**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

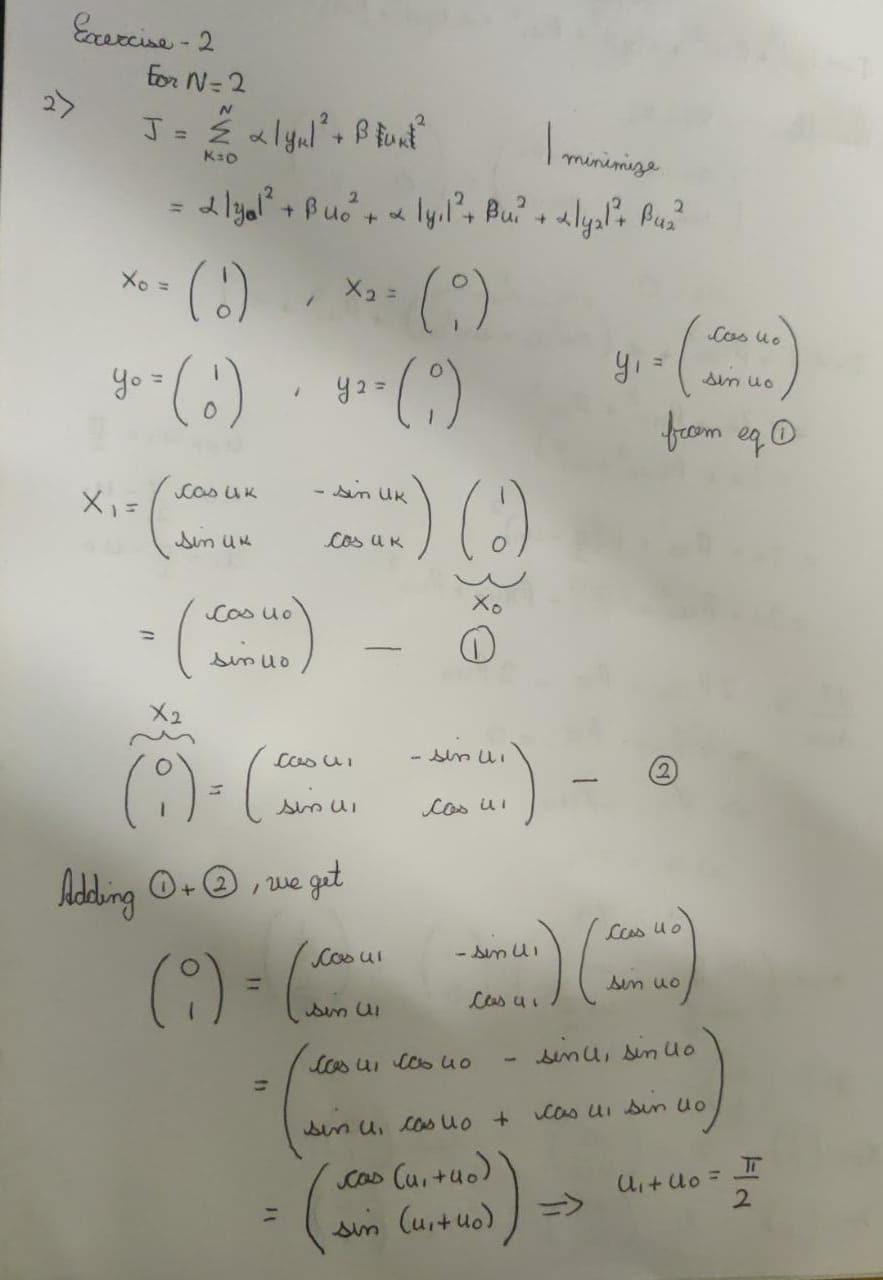
u = ones(101,2);

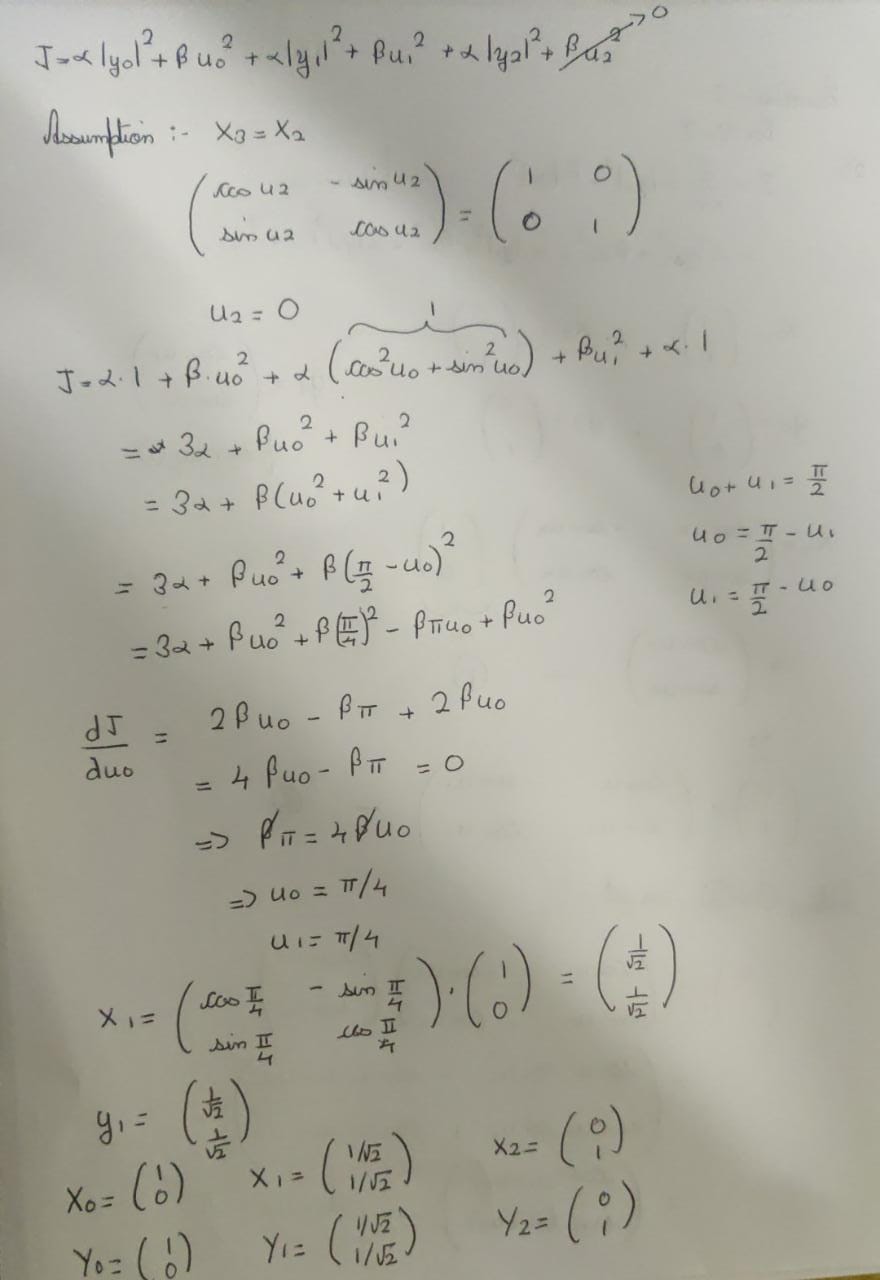
InitialGuess\_drone = struct('Time',Time,'x',x,'u',u);

save InitialGuess\_drone.mat InitialGuess\_drone

clear pn pd vn vd the thed gam gamd

# **Exercise – 2**





2) For the prediction horizon of N = 2021, the optimum trajectory of uk was calculated to be = **∏/4042**.