Controlling Vertical Rocket Landings Using Iterative Linear Quadratic Regulator Algorithms

Ray Chang

July 23, 2021

Abstract

The control of vertical rocket landings is an important problem in aerospace engineering. The ability to land rocket boosters safely back on Earth allows for the reuse of rocket parts; a key step in the expansion of space missions. In this project, I investigated the use of an iterative linear quadratic regulator (iLQR) method in finding solutions to this trajectory optimization problem. I further adapted the iLQR method to a closed loop controller, in which the iLQR solution is recomputed at each time step in the model predictive control (MPC) fashion. I found that the open loop iLQR method is able to converge to good solutions in some cases with good initialization. However, the closed loop controller, although being more computationally expensive, is more robust to poor initializations and converges to better solutions.

1 Introduction

The rocket landing problem is a special case of a trajectory optimization problem with nonlinear dynamics. My motivation for studying this specific application of control comes from both the interesting and currently relevant aerospace engineering context of this control problem, as well as a desire to understand how nonlinear control methods work when applied to real problems. As described by the principal rocket landing engineer at SpaceX: "Precision landing [of rockets] has the potential to improve exploration of the solar system and to enable rockets that can be refueled and reused like an airplane... SpaceX uses CVXGEN (Mattingley and Boyd 2012) to generate customized flight code, which enables very highspeed onboard convex optimization."[1].

To simplify the real problem looked at by SpaceX and other aerospace companies, I restricted my problem to two dimensions. I also assumed the mass of the rocket to be constant (in reality fuel is burned away causing continuous reduction of mass), and simplified the controls of the rocket to be a single thruster's force and nozzle angle.

I define the state and control vectors to be

$$s = [x, y, \dot{x}, \dot{y}, \phi, \dot{\phi}]^T \text{ and } u = [F, \theta]^T \tag{1}$$

where ϕ denotes the orientation angle of the rocket and θ denotes the nozzle angle of the thrusters. From Figure 1, I derive the dynamics of the rocket as

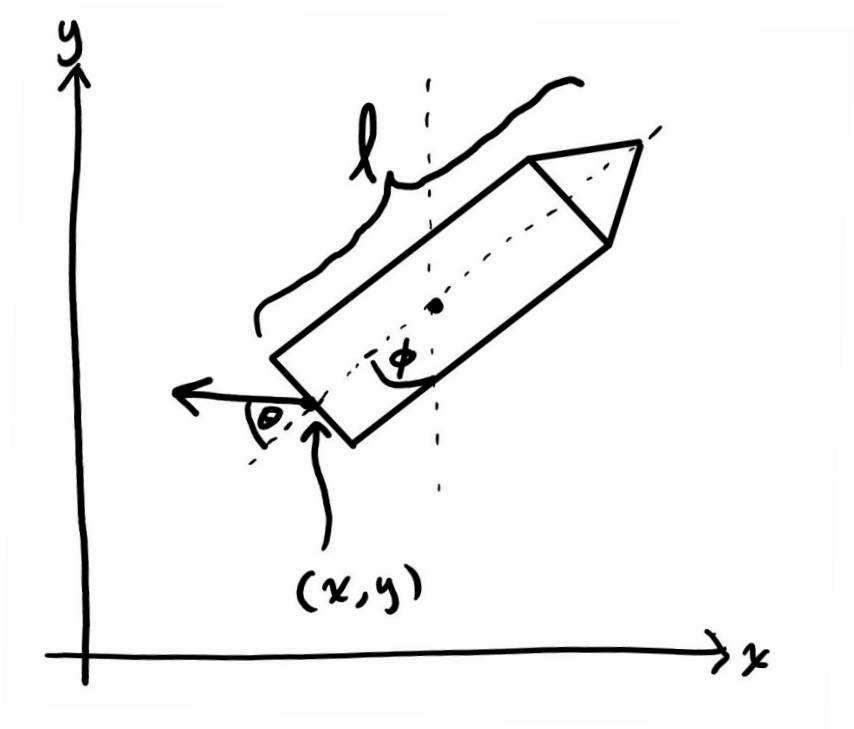

Figure 1: Diagram of the rocket

$$m\ddot{x} = F\sin(\theta + \phi) \tag{2}$$

$$m\ddot{y} = -mg + F\cos(\theta + \phi) \tag{3}$$

$$I\ddot{\phi} = -\frac{Fl}{2}\sin(\theta) \tag{4}$$

Using forward Euler discretization with time step h, the derived discrete-time dynamics are

$$x_{t+1} = x_t + h\dot{x}_t \tag{5}$$

$$y_{t+1} = y_t + h\dot{y}_t \tag{6}$$

$$\dot{x}_{t+1} = \dot{x}_t + h\frac{F_t}{m}\sin(\theta_t + \phi_t) \tag{7}$$

$$\dot{y}_{t+1} = \dot{y}_t + h(-g + \frac{F_t}{m}\cos(\theta_t + \phi_t)) \tag{8}$$

$$\phi_{t+1} = \phi_t + h\dot{\phi}_t \tag{9}$$

$$\dot{\phi}_{t+1} = \dot{\phi}_t - h\frac{F_t l}{2I}\sin\theta \tag{10}$$

Given a time horizon T and a desired final state target s^*, the problem is then

$$\begin{aligned} \min_{u_0,\ldots,u_{T-1}} \quad & (s_T - s^*)^T Q_T (s_T - s^*) + \sum_{t=0}^{T-1} s_t^T Q s_t + u_t^T R u_t \\ \text{s.t.} \quad & s_0 = s_{\text{initial}} \\ & s_{t+1} = f(s_t, u_t) \\ & s_t \in \mathcal{S} \quad \forall t \\ & u_t \in \mathcal{U} \quad \forall t \end{aligned} \tag{11}$$

2 Related work

As a well studied problem, multiple approaches have been used in the past to tackle this kind of rocket landing problem. These include classical controllers like PID, as well as other methods such as MPC and reinforcement learning algorithms[2]. As described earlier, SpaceX uses onboard convex optimization, which is only possible through a lossless relaxation of the original nonconvex problem into a convex one with a provably optimal solution[3]. This method is probably the most successful as it has been put into practice with many well known successes.

3 Algorithms

The main algorithm of my project is iLQR. This method is a first order approximation of a more general method known as differential dynamic programming

(DDP)[4]. Both iLQR and DDP are iterative algorithms that are initialized with an initial guess of the controls (nominal controls) $\{u_t\}$[5]. These control policies are then "rolled out" in a forward pass using the nonlinear dynamics to produce a nominal trajectory $\{s_t\}$. Then a backward pass is executed in which the nonlinear dynamics are linearized around the nominal trajectory and controls producing a modified LQR problem. Explicitly, this looks like

$$\Delta s_{t+1} = \nabla_s f(s_t, u_t)\Delta s_t + \nabla_u f(s_t, u_t)\Delta u_t \tag{12}$$

where letting $A_t = \nabla_s f(s_t, u_t)$ and $B_t = \nabla_u f(s_t, u_t)$ gives the LQR form

$$\Delta s_{t+1} = A_t \Delta s_t + B_t \Delta u_t \tag{13}$$

Note that this is a modified LQR problem since the dynamics are in terms of the state and control deviations from the nominal trajectory, an unavoidable consequence of using a Taylor expansion around the nominal trajectory to linearize the dynamics. Using a modified set of Riccati equations on this LQR problem produces a new set of affine control laws and control inputs. These controls are then fed into the forward pass again, beginning another iteration, which repeats until convergence.

Taken directly from [5], the affine controller and modified Riccati equations are

$$\Delta u_t = -K\Delta s_k - K_v v_{t+1} - K_u u_t \tag{14}$$
$$K = (B_t^T S_{t+1} B_t + R)^{-1} B_t^T S_{t+1} A_t \tag{15}$$
$$K_v = (B_t^T S_{t+1} B_t + R)^{-1} B_t^T \tag{16}$$
$$K_u = (B_t^T S_{t+1} B_t + R)^{-1} R \tag{17}$$
$$S_t = A_t^T S_{t+1}(A_t - B_t K) + Q \tag{18}$$
$$v_k = (A_t - B_t K)^T v_{t+1} - K^T R u_t + Q s_t \tag{19}$$
$$S_T = Q_T \tag{20}$$
$$v_T = Q_T(s_T - s^*) \tag{21}$$

Note that the control law for the actual state variables is just $u_t^* = \Delta u_t + u_t$.

What has described up to this point is the iLQR algorithm exactly as it has been presented in its original paper [5]. However, I have implemented some additional features discussed in [4] to make iLQR more applicable to the rocket

landing problem. The first deals with iLQR's convergence. When I implemented iLQR as described up to this point, the trajectories and controls would frequently diverge wildly. This is addressed in [4], which implements a backtracking line search on the magnitude of the affine terms to determine a control law update that guarantees a descent in the cost function. In this case, (14) becomes

$$\Delta u_t = -K\Delta s_k - \alpha(K_v v_{t+1} + K_u u_t) \tag{22}$$

where α is initialized as 1 and is iteratively reduced until the cost function for the rolled out trajectory using Δu_t reduces the cost function.

The second alteration comes from the need to implement the constraints of the rocket landing problem. There is a state constraint that the rocket cannot be underground, i.e. $y \geq 0$. The input constraint is on the thruster force: $0 \leq F \leq F_{\max}$. For the state constraint, I implemented a quadraticized barrier cost function. Explicitly, I used a 2nd order Taylor expansion of the barrier function

$$b(y) = \exp(-y) \tag{23}$$

to penalize states wandering into inaccessible areas. This then is incorporated into the LQR problem as a stage cost. For the control constraints, I used a simple "clamping" method in the forward pass. When the forward pass rolls out the new trajectory with controls, any control that lies outside of the linear constraint set is replaced with a control at the nearest boundary. It is important to note that [4] explains that this may harm convergence since there is no longer a guarantee of a descent direction. This algorithm works as follows:

Algorithm 1 iLQR with line search and constraints

procedure iLQR($s_0, \mathbf{u_0}, f$)
    Run forward pass on s_0 using $\mathbf{u_0}, f$ to get trajectory $\{s_0, \ldots, s_T\}$
    while $\mathbf{u}$ not converged **do**
        Compute cost of current trajectory $(\mathbf{x}, \mathbf{u})$
        Compute linearization of dynamics and quadraticization of costs around each (s_t, u_t)
        Run backward pass on current trajectory to get new $\mathbf{u}$
        Initialize alpha to 1
        while Cost of new trajectory>Cost of previous trajectory **do**
            Reduce alpha
            Run forward pass with constraint clamping to obtain new trajectory
            Compute cost of new trajectory
        end while
    end while
    Return new trajectory $\mathbf{s}$ and controls $\mathbf{u}$
end procedure

A second algorithm I implemented was an extension of iLQR using MPC style recomputing at each time step. At each time step, iLQR is run to compute the remaining controls and trajectory. The first control is used, and iLQR is rerun with the previously outputted controls as the new initial guess. Although MPC is an online closed loop controller, in the absence of disturbances in the dynamics, this can still be viewed as an open loop controller that simply updates each subsequent rerun of iLQR with a better initial guess.

Algorithm 2 iLQR with MPC style recomputation

procedure iLQR-MPC($s_0, \mathbf{u_0}, f$)
    for $k = 0$ to $T - 1$ **do**
        $\mathbf{s}, \mathbf{u}$=iLQR($s_0, \mathbf{u_0}, f$)
        Use first control in $\mathbf{u}$ to generate next state s_{k+1}
        Set s_0 to s_{k+1}
        Set $\mathbf{u_0}$ to $\mathbf{u}$ absent the first control
    end for
    Return $\mathbf{s}, \mathbf{u}$
end procedure

4 Results

An important result is the sensitivity of iLQR towards a good initialization. In this rocket problem, a good initialization is required in order to converge to an optimal solution. Examples 1 and 2 show the results of iLQR with the same problem parameters with zero control initialization and a simple constant control initialization that pushes the rocket towards the target state.

4.1 Example 1: Bad initialization

Example 1 Parameters	
Mass (m)	100
Length (l)	70
Moment of inertia	ml^2
Initial state	$[0, 5000, 10, 0, \pi/2, 0]$
Target state	$[1750, 0, 0, 0, 0, 0]$
Q_T	$10^{10} \times \text{diag}([1, 1, 1, 1, 50, 50])$
R	I_2
Algorithm 1 final cost	80292417619234.4
Algorithm 2 final cost	1175417549694.7

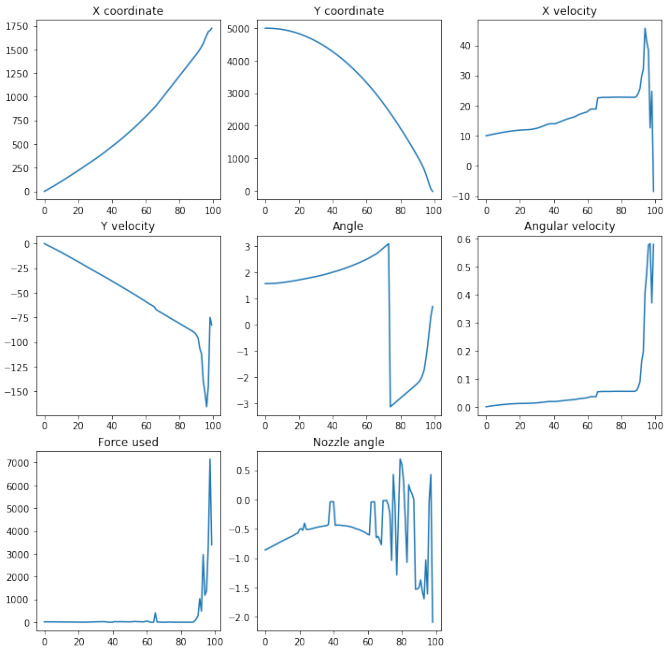

Figure 2: Example 1, algorithm 1 state and control plots

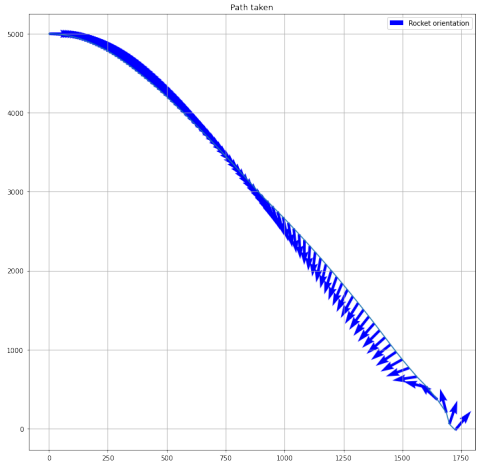

Figure 3: Example 1, algorithm 1 trajectory

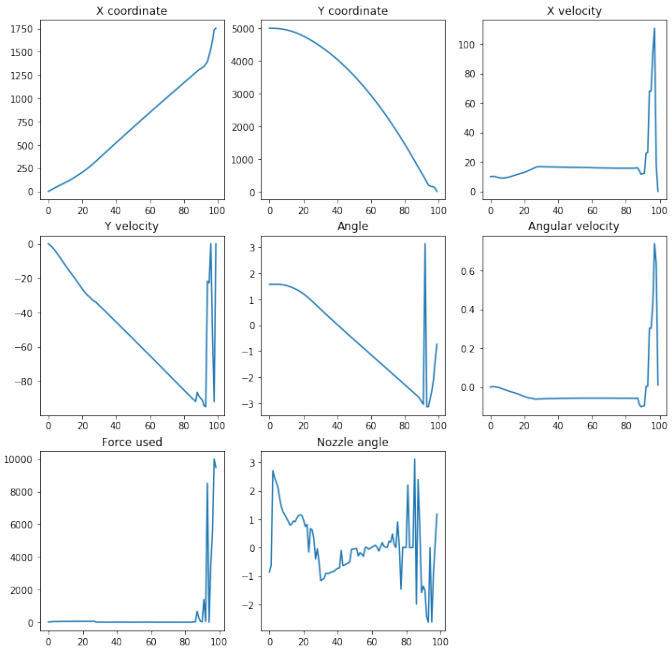

Figure 4: Example 1, algorithm 2 state and control plots

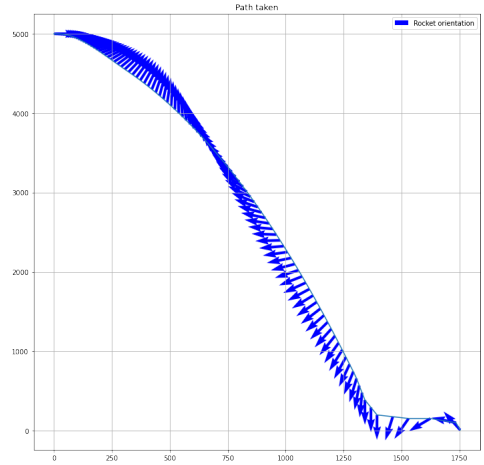

Figure 5: Example 1, algorithm 2 trajectory

With poor initialization, both algorithms converge to a local minima that is suboptimal. Although algorithm 2 performs better cost-wise, both exhibit poor

behavior in their trajectory and final state.

4.2 Example 2: Simple constant thrust initialization

In example 2, I reuse the parameters from example 1. However, the initial guess control is changed from zero to a constant thrust with zero nozzle angle.

Example 2 Parameters (Same as example 1)	
Algorithm 1 final cost	77975137817204.5
Algorithm 2 final cost	33316718.99

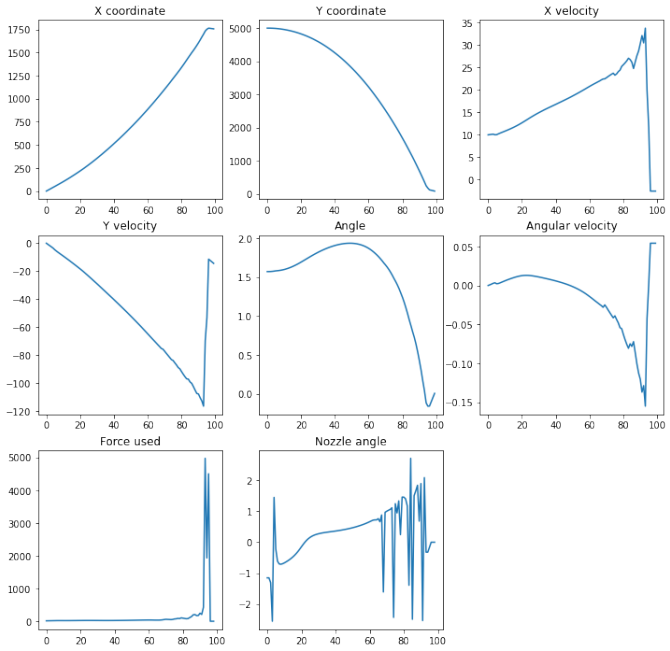

Figure 6: Example 2, algorithm 1 state and control plots

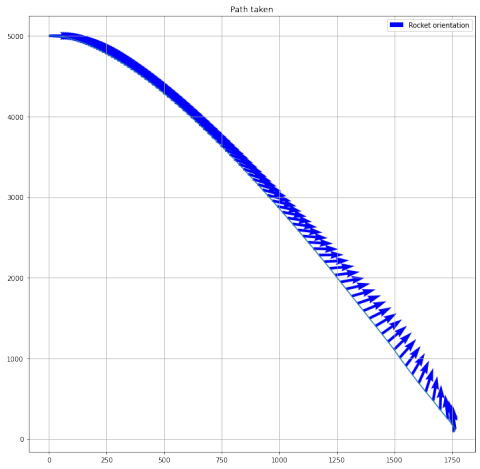

Figure 7: Example 2, algorithm 1 trajectory

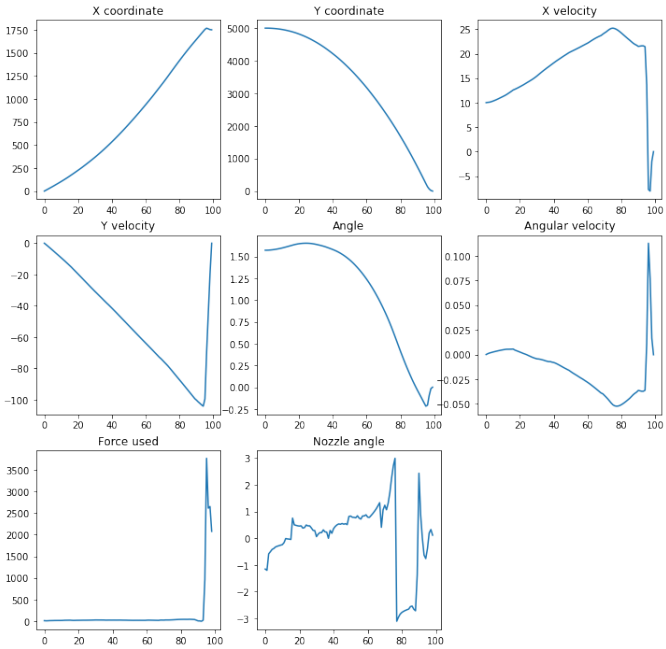

Figure 8: Example 2, algorithm 2 state and control plots

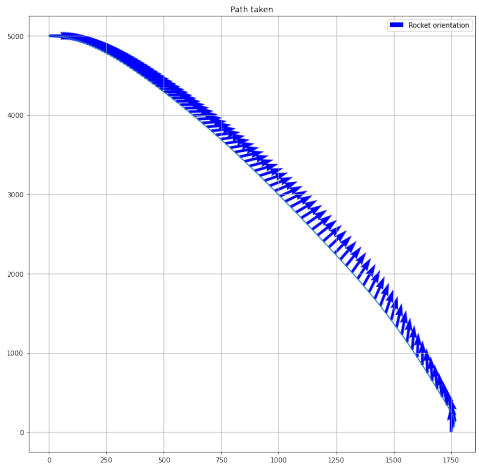

Figure 9: Example 2, algorithm 2 trajectory

With a simple but better initialization than example 1's, both algorithms converge to much better solutions, with algorithm 2 again outperforming algorithm 1.

Since algorithm 2 runs iLQR $T-1$ times, it requires a longer computation time, but yields much better control laws and trajectories.

5 Conclusion

This project introduced me to a lot of interesting literature on iterative methods in nonlinear control. There are a couple of open questions and future paths I would explore in this project if I had more time. The most constraining factor in finding good solutions with iLQR is the combined problem of needing a good initialization to not get stuck in local minima. Towards this goal, there are interesting paths to be taken such as using a batch initialization of multiple control guesses and picking the most optimal one. Another path I would've liked to explore more is the comparison between iLQR and DDP, which itself has more sophisticated techniques to incorporate control constraints [4], whereas in this project I just used a simple clamping procedure.

References

[1] Blackmore L., "Autonomous Precision Landing of Space Rockets", The Bridge, 15-20, Winter 2016

[2] Ferrante, R. (2017). A Robust Control Approach for Rocket Landing (Master's Thesis).

[3] Blackmore, L., Açikmeşe, B., and Scharf, D. P. (2010). Minimum-Landing-Error Powered-Descent Guidance for Mars Landing Using Convex Optimization. Journal of Guidance, Control, and Dynamics, 33(4), 1161–1171. https://doi.org/10.2514/1.47202

[4] Tassa, Y., Mansard, N., and Todorov, E. (2014). Control-limited differential dynamic programming. 2014 IEEE International Conference on Robotics and Automation (ICRA). https://doi.org/10.1109/icra.2014.6907001

[5] Li, W., and; Todorov, E. (2004). Iterative Linear Quadratic Regulator Design for Nonlinear Biological Movement Systems. Proceedings of the First International Conference on Informatics in Control, Automation and Robotics. https://doi.org/10.5220/0001143902220229