HW2: Modeling the Flying-Chardonnay

Work done by –

Akash Sharma

Yash Malandkar

Exercise - 1

The following MATLAB code finds the optimum trajectory (x(t),u(t)) to shift a drone horizontally by 4m while balancing a glass of water. The task is achieved while keeping the water level reasonably high. The 2 given constraints of sample frequency = 50Hz and maximum maneuver time = 2s are also met.

The following 3 code algorithms define different aspects of the optimization.

The final amount of water retained = **84.157%**

Trajectory_Generation

```
clear all
clc
close all

%% Creating a structure for the system's physical properties
drone.md = 1;
drone.mc = 1;
drone.l = 1;
drone.ld = 1;
drone.J = 1;
drone.Cd = 0.01;
drone.g = 10;

%% MPC handler config
% 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

% MPC basic control parameters
nlobj.Ts = 0.02; % sampling time equal to usual servo frequency. This ensures a sample
frequency = 50Hz
nlobj.PredictionHorizon = 100; % how many steps to look ahead (2 sec: Maneuver time)
nlobj.ControlHorizon = 100; % only first 100 actions can be variable
```

```
% this is where we link the MPC handler to our dynamics model
nlobj.Model.StateFcn = "drone2d_dynamics";
nlobj.Model.OutputFcn = "drone2d_output";
nlobj.Model.IsContinuousTime = true;

% setting the number of additional parameters of solver.
% we only use the DRONE structure as additional argument!
nlobj.Model.NumberOfParameters = 1;

% setting optimization solver options here!
nlobj.Optimization.SolverOptions.Algorithm = 'active-set';
nlobj.Optimization.SolverOptions.Display = 'iter';
nlobj.Optimization.UseSuboptimalSolution = true;
nlobj.Optimization.SolverOptions.MaxIterations = 8;

% this set the equality constraints of the MPC problem:
%  - in our case, we want all the state vectors to be 0 except for the
%    horizontal and vertical positions.
% These constraints are set keep in mind the destination of the
% drone. Assuming initial coordinates to be (0,1), the final coordinates
% should be (4,1). x = [pn, pd, vn, vd, the, thed, gam, gamd], therefore,
% the required constraints are -
% pn-4 = 0 ; pd-1 = 0 ; all other states = 0

nlobj.Optimization.CustomEqConFcn = @(X,U,data,params) [X(end,1)-4 X(end,2)-1
X(end,3:end)]';

% this set the inequality constraints of the MPC problem:
%  - in our case, we want the drone to stay above ground all times during
%  the entire flight phase.
nlobj.Optimization.CustomIneqConFcn = @(X,U,e,data,params) X(1:end,2);

% These the cost function options. The output variables y is defined as -
% y = [pn pd vn vd the thed gam gamd].
% In order to make sure the drone reaches the precise desired location, the
% position terms have been tuned to have aggressive control with a tuning of
% 10.

nlobj.Weights.OutputVariables = [10 10 0 0 0 0 0 0];
nlobj.Weights.ManipulatedVariables = [1 1];
nlobj.Weights.ManipulatedVariablesRate = [1 1];

% Input Constraints!
nlobj.ManipulatedVariables = struct(Min={0;0},Max={13;13},RateMin={-1;-
1},RateMax={3;3});

% initial conditions for the simulation
x0 = zeros(8,1);
x0(2) = 1; % assume we are 1 meter above ground
u0 = 0.5*(drone.mc+drone.md)*drone.g*ones(nu,1); % This is numerically equal to which is half
                                                 % the weight of the system
                                                 % taken by the propeller.

% this function runs some sanity checks for us
```

```
validateFcns(nlobj,x0,u0,[],{drone});

% In the following, we set up an initial guess for this problem since it is
% a difficult problem to converge to a solution.
nloptions = nlmpcmoveopt;
nloptions.Parameters = {drone};
load InitialGuess_drone.mat;
nloptions.MV0 = InitialGuess_drone.u;
nloptions.X0 = InitialGuess_drone.x;

% solve the problem!!!
[~,~,info] = nlmpcmove(nlobj,x0,u0,[],[],nloptions);

%% extract the solution data and animate the trajectory

t = info.Topt';
r = [info.Xopt(:,1)'; info.Xopt(:,2)'];
the = info.Xopt(:,5)';
gam = info.Xopt(:,7)';

drone2d_animate(drone, t, r, the, gam);
```

drone2d_dynamics

```
function x_dot = drone_dynamics(x,u, drone)
% DRONE_DYNAMICS
% All values are in S.I. units!!
%   x = [pn pd vn vd the thed gam gamd]
%   u = [T1; T2]
%   w = [wn; wd]
%   drone = structure containing all drone physical parameters (e.g., mass,
%   length)

%% Drone params[in S.I. units]

md = drone.md; % mass of drone
mc = drone.mc; % mass of cup
l = drone.l; % cup to center of mass
ld = drone.ld; % propeller to center of mass
J = drone.J; % moment of inertia
Cd = drone.Cd; % g coefficient drag
g = drone.g; % gravity
w = [2 -3]; % simulating wind

%% Init variables

the = x(5); % all angles are in radians
gam = x(7);
alp = the+gam;
```

```
T1 = u(1);
T2 = u(2); % in Newtons

%% Defining matrices
% A*sol = B - Defined in accordance with the equation 23

A = [md  0         0                 0          (sin(alp));
      0  md        0                 0          (cos(alp));
      mc 0  (-mc*l*cos(alp)) (-mc*l*cos(alp))  (-sin(alp));
      0  mc (mc*l*sin(alp))  (mc*l*sin(alp))   (-cos(alp));
      0  0         J                 0               0    ];

B = [(-(T1+T2)*sin(the)-Cd*(x(3)-w(1)));
    (md*g-(T1+T2)*cos(the)-Cd*(x(4)-w(2)));
       (-mc*l*((x(6)+x(8))^2)*sin(alp));
      (mc*g-mc*l*((x(6)+x(8))^2)*cos(alp));
                ((T2-T1)*ld)              ];

sol = A\B;

%% Obtaining x_dot from above

x_dot = zeros(8,1);
x_dot(1:2) = x(3:4);
x_dot(3:4) = sol(1:2);
x_dot(5) = x(6)*180/pi; % to obtain the angles in degrees
x_dot(6) = sol(3);
x_dot(7) = x(8)*180/pi;
x_dot(8) = sol(4);

end
```

InitialGuess_script

```
clear all
clc
close all

% Time matrix

Time = (0:0.02:2)';
d2r= pi/180;

% x matrix with an initial guess for each of the states

pn = linspace(0,4,101)';
pd = ones(101,1);
vn = [linspace(0,2,50) linspace(2,0,51)]';
vd = zeros(1,101)';
the = [linspace(0,1.5*d2r,50) linspace(1.5*d2r,0,51)]';
thed = [linspace(0,1.5*d2r,50) linspace(1.5*d2r,0,51)]';
gam = [linspace(0,1.5*d2r,50) linspace(1.5*d2r,0,51)]';
```

```
gamd = [linspace(0,1.5*d2r,50) linspace(1.5*d2r,0,51)]';

x = [pn pd vn vd the thed gam gamd];

% u matrix

u = ones(101,2);

InitialGuess_drone = struct('Time',Time,'x',x,'u',u);
save InitialGuess_drone.mat InitialGuess_drone

clear pn pd vn vd the thed gam gamd
```

Exercise – 2

Exercise - 2

For $N = 2$

2> $$J = \sum_{K=0}^{N} \alpha |y_K|^2 + \beta u_K^2 \quad \Big|_{\text{minimize}}$$

$$= \alpha |y_0|^2 + \beta u_0^2 + \alpha |y_1|^2 + \beta u_1^2 + \alpha |y_2|^2 + \beta u_2^2$$

$$x_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad x_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$y_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix}, \quad y_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix} \qquad y_1 = \begin{pmatrix} \cos u_0 \\ \sin u_0 \end{pmatrix}$$

from eq ①

$$X_1 = \begin{pmatrix} \cos u_K & -\sin u_K \\ \sin u_K & \cos u_K \end{pmatrix} \underbrace{\begin{pmatrix} 1 \\ 0 \end{pmatrix}}_{X_0}$$

$$= \begin{pmatrix} \cos u_0 \\ \sin u_0 \end{pmatrix} \quad — \quad ①$$

$$\overbrace{\begin{pmatrix} 0 \\ 1 \end{pmatrix}}^{X_2} = \begin{pmatrix} \cos u_1 & -\sin u_1 \\ \sin u_1 & \cos u_1 \end{pmatrix} \quad — \quad ②$$

Adding ① + ②, we get

$$\begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \cos u_1 & -\sin u_1 \\ \sin u_1 & \cos u_1 \end{pmatrix} \begin{pmatrix} \cos u_0 \\ \sin u_0 \end{pmatrix}$$

$$= \begin{pmatrix} \cos u_1 \cos u_0 - \sin u_1 \sin u_0 \\ \sin u_1 \cos u_0 + \cos u_1 \sin u_0 \end{pmatrix}$$

$$= \begin{pmatrix} \cos(u_1 + u_0) \\ \sin(u_1 + u_0) \end{pmatrix} \Rightarrow \quad u_1 + u_0 = \frac{\pi}{2}$$

$$J = \alpha|y_0|^2 + \beta u_0^2 + \alpha|y_1|^2 + \beta u_1^2 + \alpha|y_2|^2 + \cancel{\beta u_2^2} \to 0$$

Assumption :- $X_3 = X_2$

$$\begin{pmatrix} \cos u_2 & -\sin u_2 \\ \sin u_2 & \cos u_2 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$u_2 = 0$$

$$J = \alpha \cdot 1 + \beta \cdot u_0^2 + \alpha \underbrace{\left(\cos^2 u_0 + \sin^2 u_0\right)}_{1} + \beta u_1^2 + \alpha \cdot 1$$

$$= 3\alpha + \beta u_0^2 + \beta u_1^2$$

$$= 3\alpha + \beta(u_0^2 + u_1^2)$$

$$u_0 + u_1 = \frac{\pi}{2}$$

$$u_0 = \frac{\pi}{2} - u_1$$

$$u_1 = \frac{\pi}{2} - u_0$$

$$= 3\alpha + \beta u_0^2 + \beta\left(\frac{\pi}{2} - u_0\right)^2$$

$$= 3\alpha + \beta u_0^2 + \beta\left(\frac{\pi}{4}\right)^2 - \beta\pi u_0 + \beta u_0^2$$

$$\frac{dJ}{du_0} = 2\beta u_0 - \beta\pi + 2\beta u_0$$

$$= 4\beta u_0 - \beta\pi = 0$$

$$\Rightarrow \cancel{\beta}\pi = 4\cancel{\beta}u_0$$

$$\Rightarrow u_0 = \pi/4$$

$$u_1 = \pi/4$$

$$X_1 = \begin{pmatrix} \cos\frac{\pi}{4} & -\sin\frac{\pi}{4} \\ \sin\frac{\pi}{4} & \cos\frac{\pi}{4} \end{pmatrix} \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

$$y_1 = \begin{pmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{pmatrix}$$

$$X_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad X_1 = \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} \quad X_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$Y_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad Y_1 = \begin{pmatrix} 1/\sqrt{2} \\ 1/\sqrt{2} \end{pmatrix} \quad Y_2 = \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

2) For the prediction horizon of $N = 2021$, the optimum trajectory of u_k was calculated to be = $\Pi/4042$.