Self Landing Rocket using Deep Reinforcement Learning

Carson Kohlbrenner, Thomas Dunnington, Owen Craig

1) Link to full video of simulated mission here.
2) Link to GitHub repository here.

***Abstract*— Autonomous landing of rockets poses a significant challenge in aerospace engineering. This study investigates the effectiveness of various control methods for achieving precise landings within a continuous state space. This begins with the development of a dynamic simulation environment with two-dimensional physics, neglecting aerodynamic effects. To control the landing, three different control strategies are explored: Proportional-Derivative (PD), Deep Q-Networks (DQN), and Behavior Cloning. We begin by implementing a PD controller, which demonstrates consistent performance in guiding rockets to successful landings. Building upon traditional control methods, the application of Deep Q-Networks (DQN) and Behavior Cloning for autonomous landing tasks is explored. The best-performing Q-network achieves promising results, indicating the potential of reinforcement learning techniques in complex control tasks. Behavior Cloning yields results comparable to the PD controller, showcasing the successful learning of expert policies.**

I. Introduction

The successful landing of rocket boosters represents a pivotal advancement in the field of space exploration, allowing for a significant reduction in the cost of getting to space. Traditionally, model predictive control has been the method of choice for rocket landing maneuvers due to its precise control capabilities. However, model predictive control demands substantial computational resources, making it less feasible for real-time applications on resource-constrained platforms. As an alternative, Deep Reinforcement Learning could be a potential alternative capable of deriving complex control policies directly from raw sensor data. Although training a Deep Reinforcement Learning network requires significant computational effort, once the model is fully trained, the computational demands for onboard operations are considerably reduced, offering a more scalable solution for real-time control in aerospace applications.

In this paper, a thorough analysis of various control strategies for autonomously landing rocket boosters was conducted. The paper assesses the performance of several algorithms, focusing on their convergence speed, stability, and ability to generalize across different landing scenarios. This work seeks to showcase the efficacy and practical aspects of using Deep Reinforcement Learning in autonomous rocket landings, setting the stage for more efficient and reliable launch vehicles.

All authors are with the University of Colorado Boulder, 1111 Engineering Drive, Boulder, CO USA. `name.surname@colorado.edu`.

Fig. 1: A rocket can learn to autonomously land on a landing pad using deep reinforcement learning.

II. Background and Related Work

The self-landing rocket problem is commonly formulated as an MDP, but can also be modeled using control theory. Mikulis-Borsoi provided an in-depth MDP formulation of the problem using a dense set of three continuous actions without using any control laws and achieved a success rate of 95% [1]. Xue et al. attempted to improve upon the MDP formulation using proximal policy optimization (PPO), improved with LSTM, along with an imitation learning algorithm to train a recurrent neural network on how to land a rocket [2]. Their approach demonstrated that a basic PPO algorithm can yield greater results when trained by an expert policy through imitation and achieved a 92% landing success rate. Their approach required that a policy be learned from scratch initially.

The other approach to landing the rocket is using control theory. Ferrante used PID controls, LQR controls, and model predictive controls to land a rocket simulated in the Box2D simulator and achieved the highest rewards using the PID controller [3]. Other formulations of this problem have used convex optimization to land a rocket that relies on a well-described model of the dynamics to work. [4], [5].

III. Problem Formulation

The problem of landing a rocket can be modeled as an MDP where an agent decides what action to take given the current state of the rocket. For this model, perfect observability of the rocket states was assumed and a simplified dynamical model was utilized. The simulation environment includes a simplified two-dimensional problem with no aerodynamic effects. The rocket has a continuous state space with seven states: $\mathcal{S} = \{x, \dot{x}, y, \dot{y}, \theta, \dot{\theta}, t\}$. These states include the (x, y) position of the rocket, the attitude represented by a pitch angle θ, the corresponding rates of change $(\dot{x}, \dot{y}, \dot{\theta})$, and the time t.

The transition probabilities, $\mathcal{T}$, are deterministic based on the dynamics of the rocket system. The continuous dynamics are approximated using Euler's method of integration with a time step of 0.1 seconds. The equations to propagate the state of the rocket in the environment are shown below where F is the force due to thrust, T is the torque, m is the mass, and I is the moment of inertia. For this work, a small-scale rocket is used with a mass of 3000 kg and a moment of inertia of 625,000 kgm^2.

$$
\begin{aligned}
x(i+1) &= x(i) + \dot{x} \cdot \Delta t \\
y(i+1) &= y(i) + \dot{y} \cdot \Delta t \\
\dot{x}(i+1) &= \dot{x}(i) + \frac{F}{m}\cos(\theta + \frac{\pi}{2}) \cdot \Delta t \\
\dot{y}(i+1) &= \dot{y}(i) + (\frac{F}{m}\sin(\theta + \frac{\pi}{2}) - g) \cdot \Delta t \\
\theta(i+1) &= \theta(i) + \dot{\theta} \cdot \Delta t \\
\dot{\theta}(i+1) &= \dot{\theta}(i) + \frac{T}{I} \cdot \Delta t \\
t(i+1) &= t(i) + \Delta t
\end{aligned}
$$

Each step in the environment will propagate the states with a single time step. At each step, the agent has the following action space: $\mathcal{A} = \{2mg, mg, 0.2mg, 0\}$. These actions are thrust values scaled relative to the weight of the rocket. In this discrete action space, the agent learns what value of thrust to execute at every time step. Instead of including side thrusters as done in papers [1], [2], and [3], a vectoring angle ϕ on the main rocket exhaust is used to control the rocket torque T.

The vectoring angle is not included in the action space and is instead dictated by a PD control law that always tries to bring the rocket to the target and in an upright position. This vectoring angle is derived from an ideal torque under the following control law:

$$T_r = -K1_{\text{rot}} \cdot \theta - K2_{\text{rot}} \cdot \dot{\theta} - K3_{\text{rot}} \cdot \dot{x} - K4_{\text{rot}} \cdot K3_{\text{rot}} \cdot x \quad (1)$$

Settling time constants of 2s and 0.2s were used for the inner-loop gains. $K1_{\text{rot}} = 1.5625e6$ and $K2_{\text{rot}} = 3.4375e6$ were used after taking the eigenvalues of the reduced inner-loop state space model. Root locus plots then were used to set $K3_{\text{rot}} = -2e6$ and $K4_{\text{rot}} = 0.02$. The resulting controller was slightly underdamped. This control law outputted a control torque that would reorient the rocket.

This reference torque value was then converted to a vectoring angle, assuming that the center of gravity of the booster is exactly at the center of the booster. The following piecewise

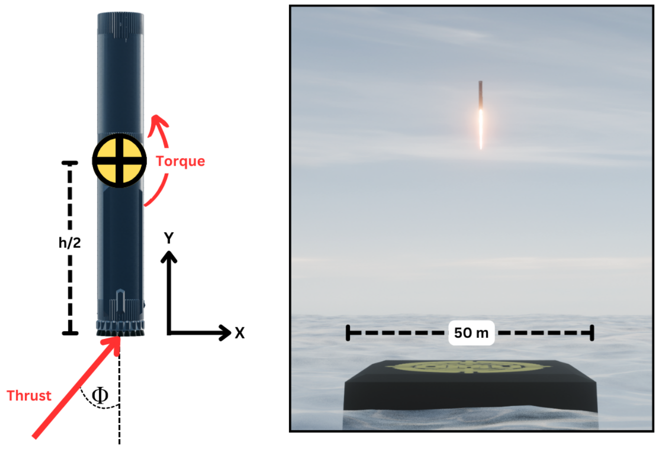

Fig. 2: Problem setup showing how thrust vectoring angle ϕ is setup to produce torque.

function converts the torque to the correct vectoring angle.

$$\phi(t) = \begin{cases} \text{sign}(T_r) \cdot \phi_{\max}, & \text{if } |\text{T}_r| > \text{F} \cdot \frac{y}{2} \cdot \sin(\phi_{\max}) \\ \arcsin\left(\frac{2 \cdot \text{T}_r}{h \cdot \text{F}}\right), & \text{otherwise} \end{cases} \quad (2)$$

This piecewise function commands a maximum vectoring angle if the torque required is greater than the torque the booster can produce within the vectoring angle range. Otherwise, the function calculates the vectoring angle that will produce the required torque.

The rewards, $\mathcal{R}$, depend on the final landing location and the intermediate actions taken in the air. For a successful landing, the rocket must land on the pad, have a low velocity, and be upright. There are continuous function rewards to incentivize learning by rewarding getting close to optimal landing conditions. Equations 3, 4, and 5 are continuous functions that increase in value when the rocket lands with a condition closer to the desired landing configuration. Namely, r_x is the reward for achieving an x position close to the target, r_θ is for having the correct orientation, and r_v is for having low velocity.

$$r_x = 15e^{\frac{-(x - x_{\text{targ}})^2}{(x_{\max} - x_{\min})}} \quad (3)$$

$$r_\theta = 50e^{-\pi|\theta|} \quad (4)$$

$$r_v = 40e^{\sqrt{\dot{x}^2 + \dot{y}^2}/v_{\text{crash}}} \quad (5)$$

These continuous functions are added to additional bonuses for having good landing parameters. This is shown below with the following tolerances: $\theta_{\text{tol}} = \pm\, 10°$, $\dot{\theta}_{\text{tol}} = \pm\, 10°/s$, $\dot{x}_{\text{tol}} = \pm\, 3$ m/s, $\dot{y}_{\text{tol}} = \pm\, 5$ m/s, $x_{\text{targ}} = 0$, and $v_{\text{crash}} = \pm\, 5$ m/s.

$$\mathcal{R}(y=0) = r_x + r_\theta + r_v + \sum \begin{cases} 50 & \text{if } x - x_{\text{targ}} < x_{\text{tol}} \\ 25 & \text{if } |\theta| < \theta_{\text{tol}} \;\; \& \;\; |\dot{\theta}| < \dot{\theta}_{\text{tol}} \\ 25 & \text{if } |\dot{x}| < \dot{x}_{\text{tol}} \;\; \& \;\; |\dot{y}| < \dot{y}_{\text{tol}} \end{cases}$$

The above rewards are added together when the rocket hits the ground at $y = 0$. There are also intermediate rewards during the descent of the rocket. These rewards are much smaller and are used to guide the learning toward achieving the larger rewards for a successful landing. Equation 6 is the reward earned during the rocket's descent. It is based on the current velocity of the rocket $\dot{x}$ and the path to the target $\vec{x}_{\text{targ}} - \vec{x}$. If the velocity is in the same direction as the path to the rocket, the dot product results in a large number while if it is directly opposite, it will be negative. This reward is in place to incentivize learning by having the agent learn to go toward the target. In addition, at any time during the trajectory, if the rocket leaves the predetermined bounds of (-100, 100) for x and (0, 2000) for y, there is a large negative reward as shown by Equation 7. There is also a discount factor of $\gamma = 0.999$ to incentivize shorter landing times. A landing is considered a success if the rocket lands on the pad and has a state within the aforementioned tolerances.

$$r_{\text{descent}} = 0.5 \frac{(\vec{x}_{\text{targ}} - \vec{x})^T}{||(\vec{x}_{\text{targ}} - \vec{x})||} \vec{\dot{x}} \tag{6}$$

$$r_{\text{bound}} = -200 \text{ if } |x| > x_{\text{bound}} \text{ or } |y| > y_{\text{bound}} \tag{7}$$

The goal of this project is to develop a model that can handle a variety of different scenarios and successfully land the rocket. To accomplish this, each simulation of the rocket includes a random initial condition. The initial condition of the rocket is randomly determined according to the following bounds in Table I

TABLE I: Initial Conditions

State	Min Value	Max Value
x	-100 m	100 m
y	2000 m	2000 m
$\dot{x}$	-5 m/s	5 m/s
$\dot{y}$	-20 m/s	-10 m/s
θ	$-45°$	$45°$
$\dot{\theta}$	0 $°/s$	0 $°/s$

The initial states have a random distribution except for the initial height and angular rate. The height is constant for each simulation to simplify the beginning of the control and the angular rate is set to zero to simplify the dynamics at the beginning of the landing burn.

IV. Solution Approach

Three separate approaches for controlling the rocket to land on the pad are explored: PD control, Deep Q-Learning, and Behavior Cloning. A PD controller is a common method for controlling dynamical systems. For this work, a PD controller was developed and utilized as the expert for both the DQN and Behavior Cloning methods. Deep Q-learning is a popular reinforcement learning algorithm that can be used to learn optimal policies in a variety of different environments. The implementation of a Deep Q-network (DQN) to land the rocket at random initial conditions was explored. Lastly, Behavior Cloning is a supervised learning technique where a function approximator learns to imitate the actions of an expert. Supervised learning with the PD controller was augmented with the Dataset Aggregation (DAgger) method to create a robust model that imitates the expert.

A. PD Controller

The first phase of the solution was to design PD control laws that operate in a continuous action space for the booster's thrust. The control law for the continuous thrust is as follows.

$$F = \text{clamp}\left(-k_1^t \cdot (y - y_{\text{ref}}) - k_2^t \cdot \dot{y}, 0.0, 2 \cdot g \cdot m\right) \tag{8}$$

This control law commands a thrust to maintain a given y_{ref}. The control law commands different y_{ref} for three flight phases. In the first flight phase, the booster is commanded to descend to 50% of the maximum height over a set descent time (t_D). In the second phase of flight, the control law commands the booster to descend to 10% of the maximum height. The final phase of flight is when the flight time is greater than twice the descent time. In this phase, the booster is commanded to land meaning y_{ref} is commanded to be 0. y_{ref} is defined by the equation 9.

$$y_{\text{ref}} = \begin{cases} y_{\max} \cdot \left(1 - \frac{t}{2 \cdot t_D}\right) & \text{for } t < t_D \\ 0.1 \cdot y_{\max} \cdot \left(1 - \frac{t - t_D}{t_D}\right) & \text{for } t < 2 \cdot t_D \\ 0 & \text{for } t \geq 2 \cdot t_D \end{cases} \tag{9}$$

Overall, this PD control law as well as the vectoring angle control law work in tandem to control and land the booster on target in a desired state.

B. DQN

The DQN solution to land the booster involves a combination of control strategies and reinforcement learning techniques. The solution used can be broken down into two main phases. The solution begins by taking the PD control laws for continuous action space (heuristic policy), converting this policy into a discrete form suitable for neural network training, and then a DQN to is trained to learn and optimize landing strategies based on the discrete actions.

To use the PD controller as the heuristic policy in the DQN, the continuous thrust values must be converted to the discrete values in the action space. To do this the closest discrete thrust value to the continuous thrust value is chosen to be the heuristic action. This action is then used by a ϵ-greedy with a linearly decaying ϵ that has an initial value of ϵ_{max} and decays to ϵ_{min} over a set number of exploring epochs. The policy uses this ϵ value to determine which action is taken. First, the policy will return the action corresponding to the maximum Q value with a probability of ϵ, otherwise, the policy will return the action from the heuristic policy 70% of the time or a random action 30% of the time. This ϵ-greedy approach improves learning because the agent will take the expert action often and a random action sometimes to explore and learn better policies than the expert.

Once the policy has been established, a Q network is designed with a single hidden layer connecting two input/output layers using the Relu activation function. Alongside training this Q network, there are two additional networks: Q best, which retains the Q network yielding the best average reward throughout training, and the target Q network, which stabilizes training by freezing the target values every five epochs. With these defined networks, the training uses the following parameters:

- **Learning Rate:** 0.0005
- ϵ_{max} 0.6
- ϵ_{min} 0.05
- **Epochs** 15000
- **Exploring Epochs** 7000
- **Buffer Size** 100000
- **Batch Size** 2000
- **Data Added to the Buffer per Epoch:** 1000
- **Max steps to Evaluate** 2000

The choice of the number of epochs was based on the complexity of the system and its large number of steps per episode. Given these factors, a substantial number of epochs is necessary for the network to effectively learn how to interact with the environment and successfully land the booster. In addition, ϵ is decayed over only 7000 epochs as the network needs to explore the environment, however, in the final 8000 epochs, the agent should take the action that returns the highest Q value most of the time, emphasizing the exploitation of learned strategies.

Before the model was trained each epoch the model collected tuples of experience using the current ϵ by interacting with the environment and storing them in the buffer. This was done for a maximum of 2000 steps or until a terminal state was reached. If adding the new experiences to the buffer caused the size to be greater than 100,000 the buffer was shifted removing old experiences first. Then from the buffer, 2000 random experiences were sampled and used to train the Q network using a loss function that aims to minimize the difference between predicted Q-values and the target Q-values. To train the network the flux.train! method was used with the ADAM optimizer. Once the network has been trained the current Q model was evaluated by finding the average reward of the network over 1000 simulations. If the average reward of the current Q network was the best performing network it was copied to the best model. At this point, the target Q was set to be the current Q network if five epochs have passed since the last reset. This was repeated for 15000 epochs. At the end of training, the best model was evaluated for 10,000 episodes and saved as a BSON file.

C. Behavior Cloning

Given the consistently good results of the heuristic PD controller, an attempt was made to clone the behavior by training a neural network to approximate the best action at a given state. Using the PD controller as the expert, the neural network is trained on large data sets of state-action pairs from full simulation trajectories. Once a supervised model is trained, the Dataset Aggregation (DAgger) method is used to further refine the function approximator to get a more robust model. The neural network used has four hidden layers and input/output layers. Within the hidden layers, there are 64 nodes and the Relu activation function is used. There are six inputs to the network for each of the states except time, and the output is a continuous thrust value. The loss function used the mean square error and the Flux library in Julia was used for training. With this structure, the following parameters were used for learning:

- **Inputs:** $[x, \dot{x}, y, \dot{y}, \theta, \dot{\theta}]$
- **Output:** F
- **Learning Rate:** 0.001
- **Number of Episodes:** 10000
- **Batch Size:** 5096
- **DAgger Epochs:** 100

With the large number of episodes, a large dataset of state-action pairs was gathered, which was then used to train the neural network. The data set was split into batches of 5096 state-action pairs and the network was trained using multiple epochs until all data points had been processed and used in training. Once the supervised training was complete, the DAgger method leveraged the expert's actions to improve the robustness of the model. During each DAgger epoch, a trajectory was simulated using the neural network's policy. At each step in the environment, the expert was used to get the optimal action which was then consolidated into another data set used to refine the model. Following the completion of an epoch, the model's performance improved slightly as the new data helped the network approximate actions at states it may have not seen before.

V. Results

A. PD Controller

Using the PD controller resulted in consistent landings within the continuous action space as shown by Figure 3. In this figure, the arrows indicate the orientation of the rocket pointing directly out of the top of the booster. A total of 10 different trajectories are depicted with all of them successfully landing within the tolerances listed in the problem formulation. Important states and actions are shown in Figure 4. The oscillatory motion due to the underdamped system can be seen in the change of θ over time. The control inputs change around 1250 time steps as the reference input to the control law shifts as shown by Equation 9. Table II shows the average cumulative reward of 10,000 episodes for each controller. Doing nothing resulted in a negative reward of -27.8 which shows the need for control to get a higher reward. With the PD controller, the average cumulative reward was 2374.6 which is significantly higher than no control. It is also evident from the trajectories in Figure 3 that the controller can consistently control the rocket's motion to land on the target pad. The PD controller successfully landed within the tolerances 95.4% of the time, demonstrating the effectiveness of PD control.

TABLE II: Landing Results

Controller	Mean Reward	Reward SEM	Total Impulse (MNs)	Landing Time (s)	Success %
Nothing	-27.8	0.812	N/A	N/A	0
PD	2374.6	5.45	4.6 ± 0.01	157 ± 0.01	95.4
DQN	5372.2	24.38	1.8 ± 0.3	60 ± 10	69.7
DAgger	2393.6	7.82	4.6 ± 0.01	155.9 ± 0.7	94.75

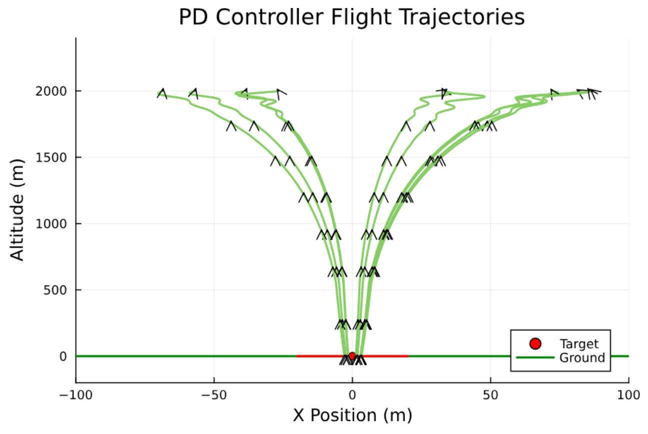

Fig. 3: PD controller sample trajectories

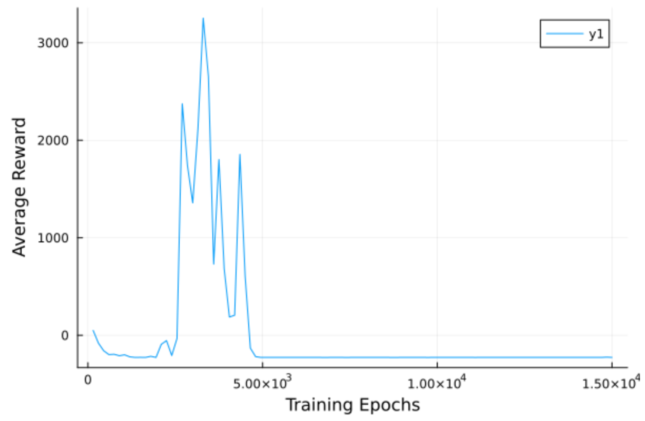

Fig. 5: DQN Learning Curve

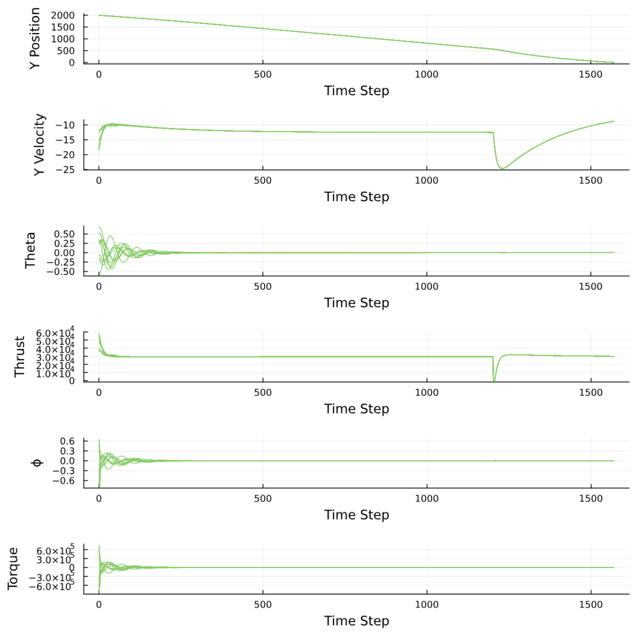

Fig. 4: PD controller states

B. DQN

Training the DQN resulted in the learning curve shown in Figure 5. This curve shows that the training starts with low average rewards, which is typical as the agent begins by randomly exploring the environment, learning from the rewards obtained from its actions. Then around epoch 3000 to 5000 the average cumulative rewards spike indicating the agent discovered particularly effective strategies. However, the well-performing network was forgotten and the average return quickly dropped. These drops suggest that the agent is exploring new strategies that do not perform as well as the previously discovered ones, or that it is failing to consistently apply the best-learned strategies due to the exploration component of the epsilon-greedy policy. The DQN model eventually converges to a non-optimal policy resulting in an average reward of around -200, despite previously reaching higher peaks in performance. Several key factors could influence this outcome, three likely causes are the utilization of the experience replay buffer, the size of the training batches, and the simple network structure. The current training method contains a relatively small amount of data which may not be sufficient to get a model to learn the complex relations required to land the booster. Increasing the size of the buffer could allow the network to remember and learn from a more extensive range of past experiences, however, this would increase the computational complexity. The other factor that could influence the training is the Q network's structure; in this training, the model only has one hidden layer which could make the network incapable of learning the complex relationships in the data. Increasing the number of hidden layers may boost performance, but like enlarging the buffer, it also increases computational complexity.

Despite the final Q network converging to a low average reward, the best Q network during training returns an average reward of 5372.2. Figures 6 and 7 show the trajectory and states over time for 10 simulations using the DQN's best policy. The trajectories show that the Q network had successfully landed the booster within the desired ranges most of the time, however, the movement is erratic and oscillatory. This is also reflected in the video simulation which showcases the erratic behavior of the DQN as compared to the smooth damping in the imitated PD controller. This type of behavior

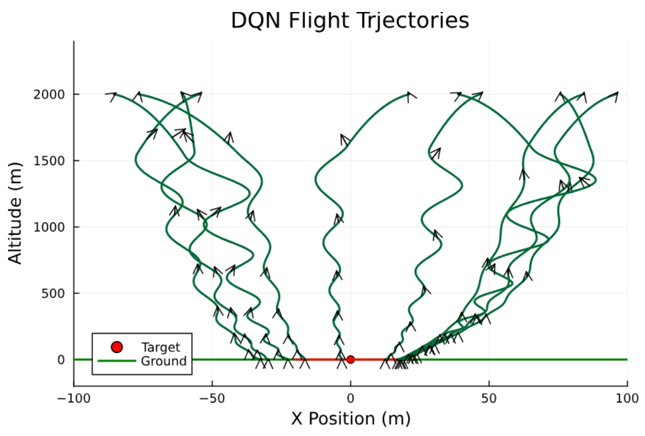

Fig. 6: DQN Sample Trajectories

is not discouraged in the reward structure which allows the DQN to achieve significantly larger rewards than the PD and DAgger controllers despite having undesirable behavior.

The Q-network achieves higher rewards when compared to other methods due to its ability to reach a terminal state faster. By including a discount factor, reaching the terminal state faster will result in larger Q values at each state. This feature incentivizes the network to prioritize quicker landings. During a DQN trajectory, the booster undergoes free fall without applying any thrust. This is immediately followed by a period of bursts of thrust to control and land the booster. By having discrete actions, the network chooses to use no thrust at certain time steps which allows for a quicker descent. However, the lack of a continuous action space causes more oscillations in its trajectory. Through the use of discrete actions, the DQN's ability to apply the exact thrust needed for smooth control is restricted. Despite the oscillatory motion, the Q-network achieves successful landings 69.7% of the time within the designated target area, while maintaining low linear and angular velocities, as illustrated in Figure 7. This demonstrates the effectiveness of the Q-network in handling the complexities of booster landing operations.

C. Behavior Cloning

Figures 8 and 9 show the results of the Behavior Cloning. Similar to the PD controller, the trained neural network can consistently land the rocket within the desired constraints. This means it successfully cloned the PD controller's behavior with comparable performance. Table II shows the differences between the PD controller and the DAgger clone. The DAgger performed slightly better but had a large standard error of the mean (SEM). However, the mean cumulative rewards for both the DAgger and PD controller are very similar indicating that the DAgger method was successful in learning the expert policy. The learning curve in Figure 10 shows the loss with the mean square error over the number of epochs. The error steadily decreases as the model is trained with batches of 5096 data points per epoch.

One of the key differences between the DAgger and DQN methods, as shown in the simulation, is the thrust level. In the

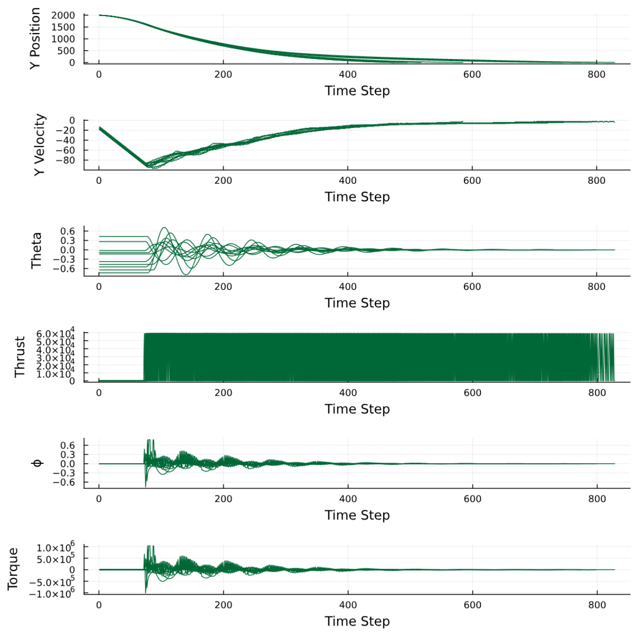

Fig. 7: DQN States

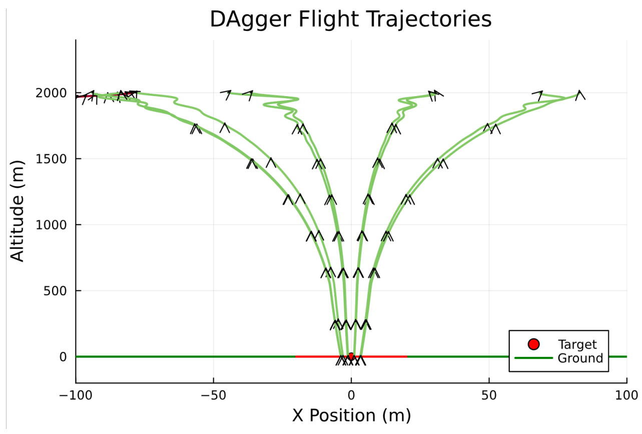

Fig. 8: DAgger Sample Trajectories

imitation network, there is a continuous value of thrust that fluctuates over time. This can also be seen in Figure 10 with the continuous curve of the thrust. Conversely, the DQN uses discrete actions which is evident in the video from the flickering thrust turning on and off in the discrete action space as outlined in Section III. The thrust plot in Figure 7 also reflects this behavior with thrust values constantly changing between discrete values of $\mathcal{A} = \{2mg, mg, 0.2mg, 0\}$. This ultimately causes a difference between the landing time and total fuel expenditure. The DQN had a significantly lower total impulse than the PD and DAgger methods as shown in Table II. This is due to the use of the discrete action space which includes using zero thrust during the landing. This shows that while the DQN has erratic movement, it uses far less fuel and has a faster landing time leading to larger rewards. Despite this, the imitation network had a significantly better policy than the DQN with a success rate of 94.75% compared to the DQN's 69.7%.

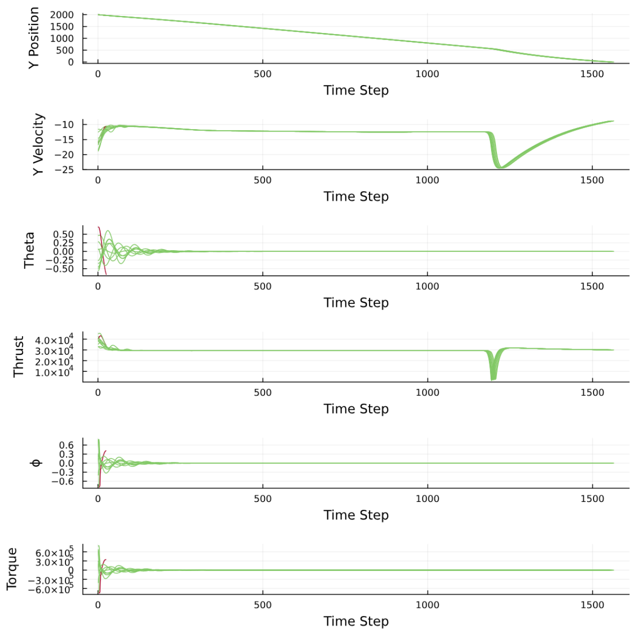

Fig. 9: DAgger States

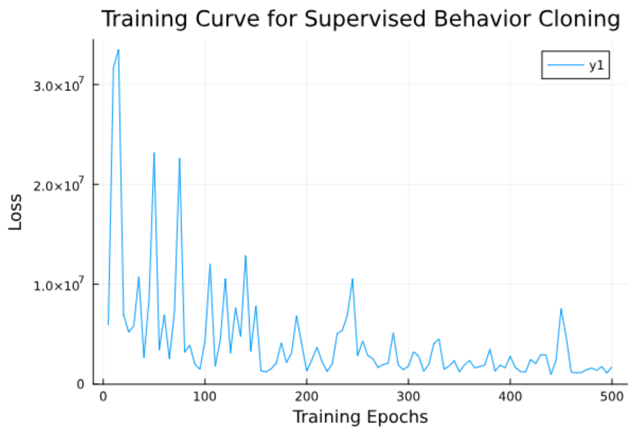

Fig. 10: DAgger Learning Curve

VI. Conclusion

In this study, different control methods for landing a rocket autonomously were explored. PD control and imitation learning yielded consistent landings as seen in Table II, whereas deep Q learning landed less consistently but accumulated more reward on average.

In terms of implementing these controllers on a physical rocket, the DQN algorithm would not be the ideal choice considering it converged on a high-risk high-reward policy that would be potentially dangerous to boarded patrons or cargo. Additionally, this policy rapidly fluctuated between high and low thrust to level out the rocket. Such actions may not be replicable on a physical rocket without precise propellant flow controls. For this algorithm to work more practically in the future, the rocket's dynamics should be modeled more accurately, and stricter constraints on parameters such as fuel and angular acceleration should be enforced. Proximal policy optimization may also yield better policies when coupled with the intrinsic vectoring angle controller and should be explored.

There are also practical limitations related to training. To train the models, large amounts of data need to be acquired which requires either an extremely accurate simulation environment or frequent testing. Using a simulation environment includes more uncertainty as the model cannot predict the real dynamics perfectly. On the other hand, frequent testing can be costly and take a lot of time. Overall, while the controllers in this work produce good results, they require large data sets to train and thus have practical limitations. To improve the models' robustness, a more accurate three-dimensional simulation environment with aerodynamic effects should be utilized.

VII. Contributions

The authors grant permission for this report to be posted publicly. All algorithms were implemented from scratch and used the CommonRLInterface and Flux libraries in Julia.

Carson Kohlbrenner: Helped with DQN tuning, parallelization, supervised learning models, and generating images/videos.

Owen Craig: Led the DQN implementation and aided in the writing of the paper.

Thomas Dunnington: Helped with the DQN implementation, the DAgger method, and writing the paper.

References

[1] F. A. S. Mikulis-Borsoi, "Landing throttleable hybrid rockets with hierarchical reinforcement learning in a simulated environment," 2020.

[2] S. Xue, H. Bai, D. Zhao, and J. Zhou, "Research on intelligent control method of launch vehicle landing based on deep reinforcement learning," *Mathematics*, vol. 11, no. 20, p. 4276, 2023.

[3] R. Ferrante, "A robust control approach for rocket landing," *Master's thesis*, 2017.

[4] X. Liu, "Fuel-optimal rocket landing with aerodynamic controls," *Journal of Guidance, Control, and Dynamics*, vol. 42, no. 1, pp. 65–77, 2019.

[5] Z. Wang and M. J. Grant, "Constrained trajectory optimization for planetary entry via sequential convex programming," *Journal of Guidance, Control, and Dynamics*, vol. 40, no. 10, pp. 2603–2615, 2017.