the rocket on the platform. Based on these rewards, adjustments are made to the neural network settings to encourage behaviors that lead to higher success rates. Through iterative trial and error, we aim to develop a machine learning system capable of achieving precise rocket landings on the platform.

We created a simulation environment using Tensorflow, taking into account real-life parameters like speed, gravity, rocket speed, and so on, during landing. We successfully landed spacecraft or rockets after constructing the simulated environment with Python.

SpaceX has recently employed a convex optimization technique to identify the optimal landing path for their rockets, leveraging real-time computer vision data. This approach enables them to dynamically replan the landing trajectory, ensuring precise landings even in the presence of significant variations in the rocket's initial state. As future flights acquire fresh data during the mission, there will be a growing need for rapid planning methods that can effectively address real-time demands.

The complexity of planning landing trajectories arises from nonlinear dynamics, nonconvex constraints on states and controls, posing a real-time challenge for existing nonconvex optimization methods. However, a study suggests that employing convex optimization techniques can significantly enhance the performance in addressing rapid trajectory optimization problems.

The approach is based on the universal approximation theorem, assuming there is an ANN that approximates the dynamical function with a known approximation error. The training set's input-output pairs are effectively stacked vectors of system states and control vectors for the input.

The ultimate result is a neural network model that includes all of the dynamics and may be used to predict future states. The usage of recurrent neural networks improves dynamics reconstruction when just artificial neural networks are used. neural networks instead of more basic feedforward networks In the literature, recurrent neural networks are utilized to carry out the job, despite their typically poor quality.

IV. METHODOLOGY

About the Rocket landing and Reinforcement Learning-

In the simulation, the rocket is randomly spawned at a distance of 5-6 kilometers from the target, facing a random direction and positioned randomly. The center of mass of a rocket is commonly positioned at two-thirds of its length. It is worth noting that the majority of the rocket's mass is concentrated within this two-thirds length. The objective for the rocket is to successfully land on the ground with a speed ranging between 4 m/s to 5 m/s.

Despite variations in the simulation, the agents consistently adopt the suicide burn landing strategy. This strategy entails igniting the main engine at the very last moment to decelerate the rocket and achieve a successful landing. Although this approach is considered risky, it is the most fuel-efficient method in terms of consumption.

The simulation results demonstrate that the agents have learned to optimize their fuel usage by consistently implementing the suicide burn strategy, irrespective of the initial conditions or random factors involved.

On getting a successful landing the agent is provided a reward, by which the agent tries to get the best possible output for every time step. On successfully landing the rocket gets a reward of +1 which makes the agent give a better output than before.

In this project we are not considering atmospheric conditions as one of the factors and the agent has only a single throttle level, the Reinforcement Learning problem can be characterized as having a finite set of possible actions. This means that the agent's movements are restricted to a discrete set of options, rather than a continuous range of possibilities. The most effective landing strategy at full speed is to approach at an angle, which is why this option is available to the agent. To achieve this, the agent employs Thrust vector control and Reaction control system to steer the Starship towards a seven-degree angle in both directions

To maintain the rocket's orientation, the agent controls the three main engines and four Reaction Control System Thrusters. This combination of Thrust vector control and Reaction control system controls the roll and pitch of the rocket.

This project has a limitation as we are not considering atmospheric conditions. Thus the velocity of the rocket and the trajectory of the rocket differs by a little to real-life circumstances. The other limitation is the rocket in this simulation is allowed to switch between the 3 engines and make them switch on and switch off, which is not similar to real-life circumstances. The real-life engines are complex to start and is a time-consuming process so it doesn’t switch on and off frequently.

The agent receives a total of 11 observations, where each observation is represented by a floating-point number indicating the rocket's position, rotation, and speed.

Those 11 observations are- Hight, initial angle roll, initial angle pitch, x offset, z offset, z speed, tvc, thrust engine, thrust rcs thrust, collision speed

So the total number of possible actions are 162 at each time step. The agent only has control over the throttle level and the TVC on X axis and Z axis and there roll and pitch of RSV of the main engine

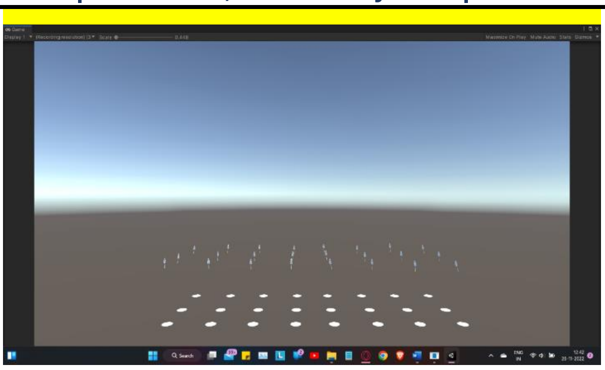
Figure.1 Training Area

The reward system is simple: the agent receives a reward of +1 for successfully landing the rocket. However, it is unlikely that the agent will achieve a positive reward by randomly selecting actions at the beginning of the training process.

To expedite the learning process, we employed two techniques: curriculum learning and imitation learning. By initially teaching the agent a less challenging task, such as launching the rocket from a lower height, it can progress towards the landing state and improve its chances of receiving rewards. Additionally, imitation learning involved providing the agent with examples to learn from, such as successfully recorded rocket launches. In summary, these techniques were utilized in the Starship Landing Support to enhance the learning process.

In summary, the Starship landing reinforcement learning problem is challenging. However, by utilizing curriculum and imitation learning techniques, we can facilitate more efficient learning and enhance the agent's chances of successfully landing the rocket.

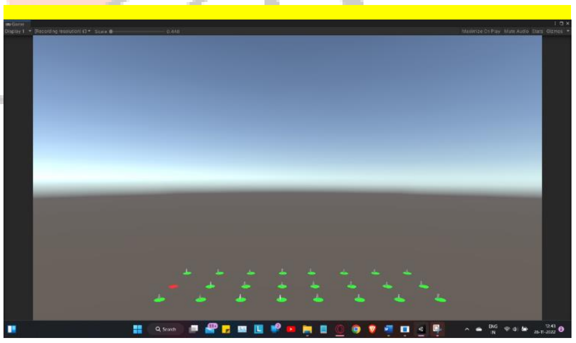
Figure.1 Training area with performed process

This approach is often considered suboptimal as it influences the agent's learning with potentially suboptimal examples. While the agent was trained with my demonstrations during only 0.0025% of its training, I would not classify it as imitation learning since it was only utilized at the initial stages and then disregarded. The agent was able to learn the optimal sequence of operations without relying on my demonstrations.

To facilitate the learning process, the agent initially relied on human examples to guide its actions and experience rewarding outcomes. This approach can be seen as an exploration strategy, where the agent explores the environment by taking actions that are similar to those leading to rewards, rather than relying solely on random actions. This combined approach, along with curriculum learning, proves to be effective when the reward function is limited. However, it's worth noting that this imitation learning technique may not be applicable in all scenarios and could even pose challenges in simpler tasks.

Random initialization was a crucial aspect utilized extensively during the entire training process to ensure that the agents could effectively adapt to different conditions. This approach aimed to enhance the agents' ability to generalize their learning. Notably, this approach proved beneficial when gradually increasing the altitude during training, from 500m to 5km. Despite significant changes in the initial conditions, the agents demonstrated robust performance. Even when the spawning altitude was considerably altered (e.g., from 900m to 1.5km), the decline in performance was less than expected. This observation indicates that the agents developed a higher level of resilience and adaptability to handle variations in the initial conditions.

In practice, the project is implemented in Unity, utilizing the Unity ML Agents toolkit. This toolkit provides a seamless integration between Python and the Unity environment, which is highly beneficial as Python is a widely used language for machine learning and reinforcement learning. Moreover, Unity ML Agents allows for the concurrent training of multiple agents within a single environment, resulting in significant speedup in the training process. This is why you can observe 32 Starships landing in close proximity to each other, as they are undergoing simultaneous training. The PPO (Proximal Policy Optimization) algorithm was utilized in its original form without any modifications to align with the desired behavior. This fact is worth mentioning and can be further explored in the Results section for more detailed information.

V. BLOCK DIAGRAM DESCRIPTION

block diagram for this project would likely include the following components:

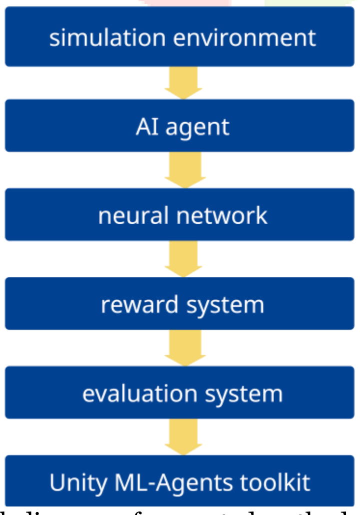

Figure.1 Block diagram of suggested methodology

The simulated environment, which could be created using the Unity game engine and would include a representation of the rocket and the landing site.

Deep reinforcement learning methods would be used to train the AI agent. It would observe the simulated setting and decide what steps to take to safely land the rocket.

The neural network, which would be used by the AI agent to make decisions and take actions. It would be trained using the data from the simulated environment and the rewards received for successful landings.

The components for the agent neural network are Height, roll angle, angle pitch angle, offset, speed, tvc angle, engine thrust, rcs thrust, collision speed

The reward system, which would provide feedback to the AI agent on its performance. The agent would receive rewards for successful landings and use this feedback to adjust the parameters of its neural network.

The evaluation system, which would be used to measure the agent's performance and determine if it has successfully learned to land the rocket. This could include metrics such as the number of successful landings within a set amount of time and the number of attempts.

The Unity ML-Agents toolkit, would be used to train the agent and also will be used to provide the interface between the agent and the 3D environment.

The block diagram would depict the interaction between these different components, with the simulated environment providing input to the AI agent, which uses its neural network to make judgements and take actions. The agent receives rewards for successful landings, which are used to adjust the parameters of the neural network, and the agent's performance is evaluated by the evaluation system.

Unity ML-Agents

The Unity ML-Agents plugin, developed by Unity Technologies, is a valuable addition to the Unity game engine that allows us to create or utilize pre-made environments for training our agents. This plugin has gained popularity among developers of AAA games, including titles like Firewatch, Gone Home, Cuphead, and more. It provides a powerful framework for integrating machine learning and artificial intelligence into game development processes.

To gain a better understanding of the RL process, let's examine it as a loop, as shown in the suggested methodology block diagram in Figure 1.

Suppose we have an agent who is learning to play a platform game.

1. The environment gives our agent access to the game's S0 beginning state.

2. Based on the state S0, our agent moves to the right in response to action A0.

3. The surroundings transition into a new state, S1.

4. We provide our agent a positive reward of R1 (+1) since it is still alive.

5. A state, action, and reward sequence are produced by this RL loop. To maximize the anticipated cumulative benefit is the agent's goal.

VI. RESULT

The neural network used in this project has approximately 40,000 parameters. To train the algorithm, we used a Ryzen 7 5800U processor and a GTX 1660ti, completing around 200 million training steps. The training process took about 26 hours. After training, the agent achieved successful Starship landings around 95% of the time, as indicated by an average reward of approximately 0.92. It was possible to complete the training in under 20 hours with a success rate of 93%, but for real-time implementation of the project, we aim for maximum efficiency, equivalent to 100%.

The neural network has roughly 40k parameters in total. The algorithm was trained on a Ryzen 7 5800U processor and GTX 1660ti for a total of roughly 200 million timesteps. About 26 hours were needed to complete the agent's training. After training, the agent successfully lands Starship around 98% of the time, as indicated by the final trained agent's mean reward of around 0.92. Even though I could have finished training much earlier than 20 hours and still achieved a good score (>95%), even a 98% performance is obviously insufficient for practical application. additional practice and effective hyperparameter tuning

Performance could be significantly higher, the "black box" effect once more makes the practical application of RL for this purpose essential. Three-layer neural networks with a learning rate of 3.0E-4, a batch size of 64, and later 128 are the parameters used.

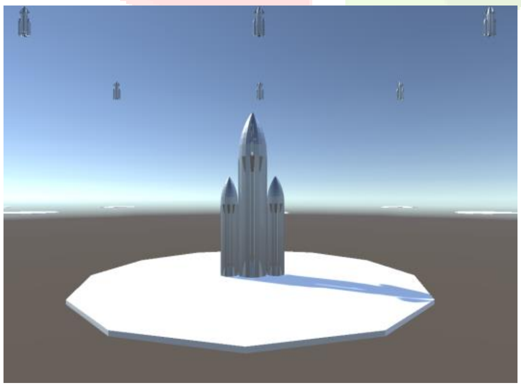
Figure.1 Rocket

In comparison, DeepMind's popular Deep Q-learning (DQN) implementation for Atari games has fewer than 10,000 parameters and is trained using a CNN neural network for a total of 10 million steps. This means that the agent observes game screenshots, similar to how humans perceive the game. The difference in the number of learning steps can be attributed to two factors. Firstly, I developed this project individually over a few weeks. Secondly, while both this project and DQN utilize deep reinforcement learning, they employ fundamentally different algorithms. DQN is utilized by Google DeepMind. The advantage of DQN lies in its high data efficiency, enabling it to learn effectively from experiences at any point during training.

The biggest challenge in creating this project was choosing the appropriate reward function. I initially tried a denser reward function that may instruct the agent on how to do the task. But because the agent maximized them in odd ways, they were incredibly difficult to employ. For instance, if you punish the agent for hitting the ground too quickly, it won't hit the ground at all and won't receive a penalty. In general there is a requirement of a reward function that instructs the agent what to do rather than how to do it, allowing the agent to act anyway it sees fit in order to accomplish your goals.

VII. REFERENCES

[1] R. S. Sutton and A. G. Barto, "Temporal Differencing," in Reinforcement Learning: An Introduction, 2013, pp. 132–163.

[2] A. Wolf, B. Acikmese, Y. Cheng, J. Casoliva, J. M. Carson and M. C. Ivanov, "Toward improved landing precision on Mars," Aerospace Conference, 2011 IEEE, Big Sky, MT, 2011, pp. 1-8.

[3] J. Gardi and J. Ross, "An Illustrated Guide to SpaceX's Launch Vehicle Reusability Plans." [Online]. Available: http://www.justatinker.com/Future/.

[4] K. J. Kloesel, J. B. Pickrel, E. L. Sayles, M. Wright, and D. Marriott, "First Stage of a Highly Reliable Reusable Launch System," AIAA Sp. 2009 Conf., no. September, pp. 1–17, 2009.

[5] SpaceX, "SES-10 Mission," no. March, pp. 9–10, 2017

[6] "Blue Origin | Our Approach to Technology." [Online]. Available: https://www.blueorigin.com/technolog. [Accessed: 07-Apr-2017].

[7] G. Sutton and O. Biblarz, "Thrust Vector Control System," in Rocket Propulsion Elements, 7th ed., vol. 17, no. 5, 1980, pp. 407–412.

[8] K. Lauer, "Box2D." [Online]. Available: https://pypi.python.org/pypi/Box2D/2.0.2b1

[9] SpaceX, "Falcon User's Guide," p. 69, 2015.

[10] M. M. Ragab et al., "Launch Vehicle Recovery and Reuse," Am. Inst.

Aeronaut. Astronaut., pp. 1–10.

[11] NASA, "Gimbaled Thrust," Beginner's Guide to Rockets, Public Domain Source. [Online]. Available:

https://spaceflightsystems.grc.nasa.gov/education/rocket/gimbaled.html. [Accessed: 07-Apr-2017].

[12] J. J. Isaac and C. Rajashekar, "Fluidic Thrust Vectoring Nozzles," no. April, 2014

[13] SpaceX, "Launch Manifest." [Online]. Available:

http://www.spacex.com/missions. [Accessed: 07-Apr-2017].

[14] N. Chris Bergin, "SpaceX's Autonomous Spaceport Drone Ship ready for action." [Online]. Available:

https://www.nasaspaceflight.com/2014/11/spacex-autonomous-spaceportdrone-ship/. [Accessed: 07-Apr-2017].