To The MOOOOOOON's Surface

Joey Caley
Aerospace Engineering Sciences
University of Colorado Boulder
Boulder, CO
joey.caley@colorado.edu

Dylan Klein
Aerospace Engineering Sciences
University of Colorado Boulder
Boulder, CO
dylan.klein@colorado.edu

Cameron Mattson
Department of Computer Science
University of Colorado Boulder
Boulder, CO
cameron.mattson@colorado.edu

***Abstract*—Landing on the moon is an iconic problem, and has been captured in a simplified form in the Gymnasium Lunar Lander environment. This sort of dynamics problem is well-suited for reinforcement learning algorithms, and so Q-Learning and Deep Q-Learning (DQN) were explored as potential solutions. Using techniques such as discretization, heuristic policies for Q-Learning, and hyperparameter tuning for DQN, both approaches achieved successful policies. When their performance was evaluated using the average reward achieved per episode after training Q-Learning was able to reach a high-performing policy with an average reward of 130. DQN improved upon this benchmark by achieving an average reward of 240, nearly doubling the average score of Q-Learning. However this score was achieved with more noticeable variation in performance during training. The quality of the policies was also noticeably different from the simulation footage, with DQN achieving a much smoother landing. Ultimately, both approaches were shown to be viable solutions to this problem, and warrant further testing under more challenging conditions.**

***Index Terms*—Reinforcement Learning, Machine Learning, Deep Learning, Entry Descent and Landing, Robotics, Guidance Navigation and Control**

I. Introduction

Landing a spacecraft on the moon or other planetary body is a challenging task requiring precise control and decision making accomplished onboard a moving vehicle. While this problem has traditionally been solved using variants of optimal control, an increasing number of solutions utilize reinforcement learning. In this paper, we applied reinforcement learning techniques to train a policy to safely land a lunar lander at a specific location in a simplified simulation of a lunar environment. Specifically, we implemented and compared Q-learning and DQN to learn an optimal policy for this sequential decision making problem. The rest of our work is summarized as follows: Section II provides a background on the lunar landing problem and related work applying reinforcement learning approaches. Section III formally defines the problem and our MDP formulation. Section IV describes our solution approach, detailing the Q-learning and DQN algorithms. Finally, Section V presents our experimental results comparing the two approaches.

II. Background

Landing on the moon is one of mankind's seminal achievements, the finish line of a decade of progress and competition between the world's superpowers. It is an interesting problem to many as a result, and its underlying trajectory optimization was crafted into a testbed for reinforcement learning algorithms in Gymnasium [1]. The problem of finding the best path to a goal given an initial state is well-studied in literature with many applications in robotics and aerospace systems. Aerospace scholars have been particularly interested recently in how to optimize a trajectory onboard a flight computer during landing. This capability is central to enabling interplanetary landings and to landing launch vehicles on Earth. As such, there have been many approaches posed that utilize optimal control theory with techniques like convex optimization or lossless convexification to compute fast, accurate, and reliable trajectories during flight [2] [3]. These sort of approaches have even been successfully deployed on real systems like the SpaceX Falcon 9 rocket, resulting in a revolution in how we approach launch vehicles.

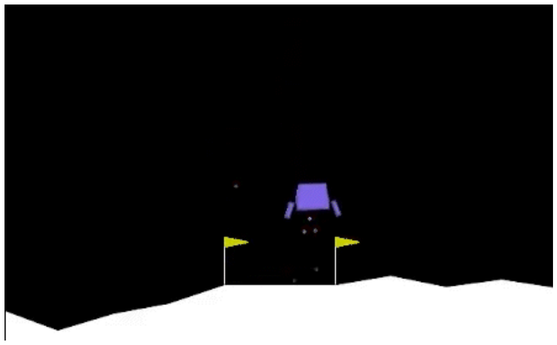


Fig. 1. Lunar Lander Simulation

However, optimal control theory is not the only possible solution to this problem. Like many problems in Guidance, Navigation, and Control (GNC), approaches grounded in artificial intelligence and reinforcement learning are increasingly being explored [4] [5] [6]. These approaches try to leverage recent machine learning advances such as computer vision or transformer neural networks to achieve potential improvements over optimal control. Improvements include direct mapping from camera images to thrust commands or faster computation time, which is a particularly key improvement for a time constrained problem. However like many other applications of reinforcement learning to traditional control problems, achieving the guarantees that control theory provides is an ongoing challenge. These solutions are still intriguing however and hopefully will soon move from simulation to the real world as spacecraft flight computers improve. Following in the footsteps of this active area of research, this work focuses on applying reinforcement learning approaches to the problem of lunar landing. Specifically within the Lunar Lander Gymnasium environment. This is a common problem for many students, with multiple published student papers and past 5264 Decision-Making Under Uncertainty final projects focused on this [7] [8] [9] [10].

III. Problem Formulation

We formulated the lunar landing task as a Markov Decision Process (MDP). The state space included the lander's position x and y, angle θ, linear velocity $\dot{x}$ and $\dot{y}$, angular velocity $\dot{\theta}$, and leg contact with the ground, which was represented with a boolean $c_{left/right\ leg}$. The action space consisted of individually boosting the left, right, bottom engines, or nothing at all. The reward function encouraged landing closer to the target pad, moving at a slower velocity, keeping the lander upright, and making leg contact with the ground, while penalizing excessive thrusting and crashing. The lander's dynamics and resulting transition probabilities are unknown, making this a model-free MDP problem.

$$\mathcal{S} = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \theta \\ \dot{\theta} \\ c_{leftfoot} \\ c_{rightfoot} \end{bmatrix} \tag{1}$$

$$\mathcal{A} = \begin{bmatrix} do\ nothing \\ fire\ left\ engine \\ fire\ main\ engine \\ fire\ right\ engine \end{bmatrix} \tag{2}$$

$$\mathcal{R}(s,a) = \begin{cases} +/-\ \text{as the lander gets closer/further from the} \\ \text{landing pad} \\ +/-\ \text{the slower/faster the lander is moving} \\ -\ \text{the more the lander is tilted from horizontal} \\ +10\ \text{for each}\ c_{left/right\ foot} = true \\ -0.03\ \text{for each frame a side engine is firing} \\ -0.3\ \text{for each frame the main engine is firing} \\ -100\ \text{for crashing} \\ +100\ \text{for landing safely} \end{cases} \tag{3}$$

This problem was then implemented in code using the Gymnasium Lunar Lander environment [1]. This is a common implementation of this problem and allows for easier development and comparison with past work on this problem. Due to limited documentation on the details of the reward function, some rewards are left vague, merely indicating the sign of the reward. We used the environmental conditions specified in Table I to test the spacecraft's landing policy.

Parameter	Description	Value
continuous	Continuous actions will be used.	False
gravity	Gravitational influence on the lunar lander.	-10.0
enable_wind	If wind is present or not.	False
wind_power	The power of the wind.	0
turbulence_power	Dictates the maximum magnitude of rotational wind applied to the craft.	0

TABLE I
Environment Parameter Descriptions

IV. Solution Approach

A. Q-Learning

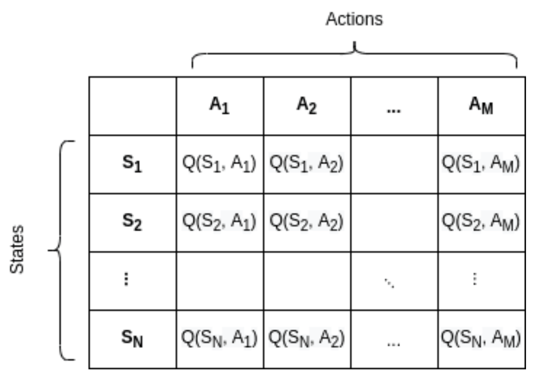
To solve the lunar lander problem, we first considered using MDP algorithms like Value Iteration and Policy Iteration. However, due to the large size of the state space, the memory and computational requirements of these methods were deemed infeasible. In addition, the transition probabilities were not known so they would need to be derived from the physics of the environment. These issues led us to skip this algorithm for something more feasible in the time we had. We first opted to use Q-learning [11], which learns a policy without requiring a model of the environment dynamics. It does this by learning the state-action value function, $Q(s,a)$ through exploration. A decaying ϵ-greedy policy is used, with the chance of selecting a random action, ϵ decreasing exponentially at each time step by a decay factor β. Otherwise, the current best action, estimated using the Q function for the current state is selected. This is shown in Algorithm 1. After each step in the environment, the estimated $Q(s,a)$ is using temporal differencing with some learning rate α as shown in Algorithm 2. Over many, many steps and simulations in the environment, it is able to converge to an optimal policy.

Algorithm 1: Decaying ϵ-Greedy Policy

```
Data: $Q, s, \mathcal{A}, \epsilon, \beta, k$
Result: $a$
$p \leftarrow Random()$;
if $p \leq \epsilon\beta^k$ then
    $a \leftarrow Random(\mathcal{A})$;
else
    $a \leftarrow max_a Q(s,a)$;
end
```

Algorithm 2: Q-Learning Update Algorithm

```
Data: $Q, s, a, r, s', \gamma, \alpha$
Result: $Q'(s,a)$
$Q'(s,a) = Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a))$;
```


Actions
States

	A_1	A_2	...	A_M
S_1	$Q(S_1, A_1)$	$Q(S_1, A_2)$		$Q(S_1, A_M)$
S_2	$Q(S_2, A_1)$	$Q(S_2, A_2)$		$Q(S_2, A_M)$
$\vdots$			$\ddots$	$\vdots$
S_N	$Q(S_N, A_1)$	$Q(S_N, A_2)$	...	$Q(S_N, A_M)$

Fig. 2. General Q-learning Table

Q-learning allows more efficient exploration since it is an off-policy method, meaning that it can be trained on any date from the environment. One limitation of Q-learning is that it does not work with continuous state spaces. Since the lunar lander environment has continuous states, discretization was necessary as a preprocessing step. This was achieved by breaking up the continuous states into n "bins" for each state variable. See Algorithm 3 for the full details. The result of this process is a set of discrete states that can be used in a Q-table, as shown in Figure 2, where each state-action pair has a corresponding state-action value. This does limit the optimality of the learned policy, but it makes the problem solvable.

```
Algorithm 3: State Discretization
Data: $s_{continuous}$, $|\mathcal{S}|$
Result: $s_{discrete}$
$s_{lower} \leftarrow [-1.5, -1.5, -5, -5, -\pi, -5, \text{false}, \text{false}]$;
$s_{upper} \leftarrow [1.5, 1.5, 5, 5, \pi, 5, \text{true}, \text{true}]$;
for $i \leftarrow 0$ to $|\mathcal{S}|$ do
    if $i \leq 6$ then
        $s_{scaled} \leftarrow \frac{s[i]-s_{lower}[i]}{s_{upper}[i]-s_{lower}[i]}$;
        $s_{discrete}[i] \leftarrow \text{Floor}(n * s_{scaled})$;
    else
        $s_{discrete}[i] \leftarrow s_{continuous}[i]$;
    end
end
```

B. DQN

To work with truly continuous states and achieve more optimal answers, a continuous Q-function is required. Deep learning is a good option for this, due to its ability to learn an approximation of nearly any function given enough training data. Deep Q-learning was developed for this and its underlying loop is summarized in Algorithm 4 using a Q_θ network parameterized by θ.

While this is useful in theory, it does not work in practice. Correlation between successive samples creates issues, small training batches makes the steps less accurate for the whole problem, and the constant adjustments to Q means that its

```
Algorithm 4: Deep Q-Learning Algorithm Loop
Data: env, $\gamma$, $\alpha$, $Q_\theta$, $s$
Result: $Q_\theta$
$a \leftarrow \epsilon - greedy(Q_\theta, s, \text{env}.\mathcal{A}, \epsilon)$;
$r \leftarrow \text{act!}(\text{env}, a)$;
$s' \leftarrow \text{observe}(\text{env})$;
if $s'$ is terminal then
    $\theta \leftarrow \theta - \alpha\nabla_\theta(r - Q_\theta(s,a))^2$ ;
else
    $\theta \leftarrow \theta - \alpha\nabla_\theta(r + \gamma\max_{a'} Q_\theta(s',a') - Q_\theta(s,a))^2$ ;
end
$s \leftarrow s'$
```

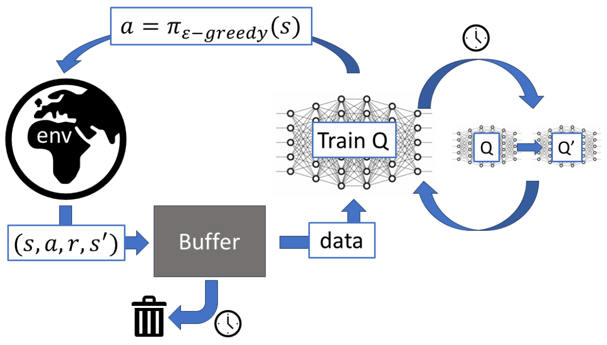


Fig. 3. Modified Deep Q-Learning

hard for the system to settle. As a result, modifications to the original DQN training loop were created [12]. These include: 1. the usage of a buffer of recent data that is sampled from for training and regularly updated with more recent data and 2. a frozen target network $Q_{\theta'}$ used in the loss function, which is only updated to the current network Q_θ periodically. When implemented, DQN is trained as shown in Algorithm 5. For this work, the ϵ-greedy policy decays linearly rather than exponentially according to Algorithm 6.

1) Training Method and Step Size: The training parameters of the DQN and their values are described in Table II in greater detail. To gather important information, simulations would end when either a terminal state was reached, or the cumulative discounted reward became greater than the discounted reward threshold defined in Table II.

2) Neural Network Construction: The architecture consisted of two (32) hidden fully-connected relu layers, the input layer (8), and the output layer (4).

V. Results

All average rewards shown were obtained by evaluating our Q function, both discrete and deep, and its resulting average reward over a set number of epsiodes. Due to differences in implementation, Q-learning evaluated using 1000 episodes and DQN used 100 episodes. See Algorithm 7 for full details.

A. Q-Learning

The Q-learning algorithm was surprisingly effective at finding a decent policy for the lunar lander. According to the

Algorithm 5: Modified Deep Q-Learning Algorithm Loop

```
Data: env, $\gamma$, $\alpha$, $Q_\theta, s, n_{buffer\ size}, k_{update}$
Result: $Q_\theta$
$a \leftarrow \epsilon - greedy(Q_\theta, s, \text{env}.\mathcal{A}, \epsilon)$;
$r \leftarrow$ act!(env, $a$);
$s' \leftarrow$ observe(env);
Add $(s, a, r, s')$ to the buffer;
Randomly sample from buffer;
if buffer has exceeded max size $n_{buffer\ size}$ then
    remove oldest data;
end
randomly sample experience tuples $(s, a, r, s')$ from
  buffer and calculate loss of $Q_\theta$;
if $s$ is terminal then
    $l(s, a, r, s') \leftarrow (r - Q_\theta(s, a))^2$
else
    $l(s, a, r, s') \leftarrow (r + \gamma \max_{a'} Q_{\theta'}(s', a') - Q_\theta(s, a))^2$
end
Train $Q_\theta$ network according to loss function $l$ and step
  size $\alpha$;
if $k_{update}$ iterations have passed since last update then
    $Q_{\theta'} \leftarrow Q_\theta$
end
$s \leftarrow s'$
```

Algorithm 6: Decaying ϵ Linearly

```
Data: $\epsilon_0$,$|epochs|$, $\epsilon_{end}$
Result: $\epsilon$
$\epsilon \leftarrow \epsilon_0$;
$\Delta\epsilon \leftarrow \frac{\epsilon - 0.2}{|epochs|}$;
for $epoch \leftarrow 1$ to $|epochs|$ do
    $\epsilon \leftarrow \epsilon - \Delta\epsilon$;
end
```

Algorithm 7: Evaluation Function

```
Data: $Q$, env, $n_{episodes}$
Result: $\overline{r}$
$r_{sum} \leftarrow 0$
for $n_{episodes}$ do
    $s \leftarrow$ reset(env) ;
    while $s$ is not terminal do
        $a \leftarrow \text{argmax}_a(Q(s))$;
        $s', r \leftarrow$ act!(env, $a$);
        $r_{sum} \leftarrow r_{sum} + r$;
        $s \leftarrow s'$
    end
end
$\overline{r} \leftarrow \frac{r_{sum}}{n_{episodes}}$
```

Parameter	Description	Initial Value
Discount (γ)	A measure of the importance of future rewards.	0.99
Epochs	The number of times the DQN trains on the experience tuples in the buffer.	3000
Episodes Per Epoch	The number of episodes accumulating experience before training the DQN.	500
Data Per Epoch	Experience tuple size sampled from the buffer for training the DQN.	10000
Buffer Size	The maximum number of experience tuples stored in the buffer.	100000
ϵ	The probability of choosing a random action in Algorithm 1 which decays linearly with each epoch.	0.9 - 0.2
Discounted Reward Threshold	The threshold the discounted cumulative reward which would allow the episode to terminate.	0.005
DQN Optimizer Learning Rate	The step size taken to optimize the weights using the ADAM optimizer.	0.0001

TABLE II
LEARNING PARAMETER DESCRIPTIONS

environment creator, a reward of 200+ per episode indicates a successful landing policy. While our learned policies did not consistently reach this threshold, they were able to achieve an average reward of around 130 per episode. Figure 4 shows the learning curve for our initial Q-learning implementation before any hyper-parameter tuning.

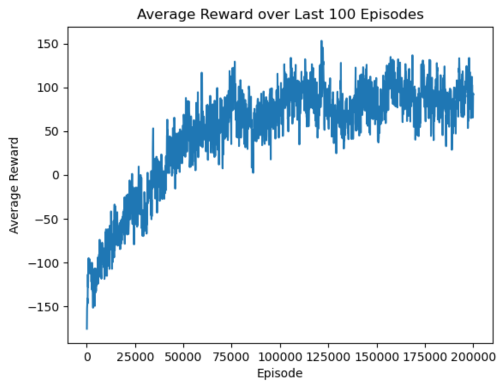


Fig. 4. Q Learning Average Reward Pre-Tuning

Performance improved by increasing the number of training episodes and the epsilon used in the epsilon-greedy behavior policy during early training. Qualitatively examining the final policy revealed that the lander would often continue firing its thrusters after touching down, in an attempt to move closer to the center of the landing pad. This prevented the lander from coming to rest, which is one of the environment's termination conditions. It also accumulated unnecessary negative reward for fuel use. To address this, we added a heuristic to the action selection that prevents firing the engine when both legs are in contact with the ground.

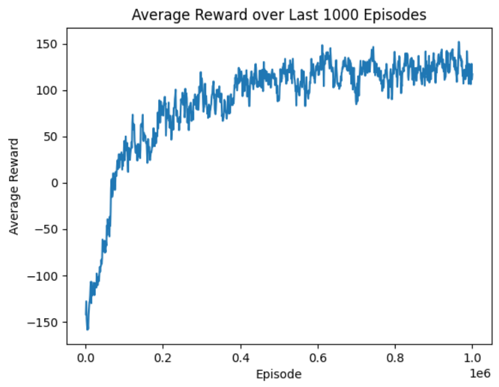


Fig. 5. Q Learning With both legs heuristic

An earlier version of this heuristic, which engaged after only one leg touched down, actually reduced performance, likely because it resulted in less stable landings. Figure 6 shows the results for our best performing Q-learning policy.

We also experimented with Double Q-learning, which can reduce the overestimation bias of regular Q-learning. However, this did not produce a measurable improvement in performance.

Evaluating the final policies visually using the simulation's visualization tools revealed a few failure modes. The policies generally handle the random initial velocities, but struggle to recover when the lander starts with a high downward velocity. This is likely because the action space only allows firing the main or side engines independently on each timestep. Slowing down requires using the main engine, so any lateral control with the side engines comes at the expense of vertical deceleration. See [13] for video examples of Q-Learning landings.

B. DQN

Through empirical evaluations we discovered that lowering the learning rate and accumulating fewer newer experiences

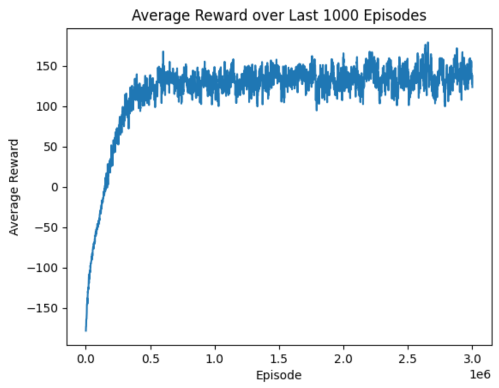


Fig. 6. Q Learning With Longer Epsilon Decay

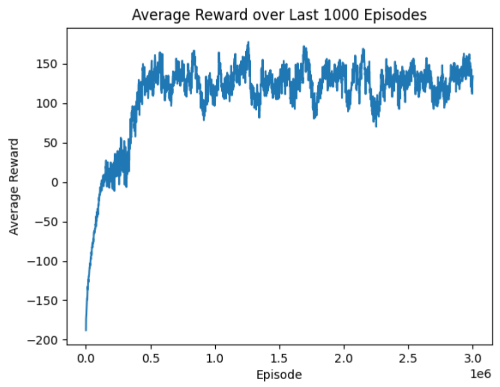


Fig. 7. Double Q Learning

in the buffer allowed the DQN to converge to higher episode-normalized rewards. Fewer experience tuples were stored by decreasing the number of episodes per epoch.

Surprisingly, a network with fewer layers and weights also performed better. One suspected reason for this discrepancy is the number of epochs required to train more complex networks compared to simpler network as it relates to ϵ-decay. The smaller network may be learning the same necessary patterns for understanding the environment in fewer epochs at larger ϵ values leading to an improved understanding of the environment through exploration. This is because of the lack of complexity in organizing each of the weights throughout the network.

DQN improved upon the Q-Learning results, achieving an average reward of 241.97 over 100 episodes. The process to get there is shown visually in Figure 8. This process was much noisier than the Q-learning approach, with drastic spikes in average reward below -1000, even after achieving a positive average reward. A lower learning rate may have helped alleviate this. Upon visual inspection of the spacecraft's landing using the environment's video visualizations, we generally noticed a gradual decent for most scenarios. The spacecraft doesn't always land between the flagged landmarks, however. In these failed scenarios sometimes the side thrusters will continue to attempt propelling the spacecraft to the targeted location even after landing on the lunar surface. This is a possible source of these incredibly negative rewards, as the negative reward from firing the thrusters may drag the score down. Implementing the heuristic used in Q-learning may have helped with this. A video of the spacecraft landing has been prepared, and can be accessed at [13].

VI. Conclusion

In this project, we applied reinforcement learning techniques to solve the lunar landing problem in simulation. We implemented and compared two approaches: Q-learning with discretized states and Deep Q-Learning (DQN) using neural networks to approximate the Q-function over continuous states. Our results showed that even the simpler Q-learning method

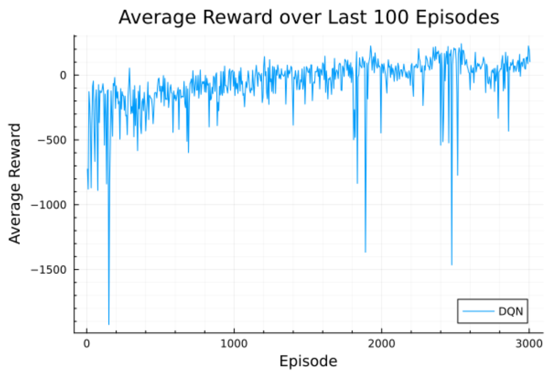


Fig. 8. Deep Q-Learning

was able to learn a policy that landed the spacecraft near the target location fairly consistently, achieving an average reward of around 130 points per episode. Adding heuristics to prevent unnecessary thruster firing after touchdown helped improve the learned policy. However, the discretization of the state space inherently limited the optimality of the Q-learning solution. The DQN approach aimed to address this limitation by working with the continuous state space directly. Through careful design of the network architecture, loss function, and training procedure, we were able to get DQN to learn an effective policy as well, achieving an average reward over 240. Key elements included using a replay buffer to decorrelate the training samples, a target network to stabilize the Q-function estimation, lowering learning rates, and a decaying exploration rate to gradually shift from exploration to exploitation.

Future work would likely focus first on tuning DQN hyper-parameters to reduce the variation seen in results and improve the speed of training. Making the problem a POMDP through adding noise to the measurements would likely also be useful, as that more accurately reflects the conditions present in a lunar landing. The environment also contains some simpler challenges, like adding wind or making the action space continuous by enabling throttling of the motors.

Overall, this project demonstrates the potential for reinforcement learning to solve complex aerospace control problems like lunar landing. With further research and development, these AI-based approaches could one day enable greater autonomy and reliability for spacecraft landing on the moon and beyond.

VII. Contributions and Release

Joey Caley was the primary author of the report and assisted with Deep Q-Learning debugging and training. Dylan Klein contributed to the Q Learning portions of this assignment. Cameron Mattson contributed to the Deep Q-Learning portions of this assignement. All Q learning and DQN algorithms were developed by scratch, no off-the-shelf algorithms were used. The authors grant permission for this report to be posted publicly.

References

[1] Oleg Klimov, “Lunar Lander Environment,” Gymnasium, 2024.
[2] L. Blackmore, B. Açıkmeşe and D. P. Scharf, “Minimum-Landing-Error Powered-Descent Guidance for Mars Landing Using Convex Optimization,” in Journal of Guidance, Control, and Dynamics, vol 10:4, pp 1161-1171, 2010.
[3] B. Açıkmeşe, J. M. Carson and L. Blackmore, “Lossless Convexification of Nonconvex Control Bound and Pointing Constraints of the Soft Landing Optimal Control Problem,” in IEEE Transactions on Control Systems Technology, vol. 21, no. 6, pp. 2104-2113, Nov. 2013.
[4] B. Gaudet, R. Linares and R. Furfaro, “Deep reinforcement learning for six degree-of-freedom planetary landing,” in Advances in Space Research, vol 65:7, pp 1723-1741, 2020.
[5] A. Scorsoglio, A. D’Ambrosio, L. Ghilardi, B. Gaudet, F. Curti and R. Furfaro, “Image-Based Deep Reinforcement Meta-Learning for Autonomous Lunar Landing,” in Journal of Spacecraft and Rockets, vol 59:1, pp 153-165, 2022.
[6] J. Briden, T. Gurga, B. J. Johnson, A. Cauligia and R. Linares, “Improving Computational Efficiency for Powered Descent Guidance via Transformer-based Tight Constraint Prediction,” in AIAA SCITECH 2024 Forum.
[7] D. Dharrao, S. Gite and R. Walambe, “Guided Cost Learning for Lunar Lander Environment Using Human Demonstrated Expert Trajectories,” in 2023 International Conference on Advances in Intelligent Computing and Applications (AICAPS), Kochi, India, pp. 1-6, 2023.
[8] S. Gadgil, Y. Xin and C. Xu, ”Solving The Lunar Lander Problem under Uncertainty using Reinforcement Learning,” in 2020 SoutheastCon, Raleigh, NC, USA, pp. 1-8, 2020.
[9] A. Diallo, B. Chupik and G. Insinger, “Autonomously Landing a Rocket on the Moon with Reinforcement Learning Techniques,” in DMU Past Projects, 2023.
[10] N. Foote, “Sticking the Landing with Deep Q-Learning,” in DMU Past Projects, 2023.
[11] C.J.C.H. Watkins, “Learning from Delayed Rewards,” in PhD Thesis, University of Cambridge, England, 1989.
[12] V. Mnih et. al, “Playing Atari with Deep Reinforcement Learning,” 2013.
[13] C. Mattson, D. Klein, J. Caley, “Q-Learning and DQN Policy Results Videos,” in DMU Final Project, CU-Boulder OneDrive, 2024, https://tinyurl.com/LunarLanderDMU.