Applying Deep Reinforcement Learning to Control a Rocket Vehicle in Hover using Thrust Vectoring

Christopher L. Osgood
Stanford University
Stanford, CA 94305
xosgood@stanford.edu

June 1, 2021

Abstract

Deep reinforcement learning allows an agent to interact with an environment and, based off rewards at given states, learn how properly navigate within this environment. Controlling vertical takeoff and vertical landing rocket propelled vehicles requires very complex non-linear control systems, especially near hover. In this project, we use deep reinforcement learning to train a neural network to hover a rocket-propelled vehicle whic uses thrust vectoring as its control authority. We implement an environment under uncertainty, and compare the results across a multitude of learning algorithms. A proximal policy optimization (PPO) algorithm performs the best.

1 Introduction

1.1 Motivation

With a boom in the development and deployment of reusable rocket boosters as well as a resurgence in sending out planetary landers, there in an increase in the need for complex control algorithms for vertical takeoff and vertical landing rocket propelled vehicles [1]. Often, the system dynamics of these vehicles, especially near hover, are highly non-linear, so traditional control algorithms are difficult to create and require large numbers of trial and error along with high-fidelity physical models. With recent advances in machine learning, specifically reinforcement learning, we can train a deep neural network to learn the complex control actuations needed for vertical takeoff and landing vehicles. Perhaps in the near future, vehicles will be able to leverage reinforcement learning to develop more robust control algorithms. For example, upcoming missions to both the moon and Mars will require these more robust control algorithms in order to increase landing accuracy, allowing for landings in regions that were previously unreachable [2, 3, 4]. We try to demonstrate how reinforcement learning can be used in this application and compare and contrast different learning algorithms.

1.2 This Project

In this project, we create a simplified version of a rocket vehicle in hover. We use OpenAI Gym to create a custom environment, which defines the characteristics of the rocket vehicle, the state and action spaces, the physics of the world, the timestep which we step the simulation by, and state and action random noise.

Once our environment is defined, we use OpenAI's Reinforcement Learning algorithms in Stable-Baselines [8] in order to train an agent to try to hover the rocket for as long as possible. The goal of each of these algorithms is to maximize the reward it gets by hovering within the constraints for the longest possible time that it can.

1.3 Related Work

Work regarding control theory and neural networks dates at least back to 1983, when Barto, et al. applied adaptive neuronlike elements to the cart and pole problem (a cart must balance a pole above it) [5], just like in PSet 4. Much more recently, Gaudet, Linares, and Furfaro have published multiple similar papers showcasing the use of reinforcement learning in planetary landers, especially Mars landers [2, 3, 4]. Their model is much more complex, applying 6-degrees of freedom to simulate a full Mars entry, descent, and landing profile. Using a proximal policy optimization algorithm similar to OpenAI's PPO2 algorithm, they were able to achieve results comparable to non-reinforcement learning optimal control algorithms. They're algorithm is developed by themselves, and is not only well tuned in its hyperparamters, it contains extra frameworks, such as introducing a variable discount rate in an attempt to resolve a conflict between terminal and shaping rewards.

Additionally, a recent master's thesis by Kaiser demonstrates the use of reinforcement learning for the hovering and landing of a rocket in 1D and 2D. Furthermore, researchers at Stanford University have used reinforcement learning under uncertainty to train an agent that can land a lunar lander in OpenAI's "LunarLander-v2" environment [6]. These researchers solved this environment and showed the superiority of a Deep Q Learning (DQN) agent over a Sarsa agent.

2 The Environment

Our environment is an extension of OpenAI Gym's base environment class. It defines a 3-dimensional, $1m^3$ bounding box which is the world that our rocket lives in. Our rocket is approximately a $\frac{1}{100}$ model of SpaceX's Falcon 9 first stage rocket booster [9]. It is modeled as a cylinder, about half a meter tall, 4 centimeters in diameter, and has a mass of 300kg. The rocket engine is assumed to be massless and is placed directly at the bottom-center of the rocket. The moment of inertia tensor can be calculated exactly, but we approximate it to be $I = diag(6, 6, 0.05)\, kg\, m^2$. We define our inertial (global) coordinate system as the conventional Cartesian coordinates such that $\hat{x}$ points to the right, $\hat{y}$ points forwards, and $\hat{z}$ points upwards. Gravity points downwards in the $-\hat{z}$ direction and is equivalent to the gravity on Earth, although this can easily be changed to match the gravity of any planetary body. We define three angles to describe orientation: pitch, which rotates about $\hat{y}$; yaw, which rotates about $\hat{x}$; and roll, which rotates about $\hat{z}$. Since we defined our rocket as a cylinder, we abstract away the roll orientation (notice that with a single point of thrust directly at the bottom-center of the rocket, we will never be able to control roll anyways). The rocket starts off randomly perturbed at most 5cm from the center of the box and at most 1 degree from vertical.

We define the body frame coordinate system of the rocket in the same way as our inertial coordinate system, except that this frame is attached to the body of our rocket. We can then use the corresponding rotation matrix to rotate a unit vector from the z-axis to obtain a vector pointing in the orientation defined by our Euler angles; our unit vector is then: $[\sin\theta_y,\ \cos\theta_y \sin\theta_x,\ \cos\theta_x \cos\theta_y]^T$ (note that we define the order that we apply our rotation angles in as pitch then yaw).

Our state space, S, corresponds to a 5-degree of freedom object. It consists of position, velocity, 2 orientation angles (pitch and yaw), and their angular rates: $s = [x, y, z, \dot{x}, \dot{y}, \dot{z}, \theta_x, \theta_y, \omega_x, \omega_y]^T$, where θ_x (yaw) is the angle about $\hat{x}$, and so on. The action space, A, consists of three total actions: two that control the thrust vectoring, and one that controls the throttle level of the engine. The thrust vectoring is restricted to $\pm 5^\circ$ in both the x and y directions with respect to the body frame. The throttle level is restricted between 75% and 125% of the nominal thrust, where the nominal thrust perfectly counters the weight of the vehicle. Both the state and action spaces are in a continuous domain.

The simulation steps forward in time with a timestep of $0.01s$ using equations of motion derived from Newton's Second Law, $F = m\ddot{\mathbf{x}}$, and Euler's rigid body dynamics equations $M = I\ddot{\theta} + \omega \times (I \cdot \omega)$. An episode ends when the rocket goes outside of the $1m^3$ box or tilts in one direction more than 15°. The simulation will continue for a maximum of 4 seconds, or 401 time steps (where it will fail on the 401st time step no matter what). The reward function is such that at each timestep where the rocket 'survives,' a reward of 1 is given, and when the rocket fails, a reward of -100 is given. This reward function was chosen in an attempt to balance giving immediate rewards for surviving a frame and penalizing failing early. Notice that the maximum reward (without a discount factor) is 300 for a full episode.

For noise, we assume that sensor data and actuator commands have a Gaussian error distribution. We

inject noise into our state measurements ('observations'). We assume that position has a standard deviation of about $1mm$, velocity $5mm/s$, angle $0.05°$, and angular rate $0.1°/s$. We assume that our actuator commands have a standard deviation of $0.02°$ and 0.05%.

3 Methods

Open AI Stable Baselines provides multiple reinforcement learning algorithms that are compatible with Open AI Gym. For the sake of conciseness, in this paper we will focus on the implementation of the 4 most insightful algorithms, although we tested our environment using 7 of the algorithms. Since our action space is continuous, we focus on policy gradient. Remember that a policy is a mapping from a state to an action. All of these algorithms are derivations of normal policy gradient, where the goal is find the optimal policy, π, where

$$\pi^* = \arg\max_a \sum_{s' \in S} P_{s,a}(s') V^*(s'),$$

where $V^\pi(s) := \mathbb{E}_{s' \in S}[\sum R(s')]$ is the value of a state and $P_{s,a}$ are the transition probabilities. We can also define a $Q^\pi(s,a) := \mathbb{E}_{s' \in S, a' \in A}[\sum R(s')]$ as the value of a state-action, and an advantage function $A^\pi := Q^\pi - V^\pi$. Many varients of policy gradient utilize these functions, i.e. Q-Learning and Advantage learning [10, 16, 18].

Additionally, the Actor-Critic method attempts to learn the value function as well as the policy. The Critic updates the parameters of the value function (V or Q), and the Actor updates the parameters of the policy as suggested by the critic. More information can be found about policy gradient techniques, Actor-Critic, and the following algorithms here [8, 10].

But before we go into the algorithms we must define a policy network. A few options are available, including a Multi-layer Perception (MLP) network, a Convolutional Neural Network (CNN), and the ability to create a custom policy network. In this project, we choose an MLP policy network, which is suitable for our purposes (a CNN would mainly be used on input data that are images) [8].

Trust Region Policy Optimization (TRPO) [11] In an attempt to increase of stability during training, TRPO ensures that changes to the policy are not too drastic. This is enforced by applying a maximum KL divergence from the current policy to the new policy. This is represented by maximizing

$$J(\theta) = \mathbb{E}_{\pi_{old}}\left[\frac{\pi(a|s)}{\pi_{old}(a|s)} \hat{A}^{\pi_{old}}(s,a)\right]$$

with the constraint

$$\mathbb{E}_{\pi_{old}}\left[D_{KL}(\pi_{old}||\pi_{new})\right] \leq \delta,$$

where δ is some small constraint. For our application, this is very useful in ensuring that the algorithm doesn't fall into continuously trying actions that are close to those that result in failure.

Proximal Policy Optimization (PPO) [12] A simpler alternative to TRPO, PPO 'clips' the policy update such that the ratio between the new and old policies are close to 1, or $1 - \epsilon \leq \frac{\pi_{new}}{\pi_{old}} \leq 1 + \epsilon$. A downside to PPO is that it can get stuck at local optima in sparse reward settings. So, value error terms and entropy terms are added to the objective function to promote enough exploration. OpenAI's implementation of PPO is called PPO2 because it has been altered to allow for training on multiple instances of environments at once. Note that throughout this paper, if I refer to PPO or PPO2, I am refering to the same algorithm.

Actor Critic using Kronecker-Factored Trust Region (ACKTR) [13] In hopes of improving sample efficiency, ACKTR uses a *natural gradient* rather than the standard gradient. This optimizes for a new policy that is a constant KL divergence 'step size' away from the previous policy, similar to TRPO, by using a scalable 'trust region' for policy updates. The natural gradient is computed efficiently useing Kronecker-factored approximation. For our purposes, ACKTR is, in theory, a less computationally expensive Actor-Critic policy optimization algorithm that increases sample efficiently to more quickly advance towards an optimum.

Soft Actor Critic (SAC) [14] SAC is a derivative of Soft Q-Learning and implements double-Q-learning (from TD3 [15]) and entropy regularization to balance the tradeoff between exploration and exploitation. SAC is similar to the DDPG algorithm [16] in that it is a cross between stochastic policy optimization and deep Q-learning [16, 8].

All algorithms were run with the default provided hyperparameters. Although reinforcement learning algorithms are often quite sensitive to changes in hyperparameters, in the interest of time, default hyperparameters were taken to be good enough and differences in algorithms were assumed to outweigh any differences in tuning hyperparameters. Although only the most insightful results are reported here, all algorithms were tested with multiple random seeds and for different amounts of timesteps to ensure a sufficient level of thoroughness. For each training session, we save the final model as well as the best model throughout the whole session.

4 Results

4.1 Training (Learning Curves)

Below are the learning curves of each algorithm run for 1 million timesteps. We should expect that TRPO is

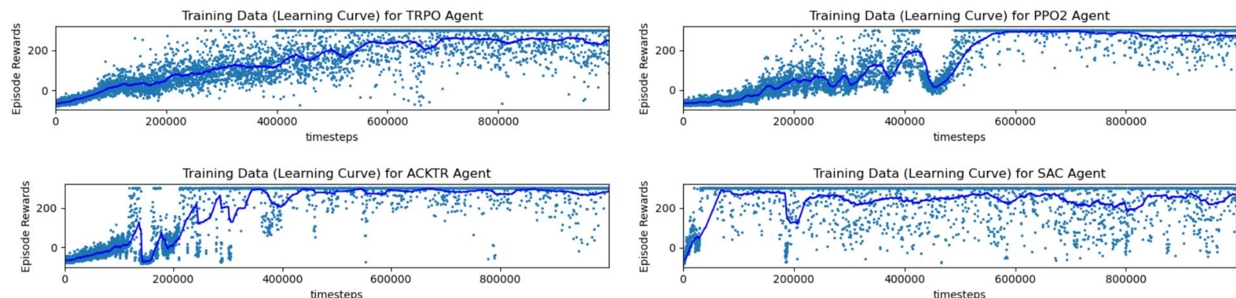

Figure 1: Plot of mean episode reward over the previous 100 episodes (evaluated every 1000 timesteps) vs. timestep of each algorithm during training for 1 million total timesteps. Random seed is 1.

the worst performing out of TRPO, PPO, and ACKTR, since PPO and ACKTR are essentially upgraded versions of TRPO. This is eactly what we see. Notice that TRPO doesn't have any large dips in episode reward like the other algorithms. That is likely due to the guaranteed monotonic improvement of the policy in the TRPO algorithm [11]. PPO has a few large dips before it reaches the global optimum, but notice that when it does dip, it does so relatively slowly (it doesn't spike) due to the strong policy trust region constraint. Also notice that after PPO stays at a good optimum for a while, it starts to dip in performance. This is likely due to the entropy term making it explore more because it has no way of knowing if it is at a local or global optimum. Perhaps this is the reinforcement learning analogue of overfit. We can see ACKTR doing a similar thing. The ACKTR algorithm acclerates to the optimum quite quickly and is able to stay there quite well. This is consistent with its heightened sampling efficiency and trust region. SAC gets to the optimum the quickest, in less than 100000 timesteps, but holds that optimum the worst. This is likely due to the entropy term that wants to make SAC explore more. Likely the hyperparameters that govern the amount of exploration for PPO, ACKTR, and SAC need to be tuned.

4.2 Testing (Performance)

To establish a baseline, a random agent was constructed and run through the same tests as the rest of the algorithms. The random agent gets a total reward (score) of about -65 per episode, with a standard deviation of about 6.

Table 1: Score of final and best model of each algorithm with three different seeds, and their averages and standard deviations.

		Score (without discount factor, max 300)							
		Final Model				Best Model			
	Seed	0	1	2	Avg	0	1	2	Avg
TRPO	mean	268.655	262.604	285.483	**272.247**	286.174	260.985	272.635	**273.265**
	std dev	63.521	59.066	38.519	**53.702**	43.713	55.317	57.470	**52.167**
PPO2	mean	282.665	275.581	287.309	**281.852**	296.788	294.526	297.506	**296.273**
	std dev	56.324	45.404	35.248	**45.659**	13.212	19.453	12.135	**14.933**
ACKTR	mean	265.764	295.939	297.91	**286.538**	293.702	295.060	296.155	**294.972**
	std dev	66.951	16.963	11.432	**31.782**	22.380	17.022	16.981	**18.794**
SAC	mean	244.855	285.294	233.356	**254.502**	285.049	284.712	293.017	**287.593**
	std dev	76.643	35.740	79.583	**63.989**	36.467	37.021	26.362	**33.283**

The algorithms we implemented all reach an average score greater than 250. But, being better than a random actor isn't all that hard. From Table 1 below, notice that PPO and ACKTR both have very good performance. Their best models get very close to maxing out the possible score on every trial. Notice that while PPO performs better amongst the best models, ACKTR performs the best among the final models. This further showcases how ACKTR reaches an optimum quicker, but chooses to explore a bit more. TRPO's final and best models are quite consistent with each other, which is indicative of its near-guaranteed monotonic policy improvement. As the most explorative algorithm, SAC shows good accuracy from its best model, while its final model can have a high degree of variability.

Regardless of algorithm, the randomness can make quite a drastic difference. PPO2 is the most stable with regards to this, ACKTR is mediocre, and TRPO and SAC are the least stable.

5 Conclusion

We successfully demonstrated that reinforcement learning algorithms can be used to develop control laws for rocket vehicles in hover in 3D space, and can withstand uncertainty in both the action and state spaces. We tested 4 different algorithms, each a derivative of policy gradient, and one tied to deep Q-learning. We identified major characteristics and investigated the advantages and disadvantages of each. TRPO is the most basic and, while it provides the worst performance, it does provide the highest confidence in monotonic improvement. Additionally, it provides a good baseline to compare the other algorithms to. PPO and ACKTR provide the highest accuracy. PPO is the most resilient to randomness. ACKTR reaches an optimum quicker, but PPO holds its optimum better. SAC reaches near-optimum in the fewest timesteps, but is the most computationally expensive by far. Additionally, SAC explores the most, and has the most trouble staying near its optimum.

This project has lots of potential for future improvements. Hyperparameter tuning, increasing the state and action spaces, state and action space normalization, increasing the scale and fidelity of the environment, adding more noise, and trying more algorithms are just a few of them. Algorithms such as PPO and ACKTR are very promising, but perhaps using an algorithm that can take advantage of an already known 'expert' trajectory, such as GAIL [17], could provide even more improvements. Perhaps a combination of multiple algorithms could be implemented, tested, and refined. Additionally, we would like to further develop the environment to be able to take in control inputs (a trajectory for the rocket to fly), and compare our results with real-world non-reinforcement learning control algorithms.

6 Contributions

I did everything myself.

References

[1] T Kaiser, "Optimal Control of Liquid Propellant Rocket Engines for Landing of Reusable Stages using Deep Reinforcement Learning," Julius-Maximilians University of Wurzburg, Institute for Computer Science VII Robotics and Telematics, Master's Thesis, Feb 2021.

[2] B Gaudet, R Linares, & R Furfaro, "Integrated Guidance and Control for Pinpoint Mars Landing using Reinforcement Learning," *AIAA*, 2018.

[3] B Gaudet & R Furfaro, "Adaptive Pinpoint and Fuel Efficient Mars Landing Using Reinforcement Learning," *IEEE/CAA Journal of Automatica Sinica*, Vol. 1, N. 4, 2014.

[4] B Gaudet, R Linares, & R Furfaro, "Deep reinforcement learning for six degree-of-freedom planetary landing," *Advances in Space Research*, 2020.

[5] AG Barto, RS Sutton, & CW Anderson, "Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problem," *IEEE Transactions on Systems, Man, and Cybernetics*, 1983.

[6] S Gadgil, Y Xin, C Xu, "Solving The Lunar Lander Problem under Uncertainty using Reinforcement Learning," *IEEE Southeast Conference*, 2020.

[7] N Renotte, youtube.com/nicholasrenotte, github.com/nicknochnack.

[8] Open AI, gym.openai.com, stable-baselines.readthedocs.io/en/master, spinningup.openai.com/en/latest/spinningup/rl_intro2.html.

[9] SpaceX, Falcon 9, spacex.com/vehicles/falcon-9/.

[10] L Weng, "Policy Gradient Algorithms," GitHub Blog Post, 2018, lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html.

[11] J Schulman, et al., "Trust Region Policy Optimization," University of California, Berkeley, Department of Electrical Engineering and Computer Sciences, 2017, arxiv.org/abs/1502.05477.

[12] J Schulman, et al., "Proximal Policy Optimization Algorithms," OpenAI, 2017, arxiv.org/abs/1707.06347.

[13] Y Wu, et al., "Scalable Trust-Region Method for Deep Reinforcement Learning using Kronecker-Factored Approximation," University of Toronto, 2017, arxiv.org/abs/1708.05144.

[14] T Haarnoja, et al., "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," Berkeley Artificial Intelligence Research, University of California, Berkeley, 2018, arxiv.org/abs/1801.01290.

[15] S Fujimoto, et al., "Addressing Function Approximation Error in Actor-Critic Methods," *Proceedings of the 35 the International Conference on Machine Learning*, 2018, arxiv.org/pdf/1802.09477.pdf.

[16] T Lillicrap, et al., "Continuous Control with Deep Reinforcement Learning," Google Deepmind, 2019, arxiv.org/abs/1509.02971.

[17] J Ho, S Ermon, "Generative Adversarial Imitation Learning," Stanford University, 2016, arxiv.org/abs/1606.03476.

[18] T Lillicrap, et al., "Asynchronous Methods for Deep Reinforcement Learning," *Proceedings of the 33rd International Conference on Machine Learning*, Google Deepmind and Montreal Institute for Learning Algorithms, University of Montreal, 2016, arxiv.org/abs/1602.01783.