ME 7247: Advanced Control Systems **Fall 2022–23**

Lecture 20: Model Predictive Control

November 18, 2022

Lecturer: Laurent Lessard *Scribe: Xavier Hubbard*

In this lecture we explore the idea of Model Predictive Control (MPC), also known as Receding Horizon Control (RHC). First we develop a constrained LQR problem with an infinite horizon. From here we discuss the drawbacks of LQR and develop MPC as an alternate strategy.

1 Model Predictive Control

1.1 Defining Constrained LQR

The dynamics of the constrained LQR system will be defined as follows:

$$\begin{aligned} J(x_0) = \underset{\substack{u_0,u_1,\dots \\ x_1,x_2,\dots}}{\text{minimize}} \quad & \sum_{k=0}^{\infty} (x_k^\mathsf{T} Q x_k + u_k^\mathsf{T} R u_k). \\ \text{s.t.} \quad & x_{k+1} = Ax_k + Bu_k \quad \text{for} \quad k = 0, 1, \dots \\ & x_k \in X, \quad u_k \in U \end{aligned} \tag{1}$$

Note the important differences between this and our previous definitions of the LQR problem. Here we are allowing k to go to infinity, we are optimizing over the states as well as the inputs, and the states and inputs are themselves constrained to X and U respectively. The feasible sets X and U can take various form, but there are a few common constraints worth talking about.

1. "box constraint": lower and upper bounds on the components of a decision variable. These linear constraints describe an n-dimensional box. They are easy to work with because each individual scalar variable is separately constrained.

   $$\text{e.g.} \quad a \le x_k \le b.$$

2. "ball constraint": norm bound on a decision variable. If using the 2-norm, this constraint couples the different components of the vector.

   $$\text{e.g.} \quad \|x_k\| \le 1.$$

3. "polyhedral": a set of linear constraints on a decision variable. In other words, the decision variable must lie inside a polytope.

   $$\text{e.g.} \quad Fx_k \le g.$$

4. "rate constraint": constraining the rate at which a variable can change between timesteps.

   $$\text{e.g.} \quad \|u_k - u_{k-1}\| \le 10.$$

As with standard LQR, we make the typical assumptions

$$Q \succeq 0 \text{ and } R \succ 0.$$

We also make the following assumptions

$$0 \in X \text{ and } 0 \in U.$$

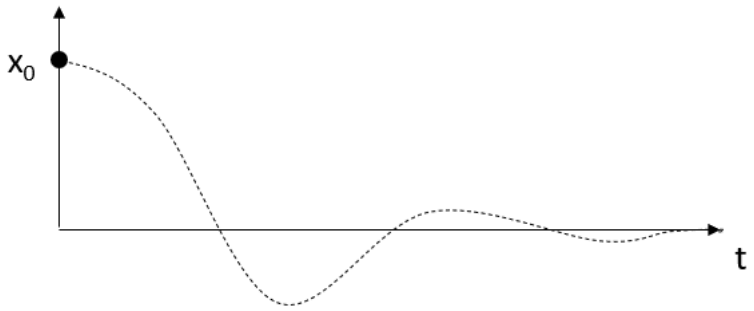
We add these constraints because we hope for a solution that doesn't have infinite cost despite having an infinite horizon. This means that x_k and u_k should eventually reach 0. Moreover, if the feasible set were to *exclude* 0, then there would be some neighborhood around 0 that could never be reached, implying that the infinite-horizon cost could never be finite.

1.2 Solving Constrained LQR

Now that we have defined the constrained LQR problem, we will focus on solving it. To solve the unconstrained LQR problem we used a few solutions: dynamic programming, solving a large least squares problem, and others. Both DP and solving the least squares problem will work, but DP requires a few extra steps and will be discussed later. For now we will focus on setting up and solving the problem as a large scale least squares optimization problem. In this case we can classify this optimization problem as a quadratic program (QP) or a quadratically constrained quadratic program (QCQP) depending on the constraints we have.

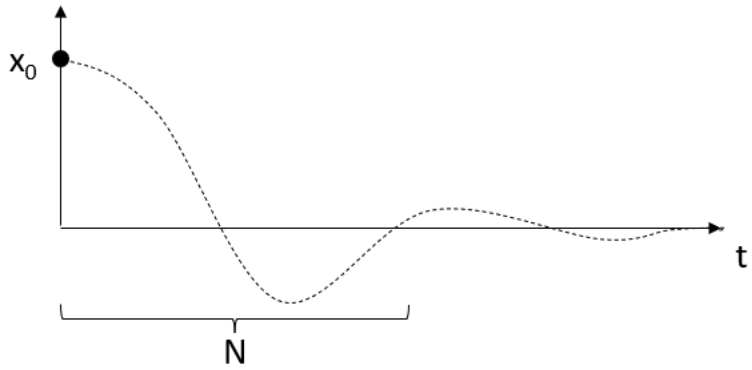
There is a problem with this approach, however. In determining the control inputs this way, you have effectively designed an "open-loop" controller, and must deal with all of the problems that come with such. Our solution to this problem will be "Model Predictive Control" (MPC). For reference, MPC is just another name for "Receding Horizon Control" (RHC).

Imagine we are trying to use a controller to drive some state, x to 0 starting from x_0 as $t \to \infty$. This can be simply visualized here:

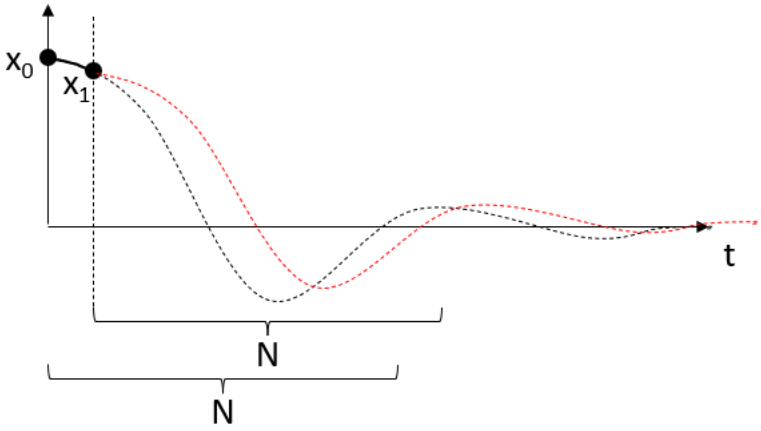


MPC can then be understood in the following steps:

1. Choose a finite horizon N.



2. Solve the finite horizon LQR problem using (x_0, N). This produces a set of optimal decisions $\{u_0^\star, u_1^\star, \ldots, u_{N-1}^\star\}$ and corresponding predicted states $\{x_1^\star, \ldots, x_N^\star\}$.
3. Apply the *first* action $u_0^\star$ to move to x_1. Although we expect to move to $x_1^\star$, there may be some noise, unmodeled dynamics, etc. So we may not end up exactly at $x_1^\star$.
4. Solve the finite horizon LQR problem using (x_1, N).
5. Apply the first action again, to move to x_2.



6. repeat.

Since you are re-evaluating the problem at each step, MPC allows you to deal with noise in the process, and can be thought of as *pseudo-open-loop*. Since this method is essentially stitching together many open-loop solutions, we can no longer guarantee optimality. However, it still serves as a useful heuristic, despite needing tuning. If 90% of the worlds controllers are PID, the other 10% are MPC. People use this because it works!

1.3 Designing an MPC Controller

There are 3 key concerns when designing an MPC controller:

1. *Computation*: The computational complexity of the problem, which is dependent on the size of the finite horizon. Longer horizons require solving larger optimization problems (more variables and constraints) at each timestep. When using the controller in real-time, we may be limited in terms of how much time we have to solve an optimization problem before we must make our next decision.

2. *Feasibility*: Finding a feasible solution for a given timestep of the process does not guarantee that a feasible solution will always exist in the future. An example of this is a car with a limited sensor range that has a simple constraint of "don't hit walls." If the car is going fast enough, there can exist a time-step where the wall enters the sensor range and there is no feasible solution to avoid hitting it despite everything being fine on the previous timestep. Selecting a larger finite horizon N increases the likelihood of continued feasibility, however there's a trade-off with additional computational complexity.

3. *Stability*: It is difficult to know whether the state, x_k, and input, u_k, will eventually go to zero. There can exist scenarios where the optimization problem is feasible at every timestep, but the trajectory of the system oscillates forever.

One way to encourage the problem to remain feasible and stable is to modify the optimization problem solved at each timestep to incorporate a suitably chosen terminal cost and terminal set constraint. Here is the modified optimization problem.

$$\begin{aligned} \hat{J}(z) \quad = \quad & \underset{\substack{u_0,u_1,\ldots,u_{N-1}\\ x_1,x_2,\ldots,x_N}}{\text{minimize}} \quad \sum_{k=0}^{N-1} \left(x_k^\mathsf{T} Q x_k + u_k^\mathsf{T} R u_k \right) + x_N{}^\mathsf{T} Q_f x_N. \\ & \text{s.t.} \quad x_0 = z, \quad x_N \in X_f, \\ & \qquad x_{k+1} = Ax_k + Bu_k \quad \text{for } k = 0, 1, \ldots \\ & \qquad x_k \in X, \quad u_k \in U \end{aligned} \tag{2}$$

Ensuring optimality or feasibility. We will discuss two simple methods of guaranteeing feasibility or optimality. These methods typically require large N so they often aren't practical, but still worth knowing about.

1. If the infinite-horizon cost is to be finite, x_t and u_t must go to zero as $t \to \infty$. Therefore, for t sufficiently large, the constraints $x_k \in X$ and $u_k \in U$ will be *slack* for all $k \geq t$. In other words, the tails of the trajectories of x and u will remain in the strict interior of X and U, respectively. From this point forward, the solution to the constrained LQR problem will be the same as that of the unconstrained LQR problem, which we know is given by the infinite-horizon LQR policy $u_t = Kx_t$ and associated cost-to-go $V(z) = z^\mathsf{T} P z$.

   Suppose we could pick N large enough that the constraints are slack for $t \geq N$. Then, the optimal cost from that point forward should be $V(x_N) = x_N^\mathsf{T} P x_N$. This means that if we pick $Q_f = P$ (we can set $X_f = X$) in (2), then the MPC approach will actually produce the *optimal* constrained infinite-horizon controller! Unfortunately, it is hard to know how large N needs to be. In general, it will be problem-dependent.

2. If you pick $X_f = \{0\}$, the set containing only 0, you can guarantee "recursive feasibility." Recursive feasibility means that you are guaranteed to have feasibility forever so long as you are feasible initially. This is a *very* strict constraint as it requires a solution that gets you to 0 at the end of your horizon. This means that if N is too small, there may not exist an initially feasible solution.

   The way to think about this is to imagine having a solution that gets you to 0 successfully within your finite horizon. If you execute the first step of that solution, you can guarantee that the optimization problem for the next timestep will be feasible, since you can use the computed path for the previous timestep and just add $u_N = 0$, which together with $x_N = 0$ yields $x_{N+1} = 0$, so we are again satisfying the terminal state constraint.

These two approaches are straightforward to implement but often require N to be too large to be practical. In the next lecture, we will see how these approaches can be refined to be made more practical.

Improving MPC performance. Simple steps can be taken to improve the practical performance of an MPC controller. A larger finite horizon N always leads to better performance, so the goal of all of these suggestions is to allow for a larger N.

1. *Simplify the model/constraints.* Using a model with smaller state dimension or simplified constraints can decrease the size of the optimization problem that must be solved at each timestep, which means it can be solved faster, which means we can use a larger N. There is a trade-off; using an approximate model means a less accurate solution, but one that can be computed more quickly.

2. *Solve approximately.* Most optimization solvers are iterative in nature, meaning they iteratively refine the solution. Decreasing the accuracy demanded of the solution means we can get to an approximate solution more quickly. Again, this is a trade-off. Solving the problem approximately means we can afford to solve larger instances, so we can increase the horizon length. Most numerical solvers solve to machine precision by default (10^{-16}). This is overkill for most applications, where a motor or a voltage source might only be able to provide increments of maybe 10^{-3}.

3. *Solve every k steps.* Instead of only implementing the first action of the plan (recomputing an optimal plan at every timestep), you can implement the first k steps of the plan and recompute a plan every k timesteps. This allows for more time to solve each optimization, so a longer horizon can be used. Again, there are trade-offs. Such an approach is obviously not a good idea if there is a lot of process noise in the system.

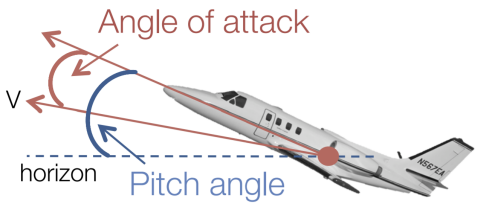
4. *Warm-starting.* By providing a very good initial guess, iterative solvers converge to the solution more rapidly. It is often a good guess to use the solution from the previous step as the initial guess for the current step. This is called *warm-starting.* Care needs to be taken if using warm-start because the first timestep will always be a cold start!

2 MPC example: Pitch-altitude control

The following[1] is a linearized continuous-time model of a Cessna Citation aircraft at an altitude of 5000 m and a speed of 128.2 m/sec.

$$\dot{x} = \begin{bmatrix} -1.2822 & 0 & 0.98 & 0 \\ 0 & 0 & 1 & 0 \\ -5.4293 & 0 & -1.8366 & 0 \\ -128.2 & 128.2 & 0 & 0 \end{bmatrix} x + \begin{bmatrix} -0.3 \\ 0 \\ -17 \\ 0 \end{bmatrix} u$$

$$y = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} x$$



The states, input, and outputs are:

- x_1: angle of attack (in radians). This is the angle the plane makes with its velocity vector.
- x_2: pitch angle (in radians). This is the angle the plane makes with respect to the horizon.
- x_3: pitch rate (in rad/sec). This is the rate of change of the pitch angle.
- x_4: altitude (in meters). Measured with respect to the nominal altitude of 5000 m.
- u: elevator angle (in radians). The elevator is a control surface on the horizontal tail that can be angled down (positive angle) or angled up (negative angle). A positive elevator angle causes the plane tail to rise, causing the airplane to pitch down.
- y: the output is the pitch angle and the altitude.

The goal is to design a controller that regulates the altitude to zero subject to the constraints:

- We assume we can measure the full state, but only at a sampling rate of $T_s = 0.25$ seconds. This is also how frequently we can send a new actuator command.
- $|u| \leq 0.262$ ($\pm 15^\circ$) and $|\dot{u}| \leq 0.524$ ($\pm 30^\circ$/sec). These are mechanical constraints on the elevator that limit the range and rate of motion.
- $|x_2| \leq 0.349$ ($\pm 20^\circ$). This is a comfort constraint for the passengers.

The first step is to discretize the dynamics, which we can do using Matlab's `c2d(...)` command using the sampling rate of 0.25. This produces dynamics:

$$x_{t+1} = \underbrace{\begin{bmatrix} 0.6136 & 0 & 0.1570 & 0 \\ -0.1280 & 1 & 0.1904 & 0 \\ -0.8697 & 0 & 0.5248 & 0 \\ -27.5636 & 32.05 & 0.4087 & 1 \end{bmatrix}}_{A} x_t + \underbrace{\begin{bmatrix} -0.4539 \\ -0.4436 \\ -3.1989 \\ 0.6068 \end{bmatrix}}_{B} u_t$$

$$y_t = \underbrace{\begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}}_{C} x_t$$

[1] Model borrowed from the course ME231A at UC Berkeley. Developed by F. Borrelli, C. Jones, and M. Morari. `http://www.mpc.berkeley.edu/mpc-course-material/MPC_handsout.pdf`

We will now design a constrained LQR controller. The corresponding optimization problem is

$$
\begin{aligned}
\underset{u,x}{\text{minimize}} \quad & \sum_{t=0}^{\infty} x_t^\mathsf{T} Q x_t + u_t^\mathsf{T} R u_t && \text{(state update)} \\
\text{subject to:} \quad & x_{t+1} = A x_t + B u_t \\
& \begin{bmatrix} -\infty \\ -0.349 \\ -\infty \\ -\infty \end{bmatrix} \leq x_t \leq \begin{bmatrix} \infty \\ 0.349 \\ \infty \\ \infty \end{bmatrix} && \text{(pitch constraint)} \\
& -0.262 \leq u_t \leq 0.262 && \text{(elevator constraint)} \\
& -0.524 \leq \frac{u_{t+1} - u_t}{T_s} \leq 0.524 && \text{(elevator rate constraint)}
\end{aligned}
$$

In Figs. 1 to 7, we simulate several different scenarios and discuss the outcomes.

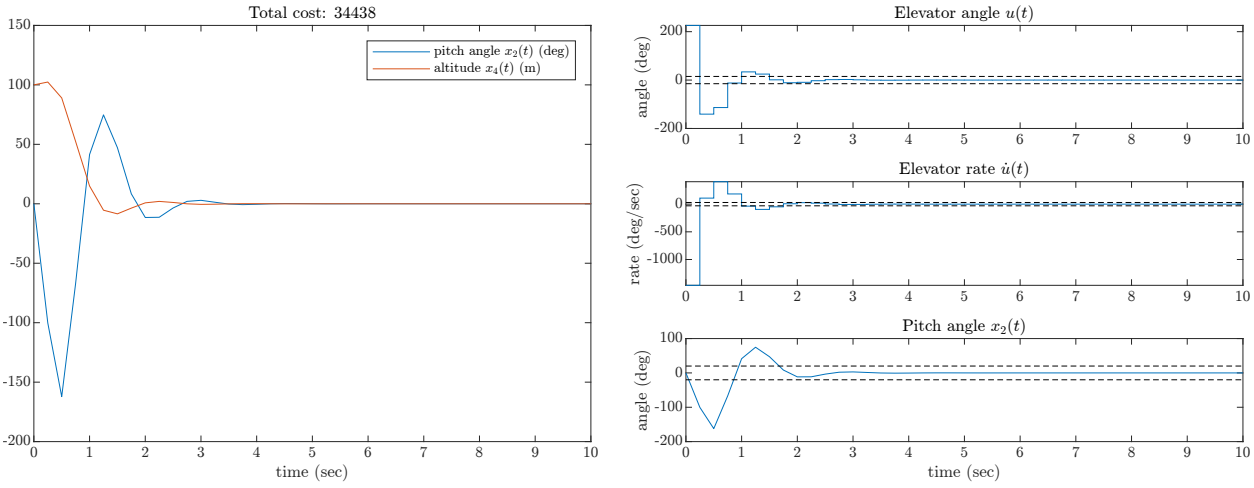


Figure 1: Ordinary LQR with $Q = I$ and $R = 100$. No constraints are imposed and naturally, the solution does not obey any of the constraints.

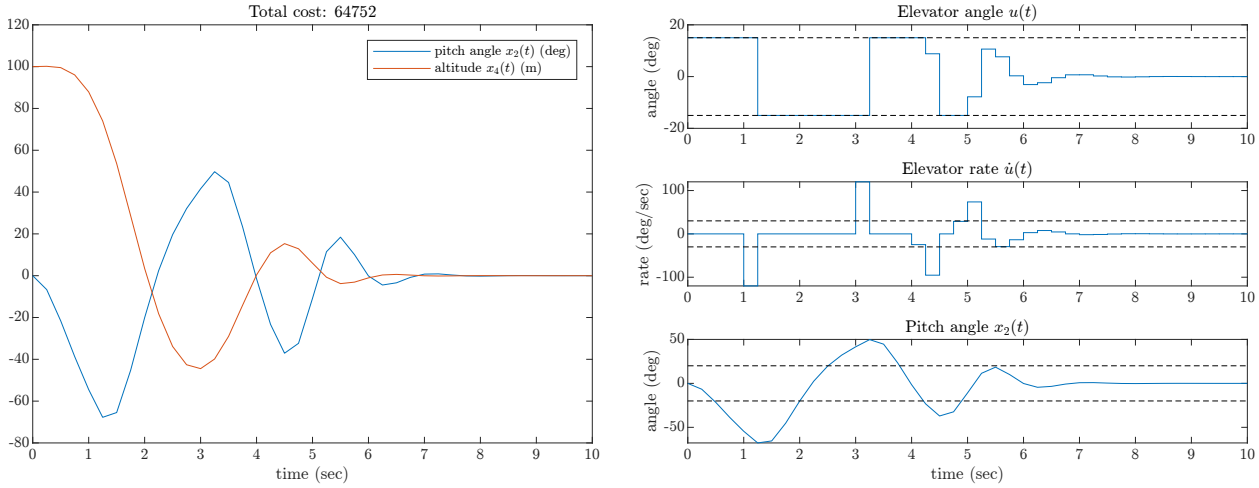


Figure 2: Ordinary LQR with $Q = I$ and $R = 100$, except this time, we clip the inputs so if $u_t > 0.262$, we set $u_t = 0.262$. Similarly, if $u_t < -0.262$, we set $u_t = -0.262$. This forces u_t to satisfy the constraints, but we do not satisfy the other two constraints. The clipping also induces some unwanted oscillation in the response.

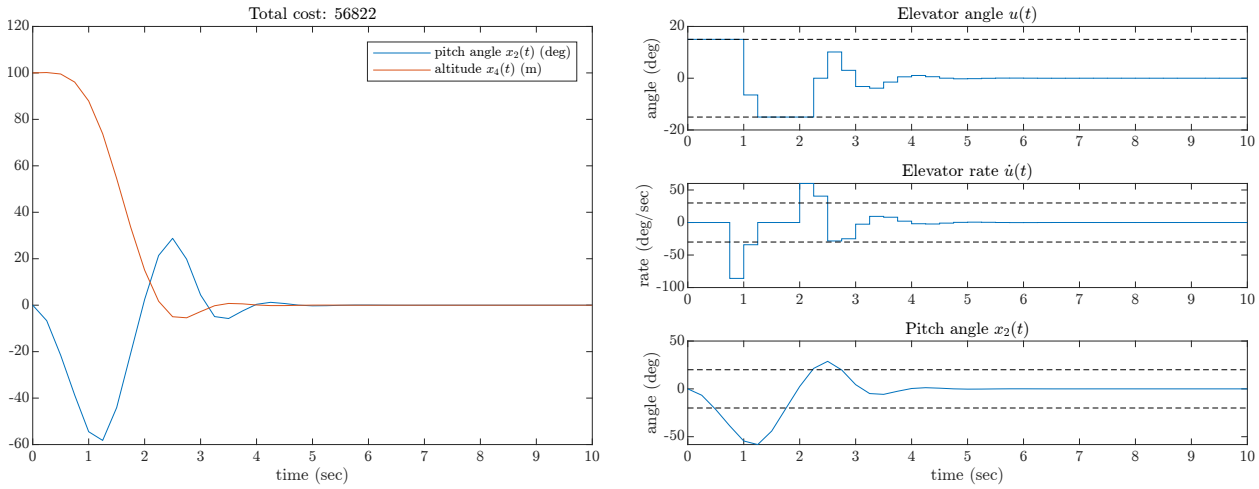


Figure 3: MPC with $Q = I$ and $R = 100$ and a horizon of $N = 10$ (10 steps corresponds to 2.5 sec since the sample time is 0.25 sec), but we only impose the constraint $|u_t| \leq 0.262$. This does not oscillate as much as LQR with clipped inputs, but the other two constraints are still not satisfied.

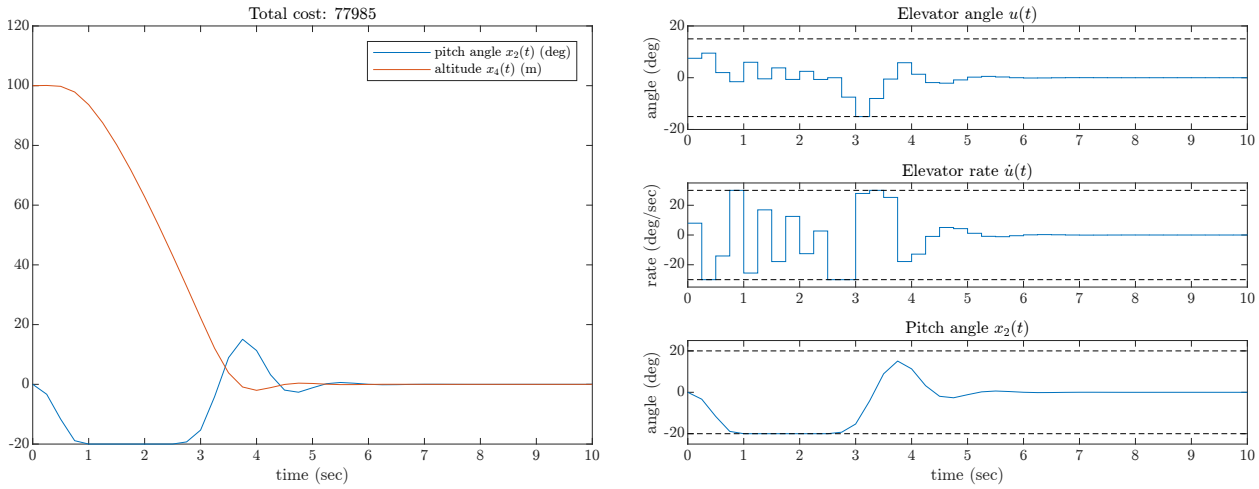


Figure 4: MPC with $Q = I$ and $R = 100$ and a horizon of $N = 10$, using all constraints. All constraints are satisfied, and the solution looks pretty good!

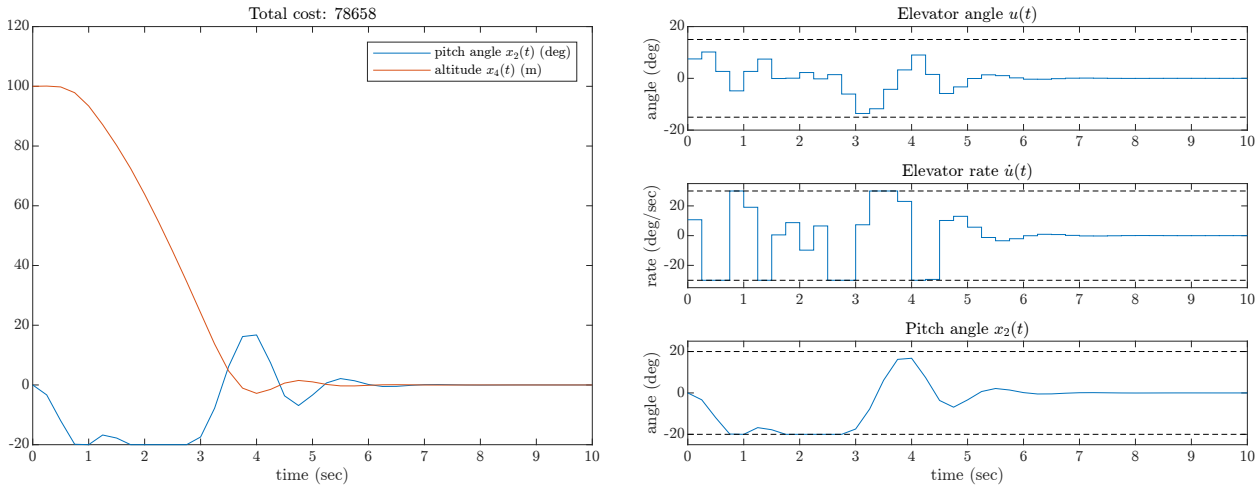


Figure 5: MPC with $Q = I$ and $R = 100$ and a horizon of $N = 5$, using all constraints. With this shorter horizon, a feasible solution is still found, but it oscillates a bit more.

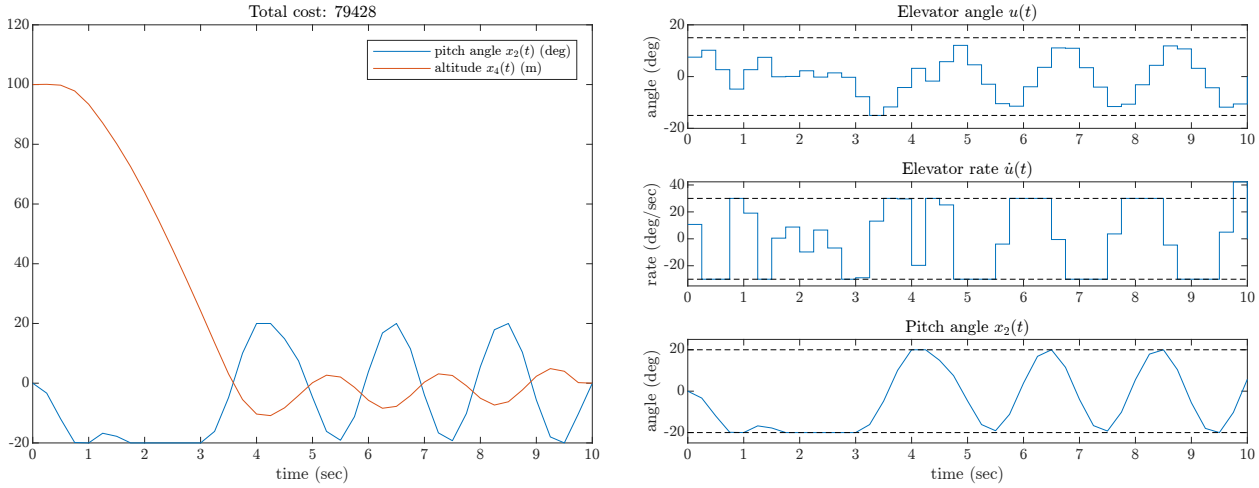


Figure 6: MPC with $Q = I$ and $R = 100$ and a horizon of $N = 3$, using all constraints. The horizon is too short: a feasible solution is found, but it is unstable and oscillates endlessly, so the infinite-horizon cost will be infinite. Note: if we use $N = 2$ we lose feasibility on the third timestep.

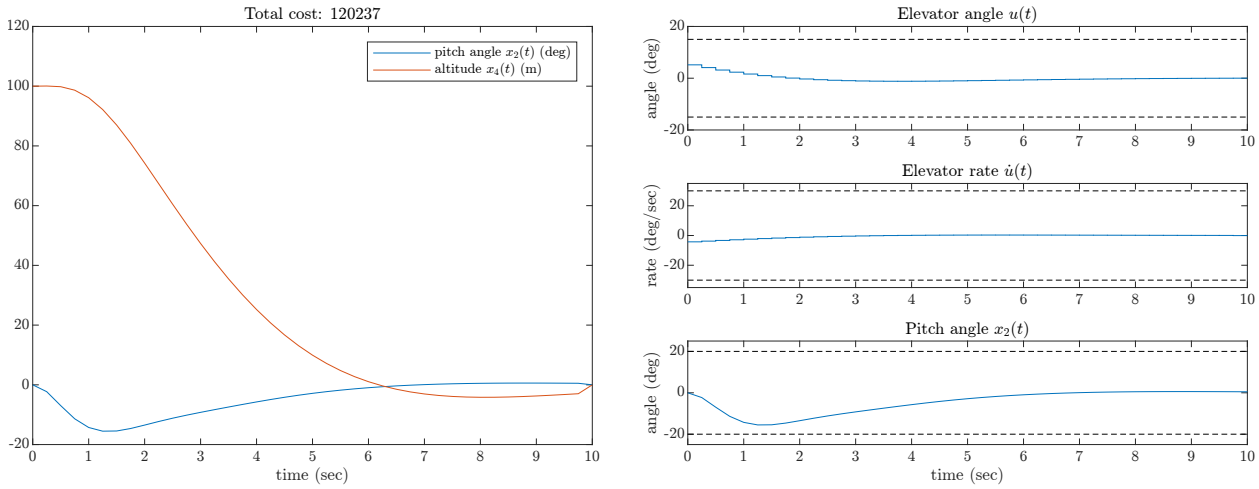


Figure 7: We can also satisfy all constraints by just using ordinary LQR but with $R = 10^6$. This is not a reliable approach (it's not clear how to pick R ahead of time), and here it leads to a very slow. A more reliable way to slow down the response and also guarantee that all constraints are satisfied is to just use a larger R with MPC!