Model Predictive Control : Exercise 6

Consider a slot car racing track for which the curve of the track in a 2D plane is parameterized by $(x(\lambda), y(\lambda))$, for $\lambda \in \mathbb{R}$.

For any given λ we will say that the slot car is located at the point $(x(\lambda), y(\lambda)) \in \mathbb{R}^2$ on the curve and thus the position of the car on the track is entirely determined by λ, its velocity by $v = \dot{\lambda}$, and the state vector of the car is (λ, v).

The curvature of the track is given by the function $\kappa(\lambda)$ and the car is known to flip out from the track if at any $\lambda \in \mathbb{R}$, the speed v exceeds $\frac{1}{1+\kappa(\lambda)}$.

Figure 1: A slot car race.

Let the car have a forward acceleration input u_1, a brake u_2 and a viscous frictional force acting on it. The dynamics of the car can be written as

$$\begin{aligned} \dot{v} &= \gamma u_1 - \alpha u_2 v - \beta v^3 \\ \dot{\lambda} &= v \end{aligned} \tag{1}$$

with $\gamma = 3$, $\alpha = 1.6$, $\beta = 0.01$, $u_1 \in [0, 1]$, $u_2 \in [0, 1]$ where γ is an accelaration constant, α is a braking constant and β a frictional constant.

For notational convenience let the continuous time dynamics from (1) be defined using a function f that takes as input a state $X = \begin{bmatrix} \lambda & v \end{bmatrix}^T$ and an input vector $U = \begin{bmatrix} u_1 & u_2 \end{bmatrix}^T$ and returns $\dot{X}$ (i.e. $\dot{X} = f(X, U)$). Let h be a fixed integration time step, and t_0 some initial time, then the state at time $t_0 + h$ is given by $X(t_0 + h) = \int_0^h f(X(t_0 + s), U(t_0 + s))ds$. There are several methods for approximating this integration under a constant input (i.e. when $U(t_0+s) = U(t_0)$ for all $s \in [0, h]$) (e.g. RK4, Euler). Let such an integration approximation be denoted by the function $f_{discrete}$ such that $X(t_0 + h) = f_{discrete}(X(t_0), U(t_0))$.

Note: You will need to have installed CASADI for this exercise.

Prob 1 | **Integration / Discretization**

a) Implement the RK4 and Forward Euler integrators and use them to define a corresponding $f_{discrete}$. Test your integrators from the initial condition $X(t_0) = (0, 0.5)$ and $U(t_0) = (0, -0.01)$ by computing $X(t_0 + h) = f_{discrete}(X_0(t_0), U(t_0)))$

b) For $X(0) = (0, 0.5)$ and a given input $U(t)$, simulate the movement of the car for 10 seconds using $f_{discrete}$ corresponding to your RK4 and Euler implementations for two cases $h = 0.1$ and $h = 0.5$. Plot and compare the integration errors for your trajectories with the ODE45 simulation given in the code template.

Prob 2 | **Gradients**

a) Write two functions `jac_x.m` and `jac_u.m` to compute the Jacobians $\nabla_X f_{discrete}$ and $\nabla_U f_{discrete}$ using a finite difference approximation.[1]

Compare the errors of finite difference approximation to the algorithmic differentiated Jacobians of your RK4 integrator provided in the template (take note of the syntax to find Jacobian and defining functions using casadi for future use).

b) Linearize $f_{discrete}$ around a point (X_0, U_0) using finite differences and algorithmic differentiation and write the linearized discrete time dynamics

$$f_{lin}(X, U) = \nabla_X f(X_0, U_0)(X - X_0) + \nabla_U f(X_0, U_0)(U - U_0) + f(X_0, U_0)$$

Simulate the linearized system using your RK4 integrator with $h = 0.5$. Compare the result to simulation of the nonlinear f using RK4.

Prob 3 | **Nonlinear MPC**

- Implement an NMPC controller with the RK4 integrator. Use horizon length $N = 10$, and a sample period of $h = 0.025$.

  Simulate the closed loop system with your controller and confirm that constraints are satisfied, and the system is performing as expected.

  Try some 'complex' tracks. Does your controller still work as expected?

- Tune your controller to minimize the use of the brakes[2]. Do you get the desired / expected behaviour from your tuning? Why, why not?

  Can you get the expected behaviour if you change the sample period and / or horizon length? Why / why not?

[1] Finite difference approximation of the derivative of a function $g(x)$ is $\dot{g}(x) \approx 0.5(g(x + \delta) - g(x - \delta))$

[2] Note that non-convex optimization can be finicky. You may find that many of your tuning configurations result in unsolvable problems.