2MAE404 MIMO control
Homework report 2

(with later improvements)

Maciej Michałów Marco Wijaya

December 2, 2022

1 Exercise 1 – Flying Chardonnay trajectory

1.1 Introduction

The goal of this excercise is using nonlinear model predictive control (NLMPC) to perform a manoeuvre with the Flying Chardonnay drone while keeping as much of the drink it transports. The manoeuvre is simply a horizontal shift $4[m]$, with a time limit of $2[s]$. An additional limitation is the frequency of the control – set to $50[Hz]$.

The code written for this excercise was based on the example code for a landing rocket, with all necessary modifications for the drone problem. The dynamics of the drone have been implemented in the previous excercise – with the only modification being two added states, r_n and r_d, which represent the position in the North-Down frame.

1.2 Drone parameters

The parameters of the drone are defined in the excercise.

```
% drone physical parameters
drone.mass = 1; % mass of drone md = 1 kg
drone.cupmass = 1; % mass of cupp mc = 1 kg
drone.l = 1; % length of rod l = 1 m
drone.l_d = 1; % distance to propeller ld = 1 m
drone.J = 1; % drone moment of inertia J = 1 kg*m^2
drone.CD = 1; % drone drag coefficients CD = 0.01
drone.g = 10; % gravity [m/s^]

% wind vector
wind = [0;0];

% parameters defining the control objectives
target.rd = 0;
target.rn = 4;
t_max = 2; % time of manoeuvre
```

1.3 NLMPC settings

The basic parameters of the control problem are:

```
% MPC handler init
nx = 8; % dimension of state vector
ny = 8; % dimension of output vector
nu = 2; % dimension of input vector
nlobj = nlmpc(nx,ny,nu); % create nonlinear MPC solver handle
```

As stated before, the manoeuvre is performed in $2[s]$ with a refresh rate of $50[Hz$. The control and prediction horizons both span the entirety of that time:

```
% MPC basic control parameters
nlobj.Ts = 1/50; % sampling time (50 Hz)
nlobj.PredictionHorizon = t_max/nlobj.Ts; % how many steps to look ahead
nlobj.ControlHorizon = t_max/nlobj.Ts; % the entire trajectory is controlled
```

The NLMPC uses the prieviously developed implementation of the Flying Chardonnay dynamics:

```
% MPC dynamics model
nlobj.Model.StateFcn = "drone_dynamics";
nlobj.Model.OutputFcn = @(x,u,drone, wind, target) x;
nlobj.Model.IsContinuousTime = true;

% setting the number of additional parameters of solver
nlobj.Model.NumberOfParameters = 3;
```

The optimisation solver is set to run for 1000 iterations max (in practise it never did, because it found a solution earlier). In case if it were not enough, it is allowed to use suboptimal solutions. A custom cost function is used: the amount of water spilled by the drone (this is implemented in the `chardonnay_animate` function, which was just stripped of the plotting functionality and signed in a way compatible with the MPC toolbox):

```
% optimisation solver settings
nlobj.Optimization.SolverOptions.Display = 'iter';
nlobj.Optimization.UseSuboptimalSolution = true;
nlobj.Optimization.SolverOptions.MaxIterations = 1000;

% custom cost function
nlobj.Optimization.CustomCostFcn = "CostFunction";
```

A custom equality constraint function is also used:

```
% equality constraints
nlobj.Optimization.CustomEqConFcn = "EqualityConstraints";
```

The equality constraint function is the following. It forces the drone go to point $[4, 0]$ at the end of the trajectory, and sets all other final states to 0:

```
function constraints = EqualityConstraints(X,U,data, drone, wind, target)
    c1 = X(end,1:6); % zero end values
    c2 = X(end,7) - target.rn; % end position
    c3 = X(end,8) - target.rd; % end position
    constraints = [c1'; c2'; c3];
end
```

For the input constraints, it was assumed that the propellers can produce $80[N]$ of thrust each. This is probably too optimistic, but lower values did tend affect the spilling performance (albeit not drastically – $50[N]$ was almost just as good).

```
% input constraints
mv_struct = struct;
mv_struct.Min = 0; % min thrust per propeller in [N]
mv_struct.Max = 80; % max thrust per propeller in [N]
mv_struct.Units = 'N';
nlobj.ManipulatedVariables = [mv_struct, mv_struct];
```

The initial state was set to 0, and the initial thrust – to a value that keeps the drone stationary:

```
% initial conditions for the simulation
x0 = zeros(8,1); % all parameters equal to 0
hover_thrust = (drone.cupmass + drone.mass)*drone.g/2; % hovering thrust
u0 = hover_thrust*[1, 1];
% sanity checks
validateFcns(nlobj,x0,u0,[],{drone, wind, target});
```

Finally, the optimal trajectory is being computed, using the previous best solution as a starting point:

```
% initial guess for the optimisation
nloptions = nlmpcmoveopt;
nloptions.Parameters = {drone, wind, target};

% initial guess u and x arrays (spills 20% of the water)
load InitialGuess81_3.mat;
nloptions.MV0 = InitialGuess.U;
nloptions.X0 = InitialGuess.X;

% computation of the MPC trajectory
[~,~,info] = nlmpcmove(nlobj,x0,u0,[],[],nloptions);
```

1.4 Results

The final trajectory satisfied the boundary conditions and preserved around 82% of the drink, which can be deemed reasonable. The manoeuvre is visualised in Figure 1. It is possible that significant improvements can be still made, since nonlinear MPC is prone to local minima.

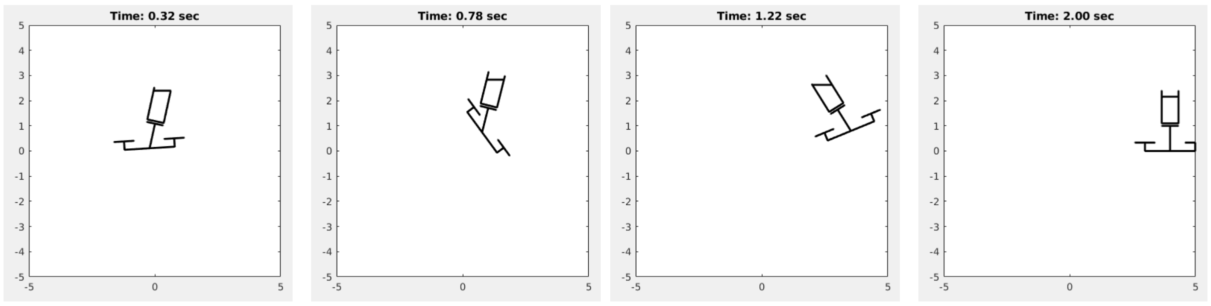

Figure 1: 4 frames of the visualisation of the generated trajectory.

2 Exercise 2 – analytical optimal trajectory for rotating vector

2.1 Introduction

The goal of this exercise is to find the optimal trajectory $(\mathbf{x}_0, u_0), (\mathbf{x}_1, u_1), ..., (\mathbf{x}_N, u_N)$ of a dynamical system described by the discrete state-space equations:

$$\mathbf{x}_{k+1} = \begin{pmatrix} \cos u_k & -\sin u_k \\ \sin u_k & \cos u_k \end{pmatrix} \mathbf{x}_k \tag{1}$$

$$\mathbf{y}_k = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \mathbf{x}_k = \mathbf{x}_k \tag{2}$$

The initial and final positions are $\mathbf{x}_0 = (1, 0)$ and $\mathbf{x}_N = (0, 1)$. The optimal trajectory should minimise the cost function:

$$J = \sum_{k=0}^{N} \alpha |\mathbf{y}_k|^2 + \beta u_k^2 \tag{3}$$

2.2 Dimensionality reduction

Looking at the state equations, it can be noticed that the state transition matrix is a rotation matrix, rotating the previous state (which is a 2D vector) by the angle u_k counterclockwise. The initial and final states indicate that the position is initialised with a 0° angle and terminated at an angle of 90°. Moreover, the magnitude of the state vector is constant, since the state transition matrix describes a pure rotation. Therefore, the system dynamics can be rewritten in terms of a single variable, θ, describing the phase angle of $\mathbf{x}$:

$$\theta_k = \arg(\mathbf{x}_k) \tag{4}$$

$$\theta_{k+1} = \theta_k + u_k \tag{5}$$

The boundary conditions expressed in terms of θ are:

$$\begin{cases} \theta_0 = 0° \\ \theta_N = \pi/2 \end{cases} \tag{6}$$

Note that the recursive state equation Equation 5 can be easily rewritten in an explicit form:

$$\theta_{k+1} = \theta_k + u_k = \sum_{i=0}^{k} u_i + \theta_0 = \sum_{i=0}^{k} u_i \tag{7}$$

Taking into account the known final value $\theta_N = \pi/2$, this yields a constraint:

$$g(u) = \sum_{k=0}^{N-1} u_k - \pi/2 = 0 \tag{8}$$

Given that $|y_k| = |x_k| = |x_0| = 1$, as noticed before, the cost function (Equation 3) can be simplified:

$$J = \sum_{k=0}^{N} \alpha|\mathbf{y}_k|^2 + \beta u_k^2 = \alpha(N+1) + \beta \sum_{k=0}^{N} u_k^2 \tag{9}$$

Since the first term is a constant, it can be ignored. Moreover, it can be noticed that u_N does not affect the final state (since the control input u_k affects the state u_{k+1} rather than the current one). Because of this, the optimal choice is obviously $u_N = 0$, which is the only value not incrementing the cost function. It also means that in Equation 9, the final (i.e. with index N) term of the series can be ignored. The final form of the cost function is therefore:

$$J = \beta \sum_{k=0}^{N-1} u_k^2 \tag{10}$$

2.3 Constrained optimisation

The final optimisation problem, consisting of the cost function in Equation 10 and constraint function Equation 8, can be solved using the method of Lagrange multipliers. The Lagrangian of this problem is:

$$\mathcal{L}(u, \lambda) = \beta \sum_{k=0}^{N-1} u_k^2 + \lambda g(u) \tag{11}$$

$$\mathcal{L}(u, \lambda) = \sum_{k=0}^{N-1} (\beta u_k^2 + \lambda u_k) - \lambda \frac{\pi}{2} \tag{12}$$

The Lagrangian must fulfill two necessary conditions at the optimal point:

$$\frac{\partial \mathcal{L}(u_k, \lambda)}{\partial u_k} = 2\beta u_k + \lambda = 0 \tag{13}$$

$$\frac{\partial \mathcal{L}(u_k, \lambda)}{\partial \lambda} = \sum_{k=0}^{N-1} u_k - \pi/2 = 0 \tag{14}$$

Equation 14 is always true. Equation 13 can be rearranged to:

$$u_k = \frac{-\lambda}{2\beta} \tag{15}$$

It can be already noted that u_k does not depend on k. Substituting equation Equation 15 into Equation 8 yields:

$$\sum_{k=0}^{N-1} \frac{-\lambda}{2\beta} = \frac{-N\lambda}{2\beta} = \frac{\pi}{2} \tag{16}$$

This can be solved for λ:

$$\lambda = -\frac{\beta\pi}{N} \tag{17}$$

Substituting back into Equation 15 yields the final formula for optimal control u_k:

$$u_k = \frac{-\lambda}{2\beta} = \frac{-\pi\beta}{N} . \frac{-1}{2\beta} = \frac{\pi}{2N} \tag{18}$$

Substituting this into Equation 7 results in the formula for the optimal trajectory θ_k

$$\theta_k = \sum_{i=0}^{k-1} u_i = \frac{\pi k}{2N} \tag{19}$$

Converting back to the original state $\mathbf{x}$, the optimal state trajectory is:

$$\mathbf{x}_k = \mathbf{y}_k = \begin{pmatrix} \cos\left(\frac{\pi k}{2N}\right) \\ \sin\left(\frac{\pi k}{2N}\right) \end{pmatrix} \tag{20}$$

2.4 Final solution

The optimal solution for $N = 2021$ is, as expressed in Equation 18 and Equation 20:

$$\begin{cases} \mathbf{x}_k = \begin{pmatrix} \cos\left(\frac{\pi k}{4042}\right) \\ \sin\left(\frac{\pi k}{4042}\right) \end{pmatrix} \text{ for } k = 0, 1, 2, ..., 2021 \\ u_k = \frac{\pi}{4042} \text{ for } k = 0, 1, 2, ..., 2020 \\ u_{2021} = 0 \end{cases} \tag{21}$$

For $N = 2$ ($k = 0, 1, 2$), the optimal trajectory is:

$$\begin{cases} \mathbf{x} = \left(\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} \sqrt{2}/2 \\ \sqrt{2}/2 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix} \right) \\ u = \left(\frac{\pi}{4}, \frac{\pi}{4}, 0\right) \end{cases} \tag{22}$$