A Safety-Critical Model Predictive Controller for a Quadrotor with Barrier States

MinGyu Kim

December 8, 2022

1 Intro - Quadrotor Dynamics

- Rotor 1 Thrust: f1
- Rotor 2 Thrust: f2
- Rotor 3 Thrust: f2
- Rotor 4 Thrust: f2
- Rotation matrix from body to inertial coordinates: R
- Quadrotor mass: m = 0.5 kg
- Quadrotor inertia in axis (x): I_{xx} = 0.0032 kgm2
- Quadrotor inertia in axis (y): I_{yy} = 0.0032 kgm2
- Quadrotor inertia in axis (z): I_{zz} = 0.0055 kgm2
- Rotor torque constant: kt = 0.01691/m
- Rotor moment arm: l = 0.17m
- Gravity constant: g = 9.81 m/s2
- Body roll rate: p
- Body pitch rate: q
- Body yaw rate: r
- Euler angle roll: ϕ
- Euler angle pitch: θ
- Euler angle yaw: ψ

$$statex = \begin{bmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ \phi \\ \theta \\ \psi \\ p \\ q \\ r \end{bmatrix} \tag{1}$$

$$thrust\ control\ u = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \\ f_4 \end{bmatrix} \tag{2}$$

1.1 Translational Equation of Motion

$$m * \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \mathbf{R} * \begin{bmatrix} 0 \\ 0 \\ f_1 + f_2 + f_3 + f_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ -mg \end{bmatrix} \tag{3}$$

where

$$R = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} cos(\theta)cos(\psi) & cos(\theta)sin(\psi) & -sin(\theta) \\ sin(\theta)sin(\phi)cos(\psi) - cos(\phi)sin(\psi) & sin(\theta)sin(\phi)sin(\psi) + cos(\phi)cos(\psi) & sin(\phi)cos(\theta) \\ sin(\phi)sin(\psi) + cos(\phi)sin(\theta)cos(\psi) & sin(\theta)sin(\psi)cos(\phi) - sin(\phi)sin(\psi) & cos(\theta)cos(\phi) \end{bmatrix}$$

1.2 Rotational Equations of Motion

$$\begin{bmatrix} I_{xx}\dot{p} \\ I_{yy}\dot{q} \\ I_{zz}\dot{r} \end{bmatrix} = \begin{bmatrix} \frac{\sqrt{2}}{2}(f_1 + f_3 - f_2 - f_4) * l - (I_{zz} - I_{yy}) * q * r \\ \frac{\sqrt{2}}{2}(f_3 + f_4 - f_1 - f_2) * l - (I_{zz} - I_{xx}) * p * r \\ k_t(f_1 + f_4 - f_2 - f_3) \end{bmatrix} \tag{4}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & tan(\theta)sin(\phi) & tan(\theta)cos(\phi) \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & \frac{sin(\phi)}{cos(\theta)} & \frac{cos(\phi)}{cos(\theta)} \end{bmatrix} \tag{5}$$

1.3 Safety Constraints (obstacle locations)

$h_1 = (x - 2.2)^2 + (y - 2.2)^2 + (z - 1)^2 - 1$
$h_2 = (x - 0)^2 + (y + 0.2)^2 + (z)^2 - 1$
$h_3 = (x - 3)^2 + (y)^2 + (z - 0.5)^2 - 1$

2 METHODS

2.1 Differential Dynamic Programming (DDP) and Backpropagation

Algorithm 1 Differential Dynamic Programming (DDP)

procedure FNDDP
    INITIALIZE VARIABLES: initial state value, desired state value, $\bar{u}, \bar{x}$, t, dt, horizon, iteration, learning rate and WEIGHTS: Q, S, R
    find dynamics $\dot{x}$
    for 1:iteration **do**
        for 1:horizon-1 (forward propagation) **do**
            Calculate running cost L(x, u)
            Calculate the linearized matrices A and B
        Obtain the value function matrices, V_{xx}, V_x, V
        for horizon-1:1 (backpropagation) **do**
            Calculate Q matrices from the value function
            Obtain feedback matrices
        update control u
    end

2.2 Model Predictive Control (a.k.a. MPC or Receding Horizon Control)

Algorithm 2 Model Predictive Control (MPC or Receding Horizon Control)

procedure FNMPC
    INITIALIZE VARIABLES: initial state value, desired state value, $\bar{u}, \bar{u}_{mpc}\bar{x}, x$, cHorizon, pHorizon, t, dt, horizon, iteration, learning rate and WEIGHTS: Q, S, R
    for 1:pHorizon-1 **do**
        Obtain $\bar{x}, \bar{u} \leftarrow$ Run fnDDP function
        update $u_{mpc} \leftarrow \bar{u}$
        update trajectory
        update $\bar{u}$ (shift horizon)
        update $\bar{x}$ (shift horizon)
    end

2.3 Safety Constraint - Barrier State (BaS)

h(x) = constraint equation
The barrier state operator **B** is expressed as

$$\mathbf{B} = \frac{1}{h(x)} \tag{6}$$

$$\dot{B} = \frac{\delta B}{\delta h}\frac{\delta h}{\delta x}\frac{\delta x}{\delta t} = \dot{B} h_x \dot{x} = \dot{B} h_x (f(x) + g(x)u) \tag{7}$$

where $\dot{x}$ = f(x) + g(x)u (control affine system).
The safety constraint equation derived from the barrier state operator is written as
β = **B**

$$\dot{z} = \dot{\beta} h_x \dot{x} - \gamma * (z - (\beta - \beta_o)) \tag{8}$$

If this term $\dot{z}$ is added to the original system $\dot{x}$, the constraint can be managed by a controller.
When multiple constraints are considered, you may integrate all constraints into one equation like below

$$\mathbf{B} = \beta = \Sigma_{i=1}^{k} \frac{1}{h_i} \tag{9}$$

However, as the combined term has more chances of facing exploding state values, you might have to spend more time adjusting the weight matrices.

3 RESULTS

3.1 Initial Configuration

time = 8
dt = 0.01
pHorizon = t/dt
cHorizon (for MPC) = 100
learning rate = 0.5
Qsafe = [1, 1, 1]
Ssafe = [10, 10, 10]
Q = diag([0, 0, 0, 0, 0, 0, 1, 1, 1, .1, .1, .1, Qsafe])
S = diag([10, 10, 10, 1, 1, 1, 1, 1, 1, 1, 1, 1, Ssafe])
R = diag([1, 1, 1, 1].*0.01)

3.2 DDP application

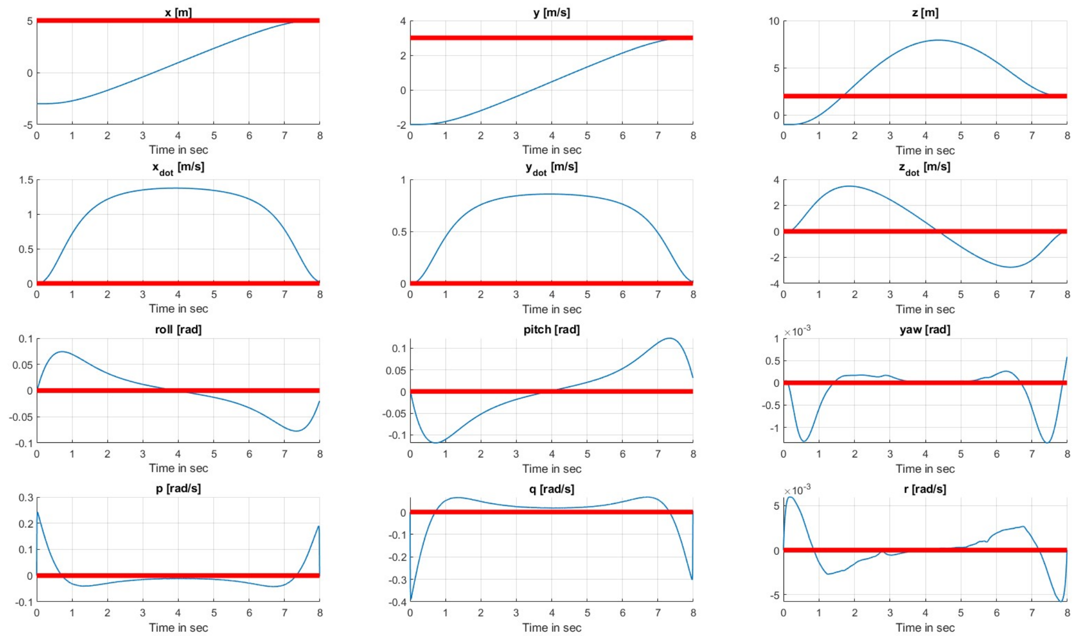

Figure 1: Optimal State Paths Determined Through DDP. The blue lines are trajectories, and the red lines are desired states.

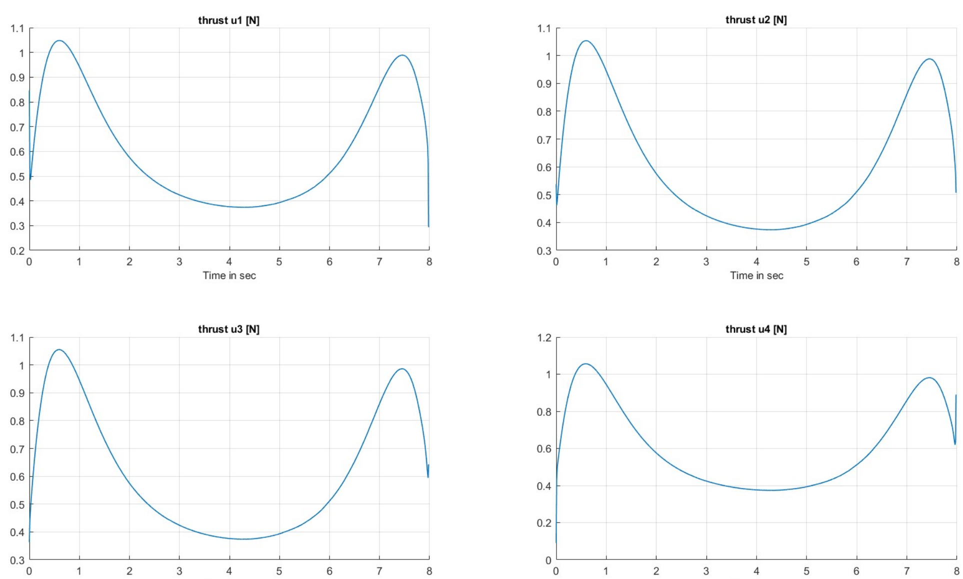

Figure 2: Optimal Control Determined Through DDP.

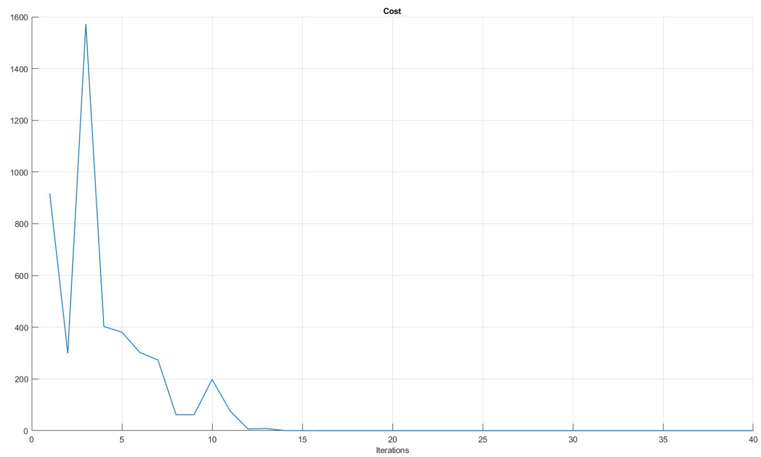

Figure 3: Corresponding Cost Through DDP

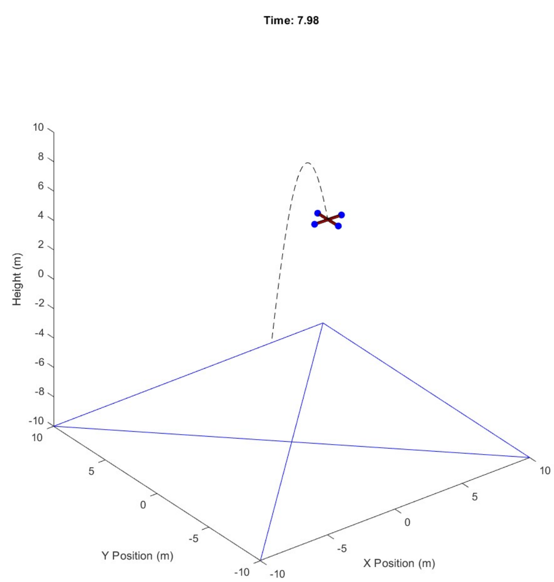

Figure 4: Trajectory Simulated by DDP

3.3 MPC application

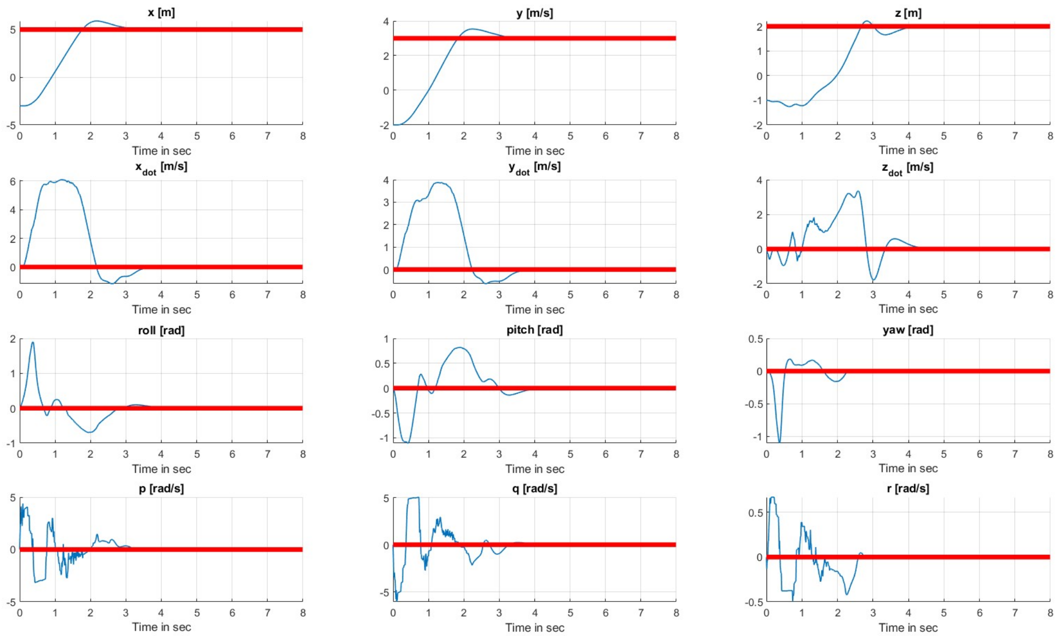

Figure 5: Optimal State Paths Determined Through MPC. The blue lines are trajectories, and the red lines are desired states.

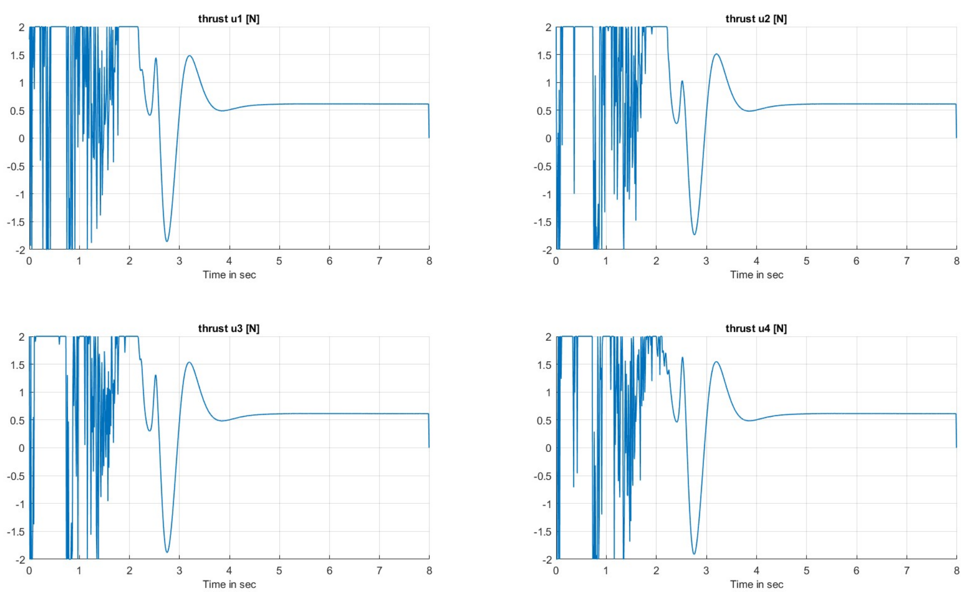

Figure 6: Optimal Control Determined Through MPC

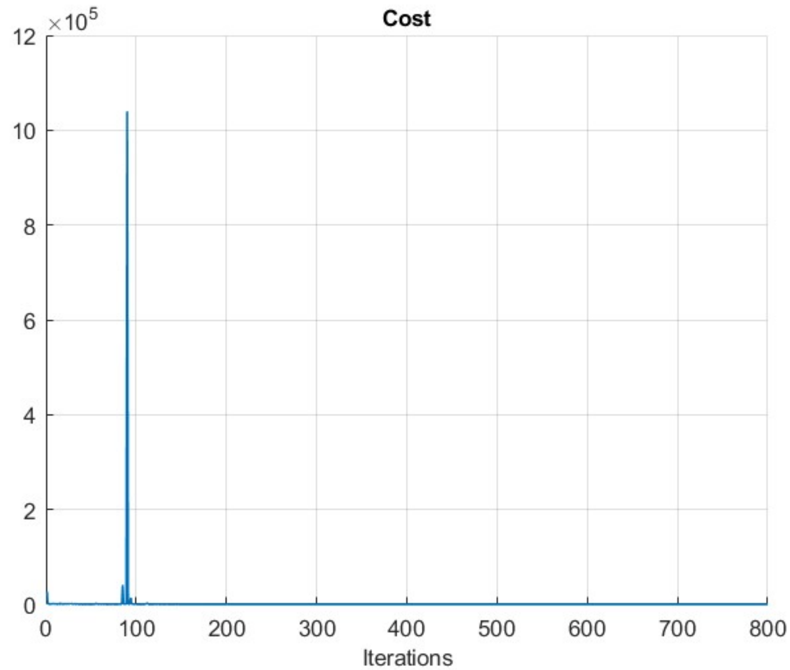

Figure 7: Corresponding Cost Through MPC

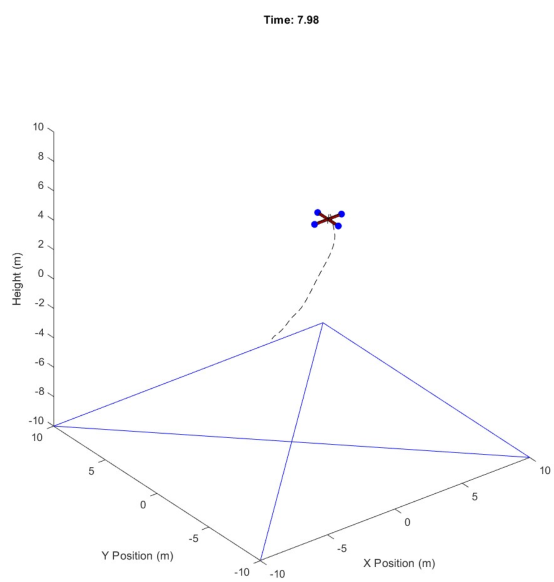

Figure 8: Trajectory Simulated by MPC

3.4 obstacle avoidance using MPC with BaS

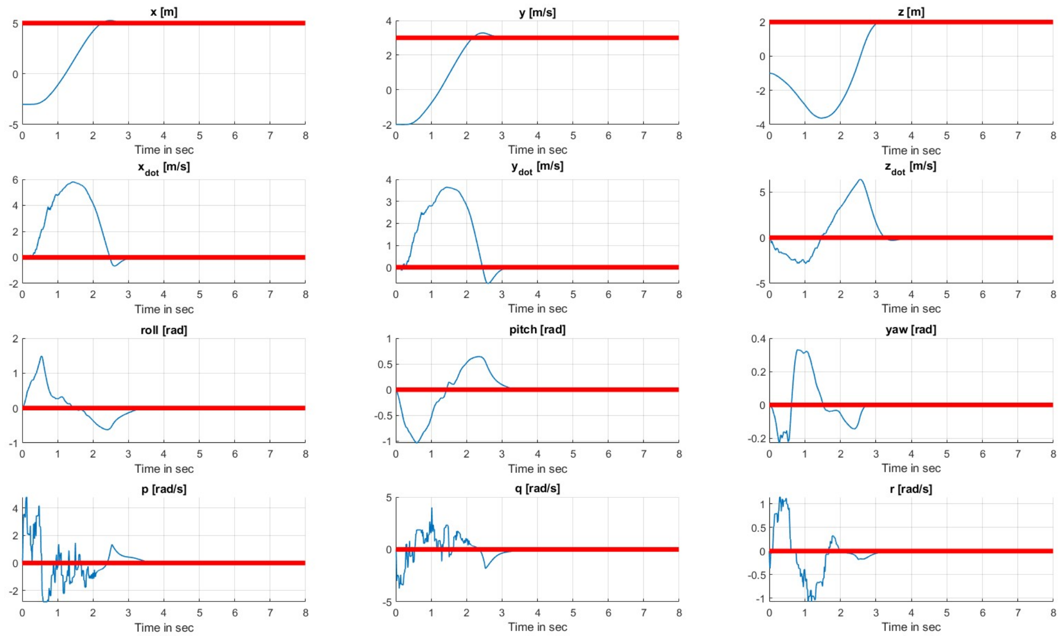

Figure 9: Optimal State Paths Determined Through MPC-BaS. The blue lines are trajectories, and the red lines are desired states.

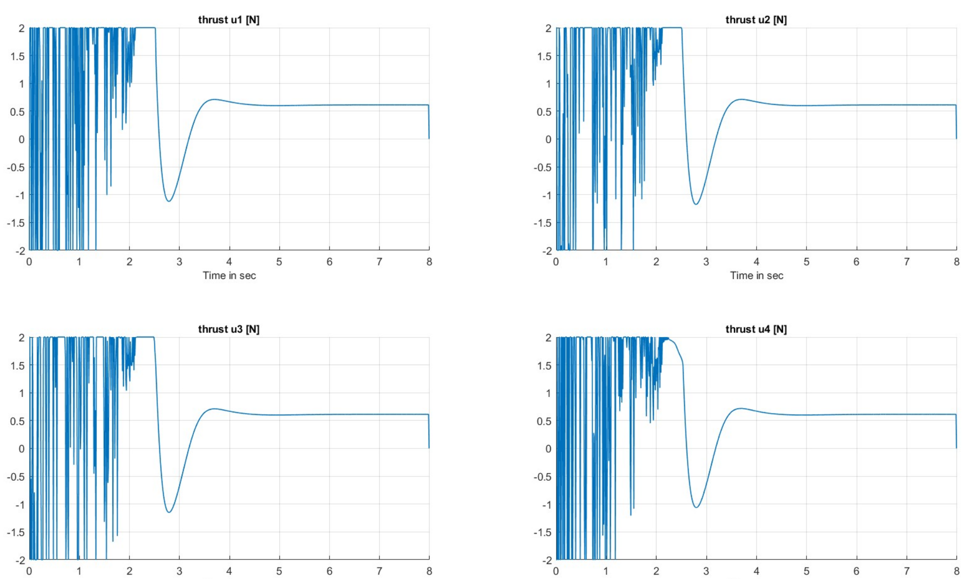

Figure 10: Optimal Control Determined Through MPC-BaS

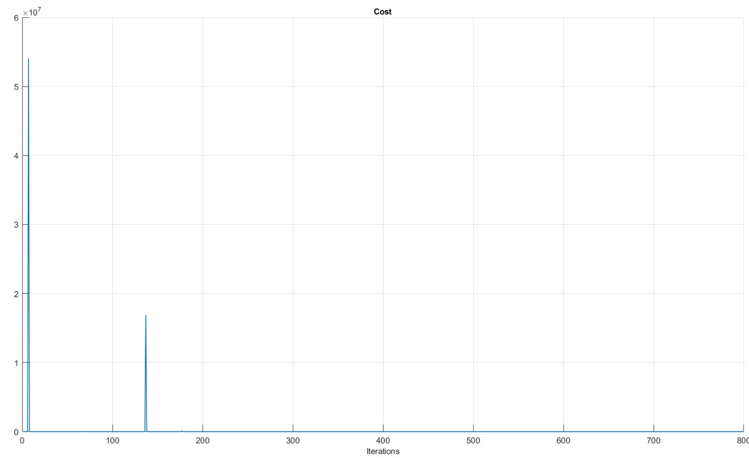

Figure 11: Corresponding Cost Through MPC-BaS

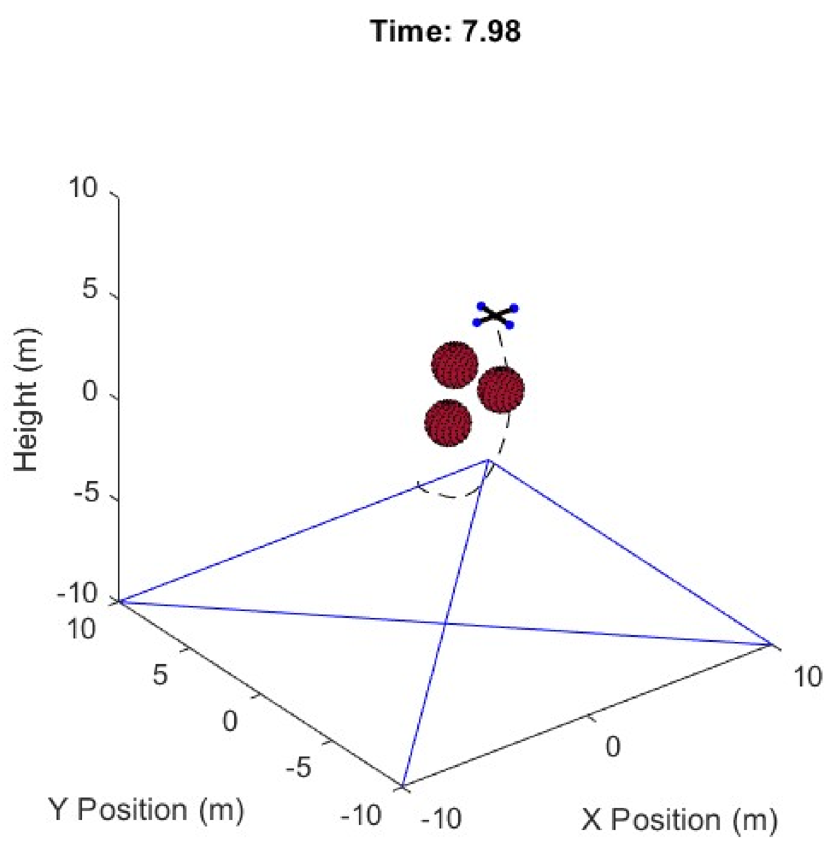

Figure 12: Trajectory Simulated by MPC-BaS

4 ROBOTARIUM SIMULATION

Following the source ($https : //www.robotarium.gatech.edu/get_started$), I set up the virtual environment and tested my DDP code on MATLAB prior to testing the code in the real environment.

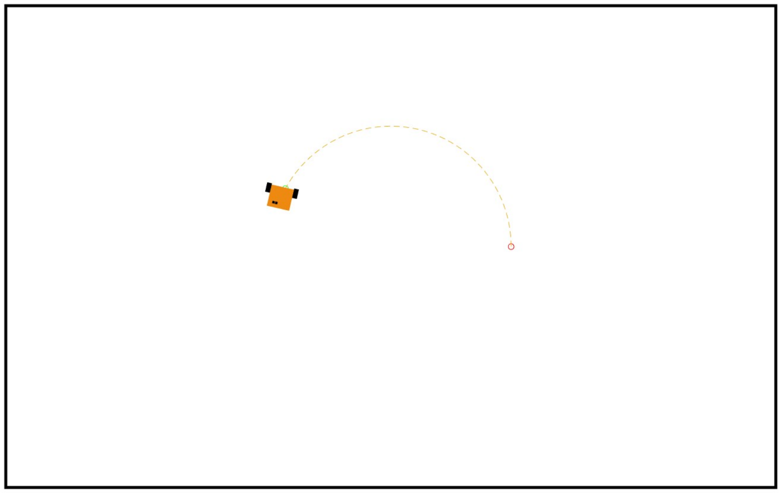

Figure 13: The virtual ROBOTARIUM simulation implemented by DDP

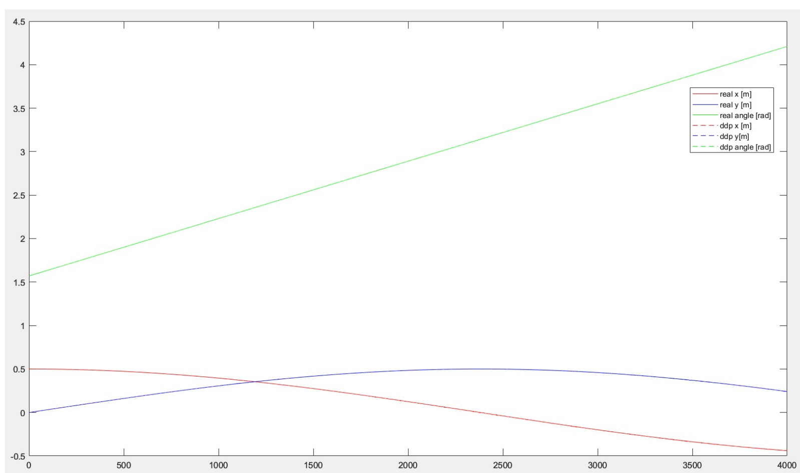

Figure 14: The state trajectories simulated by DDP

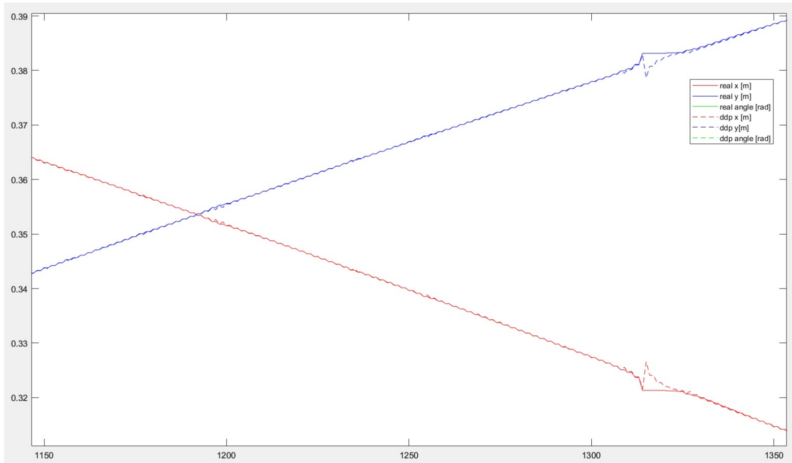

Figure 15: Differences between the real trajectory and simulated trajectory

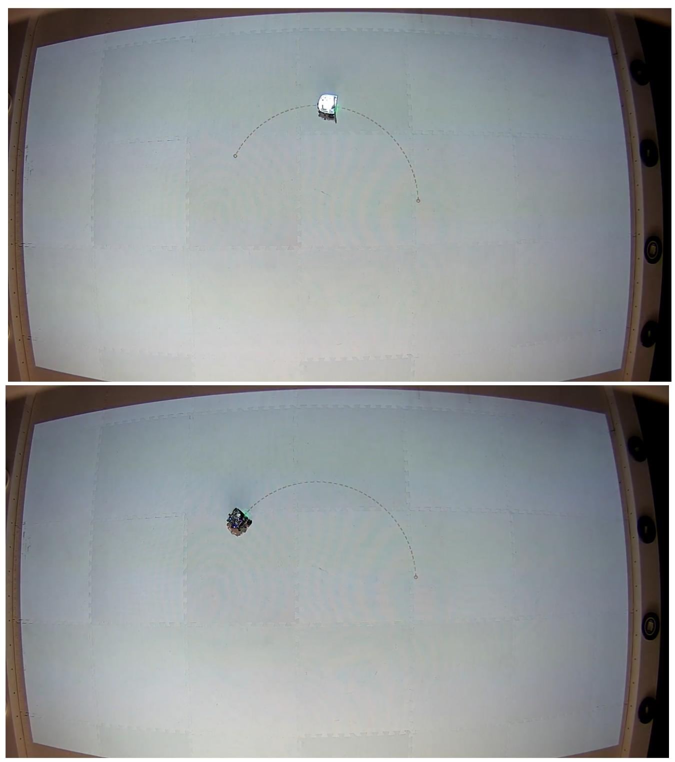

Figure 16: The real ROBOTARIUM simulation.

5 DISCUSSION

5.1 vanishing gradient Q_{uu} in backpropagation

While implementing the DDP function, one gradient of cost matrix Q_{uu} was vanishing, which caused zero feedback to the system. In other words, the quadrotor received a feedback matrix of zeros at a specific time step. To solve this issue, I tried to force the Q_{uu} to have as minimum values as it can contain so that it can keep sending small feedback to the system.

Algorithm 3 Force Backpropagation

1: **procedure** WITHIN BACKPROPAGATION OF FNDDP
2: $\lambda \leftarrow ||min(eig(Q_{uu}))|| + rand$
3: **if** $any(eig(Q_{uu})) < 1e - 10$ **then**
4: Q_{uu} += λI where I is an identity matrix
5: end

5.2 restricted controls

The first simulation was implemented without clamp of the control, however, the quadrotor initially flied too much fast and spent much time going back to the convergence. This was shown as a very unstable control. Therefore, to stabilize the motion of the drone, I gave the system a control restriction, for example, maximum control value is 4 N and minimum control value is -2.

6 Appendix: MATLAB CODE

```
clc; clear; close all;
%% INITIAL CONFIGURATION
% global m; % drone mass
% global Ixx; % MOI in x axis
% global Iyy; % MOI in y axis
% global Izz; % MOI in z axis
% global kt; % rotor torque constant
% global l; % rotor moment arm length
% global g; % gravity constant

% global p; % body roll rate
% global q; % body pitch rate
% global r; % body yaw rate
% global phi; %roll
% global theta; %pitch
% global psi; %yaw

m = 0.5;
Ixx = 0.0032;
Iyy = 0.0032;
Izz = 0.0055;
kt = 0.01691;
l = 0.17;
g = 9.81;

%% MODEL DEFINITION
% rotation matrix R123 = R1*R2*R3 = R.transpose;
syms x y z xdot ydot zdot roll pitch yaw p q r u1 u2 u3 u4
%R = R1(roll)*R2(pitch)*R3(yaw);
R = [cos(pitch)*cos(yaw), cos(pitch)*sin(yaw), -sin(pitch);
     sin(pitch)*sin(yaw) - cos(roll)*sin(yaw),  sin(pitch)*sin(roll)*
        sin(yaw) + cos(roll)*cos(yaw), sin(roll)*cos(pitch);
     sin(roll)*sin(yaw) + cos(roll)*sin(pitch)*cos(yaw), sin(pitch)*
        sin(yaw)*cos(roll) - sin(roll)*sin(yaw), cos(pitch)*cos(roll)
        ];

% translational motion
% f1 = [xddot; yddot; zddot]
f1 = 1/m.*R*[0; 0; u1+u2+u3+u4] + [0; 0; -m*g];
% f2 = [rolldot, pitchdot, yawdot] http://www.stengel.mycpanel.
    princeton.edu/Quaternions.pdf
R_f2 = [1, tan(pitch)*sin(roll), tan(pitch)*cos(roll);
     0, cos(roll), -sin(roll);
     0, sin(roll)/cos(pitch), cos(roll)/cos(pitch)];
f2 = R_f2*[p; q; r];
```

```
% rotational motion
% f3 = [pdot; qdot; rdot]
MOI = [Ixx; Iyy; Izz];
f3 = 1./MOI.*[sqrt(2)/2*(u1 + u3 - u2 - u4)*l - (Izz - Iyy)*q*r;
     sqrt(2)/2*(u3 + u4 - u1 - u2)*l + (Izz - Ixx)*p*r;
     kt*(u1 + u4 - u2 - u3)];

dyn = [xdot; ydot; zdot; f1; f2; f3];
x = [x; y; z; xdot; ydot; zdot; roll; pitch; yaw; p; q; r];
u = [u1; u2; u3; u4];
A = jacobian(dyn, x);
B = jacobian(dyn, u);
%% safety constraints
syms x y z xdot ydot zdot roll pitch yaw p q r z1 z2 z3

xc1 = 2.2;
yc1 = 2.2;
zc1 = 1;
rc1 = 1;

xc2 = 0;
yc2 = -0.2;
zc2 = 0;
rc2 = 1;

xc3 = 3;
yc3 = 0;
zc3 = 0.5;
rc3 = 1;

xvec = [x; y; z; xdot; ydot; zdot; roll; pitch; yaw; p; q; r];
h1 = (x - xc1)^2 + (y - yc1)^2 + (z - zc1)^2 - rc1;
hx1 = jacobian(h1, xvec);
b1 = 1/h1;

h2 = (x - xc2)^2 + (y - yc2)^2 + (z - zc2)^2 - rc2;
hx2 = jacobian(h2, xvec);
b2 = 1/h2;

h3 = (x - xc3)^2 + (y - yc3)^2 + (z - zc3)^2 - rc3;
hx3 = jacobian(h3, xvec);
b3 = 1/h3;
%% state shift
x0 = [-3; -2; -1; 0; 0; 0; 0; 0; 0; 0; 0; 0];
xd = [5; 3; 2; 0; 0; 0; 0; 0; 0; 0; 0; 0];

bnut1 = double(subs(b1, {x, y, z}, {x0(1), x0(2), x0(3)}));
```

```
bnut2 = double(subs(b2, {x, y, z}, {x0(1), x0(2), x0(3)}));
bnut3 = double(subs(b3, {x, y, z}, {x0(1), x0(2), x0(3)}));
zz1 = b1 - bnut1;
zz2 = b2 - bnut2;
zz3 = b3 - bnut3;
%% safety constraint states z
gamma = 1;
zdot1 = diff(b1)*hx1*dyn - gamma*(z1 - zz1);
zdot2 = diff(b2)*hx2*dyn - gamma*(z2 - zz2);
zdot3 = diff(b3)*hx3*dyn - gamma*(z3 - zz3);

%% safety constraints states' initial/desired values
xsafe = [xvec; z1; z2; z3];

z10 = double(subs(zz1, {x, y, z}, {x0(1), x0(2), x0(3)}));
z1d = double(subs(zz1, {x, y, z}, {xd(1), xd(2), xd(3)}));

z20 = double(subs(zz2, {x, y, z}, {x0(1), x0(2), x0(3)}));
z2d = double(subs(zz2, {x, y, z}, {xd(1), xd(2), xd(3)}));

z30 = double(subs(zz3, {x, y, z}, {x0(1), x0(2), x0(3)}));
z3d = double(subs(zz3, {x, y, z}, {xd(1), xd(2), xd(3)}));
%% the whole system's initial / desired values
x0safe = [x0; z10; z20; z30]; % z0 = subs(z, x1, 5);
xdsafe = [xd; z1d; z2d; z3d]; % z_des = subs(z, x1, 0);

%% system update
dyn_obs = [dyn; zdot1; zdot2; zdot3];
fxSafe = jacobian(dyn_obs, xsafe);
fuSafe = jacobian(dyn_obs, u);
%% DDP-MPC
%% initial configuration
t = 8;
dt = 0.01;
pHorizon = t/dt;
cHorizon = 100;
iter = 5;
lr = 0.5;
Qsafe = [1, 1, 1];
Ssafe = [10, 10, 10];
Q = diag([0, 0, 0, 0, 0, 0, 1, 1, 1, .1, .1, .1, Qsafe]);
S = diag([10, 10, 10, 1, 1, 1, 1, 1, 1, 1, 1, 1, Ssafe].*1);
R = diag([1, 1, 1, 1].*0.01);
ddp_message = false;
safe_mode = true;
ddp_cost = false;
%% nominal state update
```

```
ubar = zeros(4, cHorizon - 1);
ubar(:, 1) = [1, 1, 1, 1].*5;
x_des = zeros(12 + 3, cHorizon);
x_des(:, 1) = x0safe;
x_des(:, end) = xdsafe;
xbar = zeros(12 + 3, cHorizon);
xbar(:, 1) = x0safe;
for i = 1:cHorizon-1
    [x_, ~, ~] = fnStateDroneSafe(xbar(:, i), ubar(:, i));
    xbar(:, i+1) = xbar(:, i) + x_.*dt;
end
umpc = zeros(4, pHorizon);
traj = xbar;
cost_ = zeros(1, pHorizon);
%% simulation
for h = 1:pHorizon - 1
    % find optimal control u at horizon h or step t
    [xbar, u, cost] = fnDDPdrone(xbar, ubar, x_des, cHorizon, iter, lr, Q, S, R, ddp_message, safe_mode, ddp_cost); %(horizon + 1 - h)*dt,

    % save the first element of the control at step t where we generate a
    % new trajectory. ==>> will be the optimal control of this MPC model
    umpc(:, h) = u(:, 1);
    % find the next state by calculating it with the initial control input
    % and current state
    [x_, ~, ~] = fnStateDroneSafe(traj(:, h), u(:, 1));
    x_ = x_.*dt;

    % update the next state
    traj(:, h + 1) = x_ + traj(:, h);

    % update & shift nominal state/control by 1
    % this's aka receding horizon
    % At the next state (first elements of the updated state/control)
    % , the model will re-optimize control
    ubar(:, 1:end-1) = u(:, 2:end);
    xbar(:,1) = traj(:, h + 1);

    % cost display
    cost_(h) = cost;
    fprintf("%d DDP-MPC cost: %.4f with lr = %.4f \n", h, cost_(h), lr);
end
```

```
%% MPC trajectory
mpc_traj = traj;
figure(101)
plot3(mpc_traj(1, :), mpc_traj(2, :), mpc_traj(3, :), '--')
hold on
plot3(x0(1), x0(2), x0(3), 'rx')
plot3(xd(1), xd(2), xd(3), 'bx')
hold off
%% MPC animation
xd = x_des(1:3, end);
obs_info = [xc1, yc1, zc1, rc1; xc2, yc2, zc2, rc2; xc3, yc3, zc3,
    rc3];
[h,MMmpc] = quadrotor_visualize_w_obstacles(mpc_traj, u, 0:dt:t-dt,
    obs_info, xd);
%%
figure(2)
movie(MMmpc,1,length(mpc_traj(1,:))/t/2)

%% PERFORMANCE
time = 0:dt:t-dt;
horizon = pHorizon;

figure(11);
subplot(4,3,1)
hold on
plot(time,mpc_traj(1,:),'linewidth',1);
plot(time,x_des(1,end)*ones(1,horizon),'red','linewidth',4)
title('x [m]','fontsize',10);
xlabel('Time in sec','fontsize',10)
%hold off;
grid on

subplot(4,3,2)
hold on;
plot(time,mpc_traj(2,:),'linewidth',1);
plot(time,x_des(2,end)*ones(1,horizon),'red','linewidth',4)
title('y [m/s]','fontsize',10);
xlabel('Time in sec','fontsize',10)
grid on
%hold off;

subplot(4,3,3)
hold on
plot(time,mpc_traj(3,:),'linewidth',1);
plot(time,x_des(3,end)*ones(1,horizon),'red','linewidth',4)
title('z [m]','fontsize',10)
xlabel('Time in sec','fontsize',10)
```

```
grid on
%hold off;$\theta$

subplot(4,3,4)
hold on
plot(time,mpc_traj(4,:),'linewidth',1);
plot(time,x_des(4,end)*ones(1,horizon),'red','linewidth',4)
title('x_{dot} [m/s]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(4,3,5)
hold on
plot(time,mpc_traj(5,:),'linewidth',1);
plot(time,x_des(5,end)*ones(1,horizon),'red','linewidth',4)
title('y_{dot} [m/s]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on
%hold off

subplot(4,3,6)
hold on
plot(time,mpc_traj(6,:),'linewidth',1);
plot(time,x_des(6,end)*ones(1,horizon),'red','linewidth',4)
title('z_{dot} [m/s]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(4,3,7)
hold on
plot(time,mpc_traj(7,:),'linewidth',1);
plot(time,x_des(7,end)*ones(1,horizon),'red','linewidth',4)
title('roll [rad]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(4,3,8)
hold on
plot(time,mpc_traj(8,:),'linewidth',1);
plot(time,x_des(8,end)*ones(1,horizon),'red','linewidth',4)
title('pitch [rad]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(4,3,9)
hold on
plot(time,mpc_traj(9,:),'linewidth',1);
```

```
plot(time,x_des(9,end)*ones(1,horizon),'red','linewidth',4)
title('yaw [rad]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(4,3,10)
hold on
plot(time,mpc_traj(10,:),'linewidth',1);
plot(time,x_des(10,end)*ones(1,horizon),'red','linewidth',4)
title('p [rad/s]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(4,3,11)
hold on
plot(time,mpc_traj(11,:),'linewidth',1);
plot(time,x_des(11,end)*ones(1,horizon),'red','linewidth',4)
title('q [rad/s]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(4,3,12)
hold on
plot(time,mpc_traj(12,:),'linewidth',1);
plot(time,x_des(12,end)*ones(1,horizon),'red','linewidth',4)
title('r [rad/s]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on
%%
figure(13)
subplot(2, 2, 1)
hold on
plot(time,umpc(1,:),'linewidth',1);
title('thrust u1 [N]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(2, 2, 2)
hold on
plot(time,umpc(2,:),'linewidth',1);
title('thrust u2 [N]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(2, 2, 3)
hold on
plot(time,umpc(3,:),'linewidth',1);
```

```
title('thrust u3 [N]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(2, 2, 4)
hold on
plot(time,umpc(4,:),'linewidth',1);
title('thrust u4 [N]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

%%
figure(14)
hold on
plot(cost_,'linewidth',1);
xlabel('Iterations','fontsize',10)
title('Cost','fontsize',10);
grid on
%save('DDP_Data');



%% functions
function r = R1(rad)
    r = [1 0 0; 0 cos(rad) sin(rad); 0 -sin(rad) cos(rad)];
end
function r = R2(rad)
    r = [cos(rad) 0 -sin(rad); 0 1 0; sin(rad) 0 cos(rad)];
end
function r = R3(rad)
    r = [cos(rad) sin(rad) 0; -sin(rad) cos(rad) 0; 0 0 1];
end
```

```
clc; clear; close all;
%% INITIAL CONFIGURATION
global m; % drone mass
global Ixx; % MOI in x axis
global Iyy; % MOI in y axis
global Izz; % MOI in z axis
global kt; % rotor torque constant
global l; % rotor moment arm length
global g; % gravity constant

% global p; % body roll rate
% global q; % body pitch rate
% global r; % body yaw rate
% global phi; %roll
```

```
% global theta; %pitch
% global psi; %yaw

m = 0.5;
Ixx = 0.0032;
Iyy = 0.0032;
Izz = 0.0055;
kt = 0.01691;
l = 0.17;
g = 9.81;

%% MODEL DEFINITION
% rotation matrix R123 = R1*R2*R3 = R.transpose;
syms x y z xdot ydot zdot roll pitch yaw p q r u1 u2 u3 u4
R = R1(roll)*R2(pitch)*R3(yaw);
R = [cos(pitch)*cos(yaw), cos(pitch)*sin(yaw), -sin(pitch);
     sin(pitch)*sin(yaw) - cos(roll)*sin(yaw),  sin(pitch)*sin(roll)*
        sin(yaw) + cos(roll)*cos(yaw), sin(roll)*cos(pitch);
     sin(roll)*sin(yaw) + cos(roll)*sin(pitch)*cos(yaw), sin(pitch)*
        sin(yaw)*cos(roll) - sin(roll)*sin(yaw), cos(pitch)*cos(roll)
        ];

% translational motion
% f1 = [xddot; yddot; zddot]
f1 = 1/m.*R*[0; 0; u1+u2+u3+u4] + [0; 0; -m*g];
% f2 = [rolldot, pitchdot, yawdot] http://www.stengel.mycpanel.
    princeton.edu/Quaternions.pdf
R_f2 = [1, tan(pitch)*sin(roll), tan(pitch)*cos(roll);
     0, cos(roll), -sin(roll);
     0, sin(roll)/cos(pitch), cos(roll)/cos(pitch)];
f2 = R_f2*[p; q; r];
% rotational motion
% f3 = [pdot; qdot; rdot]
MOI = [Ixx; Iyy; Izz];
f3 = 1./MOI.*[sqrt(2)/2*(u1 + u3 - u2 - u4)*l - (Izz - Iyy)*q*r;
     sqrt(2)/2*(u3 + u4 - u1 - u2)*l + (Izz - Ixx)*p*r;
     kt*(u1 + u4 - u2 - u3)];

dyn = [xdot; ydot; zdot; f1; f2; f3];
x = [x; y; z; xdot; ydot; zdot; roll; pitch; yaw; p; q; r];
u = [u1; u2; u3; u4];
A = jacobian(dyn, x)
B = jacobian(dyn, u)

%% DDP-MPC
t = 8;
dt = 0.01;
```

```
pHorizon = t/dt;
cHorizon = 100;
iter = 5;
lr = 0.5;
Q = diag([0, 0, 0, 0, 0, 0, 1, 1, 1, .1, .1, .1]);
S = diag([10, 10, 10, 1, 1, 1, 1, 1, 1, 1, 1, 1].*1);
R = diag([1, 1, 1, 1].*0.01);

ubar = zeros(4, cHorizon - 1);
ubar(:, 1) = [1, 1, 1, 1].*5;
x0 = [-3; -2; -1; 0; 0; 0; 0; 0; 0; 0; 0; 0];
xd = [5; 3; 2; 0; 0; 0; 0; 0; 0; 0; 0; 0];

x_des = zeros(12, cHorizon);
x_des(:, 1) = x0;
x_des(:, end) = xd;
xbar = zeros(12, cHorizon);
xbar(:, 1) = x0;
for i = 1:cHorizon-1
    [x_, ~, ~] = fnStateDrone(xbar(:, i), ubar(:, i));
    xbar(:, i+1) = xbar(:, i) + x_.*dt;
end

umpc = zeros(4, pHorizon);
traj = xbar;
cost_ = zeros(1, pHorizon);
ddp_message = false;
safe_mode = false;
ddp_cost = false;
for h = 1:pHorizon - 1
    % find optimal control u at horizon h or step t
    [xbar, u, cost] = fnDDPdrone(xbar, ubar, x_des, cHorizon, iter, lr, Q, S, R, ddp_message, safe_mode, ddp_cost); %(horizon + 1 - h)*dt,

    % save the first element of the control at step t where we generate a
    % new trajectory. ==>> will be the optimal control of this MPC model
    umpc(:, h) = u(:, 1);
    % find the next state by calculating it with the initial control input
    % and current state
    [x_, ~, ~] = fnStateDrone(traj(:, h), u(:, 1));
    x_ = x_.*dt;

    % update the next state
```

```
    traj(:, h + 1) = x_ + traj(:, h);

    % update & shift nominal state/control by 1
    % this's aka receding horizon
    % At the next state (first elements of the updated state/control)
    % , the model will re-optimize control
    ubar(:, 1:end-1) = u(:, 2:end);
%     ubar(:, end) = 0;
%     xbar(:, 1:end-1) = xbar(:, 2:end);
    xbar(:,1) = traj(:, h + 1);

    % cost display
    cost_(h) = cost;
    fprintf("%d DDP-MPC cost: %.4f with lr = %.4f \n", h, cost_(h),
       lr);
end
%% MPC trajectory
mpc_traj = traj;
figure(101)
plot3(mpc_traj(1, :), mpc_traj(2, :), mpc_traj(3, :), '--')
hold on
plot3(x0(1), x0(2), x0(3), 'rx')
plot3(xd(1), xd(2), xd(3), 'bx')
hold off
%% MPC animation
xd = x_des(1:3, end);
[h,MMmpc] = quadrotor_visualize(mpc_traj,u, 0:dt:t-dt, xd)
%%
figure(2)
movie(MMmpc,1,length(mpc_traj(1,:))/t/2)

%% PERFORMANCE
time = 0:dt:t-dt;
horizon = pHorizon;

figure(11);
subplot(4,3,1)
hold on
plot(time,mpc_traj(1,:),'linewidth',1);
plot(time,x_des(1,end)*ones(1,horizon),'red','linewidth',4)
title('x [m]','fontsize',10);
xlabel('Time in sec','fontsize',10)
%hold off;
grid on

subplot(4,3,2)
hold on;
```

```
plot(time,mpc_traj(2,:),'linewidth',1);
plot(time,x_des(2,end)*ones(1,horizon),'red','linewidth',4)
title('y [m/s]','fontsize',10);
xlabel('Time in sec','fontsize',10)
grid on
%hold off;

subplot(4,3,3)
hold on
plot(time,mpc_traj(3,:),'linewidth',1);
plot(time,x_des(3,end)*ones(1,horizon),'red','linewidth',4)
title('z [m]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on
%hold off;$\theta$

subplot(4,3,4)
hold on
plot(time,mpc_traj(4,:),'linewidth',1);
plot(time,x_des(4,end)*ones(1,horizon),'red','linewidth',4)
title('x_{dot} [m/s]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(4,3,5)
hold on
plot(time,mpc_traj(5,:),'linewidth',1);
plot(time,x_des(5,end)*ones(1,horizon),'red','linewidth',4)
title('y_{dot} [m/s]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on
%hold off

subplot(4,3,6)
hold on
plot(time,mpc_traj(6,:),'linewidth',1);
plot(time,x_des(6,end)*ones(1,horizon),'red','linewidth',4)
title('z_{dot} [m/s]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(4,3,7)
hold on
plot(time,mpc_traj(7,:),'linewidth',1);
plot(time,x_des(7,end)*ones(1,horizon),'red','linewidth',4)
title('roll [rad]','fontsize',10)
xlabel('Time in sec','fontsize',10)
```

```
grid on

subplot(4,3,8)
hold on
plot(time,mpc_traj(8,:),'linewidth',1);
plot(time,x_des(8,end)*ones(1,horizon),'red','linewidth',4)
title('pitch [rad]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(4,3,9)
hold on
plot(time,mpc_traj(9,:),'linewidth',1);
plot(time,x_des(9,end)*ones(1,horizon),'red','linewidth',4)
title('yaw [rad]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(4,3,10)
hold on
plot(time,mpc_traj(10,:),'linewidth',1);
plot(time,x_des(10,end)*ones(1,horizon),'red','linewidth',4)
title('p [rad/s]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(4,3,11)
hold on
plot(time,mpc_traj(11,:),'linewidth',1);
plot(time,x_des(11,end)*ones(1,horizon),'red','linewidth',4)
title('q [rad/s]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(4,3,12)
hold on
plot(time,mpc_traj(12,:),'linewidth',1);
plot(time,x_des(12,end)*ones(1,horizon),'red','linewidth',4)
title('r [rad/s]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on
%%
figure(13)
subplot(2, 2, 1)
hold on
plot(time,umpc(1,:),'linewidth',1);
title('thrust u1 [N]','fontsize',10)
```

```
xlabel('Time in sec','fontsize',10)
grid on

subplot(2, 2, 2)
hold on
plot(time,umpc(2,:),'linewidth',1);
title('thrust u2 [N]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(2, 2, 3)
hold on
plot(time,umpc(3,:),'linewidth',1);
title('thrust u3 [N]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(2, 2, 4)
hold on
plot(time,umpc(4,:),'linewidth',1);
title('thrust u4 [N]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

%%
figure(14)
hold on
plot(cost_,'linewidth',1);
xlabel('Iterations','fontsize',10)
title('Cost','fontsize',10);
grid on
%save('DDP_Data');
%% functions
function r = R1(rad)
    r = [1 0 0; 0 cos(rad) sin(rad); 0 -sin(rad) cos(rad)];
end
function r = R2(rad)
    r = [cos(rad) 0 -sin(rad); 0 1 0; sin(rad) 0 cos(rad)];
end
function r = R3(rad)
    r = [cos(rad) sin(rad) 0; -sin(rad) cos(rad) 0; 0 0 1];
end
```

```
clc; clear; close all;
%% INITIAL CONFIGURATION
global m; % drone mass
global Ixx; % MOI in x axis
```

```
global Iyy; % MOI in y axis
global Izz; % MOI in z axis
global kt; % rotor torque constant
global l; % rotor moment arm length
global g; % gravity constant

% global p; % body roll rate
% global q; % body pitch rate
% global r; % body yaw rate
% global phi; %roll
% global theta; %pitch
% global psi; %yaw

m = 0.5;
Ixx = 0.0032;
Iyy = 0.0032;
Izz = 0.0055;
kt = 0.01691;
l = 0.17;
g = 9.81;

%% MODEL DEFINITION
% rotation matrix R123 = R1*R2*R3 = R.transpose;
syms x y z xdot ydot zdot roll pitch yaw p q r u1 u2 u3 u4
R = R1(roll)*R2(pitch)*R3(yaw);
R = [cos(pitch)*cos(yaw), cos(pitch)*sin(yaw), -sin(pitch);
    sin(pitch)*sin(yaw) - cos(roll)*sin(yaw),  sin(pitch)*sin(roll)*sin(yaw) + cos(roll)*cos(yaw), sin(roll)*cos(pitch);
    sin(roll)*sin(yaw) + cos(roll)*sin(pitch)*cos(yaw), sin(pitch)*sin(yaw)*cos(roll) - sin(roll)*sin(yaw), cos(pitch)*cos(roll)];

% translational motion
% f1 = [xddot; yddot; zddot]
f1 = 1/m.*R*[0; 0; u1+u2+u3+u4] + [0; 0; -m*g];
% f2 = [rolldot, pitchdot, yawdot] http://www.stengel.mycpanel.princeton.edu/Quaternions.pdf
R_f2 = [1, tan(pitch)*sin(roll), tan(pitch)*cos(roll);
    0, cos(roll), -sin(roll);
    0, sin(roll)/cos(pitch), cos(roll)/cos(pitch)];
f2 = R_f2*[p; q; r];
% rotational motion
% f3 = [pdot; qdot; rdot]
MOI = [Ixx; Iyy; Izz];
f3 = 1./MOI.*[sqrt(2)/2*(u1 + u3 - u2 - u4)*l - (Izz - Iyy)*q*r;
    sqrt(2)/2*(u3 + u4 - u1 - u2)*l + (Izz - Ixx)*p*r;
    kt*(u1 + u4 - u2 - u3)];
```

```
dyn = [xdot; ydot; zdot; f1; f2; f3];
x = [x; y; z; xdot; ydot; zdot; roll; pitch; yaw; p; q; r];
u = [u1; u2; u3; u4];
A = jacobian(dyn, x)
B = jacobian(dyn, u)

%% DDP
% Q = diag([0, 0, 0, 0, 0, 0, 1, 1, 1, .1, .1, .1]);
% S = diag([10, 10, 10, 1, 1, 1, 1, 1, 1, 1, 1, 1].*1);
% %S = diag([1, 1, 1, 10, 10, 10, 1, 1, 1, 10, 10, 10].*1);
% R = diag([1, 1, 1, 1].*0.01);
% %lr = 0.3:-0.01:0.2;
% lr = 0.6
Q = diag([0, 0, 0, 0, 0, 0, 1, 1, 1, .1, .1, .1]);
S = diag([10, 10, 10, 1, 1, 1, 1, 1, 1, 1, 1, 1].*1);
R = diag([1, 1, 1, 1].*0.01);
lr = 0.5;
t = 8;
dt = 0.01;
cHorizon = t/dt;
iter = 40;

ubar = zeros(4, cHorizon - 1);
ubar(:, 1) = [1, 1, 1, 1].*5;
x0 = [-3; -2; -1; 0; 0; 0; 0; 0; 0; 0; 0; 0];
xbar = zeros(12, cHorizon);
xbar(:, 1) = x0;
for i = 1:cHorizon-1
    [x_, ~, ~] = fnStateDrone(xbar(:, i), ubar(:, i));
    xbar(:, i+1) = xbar(:, i) + x_.*dt;
end
x_des = zeros(12, cHorizon);
x_des(:, 1) = x0;
xd = [5; 3; 2; 0; 0; 0; 0; 0; 0; 0; 0; 0];
x_des(:, end) = xd;
%% DDP
ddp_message = true;
safe_mode = false;
ddp_cost = true;
[x_traj, u, cost] = fnDDPdrone(xbar, ubar, x_des, cHorizon, iter, lr,
    Q, S, R, ddp_message, safe_mode, ddp_cost);
%%
figure(100)
plot3(x_traj(1, :), x_traj(2, :), x_traj(3, :), '--')
hold on
plot3(x0(1), x0(2), x0(3), 'rx')
```

```
plot3(xd(1), xd(2), xd(3), 'bx')
hold off
%%
xd = x_des(1:3, end);
[h,MMddp] = quadrotor_visualize(x_traj,u, 0:dt:t-dt, xd)
%%
figure(2)
movie(MMddp,1,length(x_traj(1,:))/t/2)

%% PERFORMANCE
time = 0:dt:t-dt;
horizon = cHorizon;
figure(11);
subplot(4,3,1)
hold on
plot(time,x_traj(1,:),'linewidth',1);
plot(time,x_des(1,end)*ones(1,horizon),'red','linewidth',4)
title('x [m]','fontsize',10);
xlabel('Time in sec','fontsize',10)
%hold off;
grid on

subplot(4,3,2)
hold on;
plot(time,x_traj(2,:),'linewidth',1);
plot(time,x_des(2,end)*ones(1,horizon),'red','linewidth',4)
title('y [m/s]','fontsize',10);
xlabel('Time in sec','fontsize',10)
grid on
%hold off;

subplot(4,3,3)
hold on
plot(time,x_traj(3,:),'linewidth',1);
plot(time,x_des(3,end)*ones(1,horizon),'red','linewidth',4)
title('z [m]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on
%hold off;$\theta$

subplot(4,3,4)
hold on
plot(time,x_traj(4,:),'linewidth',1);
plot(time,x_des(4,end)*ones(1,horizon),'red','linewidth',4)
title('x_{dot} [m/s]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on
```

```
subplot(4,3,5)
hold on
plot(time,x_traj(5,:),'linewidth',1);
plot(time,x_des(5,end)*ones(1,horizon),'red','linewidth',4)
title('y_{dot} [m/s]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on
%hold off

subplot(4,3,6)
hold on
plot(time,x_traj(6,:),'linewidth',1);
plot(time,x_des(6,end)*ones(1,horizon),'red','linewidth',4)
title('z_{dot} [m/s]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(4,3,7)
hold on
plot(time,x_traj(7,:),'linewidth',1);
plot(time,x_des(7,end)*ones(1,horizon),'red','linewidth',4)
title('roll [rad]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(4,3,8)
hold on
plot(time,x_traj(8,:),'linewidth',1);
plot(time,x_des(8,end)*ones(1,horizon),'red','linewidth',4)
title('pitch [rad]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(4,3,9)
hold on
plot(time,x_traj(9,:),'linewidth',1);
plot(time,x_des(9,end)*ones(1,horizon),'red','linewidth',4)
title('yaw [rad]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(4,3,10)
hold on
plot(time,x_traj(10,:),'linewidth',1);
plot(time,x_des(10,end)*ones(1,horizon),'red','linewidth',4)
title('p [rad/s]','fontsize',10)
```

```
xlabel('Time in sec','fontsize',10)
grid on

subplot(4,3,11)
hold on
plot(time,x_traj(11,:),'linewidth',1);
plot(time,x_des(11,end)*ones(1,horizon),'red','linewidth',4)
title('q [rad/s]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(4,3,12)
hold on
plot(time,x_traj(12,:),'linewidth',1);
plot(time,x_des(12,end)*ones(1,horizon),'red','linewidth',4)
title('r [rad/s]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on
%%
figure(13)
subplot(2, 2, 1)
hold on
plot(time(1:end-1),u(1,:),'linewidth',1);
title('thrust u1 [N]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(2, 2, 2)
hold on
plot(time(1:end-1),u(2,:),'linewidth',1);
title('thrust u2 [N]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(2, 2, 3)
hold on
plot(time(1:end-1),u(3,:),'linewidth',1);
title('thrust u3 [N]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on

subplot(2, 2, 4)
hold on
plot(time(1:end-1),u(4,:),'linewidth',1);
title('thrust u4 [N]','fontsize',10)
xlabel('Time in sec','fontsize',10)
grid on
```

```
%%
figure(14)
hold on
plot(cost,'linewidth',1);
xlabel('Iterations','fontsize',10)
title('Cost','fontsize',10);
grid on
%save('DDP_Data');


%% functions
function r = R1(rad)
    r = [1 0 0; 0 cos(rad) sin(rad); 0 -sin(rad) cos(rad)];
end
function r = R2(rad)
    r = [cos(rad) 0 -sin(rad); 0 1 0; sin(rad) 0 cos(rad)];
end
function r = R3(rad)
    r = [cos(rad) sin(rad) 0; -sin(rad) cos(rad) 0; 0 0 1];
end
```

```
function [x_traj, u, cost] = fnDDPdrone(xbar, ubar, x_des, h2, iter,
    gamma, Q, S, R, ddp_message, safe_mode, ddp_cost)

    %% initial configuration
    dt = 0.01;
    horizon = h2;
    u = ubar;
    x_traj = xbar;
    cost = [];
    %%
    for p = gamma
        lr = p; % learning rate
        for k = 1:iter
            %% Linearization of the dynamics
            for i = 1:(horizon-1)
                % Running cost L(x, u)
                [l0,l_x,l_xx,l_u,l_uu,l_ux] = fnCost(x_traj(:,i) -
                    x_des(:, i), u(:,i), Q, R, dt);
                L0(i) = l0;
                Lx(:, i) = l_x;
                Lxx(:, :, i) = l_xx;
                Lu(:, i) = l_u;
                Luu(:, :, i) = l_uu;
                Lux(:, :, i) = l_ux; % M = N^T

```

```
            if safe_mode
                [~, dfx, dfu] = fnStateDroneSafe(x_traj(:, i), u(:, i));
            else
                [~, dfx, dfu] = fnStateDrone(x_traj(:, i), u(:, i));
            end

            % linearized form in discrete time
            A(:, :, i) = eye(size(xbar, 1)) + dfx*dt; % Phi
            B(:, :, i) = dfu*dt; % B of linearized equation in the classnote.
        end

        %--------Search the control -------------%
        Vxx(:,:, horizon)= S;
        Vx(:, horizon) = S * (x_traj(:, horizon) - x_des(:, horizon));
        V(horizon) = 0.5*(x_traj(:, horizon) - x_des(:, horizon))'*S*(x_traj(:, horizon) - x_des(:, horizon));

        %-------- Backpropagation of the Value Function ------------(Chapter 1/ p.7)--%
        for j = (horizon-1):-1:1
            Qu(:, j) = Lu(:, j) + B(:, :, j)'*Vx(:, j+1);
            Qx(:, j) = Lx(:, j) + A(:, :, j)'*Vx(:, j+1); %% different Vx
            Quu(:, :, j) = B(:, :, j)'*Vxx(:, :, j+1)*B(:, :, j) + Luu(:, :, j);
            % Quu becomes dead in the middle. Its value ~0 or negative, which means
            % it will never update the propagation / no feedback.
            % For example, imagine the zero or negative slope in the optimization graph.
            lambda = abs(min(eig(Quu(:, :, j)))) + rand;
            if any(eig(Quu(:, :, j)) < 0.0001)
                Quu(:, :, j) = Quu(:, :, j) + lambda*eye(length(Quu(:, :, j)));
            end
            Qxx(:, :, j) = A(:, :, j)'*Vxx(:, :, j+1)*A(:, :, j) + Lxx(:, :, j);
            Qux(:, :, j) = B(:, :, j)'*Vxx(:, :, j+1)*A(:, :, j) + Lux(:, :, j)';

            %feedback
            Kfb(:, :, j) = -Quu(:, :, j)\Qux(:, :, j);
            %feedforward
```

```
            Kff(:, j) = -Quu(:, :, j)\Qu(:, j);

            Vxx(:, :, j) = round(Qxx(:, :, j), 6) + round(Kfb(:, :, j)'*Quu(:, :, j)*Kfb(:, :, j), 6) + round(2*Qux(:, :, j)'*Kfb(:, :, j), 6);
            Vx(:, j) = round(Qx(:, j), 6) + round(Kfb(:, :, j)'*Qu(:, j), 6) + round(Kfb(:, :, j)'*Quu(:, :, j)*Kff(:, j), 6) + round(Qux(:, :, j)'*Kff(:, j), 6);
%             disp(j)
%             disp([Vx(:, j)])
%             disp([Vxx(:, :, j)])
%             disp([Qx(:, j)])
%             disp([Qxx(:, :, j)])
%             disp([Quu(:, :, j)]) % becoming large at 763
%             disp([Qux(:, :, j)])
        end

    %---------------------------------------------------------------> Forward Propagaion:
    %--------------------------------> Find the controls/ forward
    if safe_mode
        for i=1:(horizon-1)
            du = Kff(:, i) + Kfb(:, :, i)*(x_traj(:, i)-xbar(:, i));
            u(:, i) = ubar(:, i) + lr * du;
            u(:, i) = min(ones(4, 1).*2, u(:, i));
            u(:, i) = max(-ones(4, 1).*2, u(:, i));
            [x_, ~, ~] = fnStateDroneSafe(x_traj(:, i), u(:, i));
            x_traj(:, i+1) = x_traj(:, i) + x_.*dt;
        end
    else
        for i=1:(horizon-1)
            du = Kff(:, i) + Kfb(:, :, i)*(x_traj(:, i)-xbar(:, i));
            u(:, i) = ubar(:, i) + lr * du;
            u(:, i) = min(ones(4, 1).*2, u(:, i));
            u(:, i) = max(-ones(4, 1).*2, u(:, i));
            [x_, ~, ~] = fnStateDrone(x_traj(:, i), u(:, i));
            x_traj(:, i+1) = x_traj(:, i) + x_.*dt;
        end
    end
    ubar = u;
    %----------------------------------------> Simulation of the Nonlinear System
    xbar = x_traj;
    [Cost(:,k)] =  fnCostComputation(x_traj,u,x_des, dt, Q, S
```

```
                , R);
            if ddp_message
                fprintf('DDP Iteration %d,  Current Cost = %.6f, Learning rate = %.4f\n',k,Cost(1,k), lr);
            end
            if ddp_cost
                cost = [cost Cost(1,k)];
            end
        end %% end iterating over the algorithm
        if ddp_cost == 0
            cost = [cost Cost(:, end)];
        end
        %figure(1);
%          subplot(2,4,1)
%          hold on
%          plot(time,x_traj(1,:),'linewidth',1);
%          plot(time,x_des(1,1)*ones(1,horizon),'red','linewidth',4)
%          title('x [m]','fontsize',10);
%          xlabel('Time in sec','fontsize',10)
%         %hold off;
%          grid on
%
%
%          subplot(2,4,2);hold on;
%          plot(time,x_traj(2,:),'linewidth',1);
%          plot(time,x_des(2,1)*ones(1,horizon),'red','linewidth',4)
%          title('x_{dot} [m/s]','fontsize',10);
%          xlabel('Time in sec','fontsize',10)
%          grid on
%         %hold off;
%
%          subplot(2,4,3);hold on
%          plot(time,x_traj(3,:),'linewidth',1);
%          plot(time,x_des(3,1)*ones(1,horizon),'red','linewidth',4)
%          title('\theta [rad]','fontsize',10)
%          xlabel('Time in sec','fontsize',10)
%          grid on
%         %hold off;$\theta$
%
%          subplot(2,4,4);hold on
%          plot(time,x_traj(4,:),'linewidth',1);
%          plot(time,x_des(4,1)*ones(1,horizon),'red','linewidth',4)
%          title('\theta_{dot} [rad/s]','fontsize',10)
%          xlabel('Time in sec','fontsize',10)
%          grid on
%
%          subplot(2,4,5);hold on
```

```
%           plot(time(1:end-1), u(:, 1:horizon-1), 'linewidth',1)
%           title('optimal control','fontsize',10)
%           xlabel('Time in sec','fontsize',10)
%           grid on;
%          %hold off
%
%
%           subplot(2,4,6);hold on
%           plot(Cost,'linewidth',1);
%           xlabel('Iterations','fontsize',10)
%           title('Cost','fontsize',10);
%           grid on
%          %save('DDP_Data');
    end
end
```

```
function [x_, dfx, dfu] = fnStateDroneSafe(xhat, u)

x = xhat(1);
y = xhat(2);
z = xhat(3);
xdot = xhat(4);
ydot = xhat(5);
zdot = xhat(6);
roll = xhat(7);
pitch = xhat(8);
yaw = xhat(9);
p = xhat(10);
q = xhat(11);
r = xhat(12);
z1 = xhat(13);
z2 = xhat(14);
z3 = xhat(15);
u1 = u(1);
u2 = u(2);
u3 = u(3);
u4 = u(4);

dfx = [0,0,0,1,0,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,1,0,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,1,0,0,0,0,0,0,0,0,0;
    0,0,0,0,0,0,0,-2*cos(pitch)*(u1 + u2 + u3 + u4), 0,0,0,0,0,0,0;
    0,0,0,0,0,0,2*cos(pitch)*cos(roll)*(u1 + u2 + u3 + u4),-2*sin(
        pitch)*sin(roll)*(u1 + u2 + u3 + u4), 0,0,0,0, 0, 0, 0;
    0,0,0,0,0,0,-2*cos(pitch)*sin(roll)*(u1 + u2 + u3 + u4),-2*cos(
        roll)*sin(pitch)*(u1 + u2 + u3 + u4), 0,0,0,0, 0, 0, 0;
    0,0,0,0,0,0,q*cos(roll)*tan(pitch) - r*tan(pitch)*sin(roll),r*cos
```

```
    (roll)*(tan(pitch)^2 + 1) + q*sin(roll)*(tan(pitch)^2 + 1),
    0,1, tan(pitch)*sin(roll), cos(roll)*tan(pitch),  0,  0,  0;
0,0,0,0,0,0,- r*cos(roll) - q*sin(roll), 0, 0,0,cos(roll), -sin(
    roll),  0,  0,  0;
0,0,0,0,0,0, (q*cos(roll))/cos(pitch) - (r*sin(roll))/cos(pitch),
    (r*cos(roll)*sin(pitch))/cos(pitch)^2 + (q*sin(pitch)*sin(
    roll))/cos(pitch)^2, 0, 0, sin(roll)/cos(pitch), cos(roll)/cos
    (pitch),  0,  0,  0;
0,0,0,0,0,0,0,0,0,0,-(23*r)/32,-(23*q)/32,  0,  0,  0;
0,0,0,0,0,0,0,0, 0, (23*r)/32,0,(23*p)/32,  0,  0,  0;
0,0,0,0,0,0,0,0, 0,0,0,0,  0,  0,  0;
(2*xdot*(2*x - 22/5)^3)/((x - 11/5)^2 + (y - 11/5)^2 + (z - 1)^2
    - 1)^3 - (xdot*(8*x - 88/5))/((x - 11/5)^2 + (y - 11/5)^2 + (z
    - 1)^2 - 1)^2 - (2*ydot*(2*y - 22/5))/((x - 11/5)^2 + (y -
    11/5)^2 + (z - 1)^2 - 1)^2 - (2*zdot*(2*z - 2))/((x - 11/5)^2
    + (y - 11/5)^2 + (z - 1)^2 - 1)^2 - (2*x - 22/5)/((x - 11/5)^2
    + (y - 11/5)^2 + (z - 1)^2 - 1)^2 + (2*ydot*(2*x - 22/5)
    ^2*(2*y - 22/5))/((x - 11/5)^2 + (y - 11/5)^2 + (z - 1)^2 - 1)
    ^3 + (2*zdot*(2*x - 22/5)^2*(2*z - 2))/((x - 11/5)^2 + (y -
    11/5)^2 + (z - 1)^2 - 1)^3, (2*xdot*(2*x - 22/5)^2*(2*y -
    22/5))/((x - 11/5)^2 + (y - 11/5)^2 + (z - 1)^2 - 1)^3 - (2*
    ydot*(2*x - 22/5))/((x - 11/5)^2 + (y - 11/5)^2 + (z - 1)^2 -
    1)^2 - (2*y - 22/5)/((x - 11/5)^2 + (y - 11/5)^2 + (z - 1)^2 -
    1)^2 + (2*ydot*(2*x - 22/5)*(2*y - 22/5)^2)/((x - 11/5)^2 + (
    y - 11/5)^2 + (z - 1)^2 - 1)^3 + (2*zdot*(2*x - 22/5)*(2*y -
    22/5)*(2*z - 2))/((x - 11/5)^2 + (y - 11/5)^2 + (z - 1)^2 - 1)
    ^3, (2*xdot*(2*x - 22/5)^2*(2*z - 2))/((x - 11/5)^2 + (y -
    11/5)^2 + (z - 1)^2 - 1)^3 - (2*zdot*(2*x - 22/5))/((x - 11/5)
    ^2 + (y - 11/5)^2 + (z - 1)^2 - 1)^2 - (2*z - 2)/((x - 11/5)^2
    + (y - 11/5)^2 + (z - 1)^2 - 1)^2 + (2*zdot*(2*x - 22/5)*(2*z
    - 2)^2)/((x - 11/5)^2 + (y - 11/5)^2 + (z - 1)^2 - 1)^3 + (2*
    ydot*(2*x - 22/5)*(2*y - 22/5)*(2*z - 2))/((x - 11/5)^2 + (y -
    11/5)^2 + (z - 1)^2 - 1)^3, -(2*x - 22/5)^2/((x - 11/5)^2 + (
    y - 11/5)^2 + (z - 1)^2 - 1)^2, -((2*x - 22/5)*(2*y - 22/5))
    /((x - 11/5)^2 + (y - 11/5)^2 + (z - 1)^2 - 1)^2, -((2*x -
    22/5)*(2*z - 2))/((x - 11/5)^2 + (y - 11/5)^2 + (z - 1)^2 - 1)
    ^2,                                                         0,

    0, 0,          0,                    0,                    0,
    -1,  0,  0;
(16*x^3*xdot)/((y + 1/5)^2 + x^2 + z^2 - 1)^3 - (2*x)/((y + 1/5)
    ^2 + x^2 + z^2 - 1)^2 - (2*ydot*(2*y + 2/5))/((y + 1/5)^2 + x
    ^2 + z^2 - 1)^2 - (8*x*xdot)/((y + 1/5)^2 + x^2 + z^2 - 1)^2 -
    (4*z*zdot)/((y + 1/5)^2 + x^2 + z^2 - 1)^2 + (8*x^2*ydot*(2*y
    + 2/5))/((y + 1/5)^2 + x^2 + z^2 - 1)^3 + (16*x^2*z*zdot)/((y
    + 1/5)^2 + x^2 + z^2 - 1)^3,
```

```
    (8*x^2*xdot*(2*y + 2/5))/((y + 1/5)^2 + x^2 + z^2 - 1)^3 - (4*
    x*ydot)/((y + 1/5)^2 + x^2 + z^2 - 1)^2 - (2*y + 2/5)/((y +
    1/5)^2 + x^2 + z^2 - 1)^2 + (4*x*ydot*(2*y + 2/5)^2)/((y +
    1/5)^2 + x^2 + z^2 - 1)^3 + (8*x*z*zdot*(2*y + 2/5))/((y +
    1/5)^2 + x^2 + z^2 - 1)^3,

    (16*x^2*xdot*z)/((y + 1/5)^2 + x^2 + z^2 - 1)^3 - (4*x*zdot)
    /((y + 1/5)^2 + x^2 + z^2 - 1)^2 - (2*z)/((y + 1/5)^2 + x^2 +
    z^2 - 1)^2 + (16*x*z^2*zdot)/((y + 1/5)^2 + x^2 + z^2 - 1)^3 +
     (8*x*ydot*z*(2*y + 2/5))/((y + 1/5)^2 + x^2 + z^2 - 1)^3,
                              -(4*x^2)/((y + 1/5)^2 + x^2 + z^2 - 1)
    ^2,                             -(2*x*(2*y + 2/5))/((y + 1/5)^2
    + x^2 + z^2 - 1)^2,                                      -(4*x*z)
    /((y + 1/5)^2 + x^2 + z^2 - 1)^2,
                                                        0,

    0, 0,           0,                     0,                    0,
    0, -1,  0;
  (2*xdot*(2*x - 6)^3)/((x - 3)^2 + (z - 1/2)^2 + y^2 - 1)^3 - (
    xdot*(8*x - 24))/((x - 3)^2 + (z - 1/2)^2 + y^2 - 1)^2 - (2*
    zdot*(2*z - 1))/((x - 3)^2 + (z - 1/2)^2 + y^2 - 1)^2 - (4*y*
    ydot)/((x - 3)^2 + (z - 1/2)^2 + y^2 - 1)^2 - (2*x - 6)/((x -
    3)^2 + (z - 1/2)^2 + y^2 - 1)^2 + (2*zdot*(2*x - 6)^2*(2*z -
    1))/((x - 3)^2 + (z - 1/2)^2 + y^2 - 1)^3 + (4*y*ydot*(2*x -
    6)^2)/((x - 3)^2 + (z - 1/2)^2 + y^2 - 1)^3,

    (4*xdot*y*(2*x - 6)^2)/((x - 3)^2 + (z - 1/2)^2 + y^2 - 1)^3 -
     (2*ydot*(2*x - 6))/((x - 3)^2 + (z - 1/2)^2 + y^2 - 1)^2 -
    (2*y)/((x - 3)^2 + (z - 1/2)^2 + y^2 - 1)^2 + (8*y^2*ydot*(2*x
     - 6))/((x - 3)^2 + (z - 1/2)^2 + y^2 - 1)^3 + (4*y*zdot*(2*x
    - 6)*(2*z - 1))/((x - 3)^2 + (z - 1/2)^2 + y^2 - 1)^3,

    (2*xdot*(2*x - 6)^2*(2*z - 1))/((x - 3)^2 + (z - 1/2)^2 + y^2
    - 1)^3 - (2*zdot*(2*x - 6))/((x - 3)^2 + (z - 1/2)^2 + y^2 -
    1)^2 - (2*z - 1)/((x - 3)^2 + (z - 1/2)^2 + y^2 - 1)^2 + (2*
    zdot*(2*x - 6)*(2*z - 1)^2)/((x - 3)^2 + (z - 1/2)^2 + y^2 -
    1)^3 + (4*y*ydot*(2*x - 6)*(2*z - 1))/((x - 3)^2 + (z - 1/2)^2
     + y^2 - 1)^3,                -(2*x - 6)^2/((x - 3)^2 + (z -
    1/2)^2 + y^2 - 1)^2,                          -(2*y*(2*x - 6))/((
    x - 3)^2 + (z - 1/2)^2 + y^2 - 1)^2,                  -((2*x - 6)
    *(2*z - 1))/((x - 3)^2 + (z - 1/2)^2 + y^2 - 1)^2,
                                                        0,

    0, 0,           0,                     0,                    0,
    0,  0, -1];
```

```

dfu = [0,0,0,0;
     0,0,0,0;
     0,0,0,0;
     -2*sin(pitch),             -2*sin(pitch),              -2*sin(pitch),              -2*sin(pitch);
     2*cos(pitch)*sin(roll), 2*cos(pitch)*sin(roll), 2*cos(pitch)*sin(roll), 2*cos(pitch)*sin(roll);
     2*cos(pitch)*cos(roll), 2*cos(pitch)*cos(roll), 2*cos(pitch)*cos(roll), 2*cos(pitch)*cos(roll);
                              0,                              0,                              0,                              0;
                              0,                              0,                              0,                              0;
                              0,                              0,                              0,                              0;
        (425*2^(1/2))/16,        -(425*2^(1/2))/16,         (425*2^(1/2))/16,         -(425*2^(1/2))/16;
       -(425*2^(1/2))/16,        -(425*2^(1/2))/16,         (425*2^(1/2))/16,          (425*2^(1/2))/16;
                     1691/550,                      -1691/550,                      -1691/550,                      1691/550
                              0,                              0,                              0,                              0;
                              0,                              0,                              0,                              0;
                              0,                              0,                              0,                              0;];

x_ = [xdot;
     ydot;
     zdot;
     -2*sin(pitch)*(u1 + u2 + u3 + u4);
     2*cos(pitch)*sin(roll)*(u1 + u2 + u3 + u4);
     2*cos(roll)*cos(pitch)*(u1 + u2 + u3 + u4) - 981/200;
     p + r*cos(roll)*tan(pitch) + q*sin(roll)*tan(pitch);
     q*cos(roll) - r*sin(roll);
     (r*cos(roll))/cos(pitch) + (q*sin(roll))/cos(pitch);
     (425*2^(1/2)*(conj(u1) - conj(u2) + conj(u3) - conj(u4)))/16 - (23*conj(q)*conj(r))/32;
     (23*conj(p)*conj(r))/32 - (425*2^(1/2)*(conj(u1) + conj(u2) - conj(u3) - conj(u4)))/16;
     (1691*conj(u1))/550 - (1691*conj(u2))/550 - (1691*conj(u3))/550 + (1691*conj(u4))/550;
     1/((x - 11/5)^2 + (y - 11/5)^2 + (z - 1)^2 - 1) - z1 - (xdot*(2*x - 22/5)^2)/((x - 11/5)^2 + (y - 11/5)^2 + (z - 1)^2 - 1)^2 - (ydot*(2*x - 22/5)*(2*y - 22/5))/((x - 11/5)^2 + (y - 11/5)^2
```

```
    + (z - 1)^2 - 1)^2 - (zdot*(2*x - 22/5)*(2*z - 2))/((x - 11/5)
    ^2 + (y - 11/5)^2 + (z - 1)^2 - 1)^2 - 25/1192;
  1/((y + 1/5)^2 + x^2 + z^2 - 1) - z2 - (4*x^2*xdot)/((y + 1/5)^2
    + x^2 + z^2 - 1)^2 - (2*x*ydot*(2*y + 2/5))/((y + 1/5)^2 + x^2
     + z^2 - 1)^2 - (4*x*z*zdot)/((y + 1/5)^2 + x^2 + z^2 - 1)^2 -
     25/306;
  1/((x - 3)^2 + (z - 1/2)^2 + y^2 - 1) - z3 - (xdot*(2*x - 6)^2)
    /((x - 3)^2 + (z - 1/2)^2 + y^2 - 1)^2 - (zdot*(2*x - 6)*(2*z
    - 1))/((x - 3)^2 + (z - 1/2)^2 + y^2 - 1)^2 - (2*y*ydot*(2*x -
     6))/((x - 3)^2 + (z - 1/2)^2 + y^2 - 1)^2 - 4/165];
end
```

```
function [x_, dfx, dfu] = fnStateDrone(xhat, u)

x = xhat(1);
y = xhat(2);
z = xhat(3);
xdot = xhat(4);
ydot = xhat(5);
zdot = xhat(6);
roll = xhat(7);
pitch = xhat(8);
yaw = xhat(9);
p = xhat(10);
q = xhat(11);
r = xhat(12);
u1 = u(1);
u2 = u(2);
u3 = u(3);
u4 = u(4);

dfx = [0, 0, 0, 1, 0, 0,
    0,

    0, 0,          0,                    0,                    0;
0, 0, 0, 0, 1, 0,
    0,

    0, 0,          0,                    0,                    0;
0, 0, 0, 0, 0, 1,
    0,

    0, 0,          0,                    0,                    0;
0, 0, 0, 0, 0, 0,
    0,                                              -2*cos(pitch)*(u1 +
     u2 + u3 + u4), 0,          0,                    0,
                    0;
```

```
0, 0, 0, 0, 0, 0,          2*cos(pitch)*cos(roll)*(u1 + u2 + u3 + u4),          -2*sin(pitch)*sin(roll)*(u1 + u2 + u3 + u4), 0,          0,          0,          0;
0, 0, 0, 0, 0, 0,          -2*cos(pitch)*sin(roll)*(u1 + u2 + u3 + u4),          -2*cos(roll)*sin(pitch)*(u1 + u2 + u3 + u4), 0,          0,          0,          0;
0, 0, 0, 0, 0, 0,      q*cos(roll)*tan(pitch) - r*tan(pitch)*sin(roll),          r*cos(roll)*(tan(pitch)^2 + 1) + q*sin(roll)*(tan(pitch)^2 + 1), 0,          1, tan(pitch)*sin(roll), cos(roll)*tan(pitch);
0, 0, 0, 0, 0, 0,          - r*cos(roll) - q*sin(roll),

    0, 0,          0,          cos(roll),          -sin(roll);
0, 0, 0, 0, 0, 0, (q*cos(roll))/cos(pitch) - (r*sin(roll))/cos(pitch), (r*cos(roll)*sin(pitch))/cos(pitch)^2 + (q*sin(pitch)*sin(roll))/cos(pitch)^2, 0,          0, sin(roll)/cos(pitch), cos(roll)/cos(pitch);
0, 0, 0, 0, 0, 0,
    0,

    0, 0,          0,          -(23*r)/32,          -(23*q)/32;
0, 0, 0, 0, 0, 0,
    0,

    0, 0, (23*r)/32,          0,          (23*p)/32;
0, 0, 0, 0, 0, 0,
    0,

    0, 0,          0,          0,          0];

dfu = [          0,          0,          0,          0;
          0,          0,          0,          0;
          0,          0,          0,          0;
          -2*sin(pitch),          -2*sin(pitch),          -2*sin(pitch),          -2*sin(pitch);
2*cos(pitch)*sin(roll), 2*cos(pitch)*sin(roll), 2*cos(pitch)*sin(roll), 2*cos(pitch)*sin(roll);
2*cos(pitch)*cos(roll), 2*cos(pitch)*cos(roll), 2*cos(pitch)*cos(roll), 2*cos(pitch)*cos(roll);
          0,          0,          0,          0;
```

```
                         0,                          0,                          0,                          0;
                         0,                          0,                          0,                          0;
         (425*2^(1/2))/16,          -(425*2^(1/2))/16,          (425*2^(1/2))/16,          -(425*2^(1/2))/16;
        -(425*2^(1/2))/16,          -(425*2^(1/2))/16,          (425*2^(1/2))/16,          (425*2^(1/2))/16;
                 1691/550,                  -1691/550,                  -1691/550,                  1691/550];

x_ = [xdot;...
    ydot;...
    zdot;...
    -2*sin(pitch)*(u1 + u2 + u3 + u4);...
    2*cos(pitch)*sin(roll)*(u1 + u2 + u3 + u4);...
    2*cos(roll)*cos(pitch)*(u1 + u2 + u3 + u4) - 981/200;...
    p + r*cos(roll)*tan(pitch) + q*sin(roll)*tan(pitch);...
    q*cos(roll) - r*sin(roll);...
    (r*cos(roll))/cos(pitch) + (q*sin(roll))/cos(pitch);...
    (425*2^(1/2)*(conj(u1) - conj(u2) + conj(u3) - conj(u4)))/16 - (23*conj(q)*conj(r))/32;...
    (23*conj(p)*conj(r))/32 - (425*2^(1/2)*(conj(u1) + conj(u2) - conj(u3) - conj(u4)))/16;...
    (1691*conj(u1))/550 - (1691*conj(u2))/550 - (1691*conj(u3))/550 + (1691*conj(u4))/550];
end
```

```
function  [l0,l_x,l_xx,l_u,l_uu,l_ux] = fnCost(x, u, Q, R, dt)

l0 = 0.5.*(u' *R *u + x'*Q*x) .* dt;
l_x = Q * x .* dt;
l_xx = Q .* dt;
l_u = R * u .* dt;
l_uu = R .* dt;
l_ux = zeros(length(Q), length(R)) .* dt;
end
```

```
function [Cost] =  fnCostComputation(x_traj, u_new,x_des,dt,Q,S,R)

    [~, Horizon] = size(x_traj);
    Cost = 0;
    for j =1:(Horizon-1)
        Cost = Cost + (u_new(:,j)' * R * u_new(:,j) + (x_traj(:, j) - x_des(:, j))'*Q*(x_traj(:, j) - x_des(:, j)))*dt;
    end
    TerminalCost= (x_traj(:,Horizon) - x_des(:, end))'*S * (x_traj(:,
```

```
        Horizon) - x_des(:, end));

    Cost = Cost + TerminalCost;

end
```