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

MinGyu Kim

December 5, 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

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

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

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)

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

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

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

1: **procedure** FNMPC
2:     INITIALIZE VARIABLES: initial state value, desired state value, $\bar{u}, u_{mpc}^{-}\bar{x}, x$, cHorizon, pHorizon, t, dt, horizon, iteration, learning rate and WEIGHTS: Q, S, R
3:     **for** 1:pHorizon-1 **do**
4:         Obtain $\bar{x}, \bar{u} \leftarrow$ Run fnDDP function
5:         update $u_{mpc} \leftarrow \bar{u}$
6:         update trajectory
7:         update $\bar{u}$ (shift horizon)
8:         update $\bar{x}$ (shift horizon)
9:     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
$\beta = \mathbf{B}$

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

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, from experience in this project, you might have to spend more time adjusting the weight matrices if you use that method (Equation 9).

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])
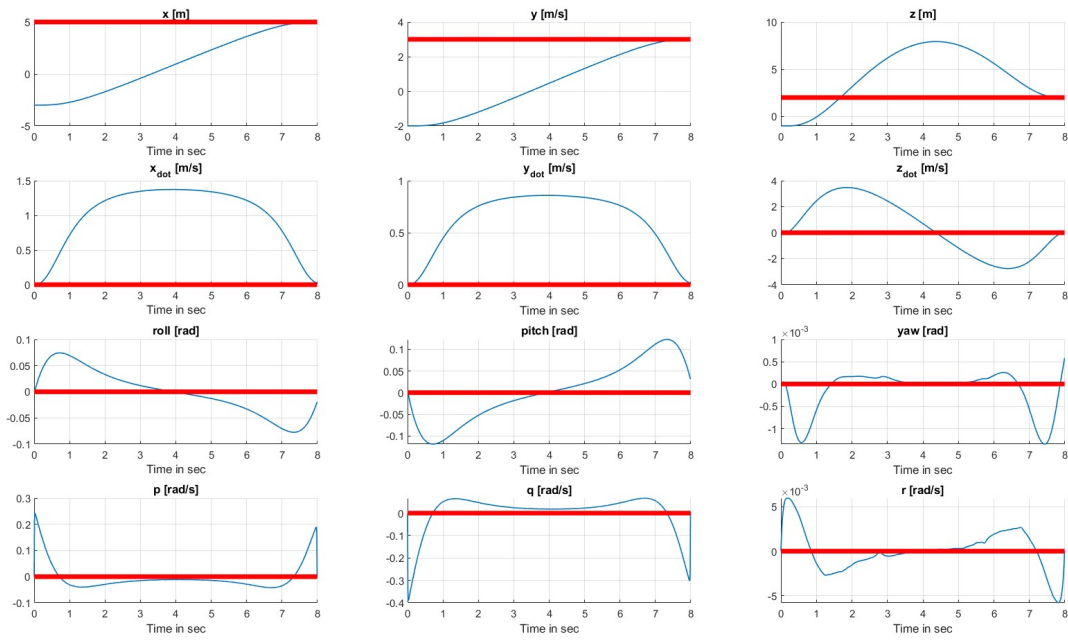R = diag([1, 1, 1, 1].*0.01)

3.2 DDP application

Figure 1: Optimal State Paths Determined Through DDP

Figure 2: Optimal Control Determined Through DDP
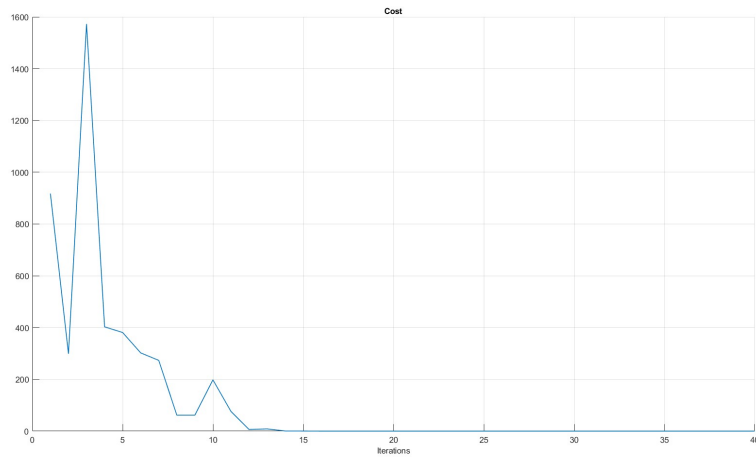
Figure 3: Corresponding Cost Through DDP

Figure 4: Trajectory Simulated by DDP

3.3 MPC application
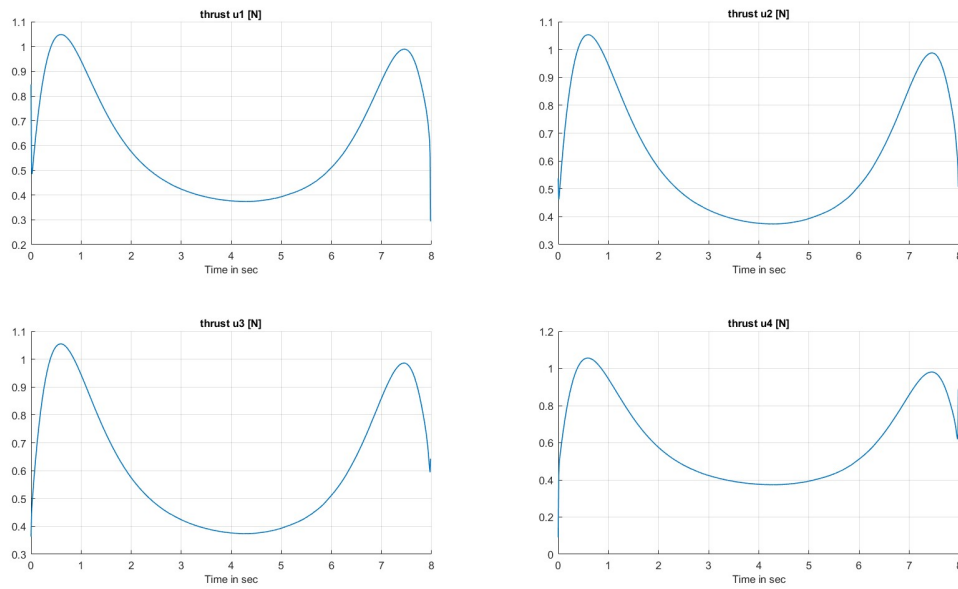

Figure 5: Optimal State Paths Determined Through MPC

Figure 6: Optimal Control Determined Through MPC

Figure 7: Corresponding Cost Through MPC
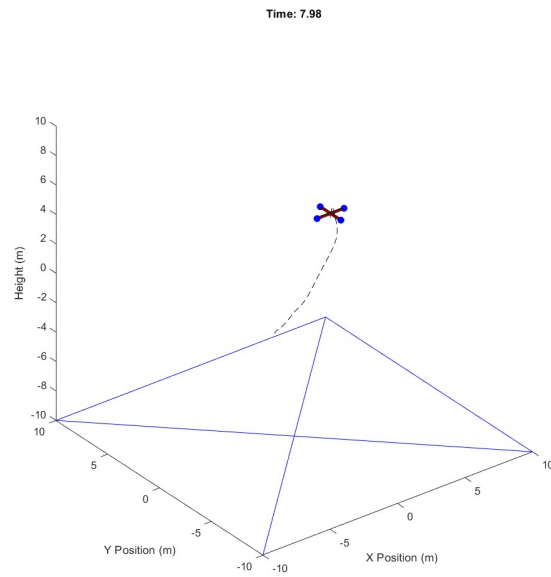
Figure 8: Trajectory Simulated by MPC

3.4 obstacle avoidance using MPC with BaS
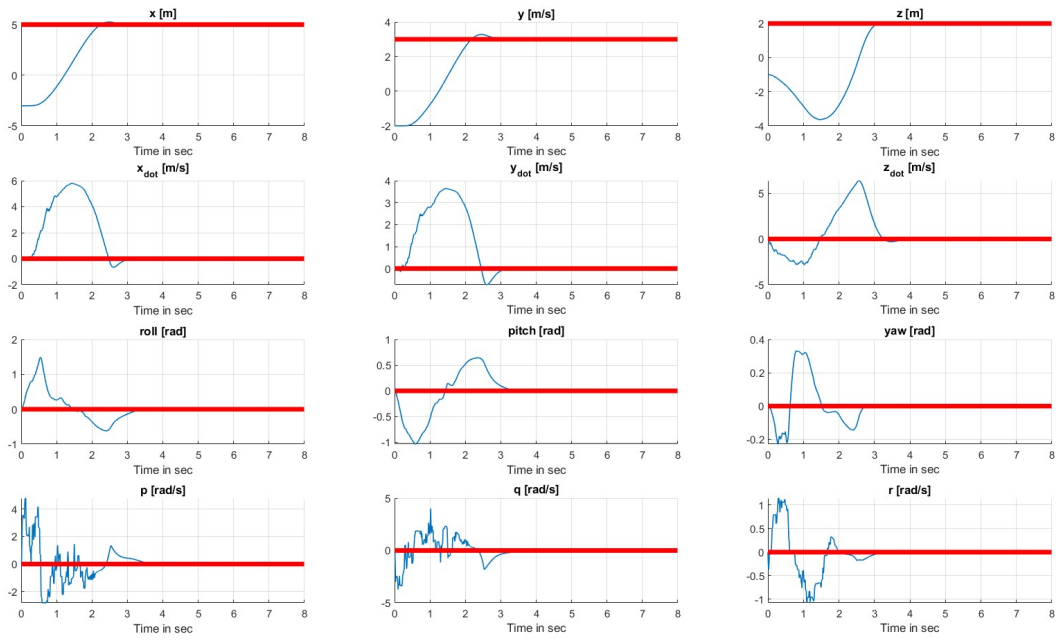
Figure 9: Optimal State Paths Determined Through MPC-BaS
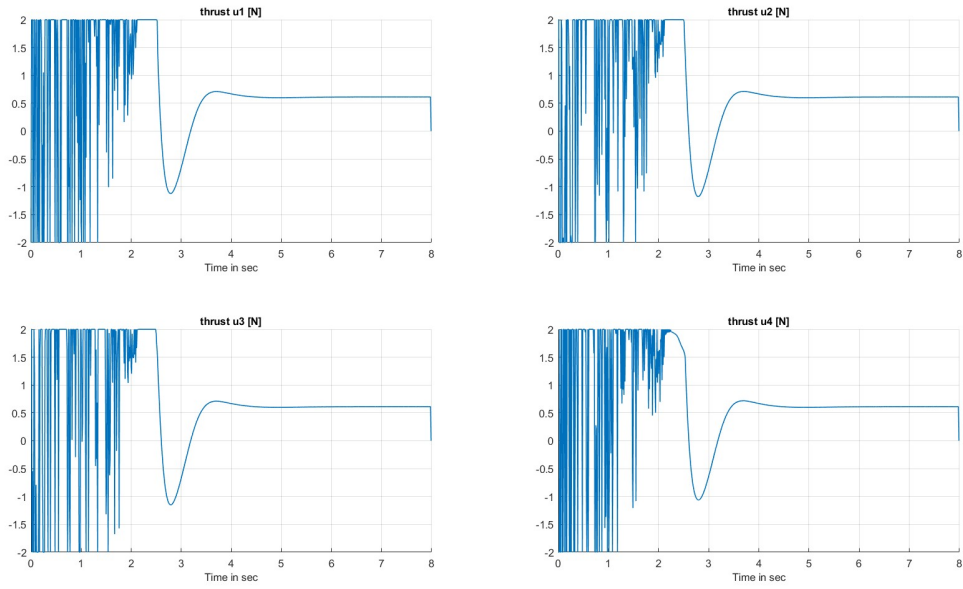
Figure 10: Optimal Control Determined Through MPC-BaS

9

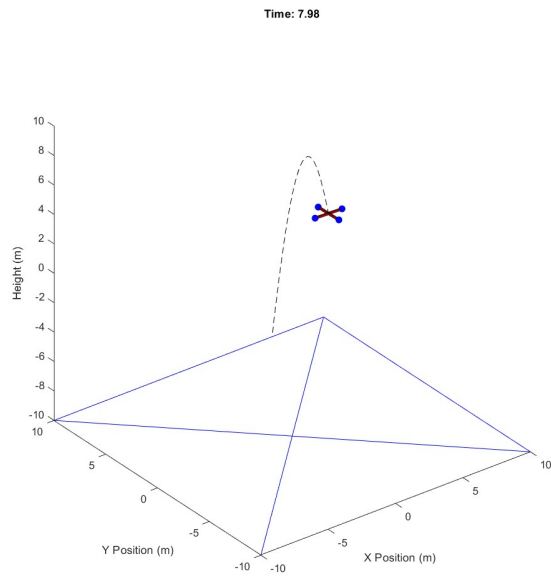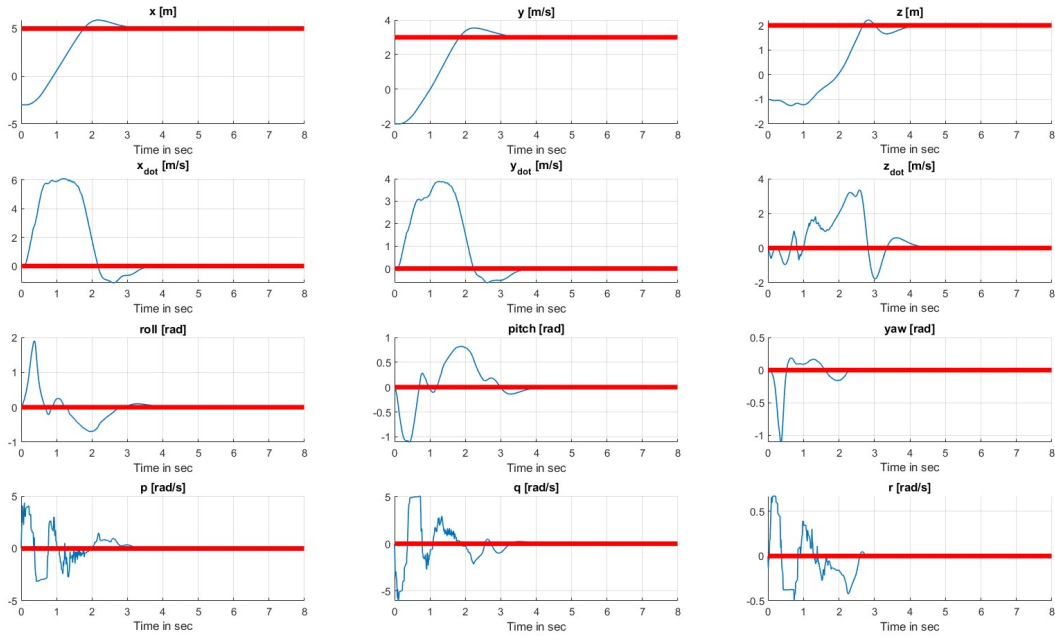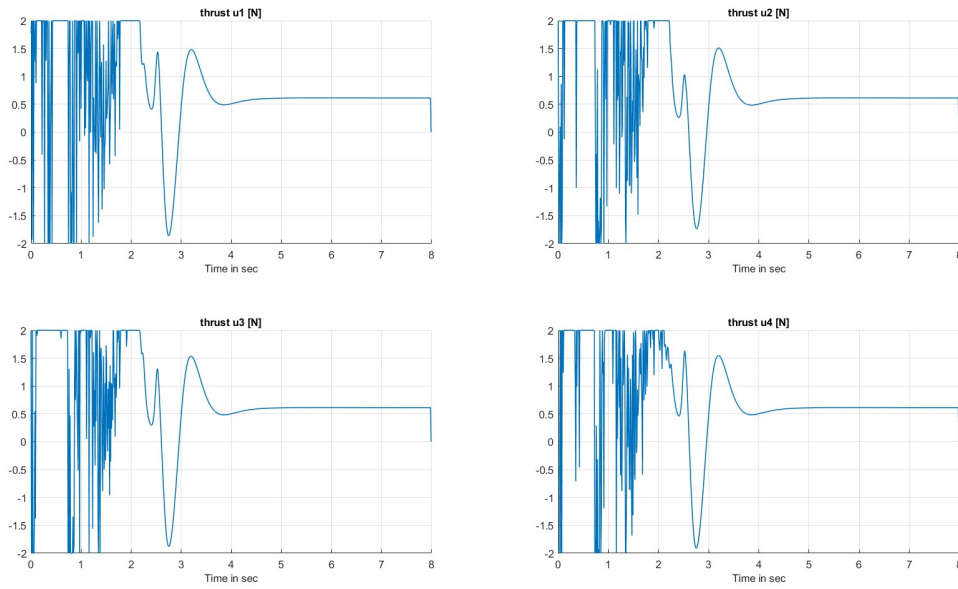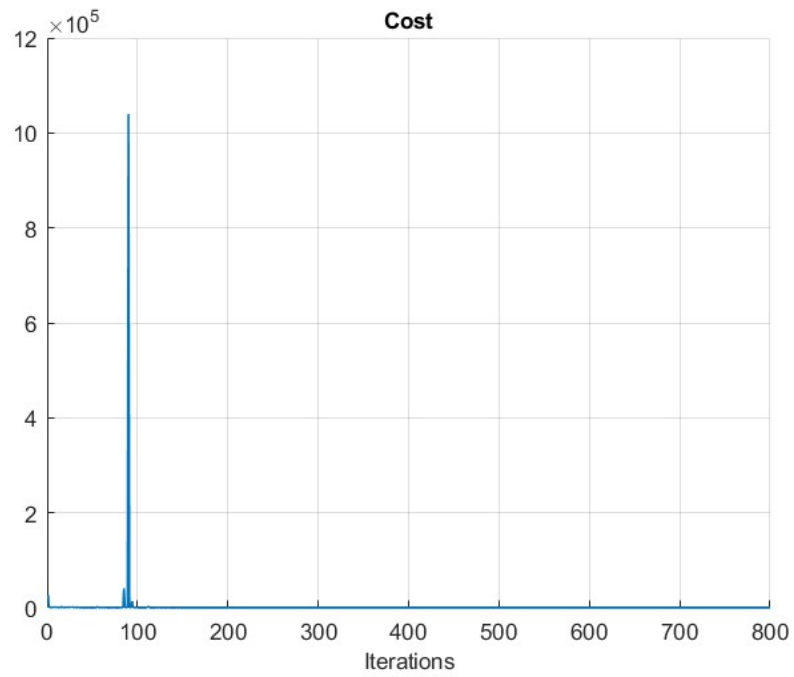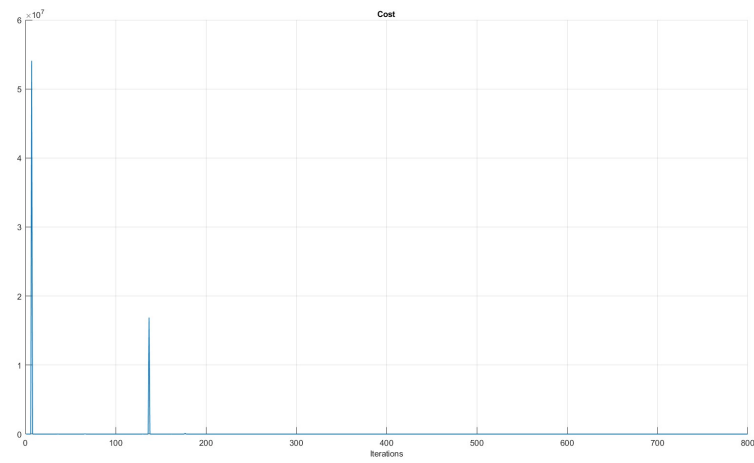Figure 11: Corresponding Cost Through MPC-BaS
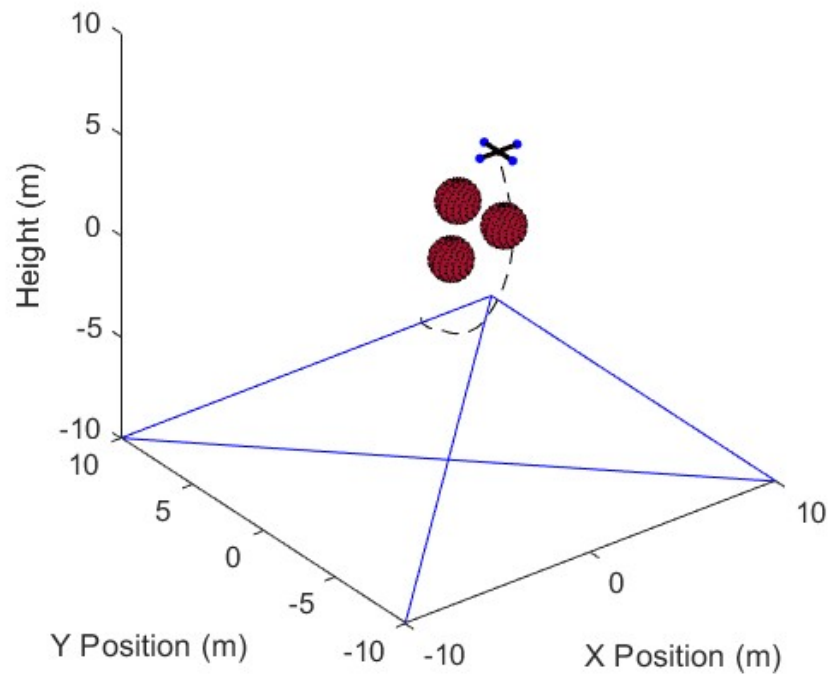
Figure 12: Trajectory Simulated by MPC-BaS

4 DISCUSSION

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.

4.1 vanishing gradient Q_{uu} in backpropagation

Algorithm 3 Force Backpropagation

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

4.2 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.

5   Appendix: MATLAB CODE

```matlab
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];
```

```matlab
% 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)}));
```

```matlab
89  bnut2 = double(subs(b2, {x, y, z}, {x0(1), x0(2), x0(3)}));
90  bnut3 = double(subs(b3, {x, y, z}, {x0(1), x0(2), x0(3)}));
91  zz1 = b1 - bnut1;
92  zz2 = b2 - bnut2;
93  zz3 = b3 - bnut3;
94  %% safety constraint states z
95  gamma = 1;
96  zdot1 = diff(b1)*hx1*dyn - gamma*(z1 - zz1);
97  zdot2 = diff(b2)*hx2*dyn - gamma*(z2 - zz2);
98  zdot3 = diff(b3)*hx3*dyn - gamma*(z3 - zz3);
99
100 %% safety constraints states' initial/desired values
101 xsafe = [xvec; z1; z2; z3];
102
103 z10 = double(subs(zz1, {x, y, z}, {x0(1), x0(2), x0(3)}));
104 z1d = double(subs(zz1, {x, y, z}, {xd(1), xd(2), xd(3)}));
105
106 z20 = double(subs(zz2, {x, y, z}, {x0(1), x0(2), x0(3)}));
107 z2d = double(subs(zz2, {x, y, z}, {xd(1), xd(2), xd(3)}));
108
109 z30 = double(subs(zz3, {x, y, z}, {x0(1), x0(2), x0(3)}));
110 z3d = double(subs(zz3, {x, y, z}, {xd(1), xd(2), xd(3)}));
111 %% the whole system's initial / desired values
112 x0safe = [x0; z10; z20; z30]; % z0 = subs(z, x1, 5);
113 xdsafe = [xd; z1d; z2d; z3d]; % z_des = subs(z, x1, 0);
114
115 %% system update
116 dyn_obs = [dyn; zdot1; zdot2; zdot3];
117 fxSafe = jacobian(dyn_obs, xsafe);
118 fuSafe = jacobian(dyn_obs, u);
119 %% DDP-MPC
120 %% initial configuration
121 t = 8;
122 dt = 0.01;
123 pHorizon = t/dt;
124 cHorizon = 100;
125 iter = 5;
126 lr = 0.5;
127 Qsafe = [1, 1, 1];
128 Ssafe = [10, 10, 10];
129 Q = diag([0, 0, 0, 0, 0, 0, 1, 1, 1, .1, .1, .1, Qsafe]);
130 S = diag([10, 10, 10, 1, 1, 1, 1, 1, 1, 1, 1, 1, Ssafe].*1);
131 R = diag([1, 1, 1, 1].*0.01);
132 ddp_message = false;
133 safe_mode = true;
134 ddp_cost = false;
135 %% nominal state update
```

14

```matlab
136  ubar = zeros(4, cHorizon - 1);
137  ubar(:, 1) = [1, 1, 1, 1].*5;
138  x_des = zeros(12 + 3, cHorizon);
139  x_des(:, 1) = x0safe;
140  x_des(:, end) = xdsafe;
141  xbar = zeros(12 + 3, cHorizon);
142  xbar(:, 1) = x0safe;
143  for i = 1:cHorizon-1
144      [x_, ~, ~] = fnStateDroneSafe(xbar(:, i), ubar(:, i));
145      xbar(:, i+1) = xbar(:, i) + x_.*dt;
146  end
147  umpc = zeros(4, pHorizon);
148  traj = xbar;
149  cost_ = zeros(1, pHorizon);
150  %% simulation
151  for h = 1:pHorizon - 1
152      % find optimal control u at horizon h or step t
153      [xbar, u, cost] = fnDDPdrone(xbar, ubar, x_des, cHorizon, iter,
              lr, Q, S, R, ddp_message, safe_mode, ddp_cost); %(horizon + 1
              - h)*dt,
154
155      % save the first element of the control at step t where we
              generate a
156      % new trajectory. ==>> will be the optimal control of this MPC
              model
157      umpc(:, h) = u(:, 1);
158      % find the next state by calculating it with the initial control
              input
159      % and current state
160      [x_, ~, ~] = fnStateDroneSafe(traj(:, h), u(:, 1));
161      x_ = x_.*dt;
162
163      % update the next state
164      traj(:, h + 1) = x_ + traj(:, h);
165
166      % update & shift nominal state/control by 1
167      % this's aka receding horizon
168      % At the next state (first elements of the updated state/control)
169      % , the model will re-optimize control
170      ubar(:, 1:end-1) = u(:, 2:end);
171      xbar(:,1) = traj(:, h + 1);
172
173      % cost display
174      cost_(h) = cost;
175      fprintf("%d DDP-MPC cost: %.4f with lr = %.4f \n", h, cost_(h),
              lr);
176  end
```

```matlab
177  %%% MPC trajectory
178  mpc_traj = traj;
179  figure(101)
180  plot3(mpc_traj(1, :), mpc_traj(2, :), mpc_traj(3, :), '--')
181  hold on
182  plot3(x0(1), x0(2), x0(3), 'rx')
183  plot3(xd(1), xd(2), xd(3), 'bx')
184  hold off
185  %%% MPC animation
186  xd = x_des(1:3, end);
187  obs_info = [xc1, yc1, zc1, rc1; xc2, yc2, zc2, rc2; xc3, yc3, zc3,
         rc3];
188  [h,MMmpc] = quadrotor_visualize_w_obstacles(mpc_traj, u, 0:dt:t-dt,
         obs_info, xd);
189  %%%
190  figure(2)
191  movie(MMmpc,1,length(mpc_traj(1,:))/t/2)
192
193  %%% PERFORMANCE
194  time = 0:dt:t-dt;
195  horizon = pHorizon;
196
197  figure(11);
198  subplot(4,3,1)
199  hold on
200  plot(time, mpc_traj(1,:), 'linewidth',1);
201  plot(time, x_des(1,end)*ones(1,horizon), 'red', 'linewidth',4)
202  title('x [m]', 'fontsize',10);
203  xlabel('Time in sec', 'fontsize',10)
204  %hold off;
205  grid on
206
207  subplot(4,3,2)
208  hold on;
209  plot(time, mpc_traj(2,:), 'linewidth',1);
210  plot(time, x_des(2,end)*ones(1,horizon), 'red', 'linewidth',4)
211  title('y [m/s]', 'fontsize',10);
212  xlabel('Time in sec', 'fontsize',10)
213  grid on
214  %hold off;
215
216  subplot(4,3,3)
217  hold on
218  plot(time, mpc_traj(3,:), 'linewidth',1);
219  plot(time, x_des(3,end)*ones(1,horizon), 'red', 'linewidth',4)
220  title('z [m]', 'fontsize',10)
221  xlabel('Time in sec', 'fontsize',10)
```

```matlab
222  grid on
223  %hold off;$\theta$
224
225  subplot(4,3,4)
226  hold on
227  plot(time,mpc_traj(4,:),'linewidth',1);
228  plot(time,x_des(4,end)*ones(1,horizon),'red','linewidth',4)
229  title('x_{dot} [m/s]','fontsize',10)
230  xlabel('Time in sec','fontsize',10)
231  grid on
232
233  subplot(4,3,5)
234  hold on
235  plot(time,mpc_traj(5,:),'linewidth',1);
236  plot(time,x_des(5,end)*ones(1,horizon),'red','linewidth',4)
237  title('y_{dot} [m/s]','fontsize',10)
238  xlabel('Time in sec','fontsize',10)
239  grid on
240  %hold off
241
242  subplot(4,3,6)
243  hold on
244  plot(time,mpc_traj(6,:),'linewidth',1);
245  plot(time,x_des(6,end)*ones(1,horizon),'red','linewidth',4)
246  title('z_{dot} [m/s]','fontsize',10)
247  xlabel('Time in sec','fontsize',10)
248  grid on
249
250  subplot(4,3,7)
251  hold on
252  plot(time,mpc_traj(7,:),'linewidth',1);
253  plot(time,x_des(7,end)*ones(1,horizon),'red','linewidth',4)
254  title('roll [rad]','fontsize',10)
255  xlabel('Time in sec','fontsize',10)
256  grid on
257
258  subplot(4,3,8)
259  hold on
260  plot(time,mpc_traj(8,:),'linewidth',1);
261  plot(time,x_des(8,end)*ones(1,horizon),'red','linewidth',4)
262  title('pitch [rad]','fontsize',10)
263  xlabel('Time in sec','fontsize',10)
264  grid on
265
266  subplot(4,3,9)
267  hold on
268  plot(time,mpc_traj(9,:),'linewidth',1);
```

```matlab
269  plot (time, x_des (9, end)*ones (1, horizon), 'red', 'linewidth',4)
270  title ('yaw [rad]', 'fontsize',10)
271  xlabel ('Time in sec', 'fontsize',10)
272  grid on
273
274  subplot (4,3,10)
275  hold on
276  plot (time, mpc_traj (10,:), 'linewidth',1);
277  plot (time, x_des (10, end)*ones (1, horizon), 'red', 'linewidth',4)
278  title ('p [rad/s]', 'fontsize',10)
279  xlabel ('Time in sec', 'fontsize',10)
280  grid on
281
282  subplot (4,3,11)
283  hold on
284  plot (time, mpc_traj (11,:), 'linewidth',1);
285  plot (time, x_des (11, end)*ones (1, horizon), 'red', 'linewidth',4)
286  title ('q [rad/s]', 'fontsize',10)
287  xlabel ('Time in sec', 'fontsize',10)
288  grid on
289
290  subplot (4,3,12)
291  hold on
292  plot (time, mpc_traj (12,:), 'linewidth',1);
293  plot (time, x_des (12, end)*ones (1, horizon), 'red', 'linewidth',4)
294  title ('r [rad/s]', 'fontsize',10)
295  xlabel ('Time in sec', 'fontsize',10)
296  grid on
297  %%
298  figure (13)
299  subplot (2, 2, 1)
300  hold on
301  plot (time, umpc (1,:), 'linewidth',1);
302  title ('thrust u1 [N]', 'fontsize',10)
303  xlabel ('Time in sec', 'fontsize',10)
304  grid on
305
306  subplot (2, 2, 2)
307  hold on
308  plot (time, umpc (2,:), 'linewidth',1);
309  title ('thrust u2 [N]', 'fontsize',10)
310  xlabel ('Time in sec', 'fontsize',10)
311  grid on
312
313  subplot (2, 2, 3)
314  hold on
315  plot (time, umpc (3,:), 'linewidth',1);
```

```matlab
316    title('thrust u3 [N]','fontsize',10)
317    xlabel('Time in sec','fontsize',10)
318    grid on
319
320    subplot(2, 2, 4)
321    hold on
322    plot(time,umpc(4,:),'linewidth',1);
323    title('thrust u4 [N]','fontsize',10)
324    xlabel('Time in sec','fontsize',10)
325    grid on
326
327    %%
328    figure(14)
329    hold on
330    plot(cost_,'linewidth',1);
331    xlabel('Iterations','fontsize',10)
332    title('Cost','fontsize',10);
333    grid on
334    %save('DDP_Data');
335
336
337
338    %% functions
339    function r = R1(rad)
340        r = [1 0 0; 0 cos(rad) sin(rad); 0 -sin(rad) cos(rad)];
341    end
342    function r = R2(rad)
343        r = [cos(rad) 0 -sin(rad); 0 1 0; sin(rad) 0 cos(rad)];
344    end
345    function r = R3(rad)
346        r = [cos(rad) sin(rad) 0; -sin(rad) cos(rad) 0; 0 0 1];
347    end
```

```matlab
1    clc; clear; close all;
2    %% INITIAL CONFIGURATION
3    global m; % drone mass
4    global Ixx; % MOI in x axis
5    global Iyy; % MOI in y axis
6    global Izz; % MOI in z axis
7    global kt; % rotor torque constant
8    global l; % rotor moment arm length
9    global g; % gravity constant
10
11   % global p; % body roll rate
12   % global q; % body pitch rate
13   % global r; % body yaw rate
14   % global phi; %roll
```

```matlab
% 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;
```

```matlab
58  pHorizon = t/dt;
59  cHorizon = 100;
60  iter = 5;
61  lr = 0.5;
62  Q = diag([0, 0, 0, 0, 0, 0, 1, 1, 1, .1, .1, .1]);
63  S = diag([10, 10, 10, 1, 1, 1, 1, 1, 1, 1, 1, 1].*1);
64  R = diag([1, 1, 1, 1].*0.01);
65
66  ubar = zeros(4, cHorizon - 1);
67  ubar(:, 1) = [1, 1, 1, 1].*5;
68  x0 = [-3; -2; -1; 0; 0; 0; 0; 0; 0; 0; 0; 0];
69  xd = [5; 3; 2; 0; 0; 0; 0; 0; 0; 0; 0; 0];
70
71  x_des = zeros(12, cHorizon);
72  x_des(:, 1) = x0;
73  x_des(:, end) = xd;
74  xbar = zeros(12, cHorizon);
75  xbar(:, 1) = x0;
76  for i = 1:cHorizon-1
77      [x_, ~, ~] = fnStateDrone(xbar(:, i), ubar(:, i));
78      xbar(:, i+1) = xbar(:, i) + x_.*dt;
79  end
80
81  umpc = zeros(4, pHorizon);
82  traj = xbar;
83  cost_ = zeros(1, pHorizon);
84  ddp_message = false;
85  safe_mode = false;
86  ddp_cost = false;
87  for h = 1:pHorizon - 1
88      % find optimal control u at horizon h or step t
89      [xbar, u, cost] = fnDDPdrone(xbar, ubar, x_des, cHorizon, iter,
            lr, Q, S, R, ddp_message, safe_mode, ddp_cost); %(horizon + 1
            - h)*dt,
90
91      % save the first element of the control at step t where we
            generate a
92      % new trajectory. ==>> will be the optimal control of this MPC
            model
93      umpc(:, h) = u(:, 1);
94      % find the next state by calculating it with the initial control
            input
95      % and current state
96      [x_, ~, ~] = fnStateDrone(traj(:, h), u(:, 1));
97      x_ = x_.*dt;
98
99      % update the next state
```

```matlab
100          traj(:, h + 1) = x_ + traj(:, h);
101
102          % update & shift nominal state/control by 1
103          % this's aka receding horizon
104          % At the next state (first elements of the updated state/control)
105          %, the model will re-optimize control
106          ubar(:, 1:end-1) = u(:, 2:end);
107 %        ubar(:, end) = 0;
108 %        xbar(:, 1:end-1) = xbar(:, 2:end);
109          xbar(:,1) = traj(:, h + 1);
110
111          % cost display
112          cost_(h) = cost;
113          fprintf("%d DDP-MPC cost: %.4f with lr = %.4f \n", h, cost_(h),
                   lr);
114 end
115 %% MPC trajectory
116 mpc_traj = traj;
117 figure(101)
118 plot3(mpc_traj(1, :), mpc_traj(2, :), mpc_traj(3, :), '—')
119 hold on
120 plot3(x0(1), x0(2), x0(3), 'rx')
121 plot3(xd(1), xd(2), xd(3), 'bx')
122 hold off
123 %% MPC animation
124 xd = x_des(1:3, end);
125 [h,MMmpc] = quadrotor_visualize(mpc_traj,u, 0:dt:t-dt, xd)
126 %%
127 figure(2)
128 movie(MMmpc,1,length(mpc_traj(1,:))/t/2)
129
130 %% PERFORMANCE
131 time = 0:dt:t-dt;
132 horizon = pHorizon;
133
134 figure(11);
135 subplot(4,3,1)
136 hold on
137 plot(time, mpc_traj(1,:),'linewidth',1);
138 plot(time, x_des(1,end)*ones(1,horizon),'red','linewidth',4)
139 title('x [m]','fontsize',10);
140 xlabel('Time in sec','fontsize',10)
141 %hold off;
142 grid on
143
144 subplot(4,3,2)
145 hold on;
```

```matlab
146  plot(time,mpc_traj(2,:),'linewidth',1);
147  plot(time,x_des(2,end)*ones(1,horizon),'red','linewidth',4)
148  title('y [m/s]','fontsize',10);
149  xlabel('Time in sec','fontsize',10)
150  grid on
151  %hold off;
152
153  subplot(4,3,3)
154  hold on
155  plot(time,mpc_traj(3,:),'linewidth',1);
156  plot(time,x_des(3,end)*ones(1,horizon),'red','linewidth',4)
157  title('z [m]','fontsize',10)
158  xlabel('Time in sec','fontsize',10)
159  grid on
160  %hold off;$\theta$
161
162  subplot(4,3,4)
163  hold on
164  plot(time,mpc_traj(4,:),'linewidth',1);
165  plot(time,x_des(4,end)*ones(1,horizon),'red','linewidth',4)
166  title('x_{dot} [m/s]','fontsize',10)
167  xlabel('Time in sec','fontsize',10)
168  grid on
169
170  subplot(4,3,5)
171  hold on
172  plot(time,mpc_traj(5,:),'linewidth',1);
173  plot(time,x_des(5,end)*ones(1,horizon),'red','linewidth',4)
174  title('y_{dot} [m/s]','fontsize',10)
175  xlabel('Time in sec','fontsize',10)
176  grid on
177  %hold off
178
179  subplot(4,3,6)
180  hold on
181  plot(time,mpc_traj(6,:),'linewidth',1);
182  plot(time,x_des(6,end)*ones(1,horizon),'red','linewidth',4)
183  title('z_{dot} [m/s]','fontsize',10)
184  xlabel('Time in sec','fontsize',10)
185  grid on
186
187  subplot(4,3,7)
188  hold on
189  plot(time,mpc_traj(7,:),'linewidth',1);
190  plot(time,x_des(7,end)*ones(1,horizon),'red','linewidth',4)
191  title('roll [rad]','fontsize',10)
192  xlabel('Time in sec','fontsize',10)
```

```matlab
193  grid on
194
195  subplot(4,3,8)
196  hold on
197  plot(time, mpc_traj(8,:),'linewidth',1);
198  plot(time, x_des(8,end)*ones(1,horizon),'red','linewidth',4)
199  title('pitch [rad]','fontsize',10)
200  xlabel('Time in sec','fontsize',10)
201  grid on
202
203  subplot(4,3,9)
204  hold on
205  plot(time, mpc_traj(9,:),'linewidth',1);
206  plot(time, x_des(9,end)*ones(1,horizon),'red','linewidth',4)
207  title('yaw [rad]','fontsize',10)
208  xlabel('Time in sec','fontsize',10)
209  grid on
210
211  subplot(4,3,10)
212  hold on
213  plot(time, mpc_traj(10,:),'linewidth',1);
214  plot(time, x_des(10,end)*ones(1,horizon),'red','linewidth',4)
215  title('p [rad/s]','fontsize',10)
216  xlabel('Time in sec','fontsize',10)
217  grid on
218
219  subplot(4,3,11)
220  hold on
221  plot(time, mpc_traj(11,:),'linewidth',1);
222  plot(time, x_des(11,end)*ones(1,horizon),'red','linewidth',4)
223  title('q [rad/s]','fontsize',10)
224  xlabel('Time in sec','fontsize',10)
225  grid on
226
227  subplot(4,3,12)
228  hold on
229  plot(time, mpc_traj(12,:),'linewidth',1);
230  plot(time, x_des(12,end)*ones(1,horizon),'red','linewidth',4)
231  title('r [rad/s]','fontsize',10)
232  xlabel('Time in sec','fontsize',10)
233  grid on
234  %%
235  figure(13)
236  subplot(2, 2, 1)
237  hold on
238  plot(time, umpc(1,:),'linewidth',1);
239  title('thrust u1 [N]','fontsize',10)
```

24

```matlab
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
```

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

```matlab
5    global Iyy; % MOI in y axis
6    global Izz; % MOI in z axis
7    global kt; % rotor torque constant
8    global l; % rotor moment arm length
9    global g; % gravity constant
10
11   % global p; % body roll rate
12   % global q; % body pitch rate
13   % global r; % body yaw rate
14   % global phi; %roll
15   % global theta; %pitch
16   % global psi; %yaw
17
18   m = 0.5;
19   Ixx = 0.0032;
20   Iyy = 0.0032;
21   Izz = 0.0055;
22   kt = 0.01691;
23   l = 0.17;
24   g = 9.81;
25
26   %%% MODEL DEFINITION
27   % rotation matrix R123 = R1*R2*R3 = R.transpose;
28   syms x y z xdot ydot zdot roll pitch yaw p q r u1 u2 u3 u4
29   R = R1(roll)*R2(pitch)*R3(yaw);
30   R = [cos(pitch)*cos(yaw), cos(pitch)*sin(yaw), -sin(pitch);
31        sin(pitch)*sin(yaw) - cos(roll)*sin(yaw), sin(pitch)*sin(roll)*
             sin(yaw) + cos(roll)*cos(yaw), sin(roll)*cos(pitch);
32        sin(roll)*sin(yaw) + cos(roll)*sin(pitch)*cos(yaw), sin(pitch)*
             sin(yaw)*cos(roll) - sin(roll)*sin(yaw), cos(pitch)*cos(roll)
             ];
33
34   % translational motion
35   % f1 = [xddot; yddot; zddot]
36   f1 = 1/m.*R*[0; 0; u1+u2+u3+u4] + [0; 0; -m*g];
37   % f2 = [rolldot, pitchdot, yawdot] http://www.stengel.mycpanel.
        princeton.edu/Quaternions.pdf
38   R_f2 = [1, tan(pitch)*sin(roll), tan(pitch)*cos(roll);
39        0, cos(roll), -sin(roll);
40        0, sin(roll)/cos(pitch), cos(roll)/cos(pitch)];
41   f2 = R_f2*[p; q; r];
42   % rotational motion
43   % f3 = [pdot; qdot; rdot]
44   MOI = [Ixx; Iyy; Izz];
45   f3 = 1./MOI.*[sqrt(2)/2*(u1 + u3 - u2 - u4)*l - (Izz - Iyy)*q*r;
46        sqrt(2)/2*(u3 + u4 - u1 - u2)*l + (Izz - Ixx)*p*r;
47        kt*(u1 + u4 - u2 - u3)];
```

```matlab
48
49  dyn = [xdot; ydot; zdot; f1; f2; f3];
50  x = [x; y; z; xdot; ydot; zdot; roll; pitch; yaw; p; q; r];
51  u = [u1; u2; u3; u4];
52  A = jacobian(dyn, x)
53  B = jacobian(dyn, u)
54
55  %%% DDP
56  % Q = diag([0, 0, 0, 0, 0, 0, 1, 1, 1, .1, .1, .1]);
57  % S = diag([10, 10, 10, 1, 1, 1, 1, 1, 1, 1, 1, 1].*1);
58  % %S = diag([1, 1, 1, 10, 10, 10, 1, 1, 1, 10, 10, 10].*1);
59  % R = diag([1, 1, 1, 1].*0.01);
60  % %lr = 0.3:-0.01:0.2;
61  % lr = 0.6
62  Q = diag([0, 0, 0, 0, 0, 0, 1, 1, 1, .1, .1, .1]);
63  S = diag([10, 10, 10, 1, 1, 1, 1, 1, 1, 1, 1, 1].*1);
64  R = diag([1, 1, 1, 1].*0.01);
65  lr = 0.5;
66  t = 8;
67  dt = 0.01;
68  cHorizon = t/dt;
69  iter = 40;
70
71  ubar = zeros(4, cHorizon - 1);
72  ubar(:, 1) = [1, 1, 1, 1].*5;
73  x0 = [-3; -2; -1; 0; 0; 0; 0; 0; 0; 0; 0; 0];
74  xbar = zeros(12, cHorizon);
75  xbar(:, 1) = x0;
76  for i = 1:cHorizon-1
77      [x_, ~, ~] = fnStateDrone(xbar(:, i), ubar(:, i));
78      xbar(:, i+1) = xbar(:, i) + x_.*dt;
79  end
80  x_des = zeros(12, cHorizon);
81  x_des(:, 1) = x0;
82  xd = [5; 3; 2; 0; 0; 0; 0; 0; 0; 0; 0; 0];
83  x_des(:, end) = xd;
84  %%% DDP
85  ddp_message = true;
86  safe_mode = false;
87  ddp_cost = true;
88  [x_traj, u, cost] = fnDDPdrone(xbar, ubar, x_des, cHorizon, iter, lr,
        Q, S, R, ddp_message, safe_mode, ddp_cost);
89  %%%
90  figure(100)
91  plot3(x_traj(1, :), x_traj(2, :), x_traj(3, :), '—')
92  hold on
93  plot3(x0(1), x0(2), x0(3), 'rx')
```

```matlab
94  plot3(xd(1), xd(2), xd(3), 'bx')
95  hold off
96  %%
97  xd = x_des(1:3, end);
98  [h,MMddp] = quadrotor_visualize(x_traj,u, 0:dt:t-dt, xd)
99  %%
100 figure(2)
101 movie(MMddp,1,length(x_traj(1,:))/t/2)
102
103 %% PERFORMANCE
104 time = 0:dt:t-dt;
105 horizon = cHorizon;
106 figure(11);
107 subplot(4,3,1)
108 hold on
109 plot(time, x_traj(1,:),'linewidth',1);
110 plot(time, x_des(1,end)*ones(1,horizon),'red','linewidth',4)
111 title('x [m]','fontsize',10);
112 xlabel('Time in sec','fontsize',10)
113 %hold off;
114 grid on
115
116 subplot(4,3,2)
117 hold on;
118 plot(time, x_traj(2,:),'linewidth',1);
119 plot(time, x_des(2,end)*ones(1,horizon),'red','linewidth',4)
120 title('y [m/s]','fontsize',10);
121 xlabel('Time in sec','fontsize',10)
122 grid on
123 %hold off;
124
125 subplot(4,3,3)
126 hold on
127 plot(time, x_traj(3,:),'linewidth',1);
128 plot(time, x_des(3,end)*ones(1,horizon),'red','linewidth',4)
129 title('z [m]','fontsize',10)
130 xlabel('Time in sec','fontsize',10)
131 grid on
132 %hold off;$\theta$
133
134 subplot(4,3,4)
135 hold on
136 plot(time, x_traj(4,:),'linewidth',1);
137 plot(time, x_des(4,end)*ones(1,horizon),'red','linewidth',4)
138 title('x_{dot} [m/s]','fontsize',10)
139 xlabel('Time in sec','fontsize',10)
140 grid on
```

```matlab
141
142 subplot(4,3,5)
143 hold on
144 plot(time,x_traj(5,:),'linewidth',1);
145 plot(time,x_des(5,end)*ones(1,horizon),'red','linewidth',4)
146 title('y_{dot} [m/s]','fontsize',10)
147 xlabel('Time in sec','fontsize',10)
148 grid on
149 %hold off
150
151 subplot(4,3,6)
152 hold on
153 plot(time,x_traj(6,:),'linewidth',1);
154 plot(time,x_des(6,end)*ones(1,horizon),'red','linewidth',4)
155 title('z_{dot} [m/s]','fontsize',10)
156 xlabel('Time in sec','fontsize',10)
157 grid on
158
159 subplot(4,3,7)
160 hold on
161 plot(time,x_traj(7,:),'linewidth',1);
162 plot(time,x_des(7,end)*ones(1,horizon),'red','linewidth',4)
163 title('roll [rad]','fontsize',10)
164 xlabel('Time in sec','fontsize',10)
165 grid on
166
167 subplot(4,3,8)
168 hold on
169 plot(time,x_traj(8,:),'linewidth',1);
170 plot(time,x_des(8,end)*ones(1,horizon),'red','linewidth',4)
171 title('pitch [rad]','fontsize',10)
172 xlabel('Time in sec','fontsize',10)
173 grid on
174
175 subplot(4,3,9)
176 hold on
177 plot(time,x_traj(9,:),'linewidth',1);
178 plot(time,x_des(9,end)*ones(1,horizon),'red','linewidth',4)
179 title('yaw [rad]','fontsize',10)
180 xlabel('Time in sec','fontsize',10)
181 grid on
182
183 subplot(4,3,10)
184 hold on
185 plot(time,x_traj(10,:),'linewidth',1);
186 plot(time,x_des(10,end)*ones(1,horizon),'red','linewidth',4)
187 title('p [rad/s]','fontsize',10)
```

```matlab
188  xlabel('Time in sec','fontsize',10)
189  grid on
190
191  subplot(4,3,11)
192  hold on
193  plot(time,x_traj(11,:),'linewidth',1);
194  plot(time,x_des(11,end)*ones(1,horizon),'red','linewidth',4)
195  title('q [rad/s]','fontsize',10)
196  xlabel('Time in sec','fontsize',10)
197  grid on
198
199  subplot(4,3,12)
200  hold on
201  plot(time,x_traj(12,:),'linewidth',1);
202  plot(time,x_des(12,end)*ones(1,horizon),'red','linewidth',4)
203  title('r [rad/s]','fontsize',10)
204  xlabel('Time in sec','fontsize',10)
205  grid on
206  %%
207  figure(13)
208  subplot(2, 2, 1)
209  hold on
210  plot(time(1:end-1),u(1,:),'linewidth',1);
211  title('thrust u1 [N]','fontsize',10)
212  xlabel('Time in sec','fontsize',10)
213  grid on
214
215  subplot(2, 2, 2)
216  hold on
217  plot(time(1:end-1),u(2,:),'linewidth',1);
218  title('thrust u2 [N]','fontsize',10)
219  xlabel('Time in sec','fontsize',10)
220  grid on
221
222  subplot(2, 2, 3)
223  hold on
224  plot(time(1:end-1),u(3,:),'linewidth',1);
225  title('thrust u3 [N]','fontsize',10)
226  xlabel('Time in sec','fontsize',10)
227  grid on
228
229  subplot(2, 2, 4)
230  hold on
231  plot(time(1:end-1),u(4,:),'linewidth',1);
232  title('thrust u4 [N]','fontsize',10)
233  xlabel('Time in sec','fontsize',10)
234  grid on
```

```matlab
235
236  %%
237  figure(14)
238  hold on
239  plot(cost,'linewidth',1);
240  xlabel('Iterations','fontsize',10)
241  title('Cost','fontsize',10);
242  grid on
243  %save('DDP_Data');
244
245
246  %% functions
247  function r = R1(rad)
248      r = [1 0 0; 0 cos(rad) sin(rad); 0 -sin(rad) cos(rad)];
249  end
250  function r = R2(rad)
251      r = [cos(rad) 0 -sin(rad); 0 1 0; sin(rad) 0 cos(rad)];
252  end
253  function r = R3(rad)
254      r = [cos(rad) sin(rad) 0; -sin(rad) cos(rad) 0; 0 0 1];
255  end
```

```matlab
1   function [x_traj, u, cost] = fnDDPdrone(xbar, ubar, x_des, h2, iter,
        gamma, Q, S, R, ddp_message, safe_mode, ddp_cost)
2
3       %% initial configuration
4       dt = 0.01;
5       horizon = h2;
6       u = ubar;
7       x_traj = xbar;
8       cost = [];
9       %%
10      for p = gamma
11          lr = p; % learning rate
12          for k = 1:iter
13              %% Linearization of the dynamics
14              for i = 1:(horizon-1)
15                  % Running cost L(x, u)
16                  [l0,l_x,l_xx,l_u,l_uu,l_ux] = fnCost(x_traj(:,i) -
                        x_des(:, i), u(:,i), Q, R, dt);
17                  L0(i) = l0;
18                  Lx(:, i) = l_x;
19                  Lxx(:, :, i) = l_xx;
20                  Lu(:, i) = l_u;
21                  Luu(:, :, i) = l_uu;
22                  Lux(:, :, i) = l_ux; % M = N^T
23
```

```matlab
24              if safe_mode
25                  [~, dfx, dfu] = fnStateDroneSafe(x_traj(:, i), u
                        (:, i));
26              else
27                  [~, dfx, dfu] = fnStateDrone(x_traj(:, i), u(:, i
                        ));
28              end
29
30              % linearized form in discrete time
31              A(:, :, i) = eye(size(xbar, 1)) + dfx*dt; % Phi
32              B(:, :, i) = dfu*dt; % B of linearized equation in
                    the classnote.
33          end
34
35          %————Search the control —————%
36          Vxx(:,:, horizon)= S;
37          Vx(:, horizon) = S * (x_traj(:, horizon) - x_des(:,
                horizon));
38          V(horizon) = 0.5*(x_traj(:, horizon) - x_des(:, horizon))
                '*S*(x_traj(:, horizon) - x_des(:, horizon));
39
40          %———— Backpropagation of the Value Function —————(
                Chapter 1/ p.7)—%
41          for j = (horizon-1):-1:1
42              Qu(:, j) = Lu(:, j) + B(:, :, j)'*Vx(:, j+1);
43              Qx(:, j) = Lx(:, j) + A(:, :, j)'*Vx(:, j+1); %%
                    different Vx
44              Quu(:, :, j) = B(:, :, j)'*Vxx(:, :, j+1)*B(:, :, j)
                    + Luu(:, :, j);
45              % Quu becomes dead in the middle. Its value ~0 or
                    negative, which means
46              % it will never update the propagation / no feedback.
47              % For example, imagine the zero or negative slope in
                    the optimization graph.
48              lambda = abs(min(eig(Quu(:, :, j)))) + rand;
49              if any(eig(Quu(:, :, j)) < 0.0001)
50                  Quu(:, :, j) = Quu(:, :, j) + lambda *eye(length(
                        Quu(:, :, j)));
51              end
52              Qxx(:, :, j) = A(:, :, j)'*Vxx(:, :, j+1)*A(:, :, j)
                    + Lxx(:, :, j);
53              Qux(:, :, j) = B(:, :, j)'*Vxx(:, :, j+1)*A(:, :, j)
                    + Lux(:, :, j)';
54
55              %feedback
56              Kfb(:, :, j) = -Quu(:, :, j)\Qux(:, :, j);
57              %feedforward
```

```matlab
                    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
```

33

```matlab
                    , 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
```

34

```matlab
141 %              plot(time(1:end-1), u(:, 1:horizon-1), 'linewidth',1)
142 %              title('optimal control','fontsize',10)
143 %              xlabel('Time in sec','fontsize',10)
144 %              grid on;
145 %              %hold off
146 %
147 %
148 %              subplot(2,4,6);hold on
149 %              plot(Cost,'linewidth',1);
150 %              xlabel('Iterations','fontsize',10)
151 %              title('Cost','fontsize',10);
152 %              grid on
153 %              %save('DDP_Data');
154     end
155 end
```

```matlab
1  function [x_, dfx, dfu] = fnStateDroneSafe(xhat, u)
2
3  x = xhat(1);
4  y = xhat(2);
5  z = xhat(3);
6  xdot = xhat(4);
7  ydot = xhat(5);
8  zdot = xhat(6);
9  roll = xhat(7);
10 pitch = xhat(8);
11 yaw = xhat(9);
12 p = xhat(10);
13 q = xhat(11);
14 r = xhat(12);
15 z1 = xhat(13);
16 z2 = xhat(14);
17 z3 = xhat(15);
18 u1 = u(1);
19 u2 = u(2);
20 u3 = u(3);
21 u4 = u(4);
22
23 dfx = [0,0,0,1,0,0,0,0,0,0,0,0,0,0,0;
24     0,0,0,0,1,0,0,0,0,0,0,0,0,0,0;
25     0,0,0,0,0,1,0,0,0,0,0,0,0,0,0;
26     0,0,0,0,0,0,0,-2*cos(pitch)*(u1 + u2 + u3 + u4), 0,0,0,0,0,0,0;
27     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;
28     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;
29     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;
30        0 ,0 ,0 ,0 ,0 ,0 ,−  r*cos ( roll ) − q*sin ( roll ), 0, 0,0, cos ( roll ), −sin (
          roll ),   0,   0,   0;
31        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;
32        0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,0 ,−(23* r )/32 ,−(23* q )/32 ,   0,   0,   0;
33        0 ,0 ,0 ,0 ,0 ,0 ,0 ,0,   0,  (23* r )/32 ,0 ,(23* p )/32 ,   0,   0,   0;
34        0 ,0 ,0 ,0 ,0 ,0 ,0 ,0,   0 ,0 ,0 ,0,   0,   0,   0;
35        (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;
36        (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];

```matlab
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;
70         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;
71         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];
72  end
```

```
1   function [x_, dfx, dfu] = fnStateDrone(xhat, u)
2
3   x = xhat(1);
4   y = xhat(2);
5   z = xhat(3);
6   xdot = xhat(4);
7   ydot = xhat(5);
8   zdot = xhat(6);
9   roll = xhat(7);
10  pitch = xhat(8);
11  yaw = xhat(9);
12  p = xhat(10);
13  q = xhat(11);
14  r = xhat(12);
15  u1 = u(1);
16  u2 = u(2);
17  u3 = u(3);
18  u4 = u(4);
19
20  dfx = [0, 0, 0, 1, 0, 0,
        0,

        0, 0,          0,                        0,                        0;
21  0, 0, 0, 0, 1, 0,
        0,

        0, 0,          0,                        0,                        0;
22  0, 0, 0, 0, 0, 1,
        0,

        0, 0,          0,                        0,                        0;
23  0, 0, 0, 0, 0, 0,
        0,                                       -2*cos(pitch)*(u1 +
        u2 + u3 + u4), 0,          0,                        0,
                        0;
```

```
24  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;
25  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;
26  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);
27  0, 0, 0, 0, 0, 0,                                 - r*cos(roll) - q*sin(roll)
       ,

       0, 0,              0,                   cos(roll),                 -sin(roll);
28  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);
29  0, 0, 0, 0, 0, 0,
       0,

       0, 0,              0,                   -(23*r)/32,                -(23*q)/32;
30  0, 0, 0, 0, 0, 0,
       0,

       0, 0, (23*r)/32,                              0,                  (23*p)/32;
31  0, 0, 0, 0, 0, 0,
       0,

       0, 0,              0,                        0,                           0];
32
33  dfu = [                          0,                              0,
                              0,                      0;
34                            0,                              0,
                              0,                      0;
35                            0,                              0,
                              0,                      0;
36          -2*sin(pitch),               -2*sin(pitch),             -2*sin(pitch
                    ),            -2*sin(pitch);
37  2*cos(pitch)*sin(roll), 2*cos(pitch)*sin(roll), 2*cos(pitch)*sin(roll
       ), 2*cos(pitch)*sin(roll);
38  2*cos(pitch)*cos(roll), 2*cos(pitch)*cos(roll), 2*cos(pitch)*cos(roll
       ), 2*cos(pitch)*cos(roll);
39                            0,                              0,
                              0,                      0;
```

```matlab
40                            0,                              0,
                               0,                              0;
41                            0,                              0,
                               0,                              0;
42        (425*2^(1/2))/16,        -(425*2^(1/2))/16,        (425*2^(1/2))
              /16,          -(425*2^(1/2))/16;
43      -(425*2^(1/2))/16,        -(425*2^(1/2))/16,        (425*2^(1/2))
              /16,          (425*2^(1/2))/16;
44              1691/550,                     -1691/550,
                  -1691/550,                     1691/550];

45
46  x_ = [xdot;...
47       ydot;...
48       zdot;...
49       -2*sin(pitch)*(u1 + u2 + u3 + u4);...
50       2*cos(pitch)*sin(roll)*(u1 + u2 + u3 + u4);...
51       2*cos(roll)*cos(pitch)*(u1 + u2 + u3 + u4) - 981/200;...
52       p + r*cos(roll)*tan(pitch) + q*sin(roll)*tan(pitch);...
53       q*cos(roll) - r*sin(roll);...
54       (r*cos(roll))/cos(pitch) + (q*sin(roll))/cos(pitch);...
55       (425*2^(1/2)*(conj(u1) - conj(u2) + conj(u3) - conj(u4)))/16 -
              (23*conj(q)*conj(r))/32;...
56       (23*conj(p)*conj(r))/32 - (425*2^(1/2)*(conj(u1) + conj(u2) -
              conj(u3) - conj(u4)))/16;...
57       (1691*conj(u1))/550 - (1691*conj(u2))/550 - (1691*conj(u3))/550 +
              (1691*conj(u4))/550];
58  end
```

```matlab
1   function  [l0,l_x,l_xx,l_u,l_uu,l_ux] = fnCost(x, u, Q, R, dt)
2
3   l0 = 0.5.*(u' *R *u + x'*Q*x) .* dt;
4   l_x = Q * x .* dt;
5   l_xx = Q .* dt;
6   l_u = R * u .* dt;
7   l_uu = R .* dt;
8   l_ux = zeros(length(Q), length(R)) .* dt;
9   end
```

```matlab
1   function [Cost] =  fnCostComputation(x_traj, u_new, x_des, dt,Q,S,R)
2
3       [~, Horizon] = size(x_traj);
4       Cost = 0;
5       for j =1:(Horizon-1)
6           Cost = Cost + (u_new(:,j)' * R * u_new(:,j) + (x_traj(:, j) -
                  x_des(:, j))'*Q*(x_traj(:, j) - x_des(:, j)))*dt;
7       end
8       TerminalCost= (x_traj(:,Horizon) - x_des(:, end))'*S * (x_traj(:,
```

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

    Cost = Cost + TerminalCost;

end
```