# NMPC Design for an overhead crane with pysolnp/SOLNP

https://www.youtube.com/user/StepanOzana

www.stepan-ozana.com

https://pypi.org/project/pysolnp/

(C) Stepan Ozana (2021)

email: stepan.ozana@vsb.cz

(C) Filip Krupa (2021)

email: filip.krupa@vsb.cz

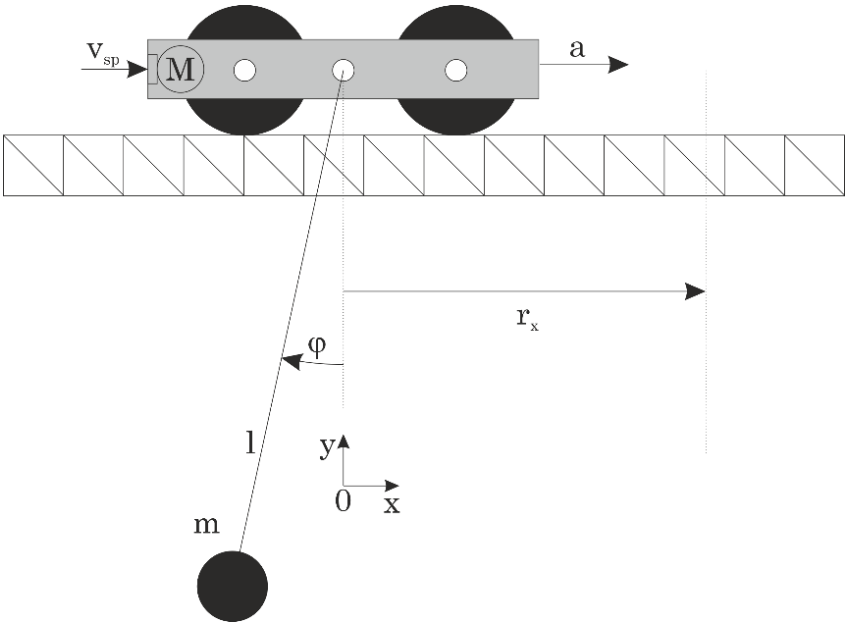



Fig. The situation scheme

Objective: The design and implementation of nonlinear model predictive control algorithm that moves an overhead crane in *x*-direction to a given desired position setpoint $r_x$ while minimizing a given cost function over prediction horizon (RHC algorithm). The cost function may represent different requirements, typically minimization of swaying of the arm and other customized constraints. The mass load is supposed to be concentrated into a single mass point. The cart is driven by a DC motor equipped with the control unit containing a speed controller and it is controlled by the speed setpoint $v_{sp}$. Dynamics of the motor is considered via approximation of the dynamics of the entire speed loop with the 1st order system having the time constant $\tau$.

Basic differential equation for the arm movement:

$$I \cdot \ddot{\varphi} + m \cdot g \cdot l \cdot \sin\varphi + b \cdot \dot{\varphi} = m \cdot l \cdot a \cdot \cos\varphi$$

$$I = m \cdot l^2$$

Where

$m$ … weight of the mass load $[kg]$

$g$ … gravity constant $[m \cdot s^{-2}]$

$l$ …length of the arm $[m]$

$b$ …friction coefficient $[kg \cdot m^2 \cdot s^{-1}]$

$I$ …moment of inertia $[kg \cdot m^2]$

$\varphi$ …angle of the arm $[rad]$

$a$ …cart acceleration $[m \cdot s^{-2}]$

The above-mentioned equation is extended by a basic trivial equation describing physical relation between the speed and the position both for the cart and the arm.

Choice and notation of state variables:

$x_1$…cart linear position $[m]$

$x_2$…cart speed $[m \cdot s^{-1}]$

$x_3 = \varphi$… angle of the arm $[rad]$

$x_4 = \dot{\varphi}$…angular speed of the arm $[rad \cdot s^{-1}]$

$u = v_{sp}$…speed setpoint $[m \cdot s^{-1}]$

Remark: The movement of the overhead crane is caused by its acceleration induced by its speed which is controlled by external speed control loop. We model the dynamics of the speed control loop by the 1st order system with time constant $\tau$ reflecting dynamics of speed PID controller and dynamics of the motor. Therefore, the input to the overall system which is subject to NMPC design is the speed setpoint $u = v_{sp}$.

Full nonlinear state-space description

Supposing a general state-space description $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$, the dynamics of the overhead crane driven by DC motor may be described by the following differential equations:

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = \frac{1}{\tau} \cdot u - \frac{1}{\tau} \cdot x_2$$

$$\dot{x}_3 = x_4$$

$$\dot{x}_4 = \frac{1}{I}\left[-m \cdot g \cdot l \cdot \sin x_3 - b \cdot x_4 + m \cdot l \cdot \left(\frac{1}{\tau} \cdot u - \frac{1}{\tau} \cdot x_2\right) \cdot \cos x_3\right]$$

With its initial state $x_{10} = x_{20} = x_{30} = x_{40} = 0$.

Simulation scheme:

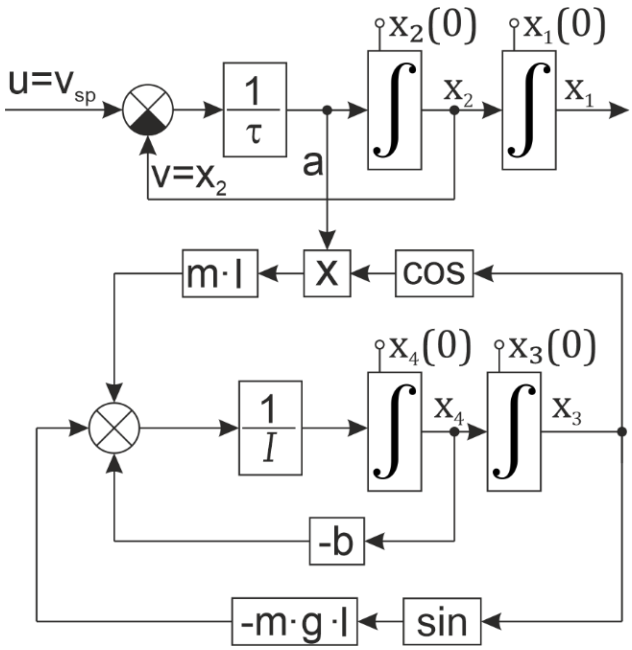



Fig. Simulation block scheme for the simulation model

NLP formulation

The continuous OCP problem of minimizing the cost function over a given prediction horizon is transformed into NLP using direct transcription described below with the time step $h = 0.25\ [s]$. The control signal and the states are discretized over the prediction horizon in a given number of knot points, $N_1 = 11$. The prediction horizon is chosen as $P = (N_1 - 1) \cdot h\ [s] = 10\ [s]$.

Nonlinear programming problem is then implemented and solved both in original version of *SOLNP* solver (https://web.stanford.edu/~yyye/matlab/) and its Python implementation *pysolnp* (https://pypi.org/project/pysolnp/) so the results can be compared, showing the identity.

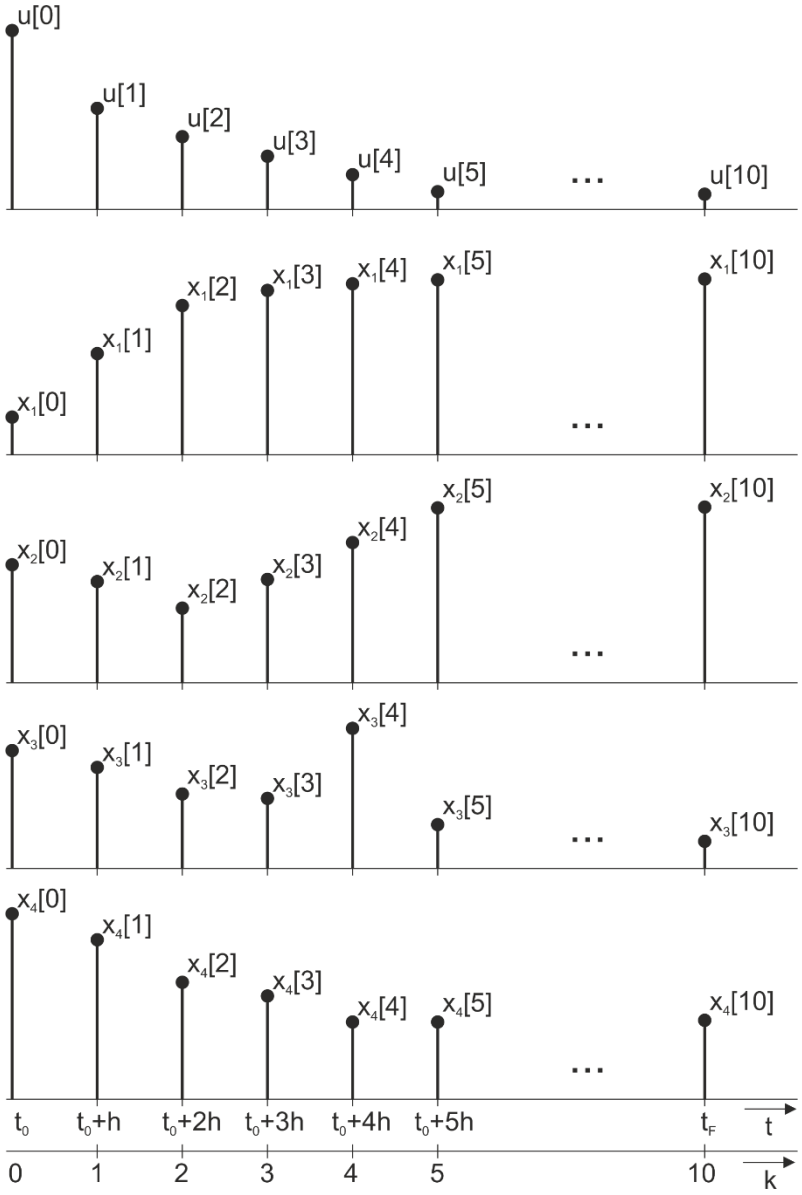



Fig. Discretization of control signal and state variables over time

NMPC design

Decision vector of variables – state variable and all states variables discretized over prediction horizon in 11 knot points:

$$\boldsymbol{z} = \{u[0], u[1], u[2], u[3], u[4], u[5], u[6], u[7], u[8], u[9], u[10],$$
$$x_1[0], x_1[1], x_1[2], x_1[3], x_1[4], x_1[5], x_1[6], x_1[7], x_1[8], x_1[9], x_1[10],$$
$$x_2[0], x_2[1], x_2[2], x_2[3], x_2[4], x_2[5], x_2[6], x_2[7], x_2[8], x_2[9], x_2[10],$$
$$x_3[0], x_3[1], x_3[2], x_3[3], x_3[4], x_3[5], x_3[6], x_3[7], x_3[8], x_3[9], x_3[10],$$
$$x_4[0], x_4[1], x_4[2], x_4[3], x_4[4], x_4[5], x_4[6], x_4[7], x_4[8], x_4[9], x_4[10]\}$$

Next step is to define the cost function $J(\boldsymbol{z})$ together with constraints.

For simplicity we may use the notation

$$\boldsymbol{z} = [\boldsymbol{u}_d, \boldsymbol{x}_d] = [\boldsymbol{u}_d, \boldsymbol{x}_{1d}, \boldsymbol{x}_{2d}, \boldsymbol{x}_{3d}, \boldsymbol{x}_{4d}]$$

where

$$\boldsymbol{u}_d = \boldsymbol{z}[0: N_1 - 1]$$

$$\boldsymbol{x}_{1d} = \boldsymbol{z}[N_1: 2N_1 - 1]$$

$$\boldsymbol{x}_{2d} = \boldsymbol{z}[2N_1: 3N_1 - 1]$$

$$\boldsymbol{x}_{3d} = \boldsymbol{z}[3N_1: 4N_1 - 1]$$

$$\boldsymbol{x}_{4d} = \boldsymbol{z}[4N_1: 5N_1 - 1]$$

are vectors containing discrete values representing discretized control and state variables over time.

The cost function is fully customizable and expert-based. Here, the quadratic function is chosen to penalize the position $x_1$ of the cart in x-direction compared to the reference $r_x$, then the rest of state variables $x_2$ to $x_4$ are penalized with respect to zero references and $x_3$ is multiplied by time to increase the importance of penalization of arm swaying with the increasing time. The cost function also penalizes the energy effort spent on manipulated value and the rate of the manipulated value. Individual weighting coefficients are used for these components.

$$J(z) = \sum_{k=0}^{N_1-1} Q_1 \cdot (x_{1d}[k] - r_x)^2 + Q_2 \cdot (x_{2d}[k])^2 + Q_3 \cdot t_k \cdot (x_{3d}[k])^2 +$$

$$+Q_4 \cdot (x_{4d}[k])^2 + R_1 \cdot (u[k])^2 + \sum_{k=0}^{N_1-2} R_2 \cdot (\Delta u[k])^2$$

where

$$t_k = t_0 + k \cdot h$$

$$\boldsymbol{r} = [r_x, 0, 0, 0]$$

$$\Delta u[k] = u[k+1] - u[k]$$

Lower and upper limits – constraints on decision variables to be used in NLP formulation:

$$x_1 \in [x_{1min}\ , x_{1max}]$$

$$x_2 \in [x_{2min}\ ,\ x_{2max}]$$

$$x_3 = \varphi \in [\varphi_{min}\ ,\ \varphi_{max}]$$

$$x_4 = \dot{\varphi} \in [\dot{\varphi}_{min}\ ,\ \dot{\varphi}_{max}]$$

$$u \in [u_{min}\ ,\ u_{max}]$$

Equality constraints to be used in NLP formulation:

a) Boundary constraints $\psi$:

$\psi_1 = x_1[0] - x_{10}$

$\psi_2 = x_2[0] - x_{20}$

$\psi_3 = x_3[0] - x_{30}$

$\psi_4 = x_4[0] - x_{40}$

$\psi_5 = x_1[N_1 - 1] - r_x = x_1[10] - r_x$

where initial state is as follows: $x_{10} = x_{20} = x_{30} = x_{40} = 0$

b) Differential defect constraints $\xi$ are expressed using RK4 integration scheme.

$\xi_k = x_{k+1} - x_k - \frac{1}{6} \cdot h \cdot (k_1 + 2k_2 + 2k_3 + k_4)$ for natural numbers $k \in [0, N_1 - 2] \in \mathbb{N}^0$

$k_1 = f(x_k, u_k)$

$k_2 = f(x_k + \frac{h}{2} \cdot k_1, \frac{u_{k+1} + u_k}{2})$

$k_3 = f(x_k + \frac{h}{2} \cdot k_2, \frac{u_{k+1} + u_k}{2})$

$k_4 = f(x_k + h \cdot k_3, u_{k+1})$

Parameters of the system and parameters used in NLP formulation:

$g = 9.81$

$m = 100$

$l = 5$

$\tau = 0.1$

$b = 500$

$r_x = 2$

$x_{1min} = -4$

$x_{1max} = 4$

$x_{2min} = -10$

$x_{2max} = 10$

$x_{3min} = -0.5$

$x_{3max} = 0.5$

$x_{4min} = -10$

$x_{4max} = 10$

$\boldsymbol{Q} = [Q_1, Q_2, Q_3, Q_4] = [1\ , 0.1\ , 50\ , 0.1]$

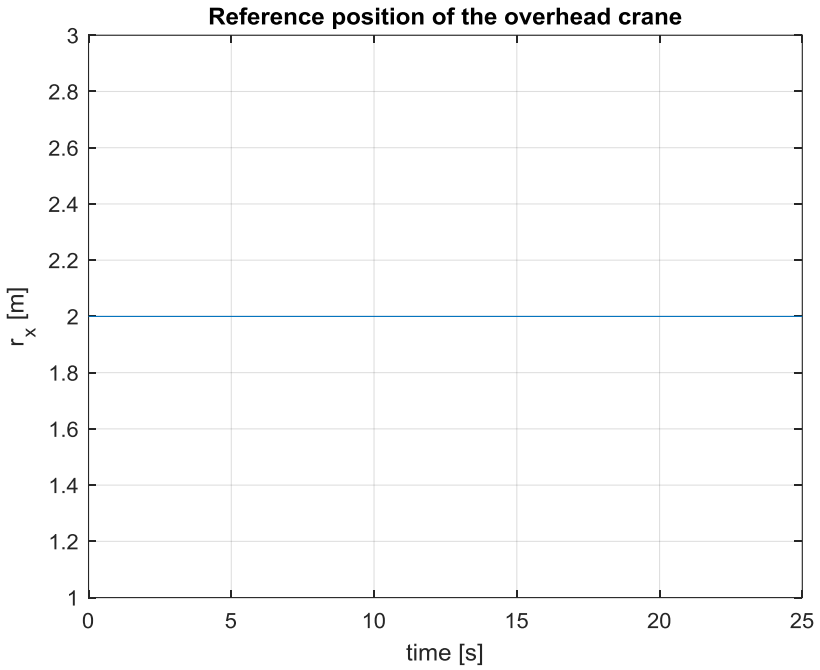
$R_1 = R_2 = 1$

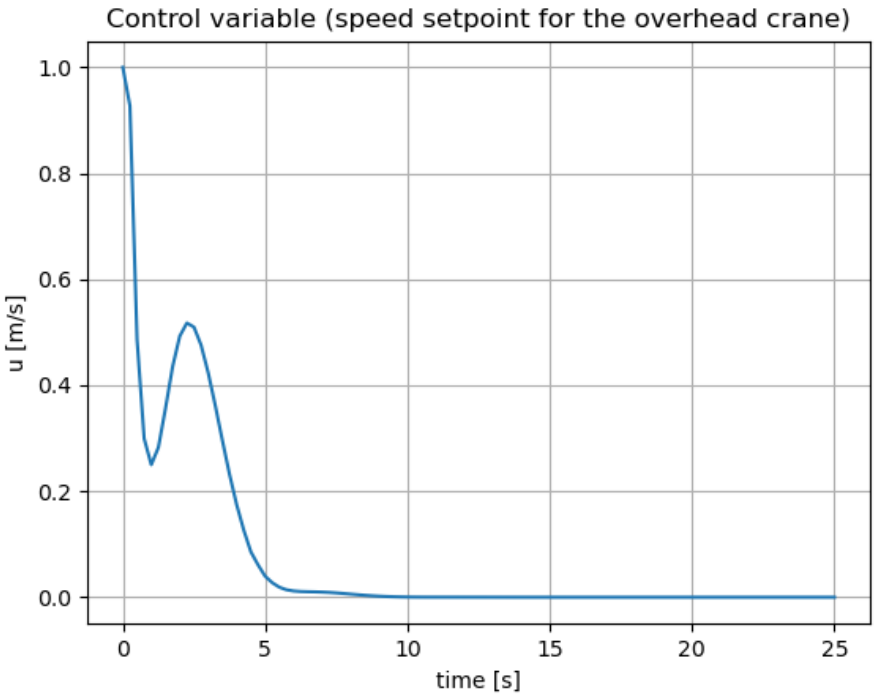
Running the case study:

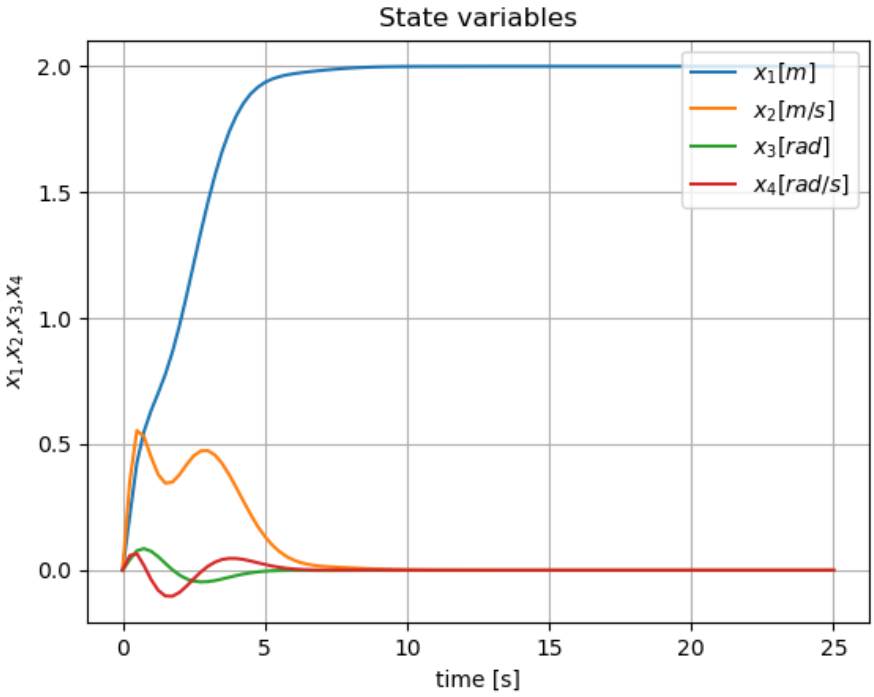
a) SOLUTION_SOLNP:  run *main.m* in Matlab

b) SOLUTION_pysolnp: run *main.py* in Python

Graphical results from NMPC loop:






Remark: The animation of the over crane movement is enclosed.

Discussion

The *pysolnp* implementation makes it possible to implement the above-mentioned NMPC algorithm for real-time embedded targets. This is carried out with the use the REXYGEN control system (www.rexygen.com). As a result of this, we can construct NMPC optimizer with the use of REXYGEN and some embedded target (Windows/Linux based, any PC or IPC or SBC). It can be operated as a single unit hosting all the functionality or in combination with some existing PLC, see the picture below. The prototype of this REXYGEN solution is already done and the tested successfully, using RPi minicomputer as the embedded target.

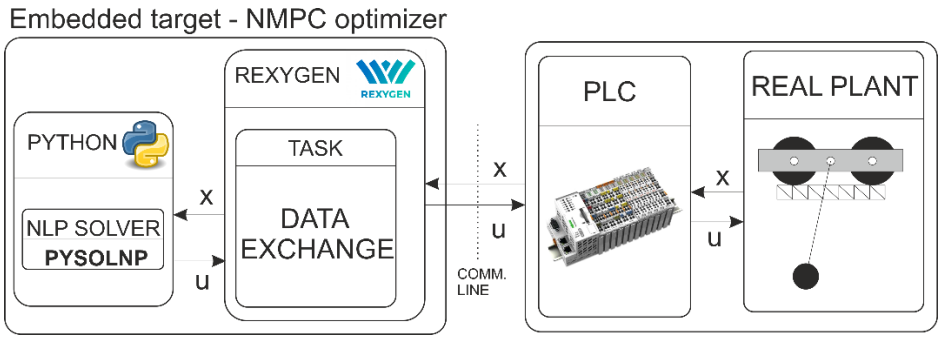



Fig. The optimizer providing NMPC algorithm for a standard PLC via communication line

Acknowledgment: Many thanks to Krister Sune Jakobsson for incredible job he made with *pysolnp*.