CART-PENDULUM SYSTEM STABILIZATION VIA MPC CONTROL SCHEMES

AEROSP 740 Project Report

Abstract

Model Predictive Control (MPC) schemes are applied to a cart-pendulum system to evaluate their stabilization capabilities. Three stabilization scenarios are considered: stabilization pendulum to an upwards position from an initial angle, cart movement while stabilizing the pendulum and wind-up maneuvers (moving pendulum from downwards position to an upwards one).

Juan Augusto Paredes Salazar

UMID: 87881020

Email: jparedes@umich.edu

Table of Contents

1. Project Overview

The cart-pendulum system represents a classic control theory problem due to its interesting nonlinear behavior. While linear controllers can be used if the initial conditions are close to the origin and nonlinear controllers can be formulated to handle more taxing tasks, Model Predictive Controller (MPC) presents itself as an attractive alternative due to being capable of optimizing the input taking into account the system model and the possible state and input constraints that the user might consider necessary. In the present project, MPC will be used in three different scenarios (stabilization from initial angle, cart movement while keeping pendulum upwards and wind-up maneuver) and compared (when appropriate) to linear feedback schemes. Throughout this project, it is assumed that the whole state is always available for the MPC to use.

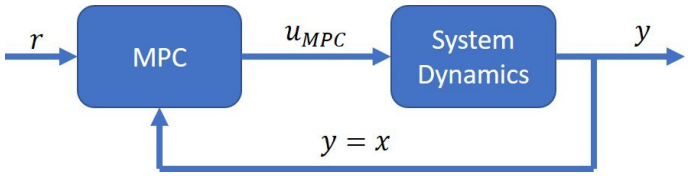
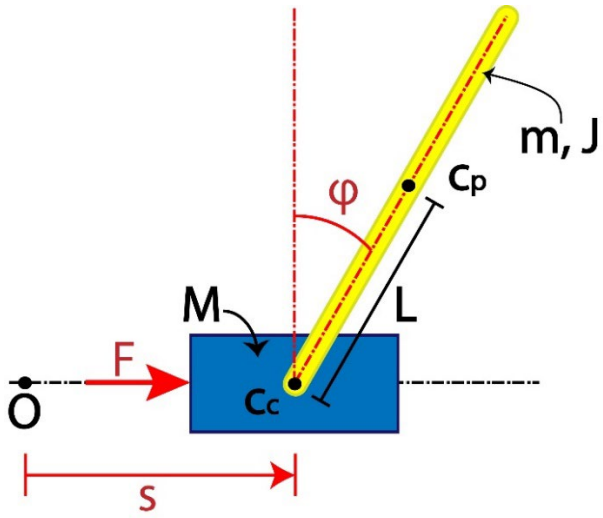


Figure 1.1.- MPC scheme and state feedback assumption.

2. Cart-Pendulum system dynamics

The cart-pendulum system is composed by a cart of mass M with a rod-shaped pendulum of mass m attached to its center of gravity. Furthermore, the pendulum's inertia around its center of gravity and in the direction perpendicular to the plane shown in Figure 1 will be denoted as J and the distance from this point to the center of gravity of the cart will be designated as L. The main variables considered in the dynamic system will be the angle between the pendulum and its upward position ϕ, the relative horizontal position of the cart with respect to the origin s and their respective derivatives $\dot{\phi}$ and $\dot{s}$. The only control force available will be a horizontal force F acting of the cart. The system diagram is displayed in Figure 1.1 along a table with the symbols used and their description.



Symbol	Description
s	Horizontal cart position from origin.
$\dot{s}$	Horizontal cart velocity.
ϕ	Angle between pendulum and its upwards position.
$\dot{\phi}$	Angular velocity of pendulum.
m	Mass of pendulum.
M	Mass of cart.
c_c	Center of gravity of cart.
c_p	Center of gravity of pendulum
L	Distance between c_c and c_p.
J	Inertia moment of pendulum at c_p in the direction perpendicular to the plane.
O	Horizontal origin.

Figure 2.1.- Cart-Pendulum system diagram with symbol diagram

The complete derivation of the dynamics of the cart-pendulum system will be displayed in Appendix 1 and will use the notation from the AEROSP 540 course taught at the University of Michigan and Dr. Bernstein's book [1]. The resulting differential equation used to represent these dynamics will be shown below. Considering the state vector $x = [x_1 \quad x_2 \quad x_3 \quad x_4]^T = [s \quad \dot{s} \quad \phi \quad \dot{\phi}]^T$, and input $u = F$, then system dynamics can be expressed in the following manner:

$$\dot{x} = f(x,u) = \begin{bmatrix} x_2 \\ \dfrac{(J+mL^2)\,mL\,x_4^2\,\sin(x_3) - (mL)^2 g \sin(x_3)\cos(x_3) + (J+mL^2)u}{(M+m)(J+mL^2) - (mL)^2\cos^2(x_3)} \\ x_4 \\ \dfrac{(M+m)mg\,L\,\sin(x_3) - (mL\,x_4)^2\,\sin(x_3)\cos(x_3) - (mL\cos(x_3))\,u}{(M+m)(J+mL^2) - (mL)^2\cos^2(x_3)} \end{bmatrix}$$

It should be taken into account that $f(x,u)$ becomes zero for $u = 0$ and either $x = [0 \quad 0 \quad 0 \quad 0]^T$ or $x = [0 \quad 0 \quad \pi \quad 0]^T$, which correspond to the upwards and downwards position respectively; therefore, both of these can be considered equilibrium points. This remains true $\forall\, s = x_1 \in \mathbb{R}$, which implies that the system should be stabilizable at any given position of the cart, implying that linear controllers are capable of moving the cart while keeping the pendulum stabilized.

Finally, from now on, the parameters to be used will be the following:

$$m = 0.2\ kg, \qquad M = 1\ kg, \qquad L = 0.2\ m, \qquad J = \frac{1}{3}mL^2 \cong 0.0027\ kg.m^2$$

3. System linearization and equilibrium points analysis

The system can be linearized about a given state x_0 and an input u_0. State matrix A and input matrix B can be obtained in the following manner:

$$A = \begin{bmatrix} \frac{df_1}{dx_1} & \frac{df_1}{dx_2} & \frac{df_1}{dx_3} & \frac{df_1}{dx_4} \\ \frac{df_2}{dx_1} & \frac{df_2}{dx_2} & \frac{df_2}{dx_3} & \frac{df_2}{dx_4} \\ \frac{df_3}{dx_1} & \frac{df_3}{dx_2} & \frac{df_3}{dx_3} & \frac{df_3}{dx_4} \\ \frac{df_4}{dx_1} & \frac{df_4}{dx_2} & \frac{df_4}{dx_3} & \frac{df_4}{dx_4} \end{bmatrix}_{\substack{x=x_0 \\ u=u_0}} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{df_2}{dx_3} & \frac{2\,(J+mL^2)\,mL\,x_4\,\sin(x_3)}{(M+m)(J+mL^2) - (mL)^2\cos^2(x_3)} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{df_4}{dx_3} & \frac{-2\,(mL\,)^2\,\sin(x_3)\cos(x_3)\,x_4}{(M+m)(J+mL^2) - (mL)^2\cos^2(x_3)} \end{bmatrix}_{\substack{x=x_0 \\ u=u_0}}$$

In which:

$$\frac{df_2}{dx_3} = \frac{(J+mL^2)\,mL\,x_4^2\,\cos(x_3) - (mL)^2 g(\cos^2(x_3) - \sin^2(x_3))}{(M+m)(J+mL^2) - (mL)^2\cos^2(x_3)} \ldots$$

$$- \frac{\big((J+mL^2)\,mL\,x_4^2\,\sin(x_3) - (mL)^2 g\sin(x_3)\cos(x_3) + (J+mL^2)u\big)(2(mL)^2\sin(x_3)\cos(x_3))}{\big((M+m)(J+mL^2) - (mL)^2\cos^2(x_3)\big)^2}$$

$$\frac{df_4}{dx_3} = \frac{(M+m)mg\ L\ \cos(x_3) - (mL\ x_4)^2\ (\cos^2(x_3) - \sin^2(x_3)) + (mL\sin(x_3))\ u}{(M+m)(J+mL^2) - (mL)^2\cos^2(x_3)} \dots$$

$$- \frac{\big((M+m)mg\ L\ \sin(x_3) - (mL\ x_4)^2\ \sin(x_3)\cos(x_3) - (mL\cos(x_3))\ u\big)(2(mL)^2\sin(x_3)\cos(x_3))}{\big((M+m)(J+mL^2) - (mL)^2\cos^2(x_3)\big)^2}$$

Afterwards, performing the same procedure to obtain the B matrix:

$$B = \begin{bmatrix} \frac{df_1}{du} \\ \frac{df_2}{du} \\ \frac{df_3}{du} \\ \frac{df_4}{du} \end{bmatrix}_{\substack{x=x_0 \\ u=u_0}} = \begin{bmatrix} 0 \\ \frac{(J+mL^2)}{(M+m)(J+mL^2) - (mL)^2\cos^2(x_3)} \\ 0 \\ \frac{mL\cos(x_3)}{(M+m)(J+mL^2) - (mL)^2\cos^2(x_3)} \end{bmatrix}_{\substack{x=x_0 \\ u=u_0}}$$

For the equilibrium point $x_0 = [0 \quad 0 \quad 0 \quad 0]^T$ (upwards position) and $u_0 = 0$, the linearized system is the following:

$$\delta\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{-(mL)^2 g}{J\,(M+m) + mML^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{(M+m)mg\ L}{J\,(M+m) + mML^2} & 0 \end{bmatrix} \delta x + \begin{bmatrix} 0 \\ \frac{(J+mL^2)}{J\,(M+m) + mML^2} \\ 0 \\ \frac{mL}{J\,(M+m) + mML^2} \end{bmatrix} \delta u$$

Obtaining the characteristic polynomial of the linearized system:

$$p(\lambda) = \det(\lambda I - A) = \lambda^2 \left(\lambda^2 - \frac{(M+m)mg\ L}{J\,(M+m) + mML^2} \right)$$

From this polynomial, it can be deduced that two of the eigenvalues will be zero, one of them will be positive and the other will be negative. Therefore, it can be stated that the upwards position constitutes an unstable equilibrium point.

For the equilibrium point $x_0 = [0 \quad 0 \quad \pi \quad 0]^T$ (downwards position) and $u_0 = 0$, the linearized system is the following:

$$\delta\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & \frac{-(mL)^2 g}{J\,(M+m) + mML^2} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{-(M+m)mg\ L}{J\,(M+m) + mML^2} & 0 \end{bmatrix} \delta x + \begin{bmatrix} 0 \\ \frac{(J+mL^2)}{J\,(M+m) + mML^2} \\ 0 \\ \frac{mL}{J\,(M+m) + mML^2} \end{bmatrix} \delta u$$

Obtaining the characteristic polynomial of the linearized system:

$$p(\lambda) = \det(\lambda I - A) = \lambda^2 \left(\lambda^2 + \frac{(M+m)mg\ L}{J\,(M+m) + mML^2} \right)$$

From this polynomial, it can be deduced that two of the eigenvalues will be zero and the rest will lie in the imaginary axis. Therefore, it can be stated that the downwards position constitutes a stable (in the sense of Lyapunov) equilibrium point.

The linearization around the upwards position $x_0 = [0 \quad 0 \quad 0 \quad 0]^T$ will be used (after a corresponding discretization) when designing linear MPC controllers and obtaining terminal cost matrices through discrete LQR. It should be noted that this linearized state space representation is controllable through u, a property which should be kept after discretization if linear MPC is to succeed or to obtain a feasible terminal cost matrix.

4. Pendulum stabilization to upwards position from initial angle

The objective in this scenario is to stabilize the system at $x = [0 \quad 0 \quad 0 \quad 0]^T$ from a starting point $x_0 = [0 \quad 0 \quad \phi_0 \quad 0]^T$, as show in figure 4.1. A way of doing this is to use the linearized dynamics to design a linear feedback controller (through LQR for example) that will successfully stabilize the pendulum up to a given ϕ_0. The proposed method will involve the usage of a LQ-MPC scheme based around the discretized linearized system described in section 3 and the methods described in [2]. (NOTE: The code used in this section can be found in Appendix 2). The LQ-MPC problem can be stated as such:

$$\min_{u_0,\dots,u_{N-1}} J_N = \sum_{k=0}^{N-1} x_k^T Q x_k + u_k^T R u_k + x_N^T P x_N \,, subject\ to \begin{matrix} x_{k+1} = Ax_k + Bu_k \\ x_0 = Current\ State \\ x_{min} \le x \le x_{max} \\ u_{min} \le u \le u_{max} \end{matrix},$$

in which J_N is the cost function to minimize, A and B are the matrices corresponding to the discretized state space model of the system, $x \in \mathbb{R}^{n_x}$, $u \in \mathbb{R}^{n_u}$, x_0 is the state at which the MPC is evaluated, x_{max} and x_{min} are the state constraints, u_{max} and u_{min} are the input constraints, Q is the state cost matrix, R is the input cost matrix, P is the terminal cost matrix and N is the prediction horizon. This problem will yield a sequence of inputs $u_0, \dots, u_{N-1}$ that minimized J_N in the given prediction horizon, although only u_0 will be used in that iteration since in the next one the LQ-MPC problem will be solved again and yield another sequence of inputs. This problem will be shaped into a quadratic programming (QP) problem of the form:

$$\min_{u_0,\dots,u_{N-1}} \mathcal{J}_N = \frac{1}{2} U^T H U + q^T U, subject\ to \begin{matrix} GU \le W + Tx_0 \\ q = Lx_0 \end{matrix},$$

that can be solved iteratively by a QP solver.

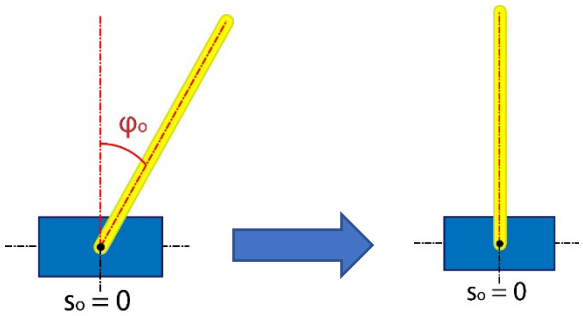


Figure 4.1.- Stabilization from initial angle.

Considering $X = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} \epsilon \mathbb{R}^{N \cdot n_x}, U = \begin{bmatrix} u_0 \\ \vdots \\ u_{N-1} \end{bmatrix} \epsilon \mathbb{R}^{N \cdot n_u}$ and rewriting J_N in matrix form:

$$J_N = X^T \begin{bmatrix} Q & 0 & \cdots & 0 & 0 \\ 0 & Q & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & Q & 0 \\ 0 & 0 & \cdots & 0 & P \end{bmatrix} X + U^T \begin{bmatrix} R & 0 & \cdots & 0 & 0 \\ 0 & R & \ddots & \ddots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & \ddots & \ddots & R & 0 \\ 0 & 0 & \cdots & 0 & R \end{bmatrix} U = X^T \bar{Q} X + U^T \bar{R} U$$

Furthermore, representing X in matrix form:

$$X = \begin{bmatrix} A_d \\ A_d^2 \\ \vdots \\ A_d^N \end{bmatrix} x_0 + \begin{bmatrix} B & 0 & \cdots & 0 \\ AB & B & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ A^{N-1}B & A^{N-2}B & \cdots & B \end{bmatrix} U = M x_0 + S U$$

Using this result in J_N:

$$J_N = U^T (S^T \bar{Q} S + \bar{R}) U + 2 x_0^T M^T \bar{Q} S U + x_0^T M^T \bar{Q} M x_0$$

Therefore, to minimize J_N with a QP solver, $\mathcal{J}_N$ takes the following form:

$$\mathcal{J}_N = \frac{1}{2} U^T (2 S^T \bar{Q} S + 2\bar{R}) U + (2 x_0^T M^T \bar{Q} S) U = \frac{1}{2} U^T H U + q^T U$$

In which:

$$\therefore H = 2 S^T \bar{Q} S + 2\bar{R}, \qquad q = 2 S^T \bar{Q}^T M x_0 = L x_0$$

Obtaining the LQ-MPC constraints in matrix form:

$$\begin{matrix} U \leq U_{max} \\ -U \leq -U_{min} \end{matrix} \quad \begin{matrix} X \leq X_{max} \\ -X \leq -X_{min} \end{matrix} \implies \begin{matrix} M x_0 + S U \leq X_{max} \\ -M x_0 - S U \leq X_{min} \end{matrix} \implies \begin{matrix} S U \leq X_{max} - M x_0 \\ -S U \leq -X_{min} + M x_0 \end{matrix}$$

$$\begin{bmatrix} S \\ -S \\ \mathbb{I} \\ \mathbb{I} \end{bmatrix} U \leq \begin{bmatrix} x_{max} \\ -x_{min} \\ u_{max} \\ -u_{min} \end{bmatrix} + \begin{bmatrix} -M \\ M \\ 0 \\ 0 \end{bmatrix} x_0$$

$$\therefore G = \begin{bmatrix} S \\ -S \\ \mathbb{I} \\ \mathbb{I} \end{bmatrix}, \qquad W = \begin{bmatrix} x_{max} \\ -x_{min} \\ u_{max} \\ -u_{min} \end{bmatrix}, \qquad T = \begin{bmatrix} -M \\ M \\ 0 \\ 0 \end{bmatrix}$$

Finally, the input u_0 to be used can be obtained from the obtained sequence of inputs in the following manner:

$$\therefore u_0 = IMPC \cdot U = \begin{bmatrix} \mathbb{I}^{n_u \times n_u} & 0^{n_u \times (N-1) n_u} \end{bmatrix} U$$

In this manner, the LQ-MPC problem can be solved (if the current conditions allow a feasible solution to be found) by any QP solver. For simplicity's sake, a simple dual projected gradient algorithm will be used to solve the problem iteratively as shown in [3].

Since matrices Q and R were chosen arbitrarily, their choice won't be discussed in the test section. The terminal cost matrix P was chosen as the solution to the DARE when solving the discrete LQR using the same Q and R matrices. Furthermore, all simulations of the MPC will be run with a sampling time T_s of

0.01 seconds and a prediction horizon of 20. Also, input constrained will be enforced, that is, if the controller input violates these constraints, then it is bounded.

Figure 4.2 shows the result of running an unconstrained simulation from an initial angle $\phi_0 = 0.7\ rad$. As it can be observed, both the Linear MPC and the LQR show similar behaviors, which is due to the choice in terminal cost matrix in the MPC case. Figure 4.3 shows the simulation results in which the input is constrained such that $|u| < 10\ N$ from the same initial angle of $\phi_0 = 0.7\ rad$. In this case, both ultimately reach the desired position, but it is noticeable that the MPC reaches the solution in a much more calm and fast manner. Finally, figure 4.4 shows the simulation results in which the input is constrained such that $|u| < 30\ N$ from the initial angle of $\phi_0 = \frac{\pi}{2}\ rad$. While the MPC controller is capable of stabilizing the system even from such a large initial angle, the LQR is unable to stabilize it.

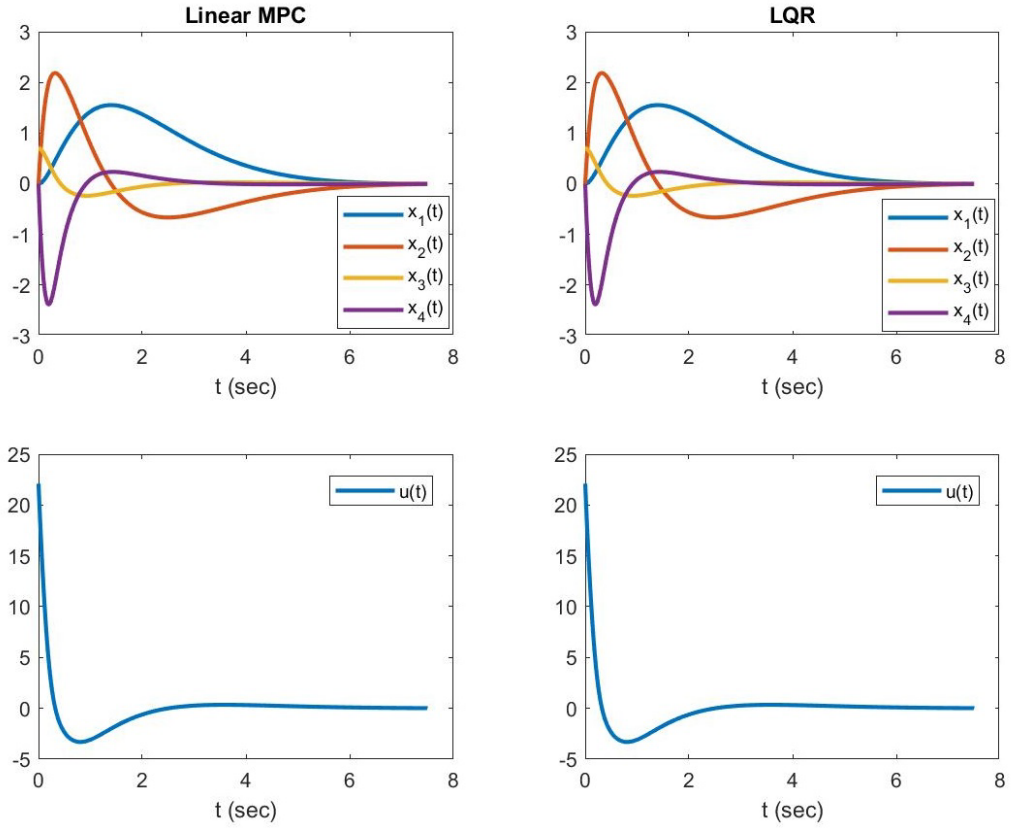


Figure 4.2.- Linear MPC vs LQR simulation, with no input constraint and with $\phi_0 = 0.7\ rad$.

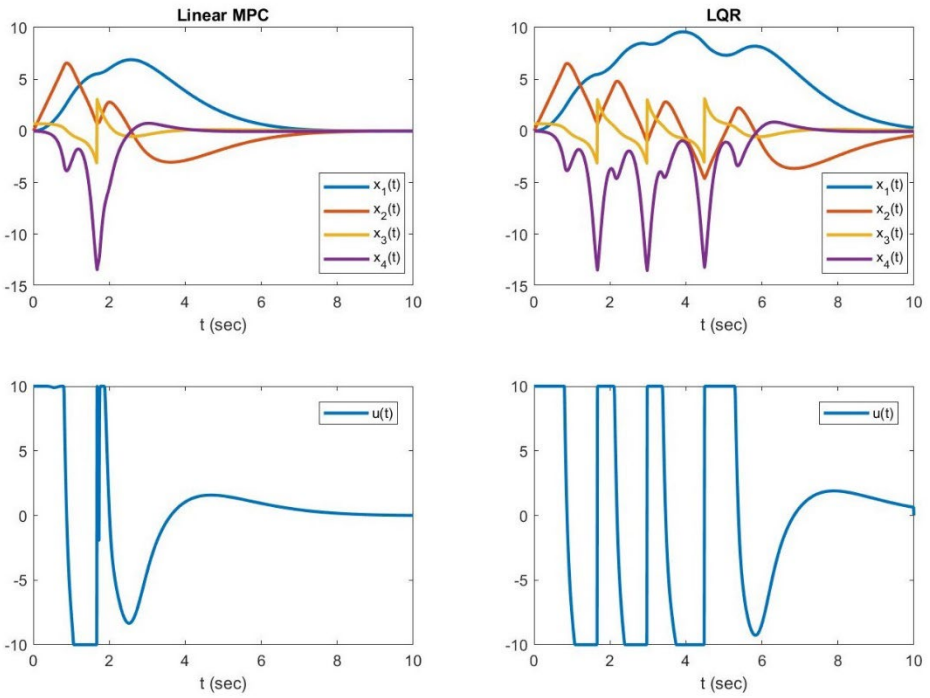


Figure 4.3.- Linear MPC vs LQR simulation, with input constraint $|u| < 10\ N$ and $\phi_0 = 0.7\ rad$.

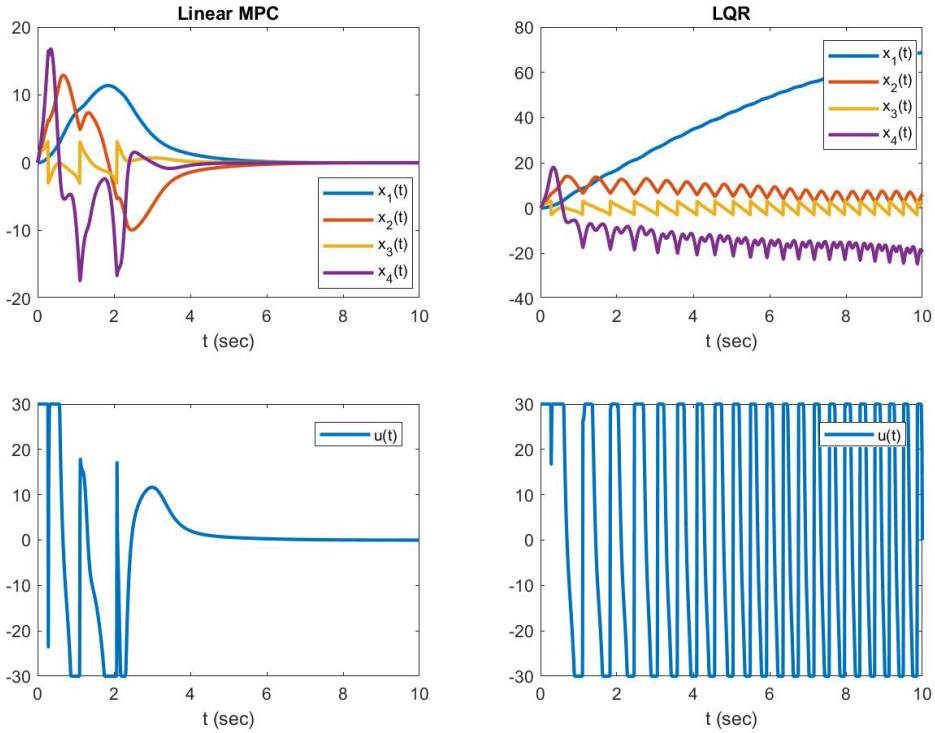


Figure 4.4.- Linear MPC vs LQR simulation, with input constraint $|u| < 30\ N$ and $\phi_0 = \frac{\pi}{2}\ rad$.

It can be concluded that Linear MPC can be setup such that it performs as well as a discrete LQR under no constraints and to perform better than it under strict constraints. This first experience revealed some

advantages of Linear MPC with respect to a well-known linear feedback scheme. The next scenarios pose more complex tasks and will prove to be more interesting.

5. Cart movement with pendulum stabilization

As illustrated in figure 5.1, The objective in this scenario is to move the cart to a new horizontal position while keeping the pendulum in an upwards position, which might seem like a trivial task to most humans; while this is true when using linear controllers for small distances, when traversing long distances most linear controllers will find it difficult to keep the pendulum upwards during the cart movement. In order to illustrate this problem, the discrete LQR controller and the LQ-MPC used in the last section will be used to try and move the cart without having the pendulum fall. (NOTE: The code used in this section can be found in Appendix 3).

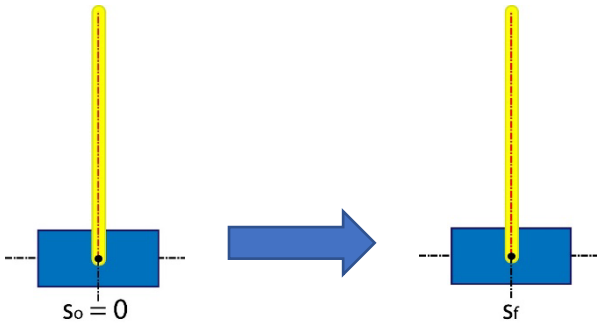


Figure 5.1.- Cart movement while keeping pendulum upwards.

A clarification should be made before proceeding regarding the LQ-MPC reference tracking scheme. Usually, in order to track a reference different from the origin, the discrete state space is augmented to a reference tracking form which increases the state dimensions and can be uncontrollable (which doesn't mean it can't be used to control the system states, although it may make obtaining a terminal cost matrix by solving a discrete LQR problem impossible). In this application, this form won't be used due to the many complications it poses. Instead, to enable reference tracking using the form LQ-MPC postulated in section 4, the QP solver will be "tricked" into believing the cart is not initially at the origin. That is, instead of feeding x_0 into the solver, $x_0 - r$ is used so that the solver moves the cart to its perceived origin, in which $x = r$. A similar approach is taken with the LQR controller, in which the feedback input becomes $u_{LQR} = -K_\infty(x - r)$.

Figure 5.2 shows the results of an unconstrained simulation with cart position goal $s_f = 15\ m$. As in the last section, when unconstrained, both controllers perform similarly. Figure 5.3 shows the results of a simulation with constrained input $|u| < 30\ N$ and $s_f = 35\ m$. Both controllers make the pendulum rotate, although the MPC controller required less swings to get to the goal. Finally, figure 5.4 shows a simulation with constrained input $|u| < 30\ N$ and $s_f = 50\ m$. While the MPC scheme required less swings and arrives at the desired position faster, the fact remains that both cause the pendulum to swing, even though the eventually arrive at the given destination. This happens even if you constrain the pendulum angle in the MPC. While this isn't dangerous to the system itself, if the cart-pendulum represented a Segway, for example, then the swings would be unacceptable. This also reveals the fact that, when implemented in the cart-pendulum system, even if the final state is the desired one, it is difficult to control how the states trajectory varies in time.

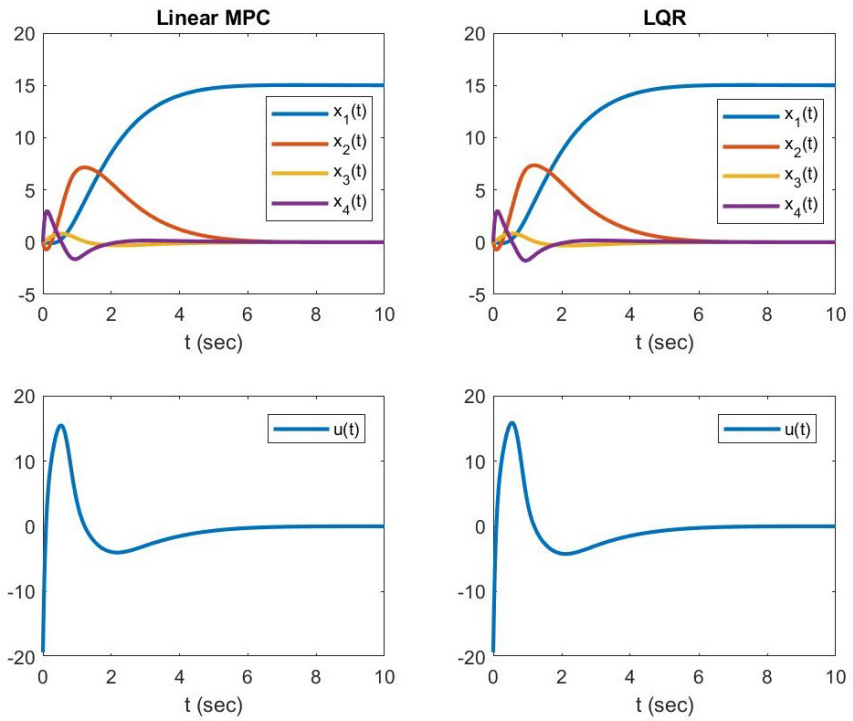


Figure 5.2.- Linear MPC vs LQR simulation, without input constraints and cart position goal $s_f = 15\ m$.

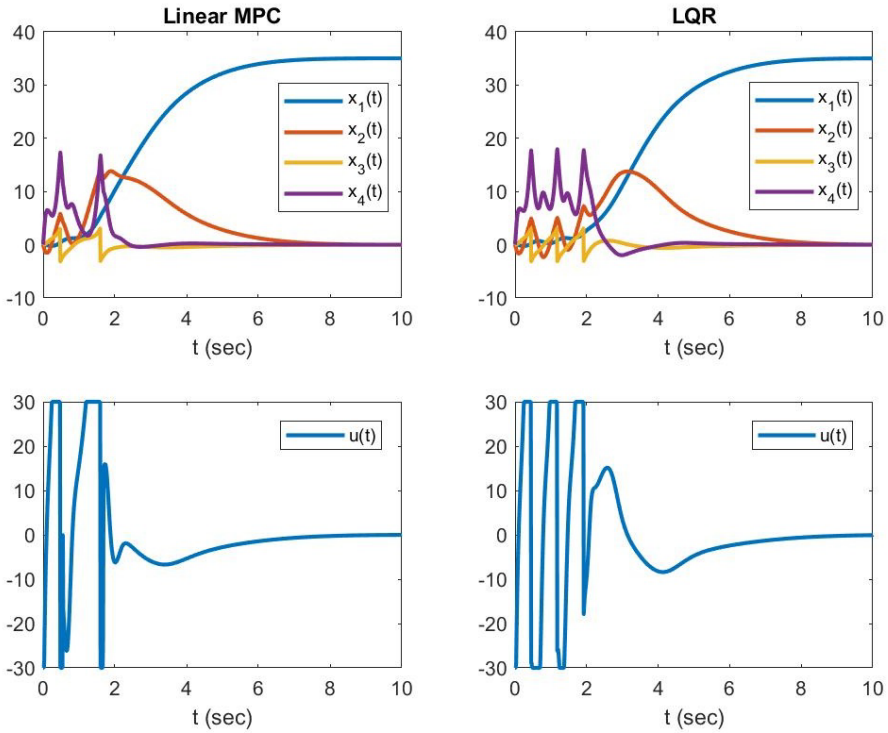


Figure 5.3.- Linear MPC vs LQR simulation, with constraint $|u| < 30\ N$ and position goal $s_f = 35\ m$.

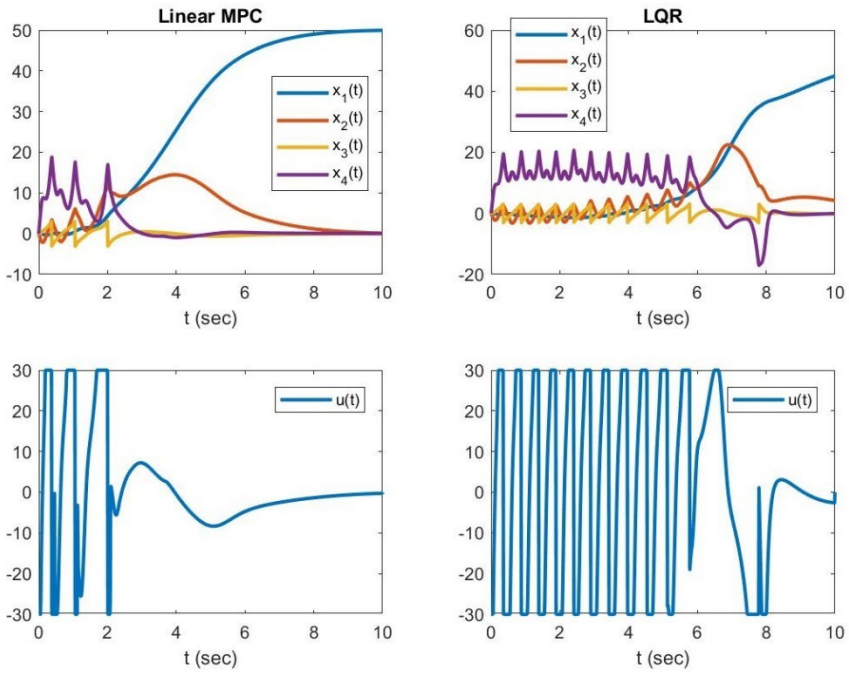


Figure 5.4.- Linear MPC vs LQR simulation, with constraint $|u| < 30\ N$ and position goal $s_f = 50\ m$.

In order to get further away from the origin without incurring in pendulum swings, a new scheme has to be adapted. The first option would be to employ a Nonlinear MPC scheme, which takes into account the nonlinearities of the system and plans ahead to prevent any constraint violation, which can be used to prevent the pendulum from straying away too far from the upwards position. A very prevalent issue in nonlinear MPC schemes revolves around the computational costs of its implementation and the time it takes each iteration to compute. Nonetheless, in a theoretical framework, this works well enough, as shown in figure 5.5, that displays a simulation in which the target position is $s = 50\ m$ and the input is constrained such that $|u| < 20\ N$. This was run using a prediction horizon $N = 30$ and the cost function matrices has to be tuned to allow good performance (which was very intuitive considering these matrices were chosen to be diagonal) since these would determine how well the optimizer would respect the system constraints. There is a small input constrain violation at the beginning and the position graph shows a slight overshoot, but overall it shows a much better performance than the linear controllers since no pendulum swings are required. However, a simpler solution can be implemented, which will be introduced in the next paragraph. The code used for the simulation displayed in figure 5.5 is the same used for the wind-up maneuver (with very slight modifications), so the nonlinear MPC formulation as well as the programming that went into it will be discussed in greater detail in the next section. (NOTE: From now on, input constraints will not be enforced to allows the controllers a greater margin of freedom).

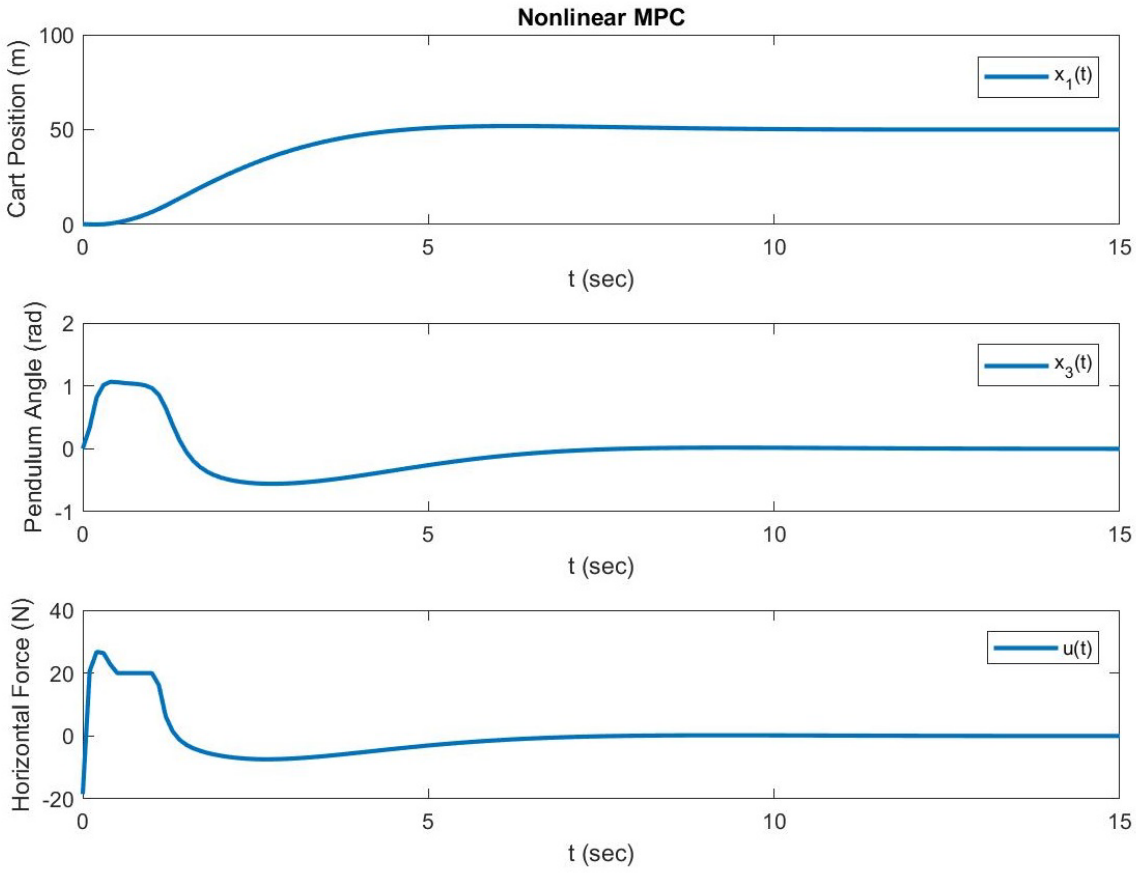


Figure 5.5.- Nonlinear MPC for cart position control with constraint $|u| < 30\ N$ and position goal $s_f = 50\ m$.

An alternative approach to solving the cart movement problem is to use a Reference Governor. Instead of directly controlling the system, the Reference Governor assumes that a known controller is already in place and control the reference being fed into it instead. Formally, the formulation of this scheme is fairly similar to the MPC one since it relies on the prediction of future system states until a given prediction horizon to prevent constraint violation. While Reference Governors can be implemented using nonlinear schemes, for this application, the linear case will be considered and coupled with a discrete LQR such that the pendulum angle is kept small enough to prevent it from escaping the region of attraction of the original controller. Figure 5.6 shows the scheme that will be considered for this implementation. The input of the reference governor will be the current state x (assuming all states can be used for feedback) and the original reference r and the output will be the controlled reference v.

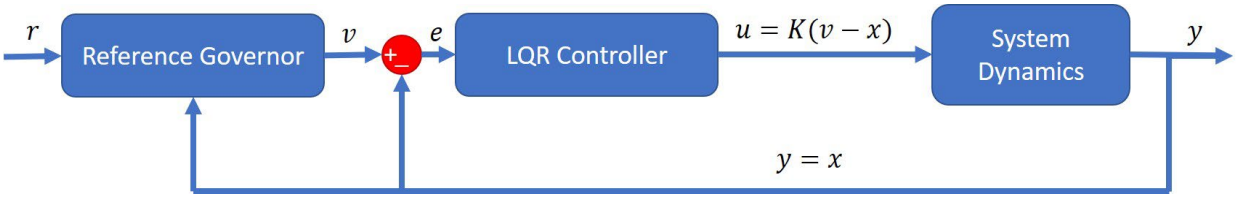


Figure 5.6.- Reference Governor scheme for cart movement

The scheme followed to implement the Reference Governor was obtained from [4]. Considering a discrete linearized system with a closed loop gain K (obtained from solving a discrete LQR problem) such that:

$$x_{k+1} = (A - BK)x_k + BKv = A_{rg}x_k + B_{rg}v$$

$$y_k = Cx_k + Dv$$

Then, taking into account set Y_c in which system constraints are satisfied, a constant v has to be computed such that the set $[x, v]^T$ stays within a Safe Set O_∞ within the prediction horizon, which is defined as follows:

$$O_\infty = \{(x(0), v) : v(t) \equiv v_r \Rightarrow y \in Y_c, \forall t \geq 0\}$$

In other words, v must be computed such that, applying it from an initial state $x(0)$, it remaining constant through the prediction horizon would imply that no constraints will be violated in the near future. Having established that, the Reference Governor can be formulated in the following fashion:

$$Maximize\ \kappa(t), \qquad subject\ to \quad \begin{matrix} v(t) = v(t-1) + \kappa(t)\,(r(t) - v(t-1)) \\ 0 \leq \kappa(t) \leq 1 \\ \begin{bmatrix} v(t) \\ x(t) \end{bmatrix} \in P \subseteq O_\infty \end{matrix}$$

It should be noted that if $\kappa(t) = 1$, then $v(t) = r(t)$ and $\kappa(t) = 0$, then $v(t) = v(t-1)$. Since the model and constraints ($x_{min} \leq x \leq x_{max}$) of the given problem are linear, then P is a Polyhedron and can be easily estimated offline.

Considering the linearized system described above, then the output can be represented by the following equation:

$$y_k = CA^k x(0) + C\left(I - A^k\right)(I - A)^{-1}Bv + Dv$$

Furthermore, defining set Y_c as a polytope such that $Y_c = \{y : Hy \leq h\}$, then P can be estimated up to a given prediction horizon N in the following manner:

$$P = \left\{ (v, x(0)) : \begin{bmatrix} HD & HC \\ HCB + HD & HCA \\ \vdots & \vdots \\ HC(I - A^N)(I - A)^{-1}B + HD & HCA^N \\ HC(I - A)^{-1}B + HD & 0 \\ K & -K \\ -K & K \end{bmatrix} \begin{bmatrix} v \\ x(0) \end{bmatrix} \leq \begin{bmatrix} h \\ h \\ \vdots \\ h \\ (1-\epsilon)h \\ u_{ub} \\ -u_{lb} \end{bmatrix} \equiv H_p \begin{bmatrix} v \\ x(0) \end{bmatrix} \leq h_p \right\}$$

While the terms in black ensure that the output within the prediction horizon remains inside polytope Y_c, the terms in blue allow small increments in v depending on the value of ϵ, while the terms in red represent an extension upon the regular Reference Governor conditions that attempt to bound the input such that $u_{lb} \leq u \leq u_{ub}$. These last input terms may be eliminated to make the computations more flexible.

At each iteration, the maximum $\kappa(t)$ must be determined such that $(v(t), x(0)) \in P$. This can be done in a simple manner by setting $\kappa(t) = 1$ at the beginning and diminishing this value until the given $v(t)$ satisfies the constraints in a determined prediction horizon N. Considering H_p and h_p can be computed offline, each iteration is performed in a fast manner. While the iteration procedure could be more

complicated and robust, the given algorithm will be proven to work well under the given system and circumstances.

Figure 5.7 shows a comparison between the performance of a single LQR controller and another one coupled to a reference governor with a prediction horizon of $N = 60$ (as mentioned before, constraints will no longer be enforced). While the single LQR was allowed to perform without having to worry about constraints, the reference governor was instructed to diminish the reference if the projected pendulum angle was higher than the set constraint or if the input required for the movement was too high (with the appended rows in the matrix). Even though the pendulum angle and the input presented some oscillation, no constraint was violated by the reference governor and the goal was reached without greater inconveniences.

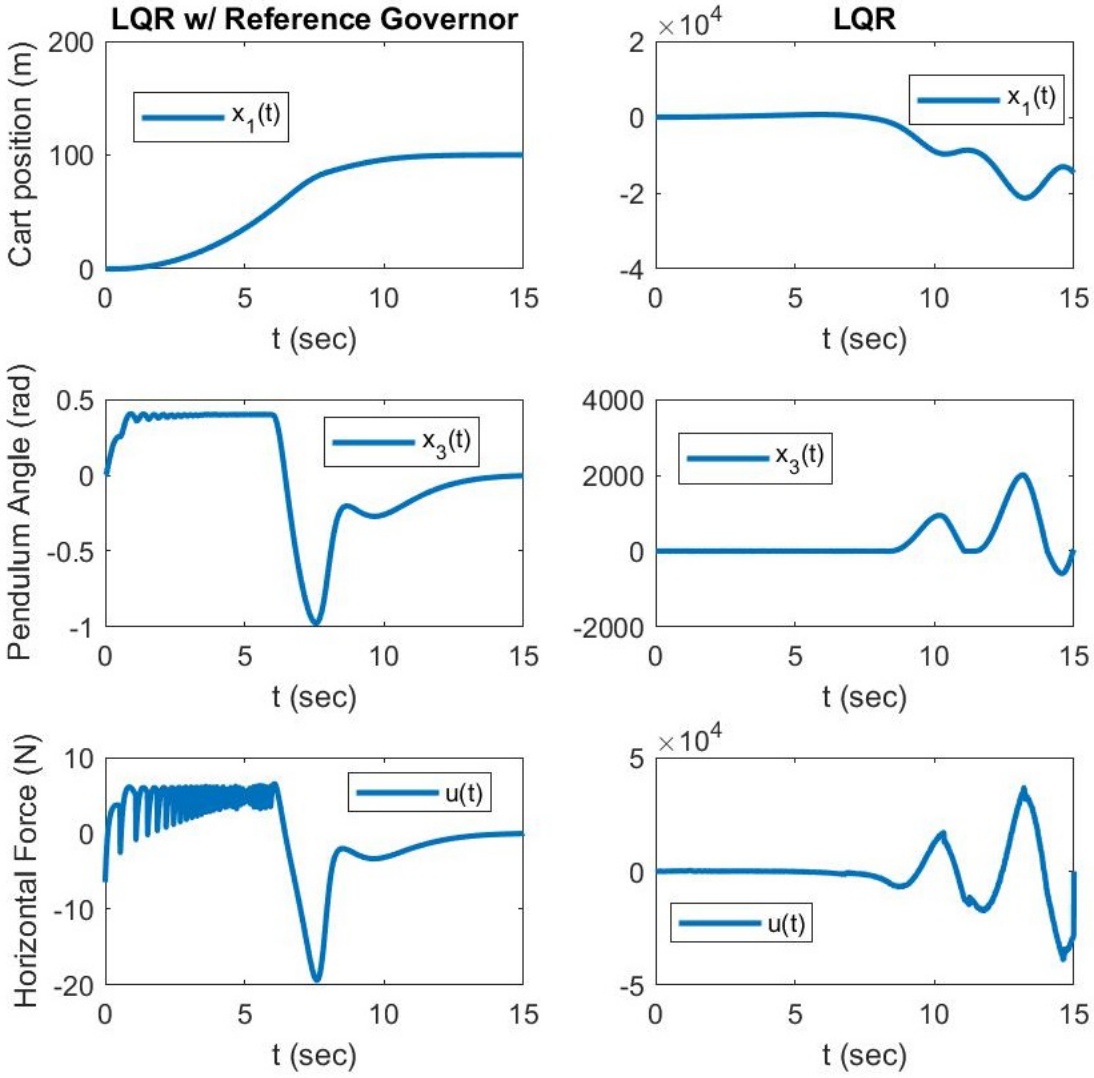


Figure 5.7.- LQR vs LQR w/ Reference Governor for cart position control with constraint $|u| < 20\ N$ (only on reference governor) and position goal $s_f = 100\ m$.

The input constraint in the reference governor must be used cautiously. While the safe zone constraints are imperative for system performance, constraining the input is performed in hopes of attaining a more optimal output. However, if these constraints are too tight, they may make the system unable to proceed and might be counterproductive for system performance. Regardless, even this simple application of the Reference Governor scheme has shown great promise. This scheme can be enhanced by considering the nonlinear dynamics and using Lyapunov functions or implementing a command governor, which allows control over the reference and the state feedback. For the moment being though, these results are satisfactory.

6. Wind-up maneuver

The wind-up maneuver consists of moving the pendulum from a downwards position to an upwards one, as shown in figure 6.1. This is an interesting problem since it requires the controller to move the system from a stable equilibrium point to an unstable one. Due to the nature of this maneuver, linear controllers would not normally be able to perform it (unless different linear controllers were used depending on the current state of the system by using, for example, fuzzy logic), which is the reason why a Nonlinear MPC approach will be preferred for this application. (NOTE: The code used in this section can be found in Appendix 4).

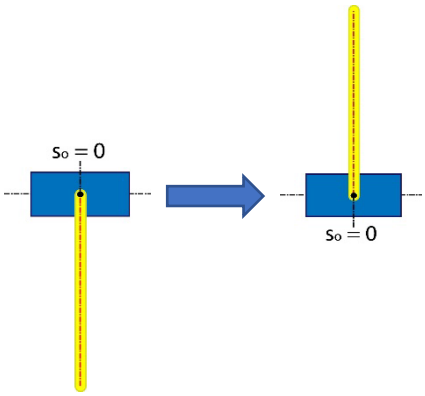


Figure 6.1.- Wind-up maneuver

For a state x_k that belongs to a sequence, its individual components will be expressed in the following manner:

$$x_k = [x_{k,1} \quad x_{k,2} \quad x_{k,3} \quad x_{k,4}]^T$$

The nonlinear MPC (NMPC) problem with slack variables (which can relax constraints to make the problem feasible) for the given system can be formulated (as shown in greater detail in [5]) as such:

$$\min_{x,u,s} \left\| \begin{matrix} x_{N,1} - x_{d,1} \\ x_{N,2} - x_{d,2} \\ \cos(x_{d,3}) - \cos(x_{N,3}) \\ x_{N,4} - x_{d,4} \end{matrix} \right\|_{Q_f}^2 + \sum_{i=0}^{N-1} \left\| \begin{matrix} x_{i,1} - x_{d,1} \\ x_{i,2} - x_{d,2} \\ \cos(x_{d,3}) - \cos(x_{i,3}) \\ x_{i,4} - x_{d,4} \end{matrix} \right\|_{Q}^2 + \|u_i - u_d\|_R^2 + \gamma s_{i+1}$$

$$s.t. \quad x_{i+1} = f_d(x_i, u_i), i = 0, \cdots, N-1,$$

$$p(u_i) \leq 0 \equiv u_i - u_{max} \leq 0, -u_i + u_{min} \leq 0, i = 0, \cdots, N-1$$

$$c(x_i) \leq 1s_i \equiv x_i - x_{max} \leq 1s_i, -x_i + x_{min} \leq 1s_i, i = 1, \cdots, N$$

$$-s_i \leq 0, i = 1, \cdots, N$$

In which the decision variables are:

$$x = [x_1 \quad \cdots \quad x_N]^T \in \mathbb{R}^{n_x \cdot N}, \qquad u = [u_1 \quad \cdots \quad u_N]^T \in \mathbb{R}^{n_u \cdot N}, \qquad s = [s_1 \quad \cdots \quad s_N]^T \in \mathbb{R}^N$$

Furthermore, the starting state x_0 is used as an input, x_d, u_d represent the target variables and, given the objective, will always remain $x_d = 0, u_d = 0$, and the effect of the slack variables is determined by the value of γ. Usually, state and input constraints are represented as nonlinear functions; however, in this case constraints are linear, which simplifies their expression to a simple inequality. Finally, $f_d(x_i, u_i)$ represents the discretized nonlinear dynamics of the system; since it would be too complicated to obtain this model analytically, it will be approximated through the explicit Euler approximation such that:

$$x_{k+1} = f_d(x_k, u_k) = x_k + T_s \cdot f(x_k, u_k),$$

in which $f(x_k, u_k)$ is the continuous dynamics function shown in section 2 and T_s is the sampling rate of the discrete system. It should be mentioned that the sampling rate used for solving the wind-up problem is $T_s = 0.1\ s$, unlike the $0.01\ s$ used on every single problem before this one. This allows the prediction horizon to be relatively small and allows the NMPC to predict the state of the system at a much later time into the future. This is useful when the controller has to plan maneuvers under constraints, which would theoretically stabilize at a future time that would require a large prediction horizon if the sampling rate was $0.01\ s$.

The most notable change from the regular NMPC scheme is that $\cos(x_3)$ is used in the cost functions. Considering that $x_d = 0$, this means that the function will try to minimize $1 - \cos(x_3)$, which will only achieve its lowest value when the pendulum is upwards. This is preferable to the regular cost function scheme, since it allows angle ϕ to grow beyond the range $(-\pi, \pi]$ rad without having to bound this value and minimizing it is similar to maximizing the potential energy of the pendulum (which can only be beneficial for our purpose).

This can be solved iteratively by shaping the problem into a Sequential Quadratic Programming (SQP), which can be stated as:

$$\min_z f(z)\,, such\ that \begin{matrix} g(z) = 0 \\ c(z) = 0 \end{matrix}, z = [u^T \quad x^T \quad s^T]^T$$

However, since general Nonlinear Programming (NLP) problems are difficult to solve, SQP can be solved iteratively by breaking it into small subproblems such that, at the k_{th} iteration, the following QP problem is solved:

$$\min_{\Delta z} \frac{1}{2} \Delta z^T H_k \Delta z + r_k^T \Delta z,$$

$$such\ that\ G_k \Delta z + g_k = 0, \qquad C_k \Delta z + c_k \leq 0,$$

in which $H_k = \nabla_z^2 L(z_k, \lambda_k, v_k), r_k = \nabla_z L(z_k), G_k = \nabla_z g(z_k), g_k = g(z_k), C_k = \nabla_z c(z_k), c_k = c(z_k)$. The following equations help define these functions:

$$J(x,u,s) = \left\| \begin{matrix} x_{N,1} - x_{d,1} \\ x_{N,2} - x_{d,2} \\ \cos(x_{d,3}) - \cos(x_{N,3}) \\ x_{N,4} - x_{d,4} \end{matrix} \right\|_{Q_f}^2 + \sum_{i=0}^{N-1} \left\| \begin{matrix} x_{i,1} - x_{d,1} \\ x_{i,2} - x_{d,2} \\ \cos(x_{d,3}) - \cos(x_{i,3}) \\ x_{i,4} - x_{d,4} \end{matrix} \right\|_{Q}^2 + \|u_i - u_d\|_R^2 + \Upsilon s_{i+1},$$

$$g(x,u) = \begin{bmatrix} x_1 - f_d(x_0, u_0) \\ \vdots \\ x_N - f_d(x_{N-1}, u_{N-1}) \end{bmatrix} = \begin{bmatrix} g_1 \\ \vdots \\ g_N \end{bmatrix},$$

$$h(x,u,s) = \begin{bmatrix} [p(u_0)^T & \cdots & p(u_{N-1})^T]^T \\ [c(x_1)^T & \cdots & c(x_N)^T]^T \\ [-s_1 & \cdots & -s_N]^T \end{bmatrix},$$

$$L = J + \lambda^T g + v^T h,$$

in which λ and v are dual variables that will satisfy the KKT conditions of the QP solved at each iteration. The iterative algorithm will require an initial guess for z_0, λ_0 and v_0 and the QP will yield optimized increments $\Delta z^*, \Delta\lambda^*$ and Δv^*, which must be added to the current values of z, λ and v respectively. The algorithm will end when the number of iterations exceed the imposed maximum or when the following condition is met:

$$\|F_{NR}\| = \left\| \begin{matrix} \nabla_z L \\ g \\ \min(-c, v) \end{matrix} \right\| \leq tolerance$$

During the first call of the algorithm, usually the values for z_0, λ_0 and v_0 can be set as zero. However, for posterior calls, it is recommendable to set them as the values obtained at the ending of the last completed run to reduce the amount of processing time.

Figure 6.2 shows a simulation of a Nonlinear MPC performing the wind-up maneuver. Both the states and the input were constrained such that $|u| \leq 30\ N$ and $|s| \leq 2.5\ m$. While the cart position constraint was violated, the input constraint wasn't, and a wind-up maneuver was performed in a relatively small amount of time. Constraint violation and problem feasibility depend directly on the cost function matrices used and the behavior of the system can be modified by tuning these. However, it should be taken into account that increasing the prediction horizon might not necessarily improve the performance of the system while performing this maneuver. This simulation was run using a prediction horizon $N = 60$, but later tests revealed that exactly the same results were obtained when using a prediction horizon $N = 100$, meaning that other factors would need to be changed to improve performance.

While the windup maneuver was successfully performed, tighter constraints will not allow the pendulum to move upwards with a single back and forth movement of the cart. Instead, the energy of the system will have to be built up in order to perform this maneuver completely and not remain in the downwards position. Under the current NMPC scheme, this would require an unfeasibly large prediction horizon N, since it may take a long time for the system to build up the required amount of kinetic and potential energy. An attempt was made using a prediction horizon $N = 500$ to no avail, since the controller would not obtain a feasible input and the simulation had to be stopped since it was taking too long to process each iteration and it was obvious that the system had become unstable. Instead, the NMPC scheme will be modified to what could be called "Switching NMPC" (SNMPC). The main idea behind this is to change

the cost function matrices Q and Q_f depending of how close the pendulum is to the upwards position. Therefore, two sets of cost function matrices will be used. The first set (corresponding to the first phase) will focus on building up the required energy to perform the wind-up, while the second one (corresponding to the second phase) will focus on finishing the stabilization process when the pendulum is near the target position (these will be the same as the ones used before).

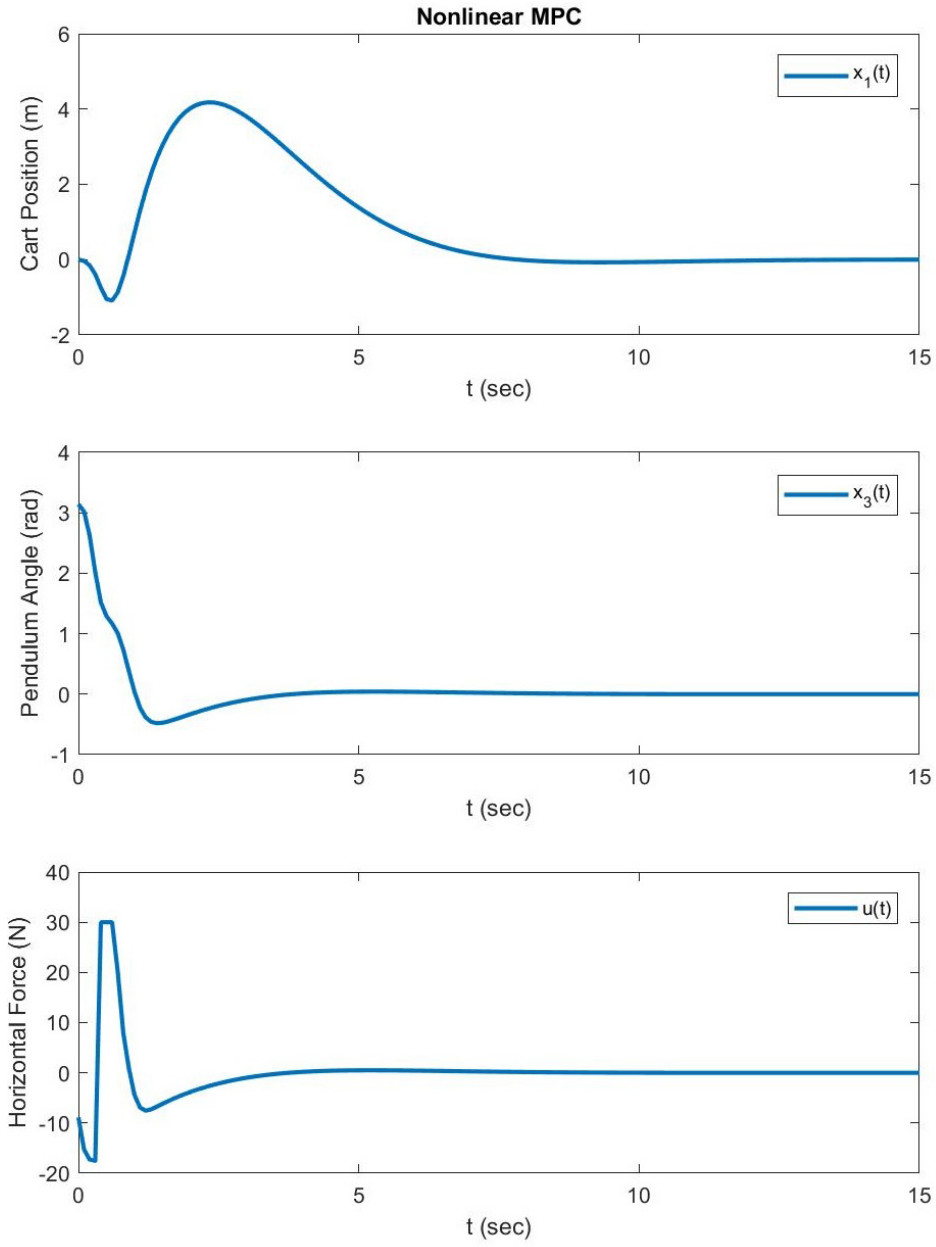


Figure 6.2.-Nonlinear MPC wind-up maneuver with constraints such that $|u| \leq 30\ N$ and $|s| \leq 2.5\ m$. This last constrained was violated during the simulation.

Since the second phase remains the same as the original NMPC scheme (although it could be changed to a linear MPC scheme due to the proximity to the upwards position at which the second phase would be activated), the first phase will be described more thoroughly, since just setting the Q and Q_f matrices is

not enough to obtain the desired behavior. In order to encourage the controller to build up energy, the matrices' diagonal term that affects state x_3 must be set positive and relatively high, while the diagonal term that affects state x_4 must be set negative and its absolute value should be relatively low. This second point is important since, even though it would be beneficial to increase the pendulum's angular velocity when it is passing through the downwards position, if the negative value is too high, then the returned input could become unfeasibly high since the optimizer may prefer increasing angular velocity infinitely to minimize the cost function depending on the current state.

Although it may seem contradicting at first, even though in the last paragraph certain precautions were taken for the solver to be able to converge to feasible solutions, the next part implemented assumes that, eventually, the solver will fail to provide one. The issue here is that once the cart is close to a cart position constraint and the pendulum is about to stop its rotation its rotation (cannot increase its height any longer in that direction), since the state cost matrix is set to maximize pendulum height and angular velocity, it is at this point that the optimizer will resolve to either keep this position by violating the cart position constraint and increasing cart acceleration greatly or increase cart speed violently in the opposite direction to maximize pendulum angular velocity, both of which result in input constraint violation. This can be detected and happens reliably at similar conditions each time. Therefore, when detected, the input is instead set to its maximum absolute value (as permitted by input constraints) in the direction of the cart position origin and the optimization duals that were passed after every iteration are reset to zero, except the last value of the input decision variable u, which is set equal to the chosen input value. This will be performed until, eventually, the state condition will allow the NMPC scheme to recover control and issue an input within the input constraints. This will happen as the cart goes back and forth until the pendulum angle is low enough for the second matrix cost to take over and stabilize the pendulum.

Lastly, a new cost function will be implemented such that the costs on pendulum states resemble more the kinetic and potential energy of this part of the system, such that the function to minimize takes the following shape:

$$\min_{x,u,s} V(x_N, x_d, Q_f) + \sum_{i=0}^{N-1} V(x_i, x_d, Q) + \|u_i - u_d\|_R^2 + \gamma s_{i+1}$$

$$s.t. V(x_k, x_d, Q) = \frac{1}{2} Q_{1,1}\left(x_{k,1} - x_{d,1}\right)^2 + \frac{1}{2} Q_{2,2}\left(x_{k,2} - x_{d,2}\right)^2 + Q_{3,3}\, mg\left(\cos\left(x_{d,3}\right) - \cos\left(x_{k,3}\right)\right) + \frac{1}{2} Q_{4,4} J \left(x_{k,4} - x_{d,4}\right)^2,$$

in which g is the gravitational constant and m and J are respectively the mass and the inertia moment around the center of gravity of the pendulum.

Figure 6.3 shows the result of a simulation running the SNMPC scheme with a prediction horizon of $N = 30$, with tight constraints such that $|u| \leq 10\ N$ and $|s| \leq 1\ m$. While the position constrain was violated, it must be notices that it did so during what is most likely the second phase of the controller. During the first phase, the cart stayed within the given boundaries and built up energy to, eventually, perform the jump required for the wind-up maneuver. Figure 6.4 shows a simulation with the exact same conditions save for the prediction horizon, which was increased to $N = 60$. In this scenario, it took the controller less swings to stabilize to the upwards position (although it seems that the second phase also causes the position constraint violation in this simulation). It seems that, even though greater prediction horizons

reduce the time it takes the system to stabilize to the upwards position, it isn't required, which is advantageous when implementing this sort of algorithms on real-time systems.

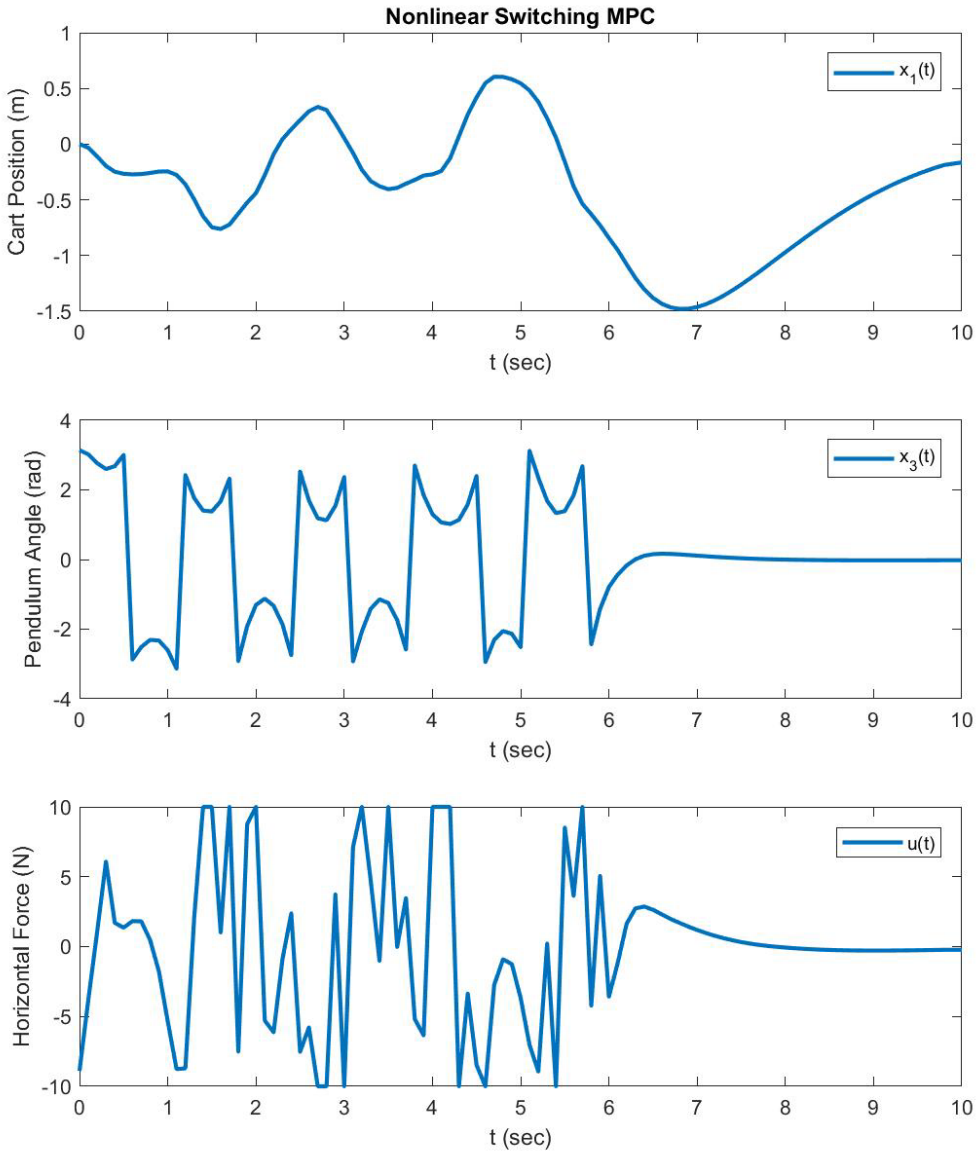


Figure 6.3.- Switching Nonlinear MPC with prediction horizon $N = 30$ and with constraints such that $|u| \leq 10\ N$ and $|s| \leq 1\ m$ (this last one is violated when stabilizing to the upwards position).

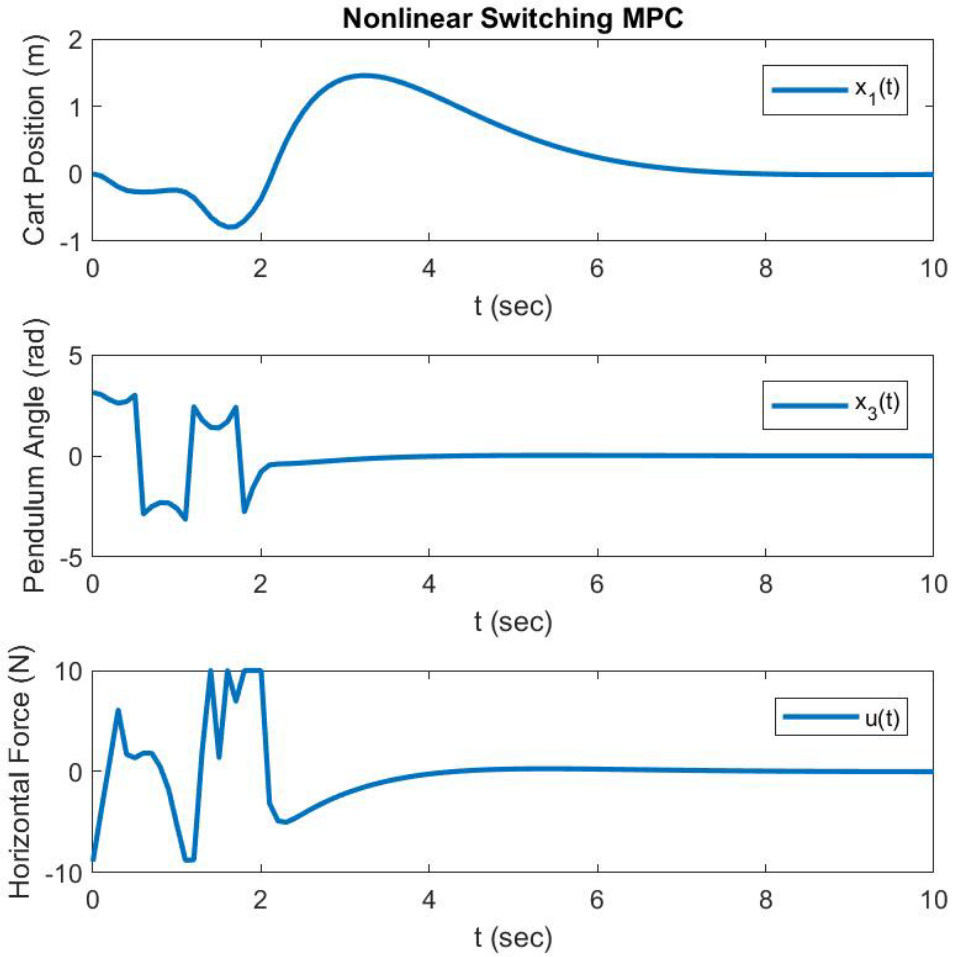


Figure 6.4.- Switching Nonlinear MPC with prediction horizon $N = 60$ and with constraints such that $|u| \leq 10\ N$ and $|s| \leq 1\ m$ (this last one is violated when stabilizing to the upwards position).

Even though this scheme works for the given parameters, it can't be guaranteed to solve the problem for any set of parameters and circumstances. Since a value of the cost matrix is negative, the QP subproblems cannot be guaranteed to be convex and thus the optimizers may exhibit unexpected behavior. Furthermore, the controller is based on observations made after running a small number of simulations, meaning that there may exist a set of conditions under which the wind-up maneuver cannot be solved by this scheme. Therefore, more testing should be performed before proceeding to implement this in a real-life situation.

7. Conclusions

- Linear Model Predictive Controllers (MPC) can reliably stabilize the pendulum to the origin under input constraints, even from high initial angles such as 90°, unlike linear controller that don't take this constraint into account.

- As demonstrated by the Linear MPC experience in the cart movement simulation, even though the ending state might be the desired one, it can be hard to guarantee constraint satisfaction along the trajectory depending on the circumstances while using the linear variant of the MPC.

- While Nonlinear MPC might extend the capabilities of Linear MPC, careful consideration should be taken into its application due to its computational requirements and difficult implementation.

- A Reference Governor can be used to extend the capabilities of linear controllers, allowing these to remain within state constraints and, thus, preventing the system state from escaping their region of attraction. As the experience with the cart movement simulation proved, its application is simple, effective and computationally light under the correct circumstances, even though one should be careful with the established constraints since these could render the reference ineffective.

- With sufficiently permissive constraints, the nonlinear MPC can reach a resolution for performing the wind-up maneuver in a single back and forth movement of the cart. For this to be possible, a sufficiently high predictive horizon must be considered.

- With sufficiently constrictive constraints, the nonlinear MPC will not be able to reach a resolution for performing the wind-up maneuver unless equipped with an unfeasibly high prediction horizon. Instead, it can be modified to prefer increase in system energy at first and then to prefer stabilization once the angle is close enough to zero. Even though this "Switching NMPC" isn't strictly a MPC, it provides a solution for the given problem under tight constraints.

Bibliography

[1] Bernstein, Dennis, *Geometry, Kinematics, Statics and Dynamics*, Lecture Notes from Intermediate Dynamics dictated at the University of Michigan in the Fall 2017 semester, updated on November 4, 2017.

[2] Kolmanovsky, Ilya, *Introduction to LQ Model Predictive Control*, Lecture Notes from Introduction to MPC course dictated at the University of Michigan in the Winter 2018 semester.

[3] Kolmanovsky, Ilya, *Numerical Methods for Solving Quadratic Programming Problems Online*, Lecture Notes from Introduction to MPC course dictated at the University of Michigan in the Winter 2018 semester.

[4] Kolmanovsky, Ilya, *Reference Governors for Control of Systems with Constraints*, Lecture Notes from Introduction to MPC course dictated at the University of Michigan in the Winter 2018 semester.

[5] Liao-McPherson, Dominic, *Module 6: On the implementation of nonlinear NMPC using SQP methods*, Lecture Notes from Introduction to MPC course dictated at the University of Michigan in the Winter 2018 semester.

Appendices

Appendix 1: System Dynamics Derivation

The notation from the AEROSP 540 course taught at the University of Michigan and Dr. Bernstein's book [1] will be used through the derivation of the dynamics system. However, the resulting differential equations will be expressed in terms of the system parameters, state variables and its only input as defined in section 2 of the report. Figure A.1 shows the system diagram and Table A.1 displays and defines important parameters and variables. A few changes and additions are shown in figure A.1. For example, origin O has been changed to fixed point w to keep congruency with the notation in [1]. Frames and extra vectors have also been defined. Frame $\boldsymbol{F_C}$ remain oriented with respect to the cart and, since it never rotates, it can be considered to be an inertial frame. Frame $\boldsymbol{F_P}$ will rotate with the pendulum such that the rotation between $\boldsymbol{F_C}$ and $\boldsymbol{F_P}$ is represented by angle ϕ around axis $\hat{k}_C$ (as represented by $F_C \xrightarrow[3]{\phi} F_P$). This means that the relationship between orthogonal axes $\hat{\imath}_C, \hat{\jmath}_C, \hat{k}_C$ and $\hat{\imath}_P, \hat{\jmath}_P, \hat{k}_P$ corresponding to both frames can be expressed as such:

$$\begin{bmatrix} \hat{\imath}_C \\ \hat{\jmath}_C \\ \hat{k}_C \end{bmatrix} = \begin{bmatrix} \cos\phi & \sin\phi & 0 \\ -\sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \hat{\imath}_P \\ \hat{\jmath}_P \\ \hat{k}_P \end{bmatrix}$$

Likewise, the angular velocity vector $\vec{\boldsymbol{\omega}}_{\boldsymbol{p/c}}$ which describes the rotation velocity between both frames can be expressed in terms of axis $\hat{k}_C$, such that $\vec{\boldsymbol{\omega}}_{\boldsymbol{p/c}} = \dot{\phi}\,\hat{k}_C$. Finally, as it is displayed in figure 1, the gravity vector $\vec{g}$ is aligned with the $\hat{\jmath}_C$ axis.

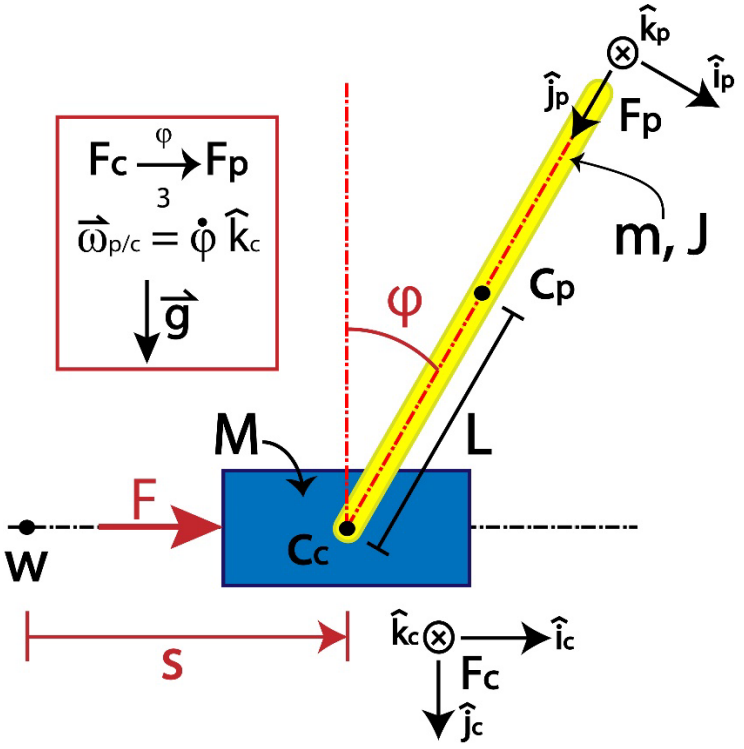


Figure A.1.- Cart-Pendulum system diagram. Notation used as seen in [1].

TABLE A.1.- Symbols and descriptions of parameters and variables used in dynamic system.

Symbol	Description
s	Horizontal cart position from origin.
$\dot{s}$	Horizontal cart velocity.
ϕ	Angle between pendulum and its upwards position.
$\dot{\phi}$	Angular velocity of pendulum.
F	Horizontal force applied to cart.
m	Mass of pendulum.
M	Mass of cart.
c_c	Center of gravity of cart.
c_p	Center of gravity of pendulum
L	Distance between c_c and c_p.
J	Inertia moment of pendulum at c_p in the direction perpendicular to the plane.
w	Fixed point embedded in an inertially nonrotating massive body.
F_C	Frame relative to cart orientation.
F_P	Frame relative to pendulum orientation.
$\vec{g}$	Gravity vector.
$\vec{\omega}_{p/c}$	Angular velocity vector of pendulum frame F_P with respect to the cart frame F_C.

The relative position vector between w and c_c, c_p each is represented by $\vec{r}_{c_c/w}$, $\vec{r}_{c_p/w}$ in the following manner:

$$\vec{r}_{c_c/w} = s\,\hat{\imath}_C, \qquad \vec{r}_{c_p/w} = s\,\hat{\imath}_C - L\,\hat{\jmath}_p = (L\sin\phi + s)\hat{\imath}_C - (L\cos\phi)j_C$$

Derivation of both terms by time yields their respective velocity vectors:

$$\vec{v}_{c_c/w} = \dot{s}\,\hat{\imath}_C, \qquad \vec{v}_{c_p/w} = \left(L\,\dot{\phi}\cos\phi + \dot{s}\right)\hat{\imath}_C + \left(L\,\dot{\phi}\sin\phi\right)j_C$$

The system dynamics will be obtained by using Lagrangian Dynamics. For that purpose, generalized coordinates s and ϕ are defined, since both can be used to describe the behavior of the entire system at all times. The next steps require both the kinetic and potential energy of the system to be defined in terms of both the generalized coordinates and their time derivatives. The kinetic energy of the system β (used to represent both the cart and the pendulum) with respect to inertial point w and inertial frame F_C is represented by $T_{\beta/w/C}$ and obtained in the following manner:

$$T_{\beta/w/C} = T_{pendulum/w/C} + T_{cart/w/C}$$

$$T_{\beta/w/C} = \frac{1}{2}m\left(\left(L\,\dot{\phi}\cos\phi + \dot{s}\right)^2 + \left(L\,\dot{\phi}\sin\phi\right)^2\right) + \frac{1}{2}J\dot{\phi}^2 + \frac{1}{2}M\dot{s}^2$$

$$T_{\beta/w/C} = \frac{1}{2}\left(J + mL^2\right)\dot{\phi}^2 + mL\,\dot{s}\,\dot{\phi}\cos\phi + \frac{1}{2}(m+M)\dot{s}^2$$

Then, the potential energy of system β with respect to inertial point w can be expressed as $U_{\beta/w}$ and obtained in the following manner:

$$U_{\beta/w} = U_{pendulum/w} + U_{cart/w} = -m\vec{r}_{c_p/w}\cdot\vec{g} + 0 = mgL\cos\phi$$

Since no energy dissipation occurs within the system and no friction is considered as part of the model, knowing that external force F is applied to the cart in the direction of the $\hat{\imath}_C$ axis, then the Langrange Equations can be built using previously obtained expressions for each of the generalized coordinates.

For s:

$$\frac{d}{dt}\frac{d}{d\dot{s}}\left(T_{\beta/w/C}\right) - \frac{d}{ds}\left(T_{\beta/w/C} - U_{\beta/w}\right) = F\hat{\imath}_C \cdot \frac{d}{ds}\left(s\hat{\imath}_C\right)$$

$$\frac{d}{dt}\left(mL\dot{\phi}\cos\phi + (m+M)\,\dot{s}\right) - (0) = F$$

$$mL\ddot{\phi}\cos\phi - mL\dot{\phi}^2\sin\phi + (m+M)\ddot{s} = F \qquad \text{(Equation A1.1)}$$

For ϕ:

$$\frac{d}{dt}\frac{d}{d\dot{\phi}}\left(T_{\beta/w/C}\right) - \frac{d}{d\phi}\left(T_{\beta/w/C} - U_{\beta/w}\right) = F\hat{\imath}_C \cdot \frac{d}{d\phi}\left(s\hat{\imath}_C\right)$$

$$\frac{d}{dt}\left((J+mL^2)\,\dot{\phi} + mL\,\dot{s}\,\cos\phi\right) - \left(-mL\,\dot{\phi}\,\dot{s}\sin\phi + mgL\sin\phi\right) = 0$$

$$(J+mL^2)\,\ddot{\phi} + mL\,\ddot{s}\,\cos\phi - mL\,\dot{s}\,\dot{\phi}\,\sin\phi + mL\,\dot{s}\,\dot{\phi}\,\sin\phi - mgL\sin\phi = 0$$

$$(J+mL^2)\,\ddot{\phi} + mL\,\ddot{s}\,\cos\phi - mgL\sin\phi = 0 \qquad \text{(Equation A1.2)}$$

From Equation A1.2, expressions for $\ddot{\phi}$ and $\ddot{s}$ can be obtained, although these depend on $\ddot{s}$ and $\ddot{\phi}$ respectively. Using both of theses expressions to replace their correspondent variables in Equation A1.1 yields the following expressions after some rearranging:

$$\therefore \ddot{s} = \frac{(J+mL^2)mL\dot{\phi}^2\sin\phi - (mL)^2 g\sin\phi\cos\phi + (J+mL^2)\,F}{(m+M)(J+mL^2) - (mL)^2\cos^2\phi}$$

$$\therefore \ddot{\phi} = \frac{(m+M)mgL\sin\phi - (mL)^2\,\dot{\phi}^2\,\sin\phi\cos\phi - mL\cos\phi\,F}{(m+M)(J+mL^2) - (mL)^2\cos^2\phi}$$

Appendix 2: Code for stabilizing at upwards position from initial angle

Linear MPC Code (CartPendulum_Linear_MPC.m)

```
%Program compares Linear MPC against LQR
%Scenario: Stabilizing pendulum at upwards position from initial angle
%close to upwards position

close all
clear all
clc

%System sampling time
Ts = 0.01;

x = sym('x',[4 1],'real');
u = sym('u',[1 1],'real');

%System parameters
m = 0.2;
M = 1;
L1 = 0.2;
```

```
J = m*L1*L1/3;

%Obtaining linearized system
fun = cartpend(0,x,u,m,M,L1,J);

Ap = jacobian(fun,x);
Bp = jacobian(fun,u);

Ac = double(subs(Ap, [x;u], [zeros(size(x));zeros(size(u))]));
Bc = double(subs(Bp, [x;u], [zeros(size(x));zeros(size(u))]));

clear x u

Cc = eye(size(Ac));
Dc = zeros(size(Bc));

sysd = c2d(ss(Ac,Bc,Cc,Dc),Ts,'zoh');

%Discretized linearized state space matrices
Ad = sysd.A;
Bd = sysd.B;
Cd = sysd.C;
Dd = sysd.D;

%Establishing input and state cost matrices
Q = diag([1 1 5 5]);
R = 0.5;
%Obtaining dlqr gain and DARE solution for terminal state cost matrix
[K,P,~] = dlqr(Ad,Bd,Q,R);

%Setting system constraints
xlim.max = [20 300 300 300];
xlim.min = -xlim.max;
%umax = 100; %Unconstrained
umax = 30;
umin = -umax;
ulim.max = umax;
ulim.min = -ulim.max;

%Setting prediction horizon
N = 20;

%Obtaining quadratic programming matrices
[H, L, G, W, T, IMPC] = formQPMatrices(Ad, Bd, Q, R, P, xlim, ulim, N);
%Setting initial guess of lambda
lam0 = ones(size(G,1),1);
lam = lam0;

t = 0:Ts:10;
h = Ts/10;

%Storing results from Linear MPC
X = zeros(4,length(t));
U = zeros(1,length(t));

%Storing results for LQR
Xk = zeros(4,length(t));
Uk = zeros(1,length(t));

%Initial position
x0 = pi/2;

x = [0;0;x0;0];
tm = 0;

X(:,1) = x;
Xk(:,1) = x;

for i = 1:length(t)-1

   [du,lam] = myQP(H, L*x, G, W + T*x, lam);
```

```
    %Determine whether or not output matrix is empty
    if ~isempty(du)
        u = IMPC*du;
    else
        u = 0;
    end
    %Implementing input constraints
    if u > umax
        u = umax;
    elseif u < umin
       u = umin;
    end
   %Simulating at each sampling time
   [tx,xm] = ode45(@(t,x) cartpend(t,x,u,m,M,L1,J), tm+h:h:tm+Ts,x);
    x = xm(end,:)';
   %Keeps angle bounded for the linear MPC
   if x(3)>pi
        x(3)= x(3) - 2*pi;
    elseif x(3)<=-pi
        x(3)= x(3) + 2*pi;
    end
   U(i) = u;
   X(:,i+1) = x;
end

x = [0;0;x0;0];
tm = 0;

for i = 1:length(t)-1
   u = -K*x; %LQR linear feedback

   %Implementing input constraints
    if u > umax
        u = umax;
    elseif u < umin
       u = umin;
    end
   %Simulating at each sampling time
   [tx,xm] = ode45(@(t,x) cartpend(t,x,u,m,M,L1,J), tm+h:h:tm+Ts,x);
   x = xm(end,:)';
    %Keeps angle bounded for the linear feedback
   if x(3)>pi
        x(3)= x(3) - 2*pi;
    elseif x(3)<=-pi
        x(3)= x(3) + 2*pi;
    end
   Uk(i) = u;
   Xk(:,i+1) = x;
end

subplot(2,2,1)
plot(t,X, 'LineWidth',2)
title('Linear MPC')
xlabel('t (sec)')
legend('x_1(t)', 'x_2(t)', 'x_3(t)', 'x_4(t)')

subplot(2,2,3)
plot(t,U,'LineWidth',2)
xlabel('t (sec)')
legend('u(t)')

subplot(2,2,2)
plot(t,Xk, 'LineWidth',2)
title('LQR')
xlabel('t (sec)')
legend('x_1(t)', 'x_2(t)', 'x_3(t)', 'x_4(t)')

subplot(2,2,4)
plot(t,Uk,'LineWidth',2)
xlabel('t (sec)')
legend('u(t)')
```

```
%Program end

function xdot = cartpend(t,x,u,m,M,L,J)

%Cart-Pendulum System Dynamcs compatible with Casadi programming scheme
%Parameters
g = 9.80655;
a = (J + m*L^2);
b = (m*L)^2;

%Cart velocity
xdot(1) = x(2);
%Pendulum angular velocity
xdot(3) = x(4);
%Cart acceleration
xdot(2) = (a*m*L*sin(x(3))*(x(4))^2 - b*g*sin(x(3))*cos(x(3)) + a*u)/((m+M)*a - b*cos(x(3)));
%Pendulum angular acceleration
xdot(4) = ((m+M)*m*g*L*sin(x(3)) - b*sin(x(3))*cos(x(3))*(x(4))^2 - m*L*cos(x(3))*u)/((m+M)*a -
b*cos(x(3)));

xdot = xdot';

end

function [H, L, G, W, T, IMPC] = formQPMatrices(A, B, Q, R, P, xlim, ulim, N)
    %This function creates matrices to set MPC as a quadratic programming (QP) problem
    %Obtaining sizes of input and state
    nu = size(B,2);
    nx = size(A,2);
    %initial S matrix, diagonal full of B matrices
    S = kron(eye(N,N),B);
    %Initial M matrix, to be filled with factors of A
    M = [];
    for i = 1:N-1
        %This process shifts the diagonal terms
        %of the identity matrix by i
        S1 = zeros(N,N);
        S2 = eye(N-i,N-i);
        S1(i+1:end,1:N-i)=S2;
        %Used to shift the (A^i)*B diagonals
        S = S + kron(S1,(A^(i))*B);
        %A factors are continuously added
        M = [M;A^i];
    end
    M = [M;A^N]; %Last A factor is added
    %Building Qbar with Qs on its diagonal
    Qbar = kron(eye(N,N),Q);
    %Last value of Qbar is P
    Qbar((end+1-size(P,1)):end,(end+1-size(P,2)):end) = P;
    %Rbar with R matrices on its diagonals
    Rbar = kron(eye(N,N),R);

    %Obtaining equivalent H for QP problem
    H = 2*(S'*Qbar*S + Rbar);
    %Obtaining equivalent L for QP problem such that
    %q = L*x0
    L = 2*S'*Qbar'*M;

    %Obtaining equivalent G, W, T for inequality constraints
    G = [S;-S;eye(N*nu,N*nu);-eye(N*nu,N*nu)];
    W = [kron(ones(N,1),xlim.max');-kron(ones(N,1),xlim.min');kron(ones(N,1),ulim.max');-
kron(ones(N,1),ulim.min')];
    T = [-M;M;zeros(2*N*nu,nx)];
    %u0 = IMPC*U
    IMPC = [eye(nu),zeros(nu,nu*(N-1))];
end

function [U, lam] = myQP(H, q, A, b, lam0)
    %Simple dual projected optimization algorithm for solving Quadratic
    %Programming (QP) problems.
```

```
    %Max number of iterations
    maxIt = 100;

    %Getting matrices Hd and qd
    Hinv = inv(H);
    A_Hinv = A*Hinv;
    Hd = A_Hinv*A';
    qd = A_Hinv*q + b;

    %Obtaining the Lyapunov constant
    L = norm(Hd,inf);
    %Setting the initial lambda guess
    lam  = lam0;

    %Iterating gradient descent until convergence (maxIt)
    for i = 1:maxIt
        lam = max(0,lam - (1/L)*(Hd*lam + qd));
    end
    %Obtaining minimizing values
    U = -Hinv*(q+A'*lam);
end
```

Appendix 3: Code for cart movement

Linear MPC code (CartPendulum_LinearMPC_HorizontalPosition.m)

```
%Program compares Linear MPC against LQR
%Scenario: Cart movement while keeping pendulum upwards
%NOTE: This program is fairly similar to the one used
%for the stabilizing case, so the functions used in this program
%can be found in Appendix 2

close all
clear all
clc

%System sampling time
Ts = 0.01;

x = sym('x',[4 1],'real');
u = sym('u',[1 1],'real');

%System parameters
m = 0.2;
M = 1;
L1 = 0.2;
J = m*L1*L1/3;

%Obtaining linearized system
fun = cartpend(0,x,u,m,M,L1,J);

Ap = jacobian(fun,x);
Bp = jacobian(fun,u);

Ac = double(subs(Ap, [x;u], [zeros(size(x));zeros(size(u))]));
Bc = double(subs(Bp, [x;u], [zeros(size(x));zeros(size(u))]));

clear x u

Cc = eye(size(Ac));
Dc = zeros(size(Bc));

sysd = c2d(ss(Ac,Bc,Cc,Dc),Ts,'zoh');

%Discretized linearized state space matrices
Ad = sysd.A;
Bd = sysd.B;
Cd = sysd.C;
Dd = sysd.D;
```

```
%Establishing input and state cost matrices
Q = diag([1 1 5 5]);
R = 0.5;
%Obtaining dlqr gain and DARE solution for terminal state cost matrix
[K,P,~] = dlqr(Ad,Bd,Q,R);

%Setting system constraints
xlim.max = [100 30 0.7 30];
xlim.min = -xlim.max;
%umax = 100; %Unconstrained
umax = 30;
umin = -umax;
ulim.max = umax;
ulim.min = -ulim.max;

%Setting prediction horizon
N = 60;

%Obtaining quadratic programming matrices
[H, L, G, W, T, IMPC] = formQPMatrices(Ad, Bd, Q, R, P, xlim, ulim, N);
%Setting initial guess of lambda
lam0 = ones(size(G,1),1);
lam = lam0;

t = 0:Ts:10;
h = Ts/10;

%Storing results from Linear MPC
X = zeros(4,length(t));
U = zeros(1,length(t));

%Storing results for LQR
Xk = zeros(4,length(t));
Uk = zeros(1,length(t));

%Desired final position
r = 50;
rvec = [r; 0; 0; 0];

%Initial conditions
x = [0;0;0;0];
tm = 0;

X(:,1) = x;
Xk(:,1) = x;

for i = 1:length(t)-1

   [du,lam] = myQP(H, L*(x-rvec), G, W + T*(x-rvec), lam);
    %Determine whether or not output matrix is empty
    if ~isempty(du)
        u = IMPC*du;
    else
        u = 0;
    end
    %Implementing input constraints
    if u > umax
        u = umax;
    elseif u < umin
       u = umin;
    end
    %Simulating at each sampling time
   [tx,xm] = ode45(@(t,x) cartpend(t,x,u,m,M,L1,J), tm+h:h:tm+Ts,x);
   x = xm(end,:)';
   %Keeps angle bounded for the linear MPC
   if x(3)>pi
        x(3)= x(3) - 2*pi;
    elseif x(3)<=-pi
        x(3)= x(3) + 2*pi;
    end
   U(i) = u;
```

```
   X(:,i+1) = x;
end

x = [0;0;0;0];
tm = 0;

for i = 1:length(t)-1
   u = K*(rvec-x); %LQR linear feedback
   %Implementing input constraints
    if u > umax
        u = umax;
    elseif u < umin
       u = umin;
    end
   %Simulating at each sampling time
   [tx,xm] = ode45(@(t,x) cartpend(t,x,u,m,M,L1,J), tm+h:h:tm+Ts,x);
   x = xm(end,:)';
    %Keeps angle bounded for the linear feedback
   if x(3)>pi
        x(3)= x(3) - 2*pi;
    elseif x(3)<=-pi
        x(3)= x(3) + 2*pi;
    end
   Uk(i) = u;
   Xk(:,i+1) = x;
end

subplot(2,2,1)
plot(t,X, 'LineWidth',2)
title('Linear MPC')
xlabel('t (sec)')
legend('x_1(t)', 'x_2(t)', 'x_3(t)', 'x_4(t)')

subplot(2,2,3)
plot(t,U,'LineWidth',2)
xlabel('t (sec)')
legend('u(t)')

subplot(2,2,2)
plot(t,Xk, 'LineWidth',2)
title('LQR')
xlabel('t (sec)')
legend('x_1(t)', 'x_2(t)', 'x_3(t)', 'x_4(t)')

subplot(2,2,4)
plot(t,Uk,'LineWidth',2)
xlabel('t (sec)')
legend('u(t)')

%Program end
```

Reference Governor Code (CartPendulum_ReferenceGovernor_HorizontalPosition.m)

```
%Program compares regular LQR against LQR with the reference signal given by a reference
%governor.
%Scenario: Cart movement while keeping pendulum upwards

close all
clear all
clc

%System sampling time
Ts = 0.01;

x = sym('x',[4 1],'real');
u = sym('u',[1 1],'real');

%System parameters
m = 0.2;
M = 1;
L1 = 0.2;
```

```
J = m*L1*L1/3;

%Obtaining linearized system
fun = cartpend(0,x,u,m,M,L1,J);

Ap = jacobian(fun,x);
Bp = jacobian(fun,u);

Ac = double(subs(Ap, [x;u], [zeros(size(x));zeros(size(u))]));
Bc = double(subs(Bp, [x;u], [zeros(size(x));zeros(size(u))]));

clear x u

Cc = eye(size(Ac));
Dc = zeros(size(Bc));

sysd = c2d(ss(Ac,Bc,Cc,Dc),Ts,'zoh');

%Discretized linearized state space matrices
Ad = sysd.A;
Bd = sysd.B;
Cd = sysd.C;
Dd = sysd.D;

%Establishing input and state cost matrices
Q = diag([1 1 5 5]);
R = 0.5;
%Obtaining dlqr gain
[K,~,~] = dlqr(Ad,Bd,Q,R);

%Setting system constraints
xlim.max = [22 100 100 100];
xlim.min = -xlim.max;
%umax = 100; %Unconstrained
umax = 20;
ulim.max = umax;
ulim.min = -ulim.max;

%Setting prediction horizon
N = 60;

%Setting matrices for Reference Governor implementation
Anew = Ad-Bd*K;
Bnew = Bd*K;
Cnew = [1 0 0 0;...
    0 0 1 0];
Dnew = zeros(2,4);
H1 = eye(2);
ub = [102;0.4];
lb = -ub;
eps = 0.01;

%Get matrices for evaluating whether future states will be within
%approximated safe zone
[Qmat,hvec] = getMats(Anew, Bnew, Cnew, Dnew, H1, K, lb, ub, -umax, umax, N, eps);

t = 0:Ts:15;
h = Ts/10;

%For storing results from LQR w/ reference governor implementation
X = zeros(4,length(t));
U = zeros(1,length(t));

%For storing results from LQR implementation
Xk = zeros(4,length(t));
Uk = zeros(1,length(t));

%Reference vector
r = [100;0;0;0];
%Initial conditions
x = [0;0;0;0];
```

```
tm = 0;

X(:,1) = x;
Xk(:,1) = x;
u = 0;%Input
v = 0; %Governed reference
Kvar = 1; %Used to determine how similar v is to r

%Reference governor implementation
for i = 1:length(t)-1
   %Searching for Kvar value that prevents constraint violation
   Kvar = 1;
   while 1
      vn = v + Kvar*(r-v);
      if prod(Qmat*[vn;x]-hvec<=0)==1
         v = vn;
         break;
      else
         Kvar = Kvar - 0.05;
      end

      if Kvar <=0
          break;
      end

   end

   %Implementing new reference
   u = K*(v-x);

   [tx,xm] = ode45(@(t,x) cartpend(t,x,u,m,M,L1,J), tm+h:h:tm+Ts,x);
   x = xm(end,:)';
   %Keeps angle bounded for the LQR
   if x(3)>pi
        x(3)= x(3) - 2*pi;
    elseif x(3)<=-pi
        x(3)= x(3) + 2*pi;
    end
   U(i) = u;
   X(:,i+1) = x;
end

x = [0;0;0;0];
tm = 0;

%LQR implementation
for i = 1:length(t)-1
   u = K*(r-x);

   [tx,xm] = ode45(@(t,x) cartpend(t,x,u,m,M,L1,J), tm+h:h:tm+Ts,x);
   x = xm(end,:)';
   %Keeps angle bounded for the LQR
   if x(3)>pi
        x(3)= x(3) - 2*pi;
    elseif x(3)<=-pi
        x(3)= x(3) + 2*pi;
    end
   Uk(i) = u;
   Xk(:,i+1) = x;
end

subplot(3,2,1)
plot(t,X(1,:), 'LineWidth',2)
title('LQR w/ Reference Governor')
xlabel('t (sec)')
ylabel('Cart position (m)')
legend('x_1(t)')

subplot(3,2,3)
plot(t,X(3,:), 'LineWidth',2)
xlabel('t (sec)')
```

```
ylabel('Pendulum Angle (rad)')
legend('x_3(t)')

subplot(3,2,5)
plot(t,U,'LineWidth',2)
xlabel('t (sec)')
ylabel('Horizontal Force (N)')
legend('u(t)')

subplot(3,2,2)
plot(t,Xk(1,:), 'LineWidth',2)
title('LQR')
xlabel('t (sec)')
legend('x_1(t)')

subplot(3,2,4)
plot(t,Xk(3,:), 'LineWidth',2)
xlabel('t (sec)')
legend('x_3(t)')

subplot(3,2,6)
plot(t,Uk,'LineWidth',2)
xlabel('t (sec)')
legend('u(t)')

%Program end

function [Q,h] = getMats(A,B,C,D,H,K,lb,ub,ulb,uub,N,eps)
    %Obtains matrices to test whether the future states considered within
    %the prediction horizon will remain inside the specified safe zone
    %under the given modified reference v
    Q = [H*D H*C; -H*D -H*C];
    h = [ub;-lb];
    invA = inv(eye(size(A))-A);
    for i = 1:N
        Q = [Q; H*C*(eye(size(A))-A^i)*invA*B+H*D H*C*A^i;...
            -H*C*(eye(size(A))-A^i)*invA*B+H*D -H*C*A^i];
        h = [h;ub;-lb];
    end
    Q = [Q;H*C*invA*B+H*D zeros(size(H*C));...
        -H*C*invA*B+H*D -zeros(size(H*C));...
        K -K; -K K];
    h = [h;(1-eps)*ub;-(1-eps)*lb; uub; -ulb];
end
```

Appendix 4: Code for wind-up maneuver

Nonlinear MPC code for Wind-Up and Cart Position Control (CartPendulum_NonlinearMPC_SwingUp.m)

```
%Program implements NMPC for wind-up maneuvers and cart movement
%Scenario: Wind-up Maneuver + Cart movement
close all
clear all
clc

import casadi.*;
%Defining problem sizes
%N = 30; %Finite Horizon for cart movement
N = 60; %Finite Horizon for windup
nx = 4; %Number of states
nu = 1; %Number of inputs
%Parameters
ts = 0.1; %Sampling time
x0 = SX.sym('x0',nx);    %State Estimate
q = SX.sym('q',4,4);     %State weight matrix
r = SX.sym('r',1,1);     %Input weight matrix
qf = SX.sym('qf',4,4);   %Terminal state weight matrix
xTarget = SX.sym('xTarget',nx);    %Target Reference
xub = SX.sym('xub',nx,1);    %state upper bound
```

```
xlb = SX.sym('xlb',nx,1);   %state lower bound
uub = SX.sym('uub',nu,1);   %control upper bound
ulb = SX.sym('ulb',nu,1);   %control lower bound
gamma = SX.sym('gamma',1,1);   %penalty weight

%System parameters
m = 0.2;
M = 1;
L1 = 0.2;
J1 = m*L1*L1/3;

%Creating stage functions
xk = SX.sym('xk',nx);   %State
uk = SX.sym('uk',nu);   %Input
sys =cartpendCas(0,xk,uk,m,M,L1,J1); %System dynamics equation
f_x = jacobian(sys,xk); %A matrix equation (linearized sys by x)
f_u = jacobian(sys,uk); %B matrix equation (linearized sys by u)
%*********************************************************
xkp1 = xk + ts*cartpendCas(0,xk,uk,m,M,L1,J1); %Discrete time dynamics obtained through
                                  %explicit Euler approximation
%*********************************************************

%Casadi function for discrete dynamics
fd = Function('f',{xk,uk},{xkp1},{'xk','uk'},'xkp1');
%Casadi function for obtaining linearized A matrix
f_x = Function('f_x',{xk,uk},{f_x},{'xbar','ubar'},{'f_x'});
%Casadi function for obtaining linearized B matrix
f_u = Function('f_u',{xk,uk},{f_u},{'xbar','ubar'},{'f_u'});
%Stage cost Casadi functions
ell = 1/2*([xk(1)-xTarget(1);xk(2)-xTarget(2);cos(xTarget(3))-cos(xk(3));xk(4)-
xTarget(4)])'*q*([xk(1)-xTarget(1);xk(2)-xTarget(2);cos(xTarget(3))-cos(xk(3));xk(4)-xTarget(4)])
+...
    1/2*uk'*r*uk;
l = Function('l',{xk,uk,q,r,xTarget},{ell},{'xk','uk','q','r','xTarget'},{'l'});

%Creating primal optimization variables within finite horizon
x = SX.sym('x',nx,N);
u = SX.sym('u',nu,N);
s = SX.sym('s',N);
%Initializing J, h and g
h = [];
g = [];
J = SX.zeros(1);

%*********************************************************
%The following loop builds the J scalar and
%the h (inequalities) and g (equalities) vectors
%using z variables
for i = 1:N
   if i == 1
       xim1 = x0;
   else
       xim1 = x(:,i-1);
   end
   xi = x(:,i);
   uim1 = u(:,i);

   %cost
   J = J + l(xim1,uim1,q,r,xTarget) + s(i)*gamma;
   %inequality constraint
   h = [h;xi-s(i)-xub;-xi-s(i)+xlb;uim1-uub;-uim1+ulb;-s(i)];
   %equality constraint
   g = [g;xi-fd(xim1,uim1)];

end
%Terminal cost
J = J + 1/2*([xi(1)-xTarget(1);xi(2)-xTarget(2);cos(xTarget(3))-cos(xi(3));xi(4)-
xTarget(4)])'*qf*([xi(1)-xTarget(1);xi(2)-xTarget(2);cos(xTarget(3))-cos(xi(3));xi(4)-
xTarget(4)]);
%*********************************************************
%Reshaping every matrix into a column
```

```
x = x(:);
u = u(:);
z = [u;x;s];

nz = length(z);
nv = length(h);
nl = length(g);

%Defining dual variables and Lagrangian
clear l
l = SX.sym('l',nl,1);   %Equality duals
v = SX.sym('v',nv,1);   %Inequality duals
L = J + l'*g + v'*h;    %Lagrangian

%*************************************************************
%On the following lines, the gradients and Hessians for functions
% in question 1d) are obtained through Casadi
%Differentiating functions
Lz = gradient(L,z); %Lagrangian gradient (vector)
Jzz = hessian(J,z); %Cost function hessian (matrix)
Jz = gradient(J,z); %Cost function gradient (vector)
gz = jacobian(g,z); %Equality constraints jacobian (matrix)
hz = jacobian(h,z); %Inequality constraints jacobian (matrix)
%*************************************************************

%*************************************************************
%After that, all of the equations from 2d) will be stored in
%struct nlp for later use
%Laplacian (L) gradient with respect to z
nlp.laplace_grad = ...
    Function('dL',{x0,x,u,s,l,v,xub,xlb,uub,ulb,...
    q,r,qf,xTarget,gamma},{Lz},...
    {'x0','x','u','s','l','v','xub','xlb','uub','ulb',...
    'q','r','qf','xTarget','gamma'},{'Lz'});
%Cost function (J) Hessian with respect to z
nlp.cost_hess = ...
    Function('HJ',{q,r,qf,x,xTarget},{Jzz},{'q','r','qf'},{'Jzz'});
%Cost function (J) Gradient with respect to z
nlp.cost_grad = ...
    Function('dJ',{x,u,q,r,qf,xTarget,gamma},{Jz},...
    {'x','u','q','r','qf','xTarget','gamma'},{'Jz'});
%Inequality constraint function (h) gradient with respect
%to z (results in matrix)
nlp.in_grad = ...
    Function('HG',{},{hz},{},{'hz'});
%Inequality constraint function (h) with respect
%to z (results in vector)
nlp.in_reg = ...
    Function('dG',{x,u,s,xub,xlb,uub,ulb},{h},...
    {'x','u','s','xub','xlb','uub','ulb'},{'h'});
%Equality constraint function (g) gradient with respect
%to z (results in matrix)
nlp.eq_grad = ...
    Function('Hh',{x,u,x0},{gz},...
    {'x','u','x0'},{'gz'});
%Inequality constraint function (h) with respect
%to z (results in vector)
nlp.eq_reg = ...
    Function('dh',{x0,x,u},{g},...
    {'x0''x','u'},{'g'});
%*************************************************************
%Maximum number of iterations in OCP
args.max_sqp_iters = 10;
%Tolerance for ending OCP program
args.tol = 1e-4;

xtrg = zeros(4,1); %Wind up maneuver
%xtrg = [50; 0; 0; 0]; %Cart Movement

% Initial Conditions
x0 = [0;0;pi;0]; %Wind up maneuver
```

```
%x0 = [0;0;0;0]; %Cart movement

% state constraints
args.xub = [2.5, 10, 10, 10]; %Wind up state constraints
%args.xub = [105, 15, 1.4, 30]; %Cart movement state constraints
args.xlb = -args.xub;
% control constraints
args.uub = 30*ones(1,1); %Windup ipnut constraints
%args.uub = 20*ones(1,1);%Cart movement input constraints
args.ulb = -args.uub;

args.N = N;          %Finite horizon
args.nlp = nlp;      %Struct with OCP required functions

%args.Q = diag([30,0,60,0]);  %State cost matrix OPT 2
args.Q = diag([30,20,60,20]);  %State cost matrix
args.R = 50;          %Input cost matrix
args.Qf = args.Q;    %Terminal state cost same as
args.gamma = 10; %Weight for s

% initial optimizer guess for u, x, s (z), l, v (duals)
ukm1 = zeros(1*N,1);
xkm1 = zeros(4*N,1);
skm1 = zeros(N,1);
lkm1 = zeros(4*N,1);
vkm1 = zeros(11*N,1);
tfinal = 15; %Total simulation time
nsim = ceil(tfinal/ts); %Number of iterations for simulation given time step

%Matrices for storing time, state and input data from simulation in case 1
X = zeros(4,nsim);
t1 = zeros(nsim,1);
U = zeros(1,nsim);
X(:,1) = x0;
x = x0;
u_nmpc = 0;
t1(1) = 0;

for i = 1:nsim-1
    %NMPC iterative solver
    [u_nmpc,ukm1,xkm1,skm1,lkm1,vkm1,res1(i)] = ...
    nmpc(x,xtrg,ukm1,xkm1,skm1,lkm1,vkm1,args);
    U(:,i) = u_nmpc;
    [tx,xm] = ode45(@(t,x) cartpendCas(t,x,u_nmpc,m,M,L1,J1),0:ts/10:ts,x);
    x = xm(end,:)';
    X(:,i+1) = x;
    t1(i) = (i-1)*ts;
end

X(3,:) = wrapToPi(X(3,:));

U(:,nsim) = U(:,nsim-1);
res1(nsim) = 0;
t1(nsim) = nsim*ts;

subplot(3,1,1)
plot(t1,X(1,:), 'LineWidth',2)
title('Nonlinear MPC')
xlabel('t (sec)')
ylabel('Cart Position (m)')
legend('x_1(t)')

subplot(3,1,2)

plot(t1,X(3,:), 'LineWidth',2)
xlabel('t (sec)')
ylabel('Pendulum Angle (rad)')
legend('x_3(t)')

subplot(3,1,3)
plot(t1,U,'LineWidth',2)
```

```
xlabel('t (sec)')
ylabel('Horizontal Force (N)')
legend('u(t)')

%End Program

function xdot = cartpendCas(t,x,u,m,M,L,J)

%Cart-Pendulum System Dynamcs compatible with Casadi programming scheme
%Parameters
g = 9.80655;
a = (J + m*L^2);
b = (m*L)^2;

xdot = [x(2);...
    (a*m*L*sin(x(3))*(x(4))^2 - b*g*sin(x(3))*cos(x(3)) + a*u)/((m+M)*a - b*cos(x(3)));...
    x(4);...
    ((m+M)*m*g*L*sin(x(3)) - b*sin(x(3))*cos(x(3))*(x(4))^2 - m*L*cos(x(3))*u)/((m+M)*a -
b*cos(x(3)))];

end

function [u_nmpc, uopt, xopt, sopt, lopt, vopt, ocp_res] = ...
    nmpc(xhat, xtrg, ukm1, xkm1, skm1, lkm1, vkm1, args)
    %Nonlinear MPC by applying Sequential Quadratic Programming (SQP) by solving QP subsystems
    %and adding up the results
    %Obtaining initial values
    x0 = xhat; %Initial state vector
    %Initial guesses for z, inequality and equality duals;
    u = ukm1; x = xkm1; s = skm1; l = lkm1; v = vkm1;
    nx = 4; %State dimension
    nu = 1; %Input dimension
    N = args.N; %Finite horizon
    n = nx*N; %Primal state dimension
    m = nu*N; %Primar input dimension
    nlp = args.nlp; % Functions required for OCP implementation
    z = [u;x;s]; %Getting initial z matrix (not used for program resolution)

    for i = 1:args.max_sqp_iters

        %Computing h (inequality function), g(equality function) and
        %dJ (Cost function gradient)
        dJ = full(nlp.laplace_grad(x0,x,u,s,l,v,args.xub,args.xlb,args.uub,args.ulb,...
            args.Q,args.R,args.Qf,xtrg,args.gamma));
        h = full(nlp.in_reg(x,u,s,args.xub,args.xlb,args.uub,args.ulb));
        g = full(nlp.eq_reg(x0,x,u));

        %Evaluating FNR to determine whether it is below the given tolerance
        FNR = [dJ;g;min(-h,v)];
        ocp_res = norm (FNR);
        if ocp_res - args.tol <= 0
            break;
        end

        %Computing G (equality gradient), C (inequality gradient)and
        %H (Cost function Hessian)
        H = full(nlp.cost_hess(args.Q,args.R,args.Qf,x,xtrg));
        d = full(nlp.cost_grad(x,u,args.Q,args.R,args.Qf,xtrg,args.gamma));
        C = nlp.in_grad();
        C = full(C.hz);
        G = full(nlp.eq_grad(x,u,x0));

        %Quadrtic programming implementation
        qpopts = optimoptions('quadprog','display','off');
        [dz,~,~,~,duals] = quadprog(H,d,C,-h,G,-g, [],[],[],qpopts);
        %If quadratic program does not converge, have the program keep going
        if isempty(dz)
            dz = zeros(size(z));
        else
            %If program converges to solution, get duals
            l = duals.eqlin;
```

```
            v = duals.ineqlin;
        end
        %Increase u, x and s
        u = u + dz(1:m);
        x = x + dz(m+1:m+n);
        s = s + dz(m+n+1:end);

    end

    %Obtain the final (optimal) values calculated
    uopt = u;
    xopt = x;
    sopt = s;
    lopt = l;
    vopt = v;

    u_nmpc = u(1);

end
```

Switching NMPC Code (CartPendulum_SwitchingMPC_SwingUp.m)

```
%Program implements Switching NMPC for extremely constrained
%wind-up maneuvers
%Scenario: Wind-up Maneuver with Small Position Constraints
close all
clear all
clc

import casadi.*;

N = 30; %Prediction Horizon
nx = 4; %Number of states
nu = 1; %Number of inputs
%Parameters
ts = 0.1; %Sampling time
x0 = SX.sym('x0',nx);    %State Estimate
q = SX.sym('q',4,4);     %State weight matrix
r = SX.sym('r',1,1);     %Input weight matrix
qf = SX.sym('qf',4,4);   %Terminal state weight matrix
xTarget = SX.sym('xTarget',nx);    %Target Reference
xub = SX.sym('xub',nx,1);    %state upper bound
xlb = SX.sym('xlb',nx,1);    %state lower bound
uub = SX.sym('uub',nu,1);    %control upper bound
ulb = SX.sym('ulb',nu,1);    %control lower bound
gamma = SX.sym('gamma',1,1);    %penalty weight

%System parameters
m = 0.2;
M = 1;
L1 = 0.2;
J1 = m*L1*L1/3;
gm = 9.81;

%Creating stage functions
xk = SX.sym('xk',nx);    %State
uk = SX.sym('uk',nu);    %Input
sys =cartpendCas(0,xk,uk,m,M,L1,J1); %System dynamics equation
f_x = jacobian(sys,xk); %A matrix equation (linearized sys by x)
f_u = jacobian(sys,uk); %B matrix equation (linearized sys by u)
%*********************************************************
xkp1 = xk + ts*cartpendCas(0,xk,uk,m,M,L1,J1); %Discrete time dynamics obtained through
                                    %explicit Euler approximation
%*********************************************************

%Casadi function for discrete dynamics
fd = Function('f',{xk,uk},{xkp1},{'xk','uk'},'xkp1');
%Casadi function for obtaining linearized A matrix
f_x = Function('f_x',{xk,uk},{f_x},{'xbar','ubar'},{'f_x'});
%Casadi function for obtaining linearized B matrix
f_u = Function('f_u',{xk,uk},{f_u},{'xbar','ubar'},{'f_u'});
```

```
%Stage cost Casadi functions
ell = 0.5.*q(1,1).*(xk(1)-xTarget(1)).^2 + 0.5.*q(2,2).*(xk(2)-xTarget(2)).^2 +
m.*gm.*q(3,3).*(cos(xTarget(3))-cos(xk(3))) + 0.5.*J1.*q(4,4).*(xk(4)-xTarget(4)).^2 + ...
    1/2*uk'*r*uk;
l = Function('l',{xk,uk,q,r,xTarget},{ell},{'xk','uk','q','r','xTarget'},{'l'});

%Creating primal optimization variables within finite horizon
x = SX.sym('x',nx,N);
u = SX.sym('u',nu,N);
s = SX.sym('s',N);
%Initializing J, h and g
h = [];
g = [];
J = SX.zeros(1);

%*********************************************************
%The following loop builds the J scalar and
%the h (inequalities) and g (equalities) vectors
%using z variables
for i = 1:N
   if i == 1
       xim1 = x0;
   else
       xim1 = x(:,i-1);
   end
   xi = x(:,i);
   uim1 = u(:,i);

   %cost
   J = J + l(xim1,uim1,q,r,xTarget) + s(i)*gamma;
   %inequality constraint
   h = [h;xi-s(i)-xub;-xi-s(i)+xlb;uim1-uub;-uim1+ulb;-s(i)];
   %equality constraint
   g = [g;xi-fd(xim1,uim1)];

end
%Terminal cost
J = J + 0.5.*qf(1,1).*(xi(1)-xTarget(1)).^2 + 0.5.*qf(2,2).*(xi(2)-xTarget(2)).^2 +
m.*gm.*qf(3,3).*(cos(xTarget(3))-cos(xi(3))) + 0.5.*J1.*qf(4,4).*(xi(4)-xTarget(4)).^2;
%*********************************************************
%Reshaping every matrix into a column
x = x(:);
u = u(:);
z = [u;x;s];

nz = length(z);
nv = length(h);
nl = length(g);

%Defining dual variables and Lagrangian
clear l
l = SX.sym('l',nl,1);   %Equality duals
v = SX.sym('v',nv,1);   %Inequality duals
L = J + l'*g + v'*h;    %Lagrangian

%*************************************************************
%On the following lines, the gradients and Hessians for functions
% in question 1d) are obtained through Casadi
%Differentiating functions
Lz = gradient(L,z); %Lagrangian gradient (vector)
Jzz = hessian(J,z); %Cost function hessian (matrix)
Jz = gradient(J,z); %Cost function gradient (vector)
gz = jacobian(g,z); %Equality constraints jacobian (matrix)
hz = jacobian(h,z); %Inequality constraints jacobian (matrix)
%*************************************************************

%*********************************************************
%After that, all of the equations from 2d) will be stored in
%struct nlp for later use
%Laplacian (L) gradient with respect to z
nlp.laplace_grad = ...
```

```
    Function('dL',{x0,x,u,s,l,v,xub,xlb,uub,ulb,...
    q,r,qf,xTarget,gamma},{Lz},...
    {'x0','x','u','s','l','v','xub','xlb','uub','ulb',...
    'q','r','qf','xTarget','gamma'},{'Lz'});
%Cost function (J) Hessian with respect to z
nlp.cost_hess = ...
    Function('HJ',{q,r,qf,x,xTarget},{Jzz},{'q','r','qf','x','xTarget'},{'Jzz'});
%Cost function (J) Gradient with respect to z
nlp.cost_grad = ...
    Function('dJ',{x,u,q,r,qf,xTarget,gamma},{Jz},...
    {'x','u','q','r','qf','xTarget','gamma'},{'Jz'});
%Inequality constraint function (h) gradient with respect
%to z (results in matrix)
nlp.in_grad = ...
    Function('HG',{},{hz},{},{'hz'});
%Inequality constraint function (h) with respect
%to z (results in vector)
nlp.in_reg = ...
    Function('dG',{x,u,s,xub,xlb,uub,ulb},{h},...
    {'x','u','s','xub','xlb','uub','ulb'},{'h'});
%Equality constraint function (g) gradient with respect
%to z (results in matrix)
nlp.eq_grad = ...
    Function('Hh',{x,u,x0},{gz},...
    {'x','u','x0'},{'gz'});
%Inequality constraint function (h) with respect
%to z (results in vector)
nlp.eq_reg = ...
    Function('dh',{x0,x,u},{g},...
    {'x0','x','u'},{'g'});
%************************************************************
%Maximum number of iterations in OCP
args.max_sqp_iters = 20;
%Tolerance for ending OCP program
args.tol = 1e-4;

xtrg = zeros(4,1); %Wind up maneuver target conditions

% Initial Conditions
x0 = [0;0;pi;0]; %Wind up maneuver

% state constraints

args.xub = [1, 5, 20, 5]; %Wind up hard constraints
args.xlb = -args.xub;
% control constraints
umax = 10;
args.uub = umax*ones(1,1);
args.ulb = -args.uub;

args.N = N;         %Finite horizon
args.nlp = nlp;     %Struct with OCP required functions

args.Q = diag([30,5,60,-0.005]); %First state cost matrix
args.Q2 = diag([20,10,60,10]);  %Second state cost matrix

args.R = 10;        %New input cost matrix
%Obtaining Terminal state cost matrix by solving DARE using linearized
%nonlinear system
[~,Qf,~] = dlqr(full(f_x(0,0)),full(f_u(0,0)),args.Q2,args.R);
%args.Qf = diag([30,0,60,-0.005]); %Terminal state cost matrix
args.Qf = diag([40,0,60,-0.005]); %First Terminal state cost matrix
args.Qf2 = Qf; %Second Terminal state cost matrix
args.gamma = 10; %Weight for s

% initial optimizer guess for u, x, s (z), l, v (duals)
ukm1 = zeros(1*N,1);
xkm1 = zeros(4*N,1);
skm1 = zeros(N,1);
lkm1 = zeros(4*N,1);
vkm1 = zeros(11*N,1);
```

```
tfinal = 10; %Total simulation time
nsim = ceil(tfinal/ts); %Number of iterations for simulation given time step

%Matrices for storing time, state and input data from simulation in case 1
X = zeros(4,nsim);
t1 = zeros(nsim,1);
U = zeros(1,nsim);
X(:,1) = x0;
x = x0;
u_nmpc = 0;
t1(1) = 0;

for i = 1:nsim-1
    %Switching NMPC iterative solver
    [u_nmpc,ukm1,xkm1,skm1,lkm1,vkm1,res1(i)] = ...
    switching_nmpc(x,xtrg,ukm1,xkm1,skm1,lkm1,vkm1,args);

    %If input obtained is unfeasible, then reset duals and
    %optimization variables and move cart towards center
    if u_nmpc > umax && x(1)<=0
        u_nmpc = umax;
        xkm1 = zeros(4*N,1);
        ukm1 = zeros(1*N,1);
        skm1 = zeros(N,1);
        lkm1 = zeros(4*N,1);
        vkm1 = zeros(11*N,1);
        ukm1(1) = umax;

    elseif u_nmpc > umax && x(1)>0
        u_nmpc = -umax;
        xkm1 = zeros(4*N,1);
        ukm1 = zeros(1*N,1);
        skm1 = zeros(N,1);
        lkm1 = zeros(4*N,1);
        vkm1 = zeros(11*N,1);
        ukm1(1) = -umax;

    elseif u_nmpc < -umax
        u_nmpc = -umax;
        xkm1 = zeros(4*N,1);
        ukm1 = zeros(1*N,1);
        skm1 = zeros(N,1);
        lkm1 = zeros(4*N,1);
        vkm1 = zeros(11*N,1);
        ukm1(1) = -umax;
    end

    [tx,xm] = ode45(@(t,x) cartpendCas(t,x,u_nmpc,m,M,L1,J1),0:ts/10:ts,x);

    x = xm(end,:)';

    X(:,i+1) = x;
    U(:,i) = u_nmpc;
    t1(i) = (i-1)*ts;
end

X(3,:) = wrapToPi(X(3,:));

U(:,nsim) = U(:,nsim-1);
res1(nsim) = 0;
t1(nsim) = nsim*ts;

subplot(3,1,1)
plot(t1,X(1,:), 'LineWidth',2)
title('Nonlinear Switching MPC')
xlabel('t (sec)')
ylabel('Cart Position (m)')
legend('x_1(t)')

subplot(3,1,2)
```

```
plot(t1,X(3,:), 'LineWidth',2)
xlabel('t (sec)')
ylabel('Pendulum Angle (rad)')
legend('x_3(t)')

subplot(3,1,3)
plot(t1,U,'LineWidth',2)
xlabel('t (sec)')
ylabel('Horizontal Force (N)')
legend('u(t)')

%End Program

function [u_nmpc, uopt, xopt, sopt, lopt, vopt, ocp_res] = ...
    switching_nmpc(xhat, xtrg, ukm1, xkm1, skm1, lkm1, vkm1, args)
    %Nonlinear MPC with similar implementation to nmpc.m.
    %This scheme switches Q and Qf matrices depending on proximity to
    %target bearing
    %Obtaining initial values
    x0 = xhat; %Initial state vector
    %Initial guesses for z, inequality and equality duals;
    u = ukm1; x = xkm1; s = skm1; l = lkm1; v = vkm1;
    nx = 4; %State dimension
    nu = 1; %Input dimension
    N = args.N; %Finite horizon
    n = nx*N; %Primal state dimension
    m = nu*N; %Primar input dimension
    nlp = args.nlp; % Functions required for OCP implementation
    z = [u;x;s]; %Getting initial z matrix (not used for program resolution)

    for i = 1:args.max_sqp_iters

       %Computing h (inequality function), g(equality function) and
       %dJ (Cost function gradient)
       if cos(abs(x0(3)-xtrg(3)))<0.90
           dJ = full(nlp.laplace_grad(x0,x,u,s,l,v,args.xub,args.xlb,args.uub,args.ulb,...
               args.Q,args.R,args.Qf,xtrg,args.gamma));
       else
           dJ = full(nlp.laplace_grad(x0,x,u,s,l,v,args.xub,args.xlb,args.uub,args.ulb,...
               args.Q2,args.R,args.Qf2,xtrg,args.gamma));
       end
       h = full(nlp.in_reg(x,u,s,args.xub,args.xlb,args.uub,args.ulb));
       g = full(nlp.eq_reg(x0,x,u));

       %Evaluating FNR to determine whether it is below the given tolerance
       FNR = [dJ;g;min(-h,v)];
       ocp_res = norm (FNR);
       if ocp_res - args.tol <= 0
           break;
       end

       %Computing G (equality gradient), C (inequality gradient)and
       %H (Cost function Hessian)
       if cos(abs(x0(3)-xtrg(3)))<0.90
           H = full(nlp.cost_hess(args.Q,args.R,args.Qf,x,xtrg));
           d = full(nlp.cost_grad(x,u,args.Q,args.R,args.Qf,xtrg,args.gamma));
       else
           H = full(nlp.cost_hess(args.Q2,args.R,args.Qf2,x,xtrg));
           d = full(nlp.cost_grad(x,u,args.Q2,args.R,args.Qf2,xtrg,args.gamma));
       end
       C = nlp.in_grad();
       C = full(C.hz);
       G = full(nlp.eq_grad(x,u,x0));

       %Quadrtic programming implementation
       qpopts = optimoptions('quadprog','display','off');
       [dz,~,~,~,duals] = quadprog(H,d,C,-h,G,-g, [],[],[],qpopts);
       %If quadratic program does not converge, have the program keep going
       if isempty(dz)
           dz = zeros(size(z));
       else
```

```
            %If program converges to solution, get duals
            l = duals.eqlin;
            v = duals.ineqlin;
        end
        %Increase u, x and s
        u = u + dz(1:m);
        x = x + dz(m+1:m+n);
        s = s + dz(m+n+1:end);

    end

    %Obtain the final (optimal) values calculated
    uopt = u;
    xopt = x;
    sopt = s;
    lopt = l;
    vopt = v;

    u_nmpc = u(1);

end
```