# Introduction to Model Predictive Control

Josiah Wai

April 12, 2023

## 1 Introduction

Model Predictive Control (MPC) is an advanced control method that has been in use in several industries since the 1980's. At its heart, MPC reformulates the controller design into a mathematical optimization with an accompanying cost function and various types of constraints. At each timestep, the MPC controller simulates the system forward a certain amount of time and selects the set of inputs that leads to the optimal system trajectory. Then, the first step of the input sequence is implemented. The states of the system are estimated or measured and the process repeats.

Due to the requirement of solving an optimization problem at each timestep, MPC is computationally intensive. For this reason, it lends itself naturally to systems that evolve on slow time scales. Additionally, the complexity of MPC makes it preferable over simpler methods (proportional-integral-derivative (PID) or the linear-quadratic (LQ) framework) only in certain cases. However, there are distinct advantages of MPC:

First the optimization algorithm explicitly enforces inequality constraints. This is most directly related to the problem of input saturation in which an actuator hits its physical limitations (e.g., a valve cannot be more than 100% open). Additionally, there may be inequality constraints on the states due to, for example, safety or efficiency concerns. It is easy to see why MPC first arose in the chemical process control industry: these are slowly-evolving systems, with sensitive input constraints in which any improvement in controller efficiency directly contributes to the bottom line! Obviously, computational efficiency has since improved dramatically, allowing for MPC to be implemented on all types of systems that can benefit from the predictive optimization.

If the underlying system, cost function, and constraints are sufficiently "nice", the optimization can be simplified substantially. For example, a linear system model with a quadratic cost function- that is, the standard LQ model- combined with feasible inequality constraints will form a convex set. Then, the techniques of convex optimization can be applied to create the controller.

Convex optimization is a discipline in itself and there are many possible methods and algorithms to solve any particular problem. For this project, I focus on the first half of the formulation: the relation between MPC and convex optimization and how to convert a control problem into the standard convex optimization form. Two different methods are presented, and simulations were created using both methods. The MPC controllers are then compared to a benchmark LQR controller.

## 2 System Overview

For this MPC formulation, I start with a discrete-time state-space model:

$$x_{k+1} = Ax_k + Bu_k \tag{1}$$

$$y_k = Cx_k \tag{2}$$

In the simplest case, I assume that the system is controllable and observable. Furthermore, I assume that the input has been shifted so that when the system has reached steady-state $u = 0$.

The cost function under consideration is the infinite-series quadratic cost function fundamental to the LQ framework. The matrices D and G, along with the vectors d and g, form the constraints on the inputs and states.

$$\text{Minimize } \Phi(x,u) := \sum_{k=0}^{\infty} x_k^T Q x_k + u_k^T R u_k \tag{3}$$

$$\begin{aligned} \text{Subject to: } & x_{k+1} = Ax_k + Bu_k \\ & Du_k \leq d \\ & Gx_k \leq g \end{aligned} \tag{4}$$

Without the constraints, this is identically the discrete-time LQR formulation which can be solved with a Ricatti equation. However, difficulties arise when treating this through convex optimization, since there are an infinite number of decision variables and an infinite number of constraints.

Several different ways of handling the infinite summation are presented in [2, 3]. In the first method, we assume that there is an input $u = -Kx$ that can drive the system to the origin in a finite number of steps, N, after which the control is turned off so that $K = 0$.

$$\sum_{n=N}^{\infty} x_k^T Q x_k + u_k^T R u_k = \sum_{k=N}^{\infty} x_k^T Q x_k = x_N^T \bar{Q} x_N \tag{5}$$

The last equality (eliminating the summation) can be made if A is stable [3] (If A is unstable, replace A with $A_s$ from the from the Schur stability decomposition) and $\bar{Q}$ is identified through the Lyapunov equation:

$$\bar{Q} - A^T \bar{Q} A = Q \tag{6}$$

The objective is now in a tractable form:

$$\text{Minimize } \Phi = \sum_{k=0}^{N-1} (x_k^T Q x_k + u_k^T R u_k) + x_N^T \bar{Q} x_N \tag{7}$$

If N is too small this results in a suboptimal control solution, because the assumption that $K = 0$ after N steps is violated. In [2], the authors present the constrained LQR method which does not allow for suboptimal solution. The key is to recognize that if the constraints are inactive, the optimization is identically the LQR formulation at which point an LQR control can be implemented. Then N is adaptively selected to be $N = N_\infty$ where $N_\infty$ is the timestep at which all constraints become inactive. This has an efficiency advantage since $N_\infty$ is lower than the N at which K = 0.

The more modern method of this scheme [3] is to replace $\bar{Q}$ with that of the LQR solution, so that the system naturally relaxes to LQR when the constraints are inactive. $\bar{Q}$ is then defined by the discrete-algebraic Ricatti equation.

$$\bar{Q} = Q + A^T \bar{Q} A - (A^T \bar{Q} B)(R + B^T \bar{Q} B)^{-1}(B^T \bar{Q} A) \tag{8}$$

The formulation (eqns (7) and (4)) is now optimal if N is larger than $N_\infty$. The next step is to replace the summation term with a convex-optimization formulation. There several possible ways, but I will focus on two methods labelled here as traditional and efficient.

## 3 Traditional Formulation

Traditional methods use the state-transition 1 to eliminate the variables x and then optimize for the sequence of inputs $U := [u_k\ u_{k+1}\ \dots\ u_{N-1}]'$. Writing the current state as $x_k$ we predict the next states as:

$$\begin{aligned}
x_{k+1} &= Ax_k + Bu_k \\
x_{k+2} &= Ax_{k+1} + Bu_{k+1} \\
&= A(Ax_k + Bu_k) + Bu_{k+1} \\
&= A^2X_k + ABu_k + Bu_{k+1} \\
x_{k+3} &= Ax_{k+2} + Bu_{k+2} \\
&\vdots
\end{aligned}$$

Which can be efficiently written as:

$$X = Mx_k + VU_k \tag{9}$$

$$X := \begin{bmatrix} x_{k+1} \\ x_{k+2} \\ x_{k+3} \\ \vdots \\ x_N \end{bmatrix},\ M := \begin{bmatrix} A \\ A^2 \\ A^3 \\ \vdots \\ A^N \end{bmatrix},\ V := \begin{bmatrix} B & 0 & 0 & \dots & 0 \\ AB & B & 0 & \dots & 0 \\ A^2B & AB & B & \dots & 0 \\ \vdots & \vdots & \vdots & & \vdots \\ A^{N-1}B & A^{N-2}B & A^{N-3}B & \dots & B \end{bmatrix}$$

Defining new variables F, H, $\hat{Q}$ and $\hat{R}$,

$$\begin{aligned}
F &:= V^T\hat{Q}M \\
H &:= V^T\hat{Q}V + \hat{R}
\end{aligned}$$

$$\hat{Q} := \begin{bmatrix} Q & & & & \\ & Q & & & \\ & & Q & & \\ & & & \ddots & \\ & & & & \bar{Q} \end{bmatrix}, \; \hat{R} := \begin{bmatrix} R & & & & \\ & R & & & \\ & & R & & \\ & & & \ddots & \\ & & & & R \end{bmatrix}$$

The cost function (7) can be efficiently expressed as:

$$\Phi = U^T H U + 2x_k^T F^T U \tag{10}$$

Note that if there are no constraints, the optimal input sequence is $H^{-1}F$ which also gives the LQR solution. For m inputs and N predicted steps, this is an optimization for $N \times m$ decision variables, where the matrices in the objective and constraints are dense. This formulation was employed using a convex optimization solver [5] with results given later.

# 4 Efficient Formulation

Instead of creating an $N \times m$ dimensional dense optimization which requires $O(N^3)$ operations to solve, [3] proposes an interior point method that is $O(N \times (m+n)^3)$. In this formulation the state-transition equation (1) is kept as an equality constraint and the solver must optimize over more $(N \times (m+n))$ decision variables. The benefit is that the matrices in the objective and constraints are less dense. Additionally, if the original cost matrices Q and R are symmetric positive definite, special structure can be exploited to make the optimization $O(N)$.

## 4.1 MPC Formulation

In this case, we use the state equation to write:

$$\hat{A}X + \hat{B}U = \vec{0} \tag{11}$$

U is defined as before and X is modified slightly:

$$X := \begin{bmatrix} x_k \\ x_{k+1} \\ x_{k+2} \\ \vdots \\ x_N \end{bmatrix}, \ \hat{A} := \begin{bmatrix} A & -1 & 0 & 0 & \dots & 0 \\ 0 & A & -1 & 0 & \dots & 0 \\ 0 & 0 & A & -1 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & & \vdots \\ 0 & 0 & 0 & 0 & A & -1 \end{bmatrix}, \ \hat{B} := \begin{bmatrix} B & & & & \\ & B & & & \\ & & B & & \\ & & & \ddots & \\ & & & & B \end{bmatrix}$$

The resulting optimization (12) uses the inputs and states as decision variables and was similarly solved using [5].

$$\text{Minimize} \ \sum_{k=0}^{N-1} X^T \hat{Q} X + U^T \hat{R} U \tag{12}$$

$$\begin{aligned} \text{Subject to:} \ & X = \hat{A}X + \hat{B}U \\ & \hat{D}U \leq \hat{d} \\ & \hat{G} \leq \hat{g} \end{aligned} \tag{13}$$

## 5 Constraint Softening for Controller Robustness

In physical applications, input constraints are generally hard (they cannot be violated) whereas state constraints are less rigid, for example corresponding to a desired operating point. If there is no feasible solution to the given input and state constraints, rather than have the controller give up entirely, it would be preferable to violate the state constraints slightly.

This can be done by introducing the slack variable $\epsilon$ which is a vector with dimension equal to that of the state constraints vector d. We soften the constraints by modifying the constraints (4) to read

$$\begin{aligned} Hx_k - \epsilon &\leq h \\ \epsilon &\geq 0 \end{aligned} \tag{14}$$

And simultaneously adding a linear $z^T \epsilon$ and quadratic $\epsilon^T Z \epsilon$ penalties for the slack variables into the cost function

$$\Phi = \sum_{k=0}^{N-1} (x_k^T Q x_k + u_k^T R u_k + z^T \epsilon + \epsilon^T Z \epsilon) + x_N^T \bar{Q} x_N \tag{15}$$

The linear penalty is considered more important, and if the values in the vector z are sufficiently high the controller has the desirable property that, if possible, it will give the same solution as the hard-constraint formulation. If there is no solution that does not violate the constraints, the controller is robust in that the optimization still provides a solution.

# 6 Results

First, I simulated a system presented in [2], using my MPC formulations to replicate the constrained LQR results obtained. There is good agreement across the simulations.

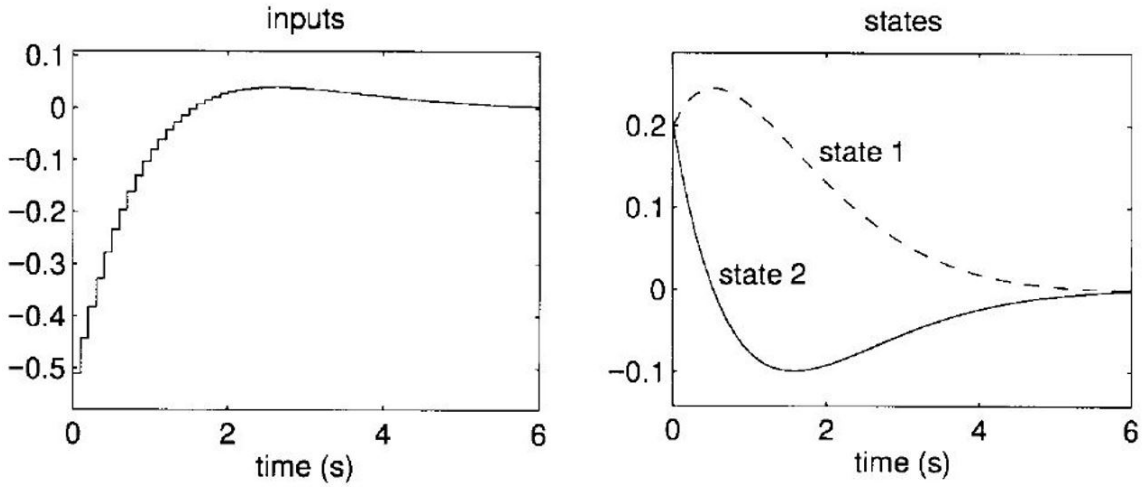


Figure 1: Constrained LQR results of [2]

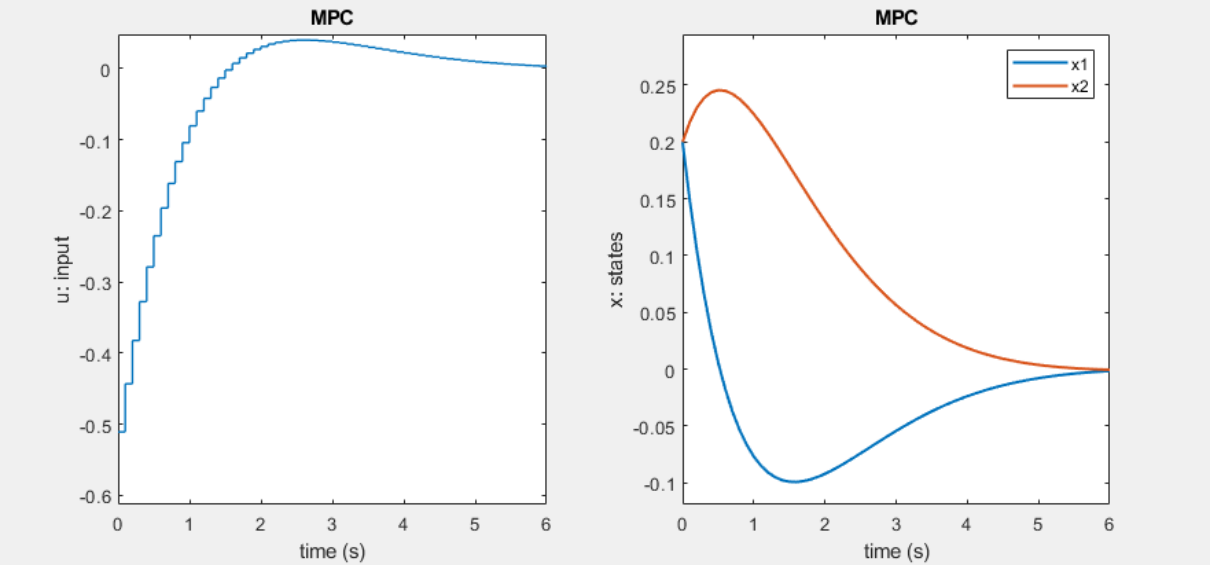


Figure 2: MPC results of the same system of Fig.1 showing good agreement in solvers.

For the remaining results, I focus on the system presented in [4] sampled at 1 Hz.

$$A = \begin{bmatrix} 1.1 & 2 \\ 0 & 0.95 \end{bmatrix},\ B = \begin{bmatrix} 0 \\ 0.0787 \end{bmatrix},\ C = \begin{bmatrix} -1 & 1 \end{bmatrix},\ D = 0 \tag{16}$$

$$x0 = \begin{bmatrix} 0.5 \\ -0.5 \end{bmatrix},\ Q = C^T C,\ R = 0.01;$$

### 6.1 No Constraints

Without constraints, the MPC formulations give exactly the same result as LQR. (Additionally both forms of MPC always give the same result, so that only one MPC result is shown.) There is clearly no advantage to MPC in this case.

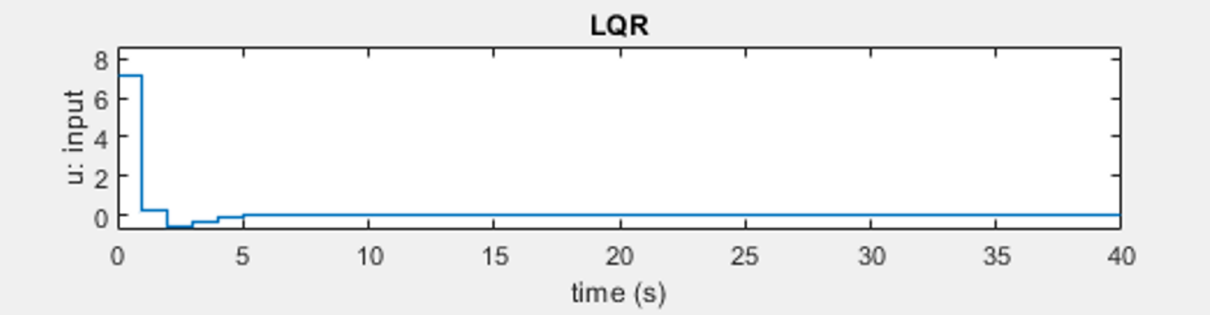


Figure 3: LQR inputs for the system (16) without constraints.

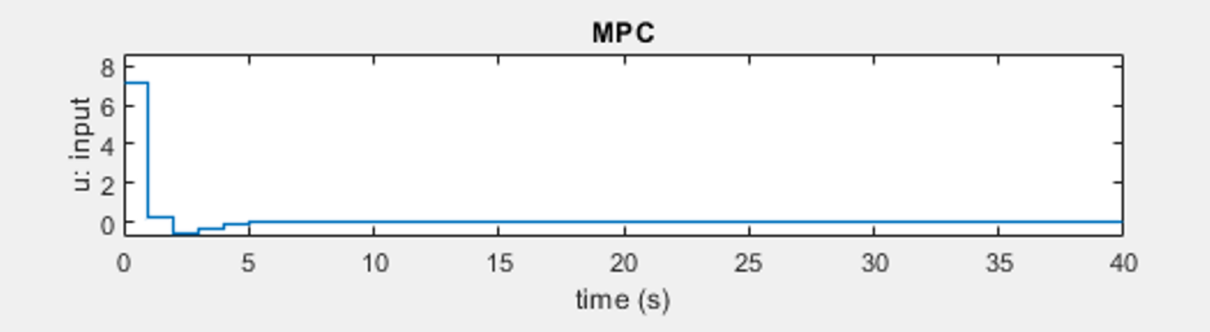


Figure 4: Without constraints the MPC inputs (and thus, states and outputs) are identical to LQR (Fig3).

### 6.2 Input Constraints $|u| \leq 1$

With input constraints, we see the motivating reason for MPC over LQR. Both controllers saturate in both directions, but LQR maintains saturation for too long and the output does not stabilize for a full 10 seconds longer than MPC.

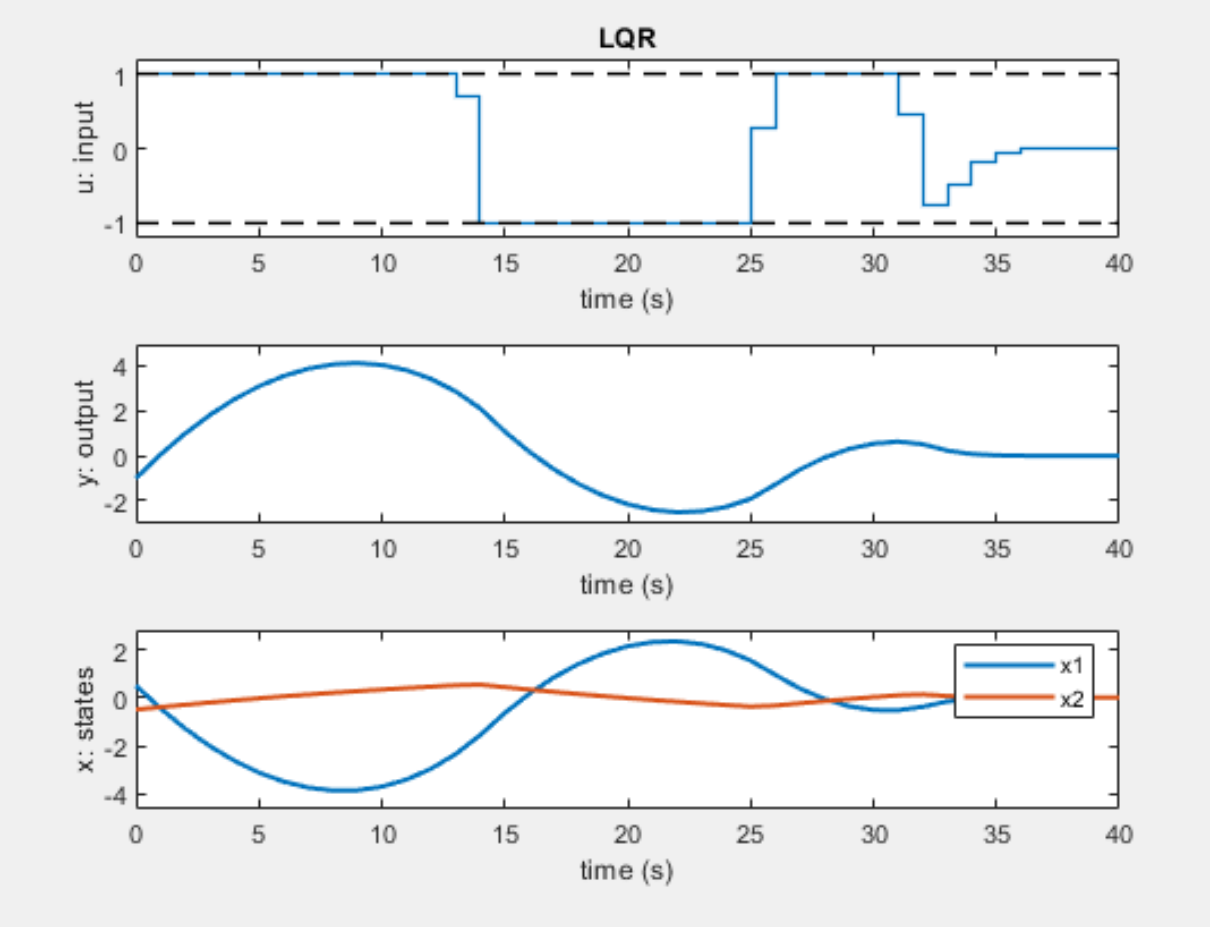


Figure 5: LQR with input constraints.

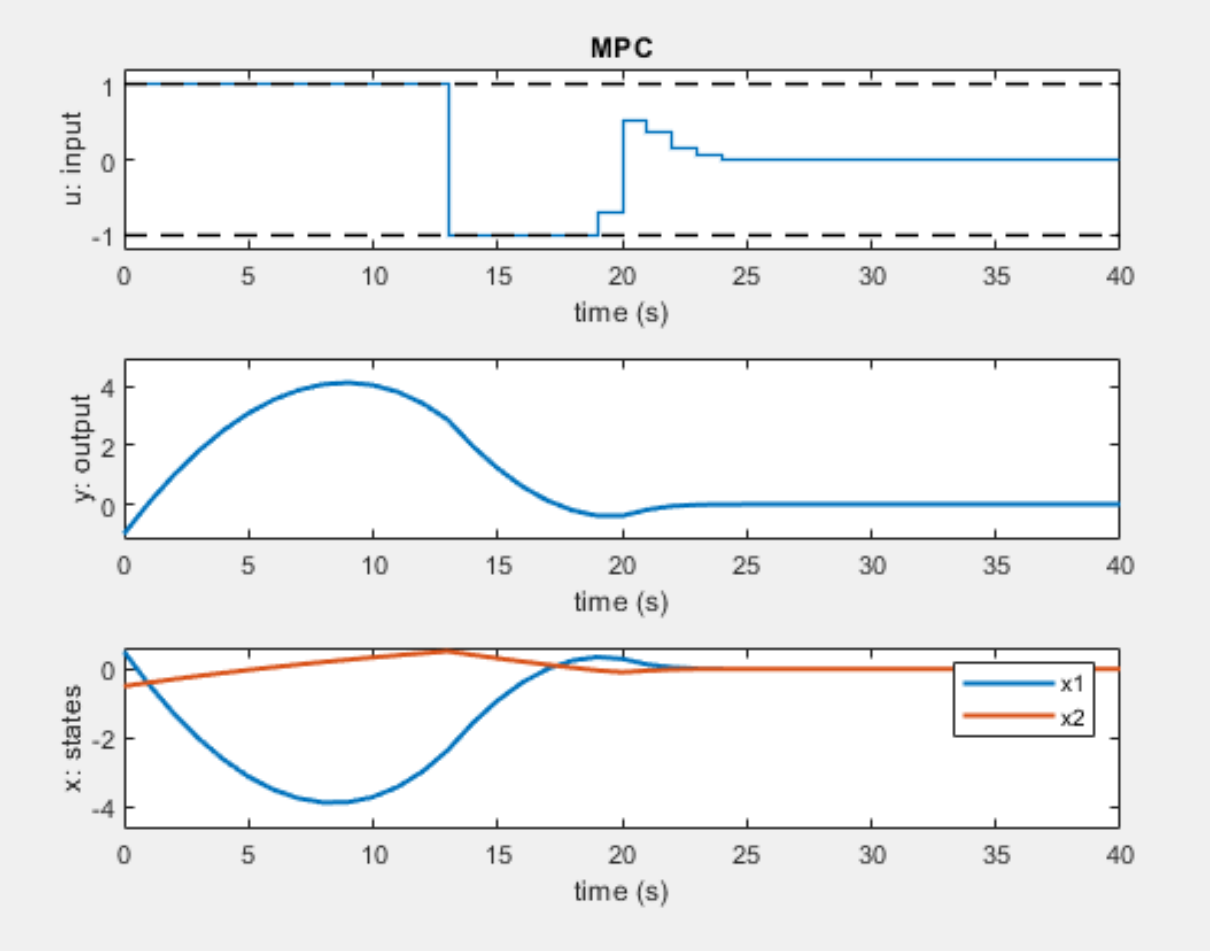


Figure 6: MPC with input constraints.

### 6.3 Hard State Constraints

There is no good way to enforce state constraints with LQR, so LQR results are not presented. Evaluating the previous results for MPC, we see that the maximum magnitude of $x_1$ is 0.51 whereas the maximum magnitude of $x_2$ is much larger at 3.86. Likely, the input is already trying to maximize its effect on the $x_2$. I expect that the controller may not be able to enforce additional constraints on $x_2$. Indeed, this is what occurs when attempting to implement a state constraint of $|x_{1,2}| \leq 3.6$: the optimization fails quickly and does not provide a solution.

However, since the controller was likely focused more on $x_2$, we can probably enforce a hard con-

straint on $x_1$, for example $x_1 \leq 0.35$. In this case, there is a feasible solution to the optimization.

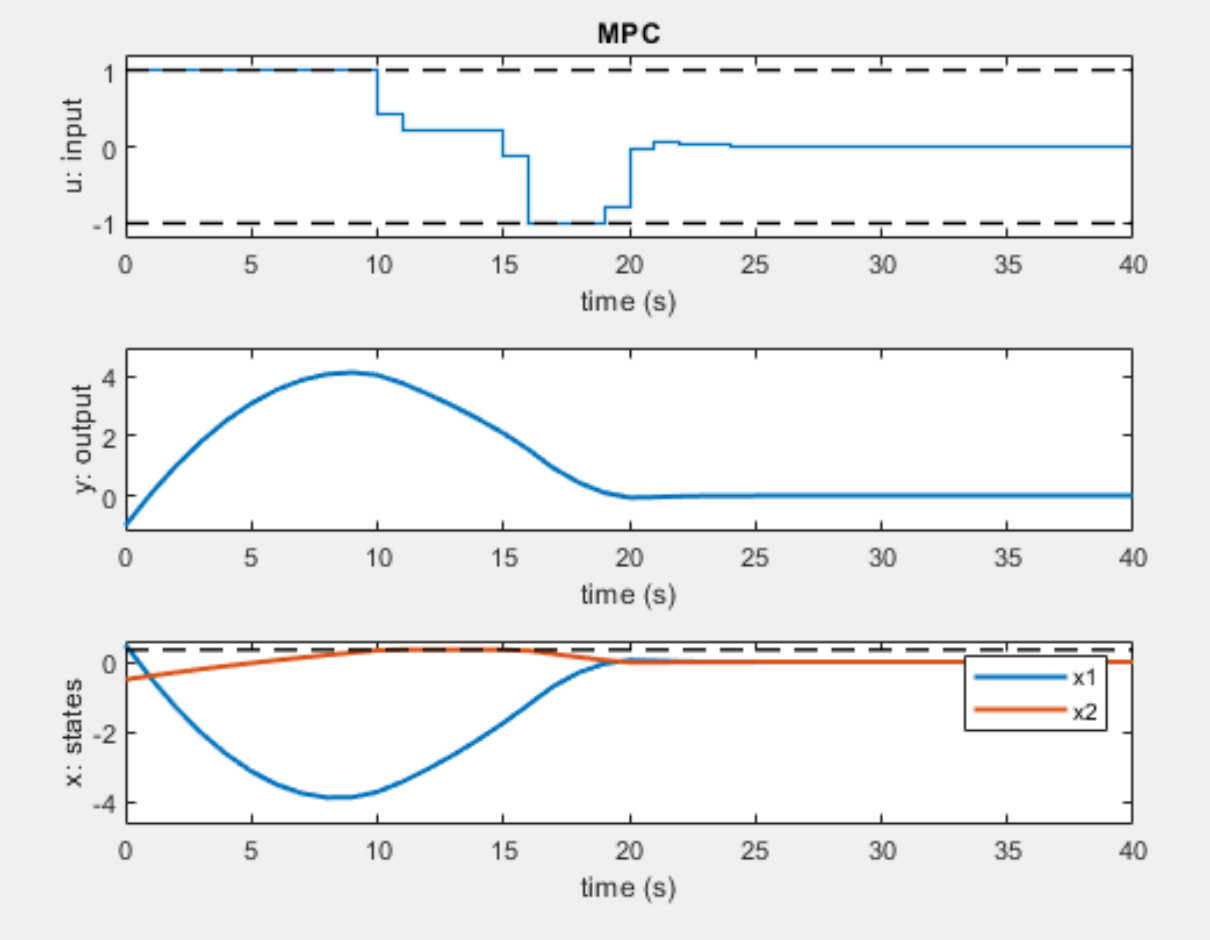


Figure 7: MPC with the hard state constraint $x_1 \leq 0.35$.

### 6.4 Soft State Constraints

I now simulate the soft state constraint approach, with the diagonal elements of Z and every element of z equal to 100. With the same constraints $|x_{1,2}| \leq 3.6$, the controller is more robust and provides an output, although the constraint was violated.

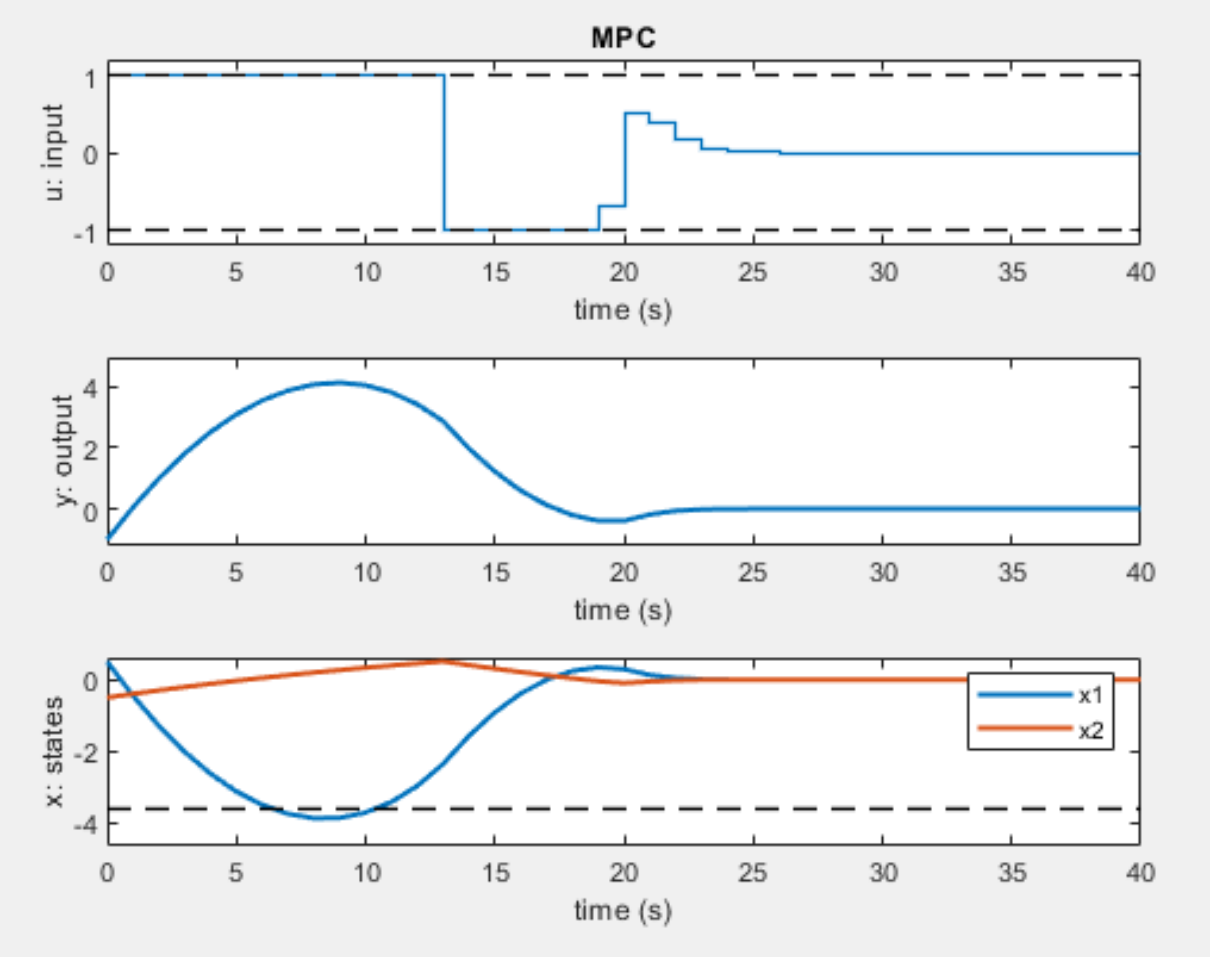


Figure 8: MPC with the soft state constraint $|x_2| \leq 3.6$. A solution is obtained even though the constraint is violated.

Now I simulate the soft-constraint approach to $x_1 \leq 0.35$, in which it was seen previously that the hard constraint problem was feasible. In this case, the elements of z and Z are sufficiently large so that the exact same solution is obtained.

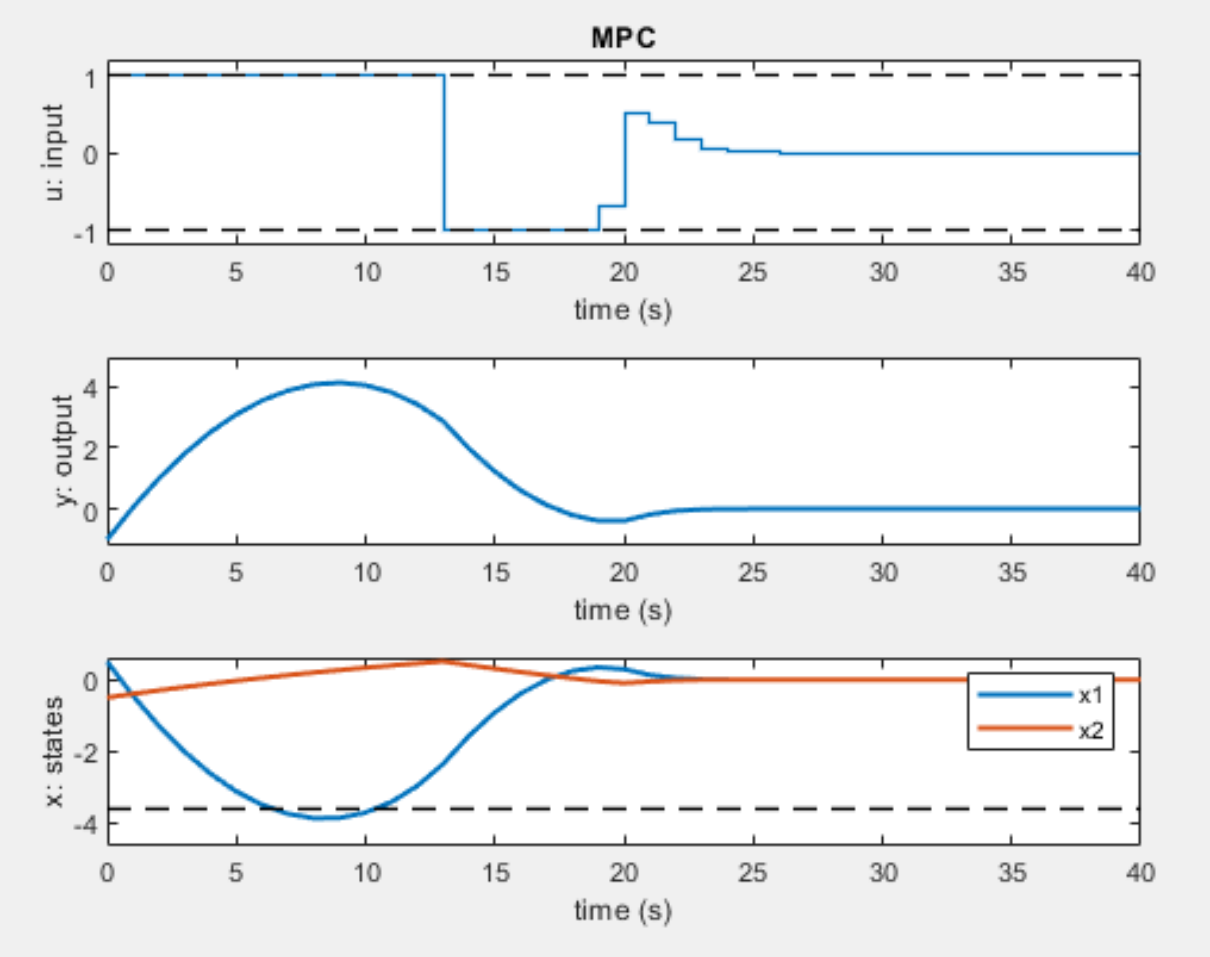


Figure 9: MPC with the soft state constraint $|x_1| \leq 0.35$. The solution is identical to the hard-constraint version in Fig. 7

## 7 Conclusion

The relationship between MPC, LQR, and convex optimization was explored. It was shown how to define an MPC problem into a standard convex optimization form, which can be solved by general optimization software such as cvx [5]. Simulations were presented comparing the results of MPC and LQR subject to varying types of input and state constraints.

This is an in-text citation [2].

## References


[1] J. Rawlings, "Tutorial Overview of Model Predictive Control," IEEE Control Systems Magazine, pp. 38-52, 2000.

[2] P. Scokaert and J. Rawlings, "Constrained Linear Quadratic Regulation," *IEEE Transactions on Automatic Control*, vol. 43, pp. 1163-1169, 1998.

[3] C. Rao, S. Wright, and J. Rawlings, "Application of Interior-Point Methods to Model Predictive Control", *Journal of Optimization Theory and Applications*, vol. 99 No. 3, pp. 723-757, 1998.

[4] "Solve Custom MPC Quadratic Programming Problem and Generate Code." *MathWorks. https://www.mathworks.com/help/mpc/ug/solve-custom-mpc-quadratic-programming-problem-and-generate-code.html*. 10 May 2019.

[5] M. Grant and S. Boyd, CVX [Computer software]. Retrieved from http://cvxr.com/cvx/