Landing Rockets with Reinforcement Learning

Govind Chari*
University of Washington, Seattle, WA, 98195

In this paper I use a linear policy to solve the powered descent guidance problem colloquially referred to as the rocket landing problem. This paper is written as the final report for CSE 579: Intelligent Control.

I. Nomenclature

$\boldsymbol{r}$	=	position vector of rocket
$\boldsymbol{v}$	=	velocity vector of rocket
θ	=	attitude of rocket
ω	=	angular velocity vector of rocket
T	=	engine thrust
θ_g	=	gimbal angle
$\boldsymbol{\phi}$	=	policy parameters

II. Introduction

The powered descent guidance problem (PDG) is concerned with finding an optimal control law that allows a vehicle to use its propulsion system to decelerate itself from some initial state to perform a soft landing at some specified target on the ground while consuming as little fuel as possible. This problem first came to prominence with the landing of the Apollo Lunar Module and still has importance today for the landing of rovers on the surface of Mars by NASA's Jet Propulsion Lab and the landing of rocket boosters on earth by private companies such as SpaceX and Blue Origin.

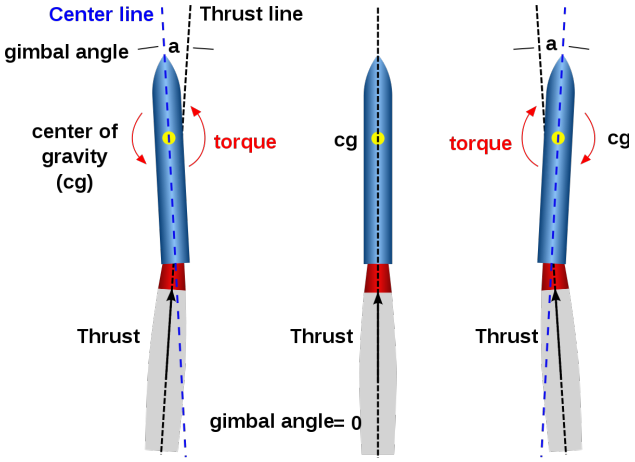
The PDG problem can be solved using model-based methods where the problem is formulated as a convex optimization problem and the solution to this problem is the optimal policy and optimal state trajectory [1] [2]. However, these methods assume you have a deterministic dynamics model for your system. In this work, I used RL techniques on a simplified 2D planar version of this problem with the assumption that the agent does not have access to the transition model.

There is already some existing literature that applies RL to the PDG problem such as [3] which uses a policy gradient method to solve the full 6DOF PDG problem with full translational and attitude dynamics as well as safety constraints such as pointing constraints and glideslope constraints. Pointing constraints are in place to ensure that the rocket remains within some angle of vertical, and the glideslope constraint ensures that the rocket remains within some cone whose apex is at the landing site and opens upwards. In this work they use a deep neural network for the policy. Given that the problem I am attempting to solve does not have these constraints and is a 3DOF problem I wanted to see if I could solve this problem using only a linear policy with intelligently chosen features and policy architecture.

III. Problem Setup

To test my policy I used an existing gym environment for 2D planar rocket landing [4]. This environment has the states: position, velocity, attitude (orientation), and angular rate. It also has the actions of engine thrust and engine gimbal angle. Gimbaling is a common term in aerospace engineering and it refers to when the engine of the rocket can be tilted to alter the direction of the thrust vector. Gimabling is used to provide attitude control for the rocket. The diagram below is helpful for visualizing gimbaling.

*PhD Student, AeroAstro



We want the rocket to land as close to the landing location as possible. Additionally, when it lands, we want to minimize the speed at landing and want the rocket to be as close to vertical as possible. Thus, the reward takes the following form where the T subscript indicates the terminal timestep and we assume we want to land at the origin.

$$reward = -(\|\boldsymbol{r}_T\| + \|\boldsymbol{v}_T\| + |\theta_T|) \tag{1}$$

IV. Deep Q-Network

This AI gym allows you to choose between a discrete action space or continuous action space. Before I implemented the linear policy, I decided to use a Deep Q-Network (DQN) to solve this problem. I wanted to use a DQN because I thought it was an interesting algorithm and I wanted to implement it to better learn how it works.

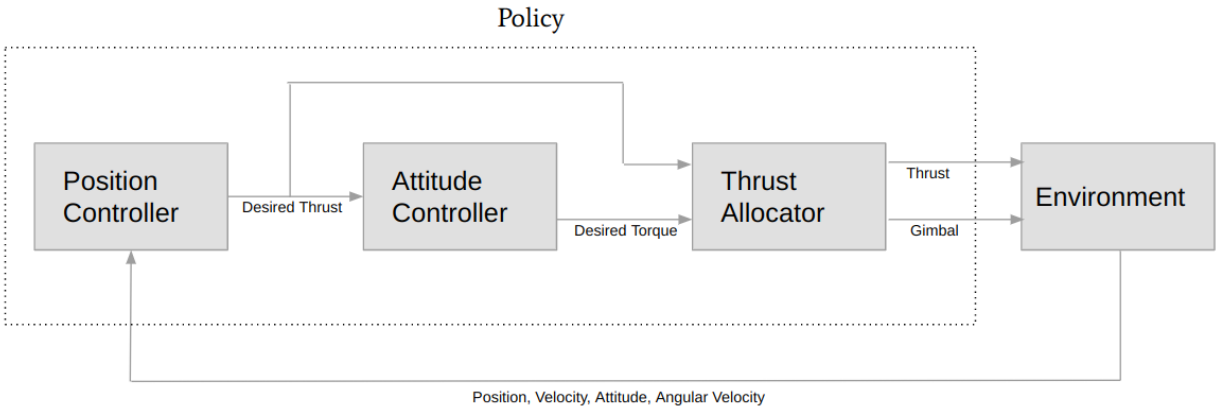
Deep Q-Learning is an extension of traditional Q-Learning, but uses a neural network to parameterize the Q-function rather than using a Q-table. A replay buffer and a target Q-network are used to reduce the correlation in Q-target updates. The former prevents overfitting to a specific episode and the latter reduces the correlation between the evaluation of the target Q-function and the update of our Q-network. Since Q-Learning is an online learning algorithm an exploration strategy is needed to adequately explore the state-action space. For this I used the typical ϵ-greedy strategy, where a random action is taken with some probability and the best action according to the current Q-function is taken the rest of the time. The Q-network took in state of the game as an input and returned the Q-values for that state and each action. Then the action with the maximum Q-value was selected.

I tested this approach on the Lunar Landing OpenAI gym and it worked very well. However, when I tried to apply it to the rocket landing gym I was using for this project, it was not able to learn at all. I spent many hours trying to figure out why it was not working, but I was running out of time for the project, so I decided to use a simpler approach that would be much easier to debug, such as a linear policy which is closely related to a cascaded PID controller.

V. Linear Policy

A. Structure

The policy is a function that maps position, velocity, attitude, and angular rate to thrust and gimbal angle. In order to design a policy structure, I used ideas from cascaded PID control where a position controller generates a desired force which is fed to an attitude controller to get a desired torque, then the desired force and torque are fed to an allocator which computes a throttle and gimbal angle. A diagram of this is shown below:



Based on the current position and velocity, a desired thrust vector is computed. The attitude controller then computes a desired torque in order to point the rocket in the direction of the desired thrust vector based on the current attitude and angular rate. Then the desired thrust vector and desired torque is fed to the thrust allocator to compute thrust and gimbal angle. The thrust is simply the norm of the desired thrust vector and the gimbal angle is proportional to the desired torque normalized by thrust. This makes sense since torque is moment arm multipled by force, and the thrust is the force and gimbal angle is a proportional to moment arm (for small angle). The full policy is shown below.

Algorithm 1: Policy

Data: $\boldsymbol{r}, \boldsymbol{v}, \theta, \omega, \boldsymbol{\phi}$
Result: T, θ_g
$\boldsymbol{F}_{des}[0] \leftarrow -\boldsymbol{\phi}[0]\boldsymbol{r}[0] - \boldsymbol{\phi}[1]\boldsymbol{v}[0];$
$\boldsymbol{F}_{des}[1] \leftarrow -\boldsymbol{\phi}[2]\boldsymbol{r}[1] - \boldsymbol{\phi}[3]\boldsymbol{v}[1] + \boldsymbol{\phi}[4];$
$\theta_{des} = -sgn(\boldsymbol{F}_{des}[0])arccos(\boldsymbol{F}_{des}[1]/\|\boldsymbol{F}_{des}\|);$
$\tau_{des} - \boldsymbol{\phi}[5](\theta_{des} - \theta) - \boldsymbol{\phi}[6]\omega;$
$\theta_g = -\boldsymbol{\phi}[7](\tau_{des}/\|\boldsymbol{F}_{des}\|);$

In the above algorithm the square bracket operator refers to indexing a vector. There are a few notable things in the algorithm. Firstly, there is a feed-forward term in the desired y-force but not desired x-force. This is to account for the fact that there is a constant force in the y-direction (gravity) that needs to be compensated for. Additionally, there is a sgn term in the calculation of θ_{des}. This is to ensure that we get the right sign for the desired attitude of the rocket.

B. Optimization

Now that we have the policy structure, we have to get the optimal set of parameters, $\boldsymbol{\phi}^*$ which maximize our reward, meaning we achieve an accurate, upright, soft landing. Since the policy space is $\mathbb{R}^8$ this is small enough that we can use derivative-free optimization techniques to get $\boldsymbol{\phi}^*$. To do this, we need to have an objective function which will be the average reward over a set of rollouts of the policy π_ϕ. To carry out the policy optimization I used the minimize function in the scipy package which implemented Nelder-Mead.

Nelder-Mead is an derivative-free optimization technique, meaning it finds the minimum of a function using only function evaluations. In this case, the objective function is the negative expected reward of running the policy with fixed parameters over many episodes. Nelder-Mead creates an $N + 1$ dimensional simplex in $\mathbb{R}^N$, where N is the dimension of the search space, and changes this simplex according to some predetermined rules. At convergence, each of the vertices of the simplex are very close to each other and surround the local optimal solution.

VI. Results

In order for the policy optimization to work well, a good initial guess must be supplied for the parameters $\boldsymbol{\phi}$, since the optimization landscape is nonconvex. To do this, I played around with the parameters and used intuition from classical control until I was able to successfully get the rocket to land near the desired location. I then fed this initial

guess into the optimizer and was able to get even better parameters that resulted in softer landings closer to the desired landing location. A video of a landing using these optimal parameters can be found here. As you can see, the rocket was able to land accurately, upright, and softly.

VII. Conclusion

After submitting this project, I want to revisit my DQN implementation and try to figure out why it was failing. Additionally, I want to try some other approaches for example PPO and compare its performance to my linear controller.

One of the biggest takeaways from this project is that you should not use model-free RL methods unless you absolutely need to. If you have a physics-based problem use your domain knowledge of the problem to set up a good policy structure and then use optimization to find the right parameter set. When I tried debugging my DQN implementation there was no clear way for me to see why it was failing, but one problem is that with the DQN method, the agent has absolutely no knowledge about the environment and must figure it all out for itself which is a waste of resources if the designer of the agent can use their knowledge to point the agent in the right direction.

Specifically looking at the full 6DOF powered descent guidance problem, I do not think anyone should ever use model-free RL methods such as policy gradient to solve the problem. Even if you can get the agent to land the rocket well enough, you have no guarantees that the agent might mess up if it encounters a part of the state space that it has not encountered before. On the other hand, optimization-based methods such as [1] have guarantees that if the optimization problem is feasible, the optimal can be found accurately in a bounded amount of time.

This all goes to say that RL should be a last resort only for cases where classical methods which have theoretical guarantees fail such as in Go or in predicting protein structure. After all it is the job of an engineer to pick the simplest possible solution that works to solve a problem.

The code for this project can be found here.

References

[1] Acikmese, B., and Ploen, S. R., "Convex Programming Approach to Powered Descent Guidance for Mars Landing," *Journal of Guidance, Control, and Dynamics*, Vol. 30, No. 5, 2007, pp. 1353–1366. https://doi.org/10.2514/1.27553, URL https://doi.org/10.2514/1.27553.

[2] Szmuk, M., Reynolds, T. P., and Açıkmeşe, B., "Successive Convexification for Real-Time Six-Degree-of-Freedom Powered Descent Guidance with State-Triggered Constraints," *Journal of Guidance, Control, and Dynamics*, Vol. 43, No. 8, 2020, p. 1399–1413. https://doi.org/10.2514/1.g004549, URL http://dx.doi.org/10.2514/1.G004549.

[3] Gaudet, B., Linares, R., and Furfaro, R., "Deep Reinforcement Learning for Six Degree-of-Freedom Planetary Landing," *Advances in Space Research*, Vol. 65, 2020. https://doi.org/10.1016/j.asr.2019.12.030.

[4] Niederberger, S., "gym-rocketlander," https://github.com/EmbersArc/gym-rocketlander, 2021.