Lunar Lander Optimal Control

Kaela Nelson*, Sydney Thompson*, Casey Wahl*, Joseph Ward*

December 15, 2018

Abstract

Our project aims to successfully navigate a lunar lander to a landing pad using control theory taught in class. We first derived the physics behind the system of the lunar lander. We then solved for state equations using a nonlinear approach to the LQR method that resulted in a state dependent Riccati equation. We found that solving this problem turned out to be more difficult than expected due to the noisy environment that OpenAI Gym provided and other errors unknown to us. As a result, we simplified the problem into a BVP, and successfully landed the lander.

1 Introduction

Lunar Lander is a popular arcade game. The point of this game is to navigate a small spaceship to the surface of the moon without crashing. Variations of the game include various sensitivities to the velocity of the lander at impact, inclusion and exclusion of angular velocity of the lander, position of the engines, and ability to rotate the lander versus simply turning on and off the engines.

In our version of the game from OpenAI Gym [1], our lander has three engines - a main engine on the bottom of the ship, and two side engines. These engines can be turned on between 50 and 100% of their full power, or can be turned off completely. There is a single landing pad marked with flags (centered at $(0, 0)$) where we must land the spacecraft. The exact position of the lander on the pad is irrelevant to us — we only care about landing the spaceship on the landing pad softly enough that the ship will not break upon impact.

At each time step in the game, we are provided with a state vector of the following form: $\mathbf{x} = [x, y, \dot{x}, \dot{y}, \theta, \dot{\theta}]$. We are also provided with constants governing the behavior of the environment and the ship, such as the exact position and power output of each engine, the force of gravity, and the torque of the springs on the lander's legs (which governs with how much force we can hit the ground, and whether we will be pushed back into the air).

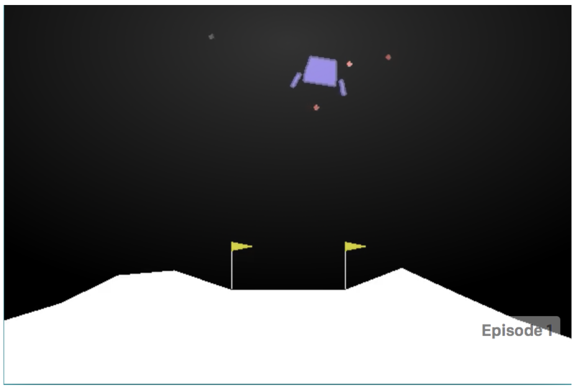

Figure 1: A screen shot from the Lunar Lander game.

In our problem, we have reformatted the lander to be a 14x24 meter rectangle, and have reduced the mass and torque of the lander's legs so that they will not impact the flight of the lander. We also assumed that the lander's center of mass was in the center of the ship, since we could not find any information indicating uneven density throughout the ship.

2 Algorithm

2.1 Physics and State Equations

Because we have three actions - activating the main, left, or right engines - we must consider the effect of each on the state of the lander. In the state vector, x and y are the horizontal and vertical coordinates of the lander relative to the center of the landing pad, and θ is the angle of the lander with respect to vertical. When the lander tilts to the left, θ is positive; when it tilts to the right, θ is negative. In Figure 2, we can see and example of the lander rotated to a positive value of θ.

Because of the positioning of the side engines, activating a side engine will not only change the x and y velocities of the lander, it will also change the angular velocity, $\dot{\theta}$. Thus, we must decompose the force applied to the lander by a side engine into the force that will contribute to changing the angular acceleration, and the force that will contribute to the positional acceleration.

In Figure 2 we see the force diagram for this event.

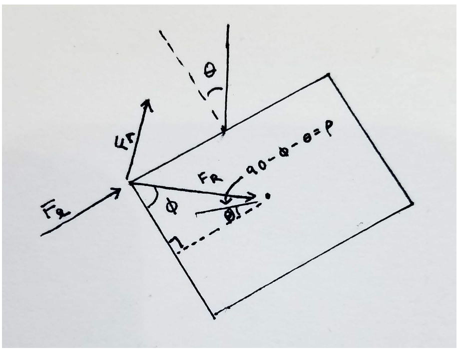

Figure 2: Force diagram for firing the left side engine.

Notice that we need to decompose F_l – the force from the left engine – into two components: F_A and F_R. F_R is the force that acts in a line directly to the center of mass. Correspondingly, F_A is the force responsible for the angular acceleration and is acting perpendicular to F_R. ϕ denotes the angle between the side of the lander and the force applied towards the center of mass.

Thus, we have $F_A = F\cos(\phi)$ and $F_R = F\sin(\phi)$. Notice in the diagram that we define a new angle, ρ, which describes the angle between the direction of F_R and the horizontal axis. Here, $\rho = 90 - \phi - \theta$. Then, the force applied only from firing the left engine can be described as follows:

$$F_x = F_R\cos(\rho) = F_R\cos(90 - (\theta + \phi)) = F_R\sin(\theta + \phi) = F_l\sin(\phi)\sin(\theta + \phi)$$

and

$$F_y = F_R\cos(\rho) = F_R\cos(90 - (\theta + \phi)) = F_R\cos(\theta + \phi) = F_l\sin(\phi)\cos(\theta + \phi).$$

As a result, when only the left engine fires with force F_l, we have the following state equations:

$$\ddot{x} = \frac{F_l}{M}\sin(\phi)\sin(\phi)\cos(\theta) + \frac{F_l}{M}\sin(\phi)\cos(\phi)\sin(\theta),$$

$$\ddot{y} = \frac{F_l}{M}\sin(\phi)\cos(\phi)\cos(\theta) - \frac{F_l}{M}\sin(\phi)\sin(\phi)\sin(\theta) - g.$$

When firing the right engine, the results are almost identical. The only difference is that instead of ρ being defined as $\rho = 90 - \phi - \theta$, now $\rho = 90 - \phi + \theta$ because we are now considering F_R as a component of a force from the right engine. F_r will now be negative since it is in the opposite direction from F_l, but the equations already account for this. Thus, the resulting force equations are now functions of $\phi - \theta$. Overall, this is equivalent to replacing θ with $-\theta$ in the state equations that describe the effect of activating the right engine. These state equations are as follows:

$$\ddot{x} = \frac{F_r}{M}\sin(\phi)\sin(\phi)\cos(\theta) - \frac{F_r}{M}\sin(\phi)\cos(\phi)\sin(\theta),$$

$$\ddot{y} = \frac{F_r}{M}\sin(\phi)\cos(\phi)\cos(\theta) + \frac{F_r}{M}\sin(\phi)\sin(\phi)\sin(\theta) - g,$$

where F_r represents the right force.

Now we consider the angular acceleration when we fire the left or the right engine. For simplicity's sake, we only consider F_l here and will address F_r later. Recall that above we de-constructed F_l into two orthogonal components: F_A and F_R. We have already considered the effects of F_R, which is the force component that affects the position of the lander. Now, we consider the effects of F_A.

A commonly used formula for Newtonian mechanics is $\ddot{\theta} = \frac{\sum_i \tau_i}{I}$ where each τ_i is a torque applied to the lander, and I is the moment of inertia for the lander. Since there is only one torque applied to the lander, this simplifies to $\ddot{\theta} = \frac{\tau}{I}$. The moment of inertia for a rectangle is described as $I = \frac{M}{12}(h^2 + w^2)$ where M is the mass of the lander (we used a density of 5 and multiplied this density by the area of the lander to construct the mass). For simplicity, we made the legs of the lander be massless in our simulation so that they would not affect the moment of inertia.

The torque applied to an object is the distance to the center of mass of the object multiplied by the force perpendicular to the line to the center of mass. F_A describes this force, and the distance is described by $d = \sqrt{h^2 + w^2}$ where h is the height of the lander and w is the width of the lander - in our experiments, 14 and 24, respectively.

With these two variables defined, we can now define the angular acceleration of the lander when we fire the left engine as follows:

$$\ddot{\theta}_l = \frac{F_{\theta_l} d}{I} = \frac{F_l \cos(\phi)\frac{1}{2}\sqrt{h^2 + w^2}}{\frac{M}{12}(h^2 + w^2)} = \frac{6F_l \cos(\phi)}{M\sqrt{h^2 + w^2}}$$

.

Considering the effects of firing the engine on the right side of the lander, we note that the only difference is from the relative magnitude of the force. Since F_r is from the opposite direction of F_l, F_R is negative. Thus,

$$\ddot{\theta}_r = \frac{F_{\theta_r} d}{I} = \frac{F_r \cos(\phi)\frac{1}{2}\sqrt{h^2 + w^2}}{\frac{M}{12}(h^2 + w^2)} = \frac{6F_r \cos(\phi)}{M\sqrt{h^2 + w^2}}$$

Now we consider the effects of firing the bottom engine. If this engine fires with a force of F_m, then it provides a force of $F_m \sin(\theta)$ in the x direction and $F_m \cos(\theta)$ in the y direction. Thus, the state equations for the whole system (including all forces) are given by:

$$\begin{aligned}
\ddot{x} = &\frac{1}{M} F_l \sin^2(\phi)\cos(\theta) + \frac{1}{M} F_l \sin(\phi)\cos(\phi)\sin(\theta) \\
&+ \frac{1}{M} F_r \sin^2(\phi)\cos(\theta) - \frac{1}{M} F_r \sin(\phi)\cos(\phi)\sin(\theta) \\
&- \frac{1}{M} F_m \theta,
\end{aligned}$$

$$\begin{aligned}
\ddot{y} = &\frac{1}{M} F_l \sin(\phi)\cos(\phi)\cos(\theta) - \frac{1}{M} F_l \sin^2(\phi)\sin(\theta) \\
&+ \frac{1}{M} F_r \sin(\phi)\cos(\phi)\cos(\phi) + \frac{1}{M} F_r \sin^2(\phi)\sin(\theta) \\
&+ \frac{1}{M} F_m - g,
\end{aligned}$$

$$\ddot{\theta} = \frac{6F_l \cos(\phi)}{M\sqrt{h^2 + w^2}} + \frac{6F_r \cos(\phi)}{M\sqrt{h^2 + w^2}}.$$

We made the simplifying assumption θ is close to 0. This is because the unstable steady state occurs at $\theta = 0$, representing that the lunar lander is parallel to the ground, dropping straight down. Thus, we can linearize about 0 for θ. So, $\sin(\theta) \to \theta$, and $\cos(\theta) \to 1$. This gives the approximate state equations:

$$
\begin{aligned}
\ddot{x} \approx & \left(\frac{1}{M}F_l \sin^2(\phi)\right) + \left(\frac{1}{M}F_l \sin(\phi)\cos(\phi)\right)\theta \\
& + \left(\frac{1}{M}F_r \sin^2(\phi)\right) - \left(\frac{1}{M}F_r \sin(\phi)\cos(\phi)\right)\theta \\
& - \left(\frac{1}{M}F_m \theta\right)
\end{aligned}
$$

$$
\begin{aligned}
\ddot{y} \approx & \left(\frac{1}{M}F_l \sin(\phi)\cos(\phi)\right) + \left(-\frac{1}{M}F_l \sin^2(\phi)\right)\theta \\
& + \left(\frac{1}{M}F_r \sin(\phi)\cos(\phi)\right) + \left(\frac{1}{M}F_r \sin^2(\phi)\right)\theta \\
& + \left(\frac{1}{M}F_m\right) - g
\end{aligned}
$$

$$
\ddot{\theta} \approx \frac{6F_l\cos(\phi)}{M\sqrt{h^2+w^2}} + \frac{6F_l\cos(\phi)}{M\sqrt{h^2+w^2}}
$$

2.2 The LQR Method

Note that linearizing our state equations gives us a non linear system represented in the form

$$
\begin{aligned}
\dot{\mathbf{x}} &= f(\mathbf{x}) + B(\mathbf{x})\mathbf{u}(t) \\
\mathbf{x}(0) &= \mathbf{x}_0.
\end{aligned}
$$

Note that we have a component of $\ddot{y}$ that does not depend on any component of $\mathbf{x}$; it is g, the gravitational constant used by the Lunar Lander game. Thus, to account for this, we need to extend $\mathbf{x}$ to include g. Thus, we create a new variable, $\mathbf{z}$, where

$$
\mathbf{z} = \begin{bmatrix} x \\ y \\ \dot{x} \\ \dot{y} \\ \theta \\ \dot{\theta} \\ -1 \end{bmatrix}.
$$

The -1 in the last row will be multiplied by g to account for the effect of gravity. Now, we can define

$$
\begin{aligned}
\dot{\mathbf{z}} &= f(\mathbf{z}) + B(\mathbf{z})\mathbf{u}(t) \\
\mathbf{z}(0) &= \mathbf{z}_0
\end{aligned}
$$

.

Then by the results on nonlinear LQR found in [3], extended linearization allows us to write

$$
f(\mathbf{z}) = A(\mathbf{z})\mathbf{z}.
$$

So our problem can be formatted as

$$
\dot{\mathbf{z}} = A(\mathbf{z})\mathbf{z} + B(\mathbf{z})\mathbf{u}(t),
$$

at each state $\mathbf{z}$, where

$$B(\mathbf{z}) = \begin{bmatrix} 0 & 0 & 0 \\ \frac{-P_m\theta}{M} & \frac{P_s\theta}{M}(\sin(\phi)^2 + \theta\sin(\phi)\cos(\phi)) & \frac{P_s\theta}{M}(\sin(\phi)^2 - \theta\sin(\phi)\cos(\phi)) \\ 0 & 0 & 0 \\ \frac{P_m}{M} & \frac{P_s\theta}{M}(\sin(\phi)\cos(\phi) - \theta\sin(\phi)^2) & \frac{P_s\theta}{M}(\sin(\phi)cos(\phi) + \theta\sin(\phi)^2) \\ 0 & 0 & 0 \\ 0 & \frac{6P_s\cos(\phi)}{M\sqrt{h^2+w^2}} & \frac{6P_s\cos(\phi)}{M\sqrt{h^2+w^2}} \\ 0 & 0 & 0 \end{bmatrix}, \mathbf{u}(t) = \begin{bmatrix} u_M \\ u_L \\ u_R \end{bmatrix},$$

where u_M is the main engine control, u_L is the left engine control, and u_R is the right engine control, and

$$A(\mathbf{z}) = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & g \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

By the state dependent Riccati equation (SDRE) method formulation [3], we minimize the following integral:

$$J[\mathbf{z}_0, \mathbf{u}(t)] = \int_0^\infty [\mathbf{z}^T Q(\mathbf{z})\mathbf{z} + \mathbf{u}^T R(\mathbf{z})\mathbf{u}]\ dt.$$

Notice that $\mathbf{u}$ and $\dot{\mathbf{z}}$ can be approximated with

$$\dot{\tilde{\mathbf{z}}} = (A - BR^{-1}B^T P)\tilde{\mathbf{z}}$$

$$\tilde{\mathbf{u}} = -R^{-1}B^T P\tilde{\mathbf{z}},$$

where P is a matrix satisfying the Riccati equation

$$\frac{dP(\mathbf{z},t)}{dt} = P(\mathbf{z})A(\mathbf{z}) + A^T(\mathbf{z})P(\mathbf{z}) + Q(\mathbf{z}) - P(\mathbf{z})B(\mathbf{z})R^{-1}(\mathbf{z})B^T(\mathbf{z})P(\mathbf{z}).$$

.

Note that P must satisfy the Riccati equation because of the Hamiltonian of $J[\mathbf{u}]$.

Since we are considering an infinite time horizon problem, we have $\dot{P} = 0$. Thus, P is a constant matrix, and we do not have to include a time dependency. Hence, we need only satisfy the equation

$$0 = P(\mathbf{z})A(\mathbf{z}) + A^T(\mathbf{z})P(\mathbf{z}) + Q(\mathbf{z}) - P(\mathbf{z})B(\mathbf{z})R^{-1}(\mathbf{z})B^T(\mathbf{z})P(\mathbf{z}).$$

Then, we can estimate $\dot{\mathbf{z}}$ with $\dot{\tilde{\mathbf{z}}}$ and $\mathbf{u}$ with $\tilde{\mathbf{u}}$. Note that these are estimations because we use approximate solvers for P. Note that here, we must re-solve the Riccati equation for each new state, since the solution to the Riccati equation is state-dependent in this version of LQR.

2.3 Optimizing the Algorithm

Since the setup of the LQR problem does not include the option to optimize over the final time, we would ideally be able to quicken the time it takes to land the ship on the landing pad by changing the values of Q and R. It is necessary that Q is positive semidefinite and R is positive definite. We chose Q and R to be diagonal matrices that satisfy these conditions.

$$Q = \begin{bmatrix} q_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & q_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & q_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & q_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & q_5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & q_6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & q_7 \end{bmatrix} \quad R = \begin{bmatrix} r_1 & 0 & 0 \\ 0 & r_2 & 0 \\ 0 & 0 & r_3 \end{bmatrix}$$

Notice that since we chose to extend the state variable from $\mathbf{x}$ to $\mathbf{z}$ by including the 1 on the end for the acceleration due to gravity, that q_7 is multiplied by 1 each time. Thus, in the integral, if $q_7 \neq 0$, then the integral will always diverge. Thus, $q_7 = 0$, and therefore

$$Q = \begin{bmatrix} q_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & q_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & q_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & q_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & q_5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & q_6 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Since we want the ship to get to the ground as quickly as possible, we need to heavily weight q_2, which corresponds with the penalty for being far from the ground. Similarly, we want to keep $q_1 > 0$ since this coefficient corresponds with the penalty of being far from the center of the landing pad. However, we should note that since the landing pad is fairly wide, we can let $0 < q_1 < q_2$ to optimize for quick landing instead of optimizing for landing in the exact center of the landing pad. We also care a great deal about q_4, since this coefficient describes $\dot{y}$. If our lander touches the ground with a vertical velocity that is too large, the lander will crash. Thus, we want to keep the vertical velocity fairly low so that the ship will land smoothly without crashing.

Because R must be invertible, $r_1, r_2, r_3 > 0$. However, since we do not consider constraints on fuel in this problem, we should not penalize use of the engines. Thus, when we were testing our algorithm, we kept the values of r_1, r_2, r_3 very close to 0 at 0.01 to prevent penalizing the use of our engines.

Because of the difficulty of our problem, we were unable to experiment with these values past reasoning through which coefficients we want to heavily weight. Thus, we do not know exact relationships between these values that will allow us to quickly land the spaceship.

3 Results

Our code has repeatedly run into errors, and we are not sure of all the reasons. The biggest issue that we faced was trying to understand and include constraints on our controls. We think that because our treatment of the control was imperfect, we were unable to land the ship under our original setup.

We tried to simplify the problem by reducing the amount of noise introduced in the environment. This was accomplished by adjusting the environment's variable 'SCALE' by a factor of 6. This 'SCALE' parameter controlled every part of the behavior of the environment - from how quickly the ship moved to the amount of force exerted on the ship to the original noise introduced to the system in the initial random state. By reducing this system parameter, we were able to land the ship on the landing pad using our optimal control. This is because the initial state had a much lower variance around 0, so the ship essentially began from rest at the top of the screen with no initial angle or angular velocity. However, we also noticed that by adjusting this parameter, the force exerted by the main engine was insufficient to combat the force of gravity exerted by the moon, and thus the optimal control was simply to fire the main engine at full power for the duration of the lander's flight. Thus, the rocket appears to simply fall to the ground. However, by firing the main engine, the force of impact of the lander on the ground is less than required to lose the game. Thus, though this solution appears simple, it is still a successful control from the perspective of the game, though it did not accomplish our goal of a 'soft' landing.

We were able to solve a simpler version of the problem by assuming that the lander starts upright and directly above the landing pad. This can be modeled as a boundary value problem in a single variable, since we need only fire the main engine. If y_1 describes the vertical position of the ship, and $y_2 = \dot{y_1}$, then if the main engine is not firing,

$$\dot{\mathbf{y}} = \dot{\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}} = \begin{bmatrix} \dot{y_1} \\ \dot{y_2} \end{bmatrix} = \begin{bmatrix} y_2 \\ -g \end{bmatrix}$$

.

If we are firing the main engine, this problem is modeled as

$$\dot{\mathbf{y}} = \dot{\begin{bmatrix} y_1 \\ y_2 \end{bmatrix}} = \begin{bmatrix} y_2 \\ -g + \frac{F_m}{m} \end{bmatrix}$$

where F_m is again the force exerted by the main engines.

This problem has a bang-bang solution. For simplicity we rescale time so the engine is off for one unit of time in one variable for y_1 and on for one unit of time in another variable for y_2. To do this, we let t_s be the switching time and t_f be the final landing time; then $t = t_s\tilde{t}$ and $t = (t_f - t_s)\tilde{\tilde{t}} + t_s$.

Then our system changes to the following:

$$\dot{\mathbf{y}} = \begin{bmatrix} t_s y_2 \\ -t_s g \end{bmatrix}$$

$$\dot{\mathbf{y}} = \begin{bmatrix} (t_f - t_s) y_2 \\ (t_f - t_s)(-g + \frac{F_m}{M}) \end{bmatrix}$$

Notice that by renaming variables, we can stack these two systems together since they now use the same time constraints, we can construct a single ODE with cohesive boundary conditions that can be solved with a regular BVP solver.

$$\dot{\mathbf{y}} = \begin{bmatrix} y_1 \\ \dot{y}_1 \\ y_2 \\ \dot{y}_2 \end{bmatrix} = \begin{bmatrix} t_s \dot{y}_1 \\ -t_s g \\ (t_f - t_s)\dot{y}_2 \\ (t_f - t_s)(-g + \frac{F_m}{M}) \end{bmatrix}$$

where we have boundary conditions $y_1(0) = y_0$, $\dot{y}_1(0) = \dot{y}_0$, $y_2(t_f) = 0$, $\dot{y}_2(t_f) = 0$, $y_1(1) = y_2(0)$, and $t_s\dot{y}_1(1) = (t_f - t_s)\dot{y}_2(0)$.

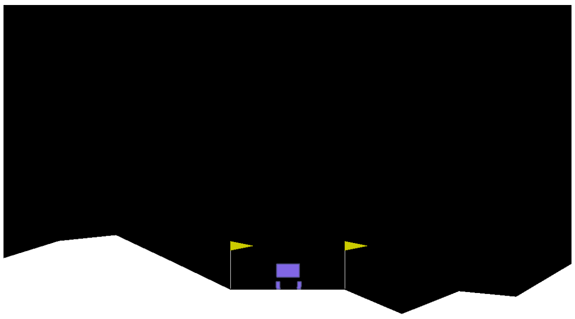

Figure 3: The ship landed on the landing pad using the bang-bang solution.

In the above figure we can see that this solution landed the ship on the landing pad. The optimal switching time to turn on the engine is at $t_s = 1.1$ seconds. At this point, we turn on the main engines at full power and the ship slows down just in time to land on the pad.

4 Conclusion

This problem was much more difficult than we originally anticipated. This was caused by a combination of confusing, messy code from the original authors of the environment, and complicated physics with constraints on the system that prevented us from accurately modeling the lander's flight patterns. The heuristic approach included in the code base worked very well to solve the landing problem. This is because it utilized a reinforcement learning method that does not require a complete environment model upfront. Thus, it is very likely that there is some small error in our model of the environment. Because of the noise introduced throughout the environment, this error is magnified. However, when simplifying the system, we were able to successfully land the lander by turning on the engines at full power for the duration of the flight. Because the game did not classify the impact of the landing high enough to qualify as a 'crash', we were able to land the ship in a de-noised environment.

We were able to find an optimal control for a simplified version of the problem using a boundary value problem solver. From this, we found that the optimal way to land the ship with from rest with no initial angle or angular velocity is to let the ship fall for 1.1 seconds and then turn on the main engines at full power.

Some natural extensions of this problem include expanding the working BVP solution to account for a wider range of initial conditions. That is, to create a separate optimal control to stabilize the lander to the center of the screen with $\theta \approx 0$ and $\dot{\theta} \approx 0$, and then use our working BVP solver to land the ship onto the pad.

5 References

[1] https://www.intechopen.com/books/parallel_manipulators_towards_new_applications/mobility_of_spatial_parallel_manipulators
[2] Whitehead, Jared. "Chapter 22. Linear Quadratic Regulator (the 'right' way to optimize)"
[3] Nonlinear Systems and Control - Spring 2015. Lecture: State Dependent Riccati Equations. http://people.ee.ethz.ch/~apnoco/Lectures2015/09_SDRE_Hinfinity.pdf