Table of Contents

.... 1
STATE-SPACE: 1
LQR BASED COST: 2
SIMULATION: IC1 2
PLOTTING RESULTS: PART (B): IC1: 4


```
% This script runs an DT-LQR based controller for stabilization of a
% linearized (DLTI) approximation of a CNL system about its
 equilibrium
% trajectory.

% Done assuming that WE KNOW ALL OUR STATES!!!!
```

STATE-SPACE:

The given system is linearized about the hovering fixed point/equilibrium. Then a simple discretization process is carried out and the new matrices are denoted by Abar and Bbar.

```
% Simulation Parameters:
t         = 5;          % Length of simulation.
dt        = 0.01;         % Step size.
N         = t/dt + 1;     % Number of steps.
T         = 0:dt:t;       % This is our time series vector.

% Other properties:
% mass m = 1; I = 0.01; T1 = g/2; T2 = g/2;
% We can see that T1 and T2 together balance the drone perfectly if
 there
% are no external disturbances.

Abar = [zeros(3,3),eye(3,3); % Discretized system matrix.
       zeros(1,6);
       9.81,0,0,0,-0.1,0;
       0,0,0,0,0,-0.1];

Abar = eye(6,6) + Abar*dt; % 1st order approximation of Ad.

fprintf('\nDiscretized system matrix Abar:\n');
disp(Abar);
% The determinant of Abar is non-zero. Hence invertible.

fprintf('\nDiscretized input matrix Bbar:\n');
Bbar = [zeros(3,3);
       100,-100,0;
       zeros(1,3);
       1,1,-1];

Bbar = Bbar*dt; % 1st order approximation of Bd.
disp(Bbar);
```

```
Discretized system matrix Abar:
    1.0000         0         0    0.0100         0         0
         0    1.0000         0         0    0.0100         0
         0         0    1.0000         0         0    0.0100
         0         0         0    1.0000         0         0
    0.0981         0         0         0    0.9990         0
         0         0         0         0         0    0.9990


Discretized input matrix Bbar:
         0         0         0
         0         0         0
         0         0         0
    1.0000   -1.0000         0
         0         0         0
    0.0100    0.0100   -0.0100
```

LQR BASED COST:

Here we will describe the LQR associated cost matrices and they will remain constant throught the control time horizon. Meaning our control objective will not change during the simulation.

```
q = (10)*eye(6,6); % Cost associated with state deviation;
r = 0.001*eye(3,3);    % Cost associated with input usage;
                       % Third input is uncontrollable.
r(3,3) = 0;
% Basically, I am choosing to penalize deviation from optimal state
 10^4
% more than input usage.

% Control objective:

p =(10)*eye(6,6);
```

SIMULATION: IC1

```
% State space definition/initialization:
n = 6;             % Number of states.
m = 3;             % Number of inputs.
x = zeros(n,1,N);    % Time-series of states.
u = zeros(m,1,N);    % Time-series of applied input.
A = zeros(n,n,N);    % Time-series of system matrix.
B = zeros(n,m,N);    % Time-series of input matrix.
B62 = zeros(n,m-1,N); % Time-series of truncated input matrix.

A(:,:,1) = Abar; % State-space is constant in time.
% We know that our third input is uncontrollable ACCELERATION DUE TO
 GRAVITY!
% Hence choosing only B6x2 block:
B(:,:,1) = Bbar;
```

```
B62(:,:,1) = Bbar(:,1:m-1); % This will give me the 6x2 block.

x(:,1,1) = [1.5,-0.5,0.5,2,5,3]'; % Let us attempt without considering
 the final
                            % state for now.
                            % Literally dropping my drone from a
 height of
                            % 10 meters.
u(:,1,1) = [0,0,9.81];     % Initial input.

K = zeros(m-1,n,N); % Coefficient of Optimal input. Third state unc!!
 2x6
P = zeros(n,n,N); % Coefficient of Optimal cost.
J = zeros(1,N);   % Our scalar cost at each time.
Q = zeros(n,n,N)*dt; % State-Cost time series.
R = zeros(m,m,N)*dt; % Input-Cost time series.
R22 = zeros(m-1,m-1,N)*dt; % Input-comp-Cost time series. 2x2

% Iteratively find our value for P and K matrix backwards:
for i = N:-1:1
    if i == N
        P(:,:,i) = p; % Cost associated with final state.
        K(:,:,i) = zeros(m-1,n); % Optimal input coefficient at final
 time.
    else
        Q(:,:,i) = q*dt; % Discretized Q and R.
        R(:,:,i) = r*dt;
        R22(:,:,i) = R(1:m-1,1:m-1,i); % Computing recursive block.
        A(:,:,i) = A(:,:,1);
        B(:,:,i) = B(:,:,1);
        B62(:,:,i) = Bbar(:,1:m-1,1);

        K(:,:,i) = (R22(:,:,i) + transpose(B62(:,:,i))*P(:,:,i
+1)*B62(:,:,i))\...
        transpose(B62(:,:,i))*P(:,:,i+1)*A(:,:,i);
        P(:,:,i) = transpose(A(:,:,i) - B62(:,:,i)*K(:,:,i))*P(:,:,i
+1)*(A(:,:,i)...
        - B62(:,:,i)*K(:,:,i)) +
 transpose(K(:,:,i))*R22(:,:,i)*K(:,:,i) + Q(:,:,i);
    end
end

% Start Simulation:
% Here we might not know all of our states. We only kno
for i = 1:N
    if i == 1
        % Initial state is already defined.
        u(1:m-1,1,i) = -K(:,:,i)*x(:,1,i); % Calculating intial
 optimal input.
        % Here we know exactly what my initial states is.
        u(m,1,i)     = 9.81; % Uncontrolled g!
        J(1,i)   = 0.5*transpose(x(:,1,i))*P(:,:,i)*x(:,1,i);
        % I will be zeroing out my gravity if the deviation is smaller
 than
```

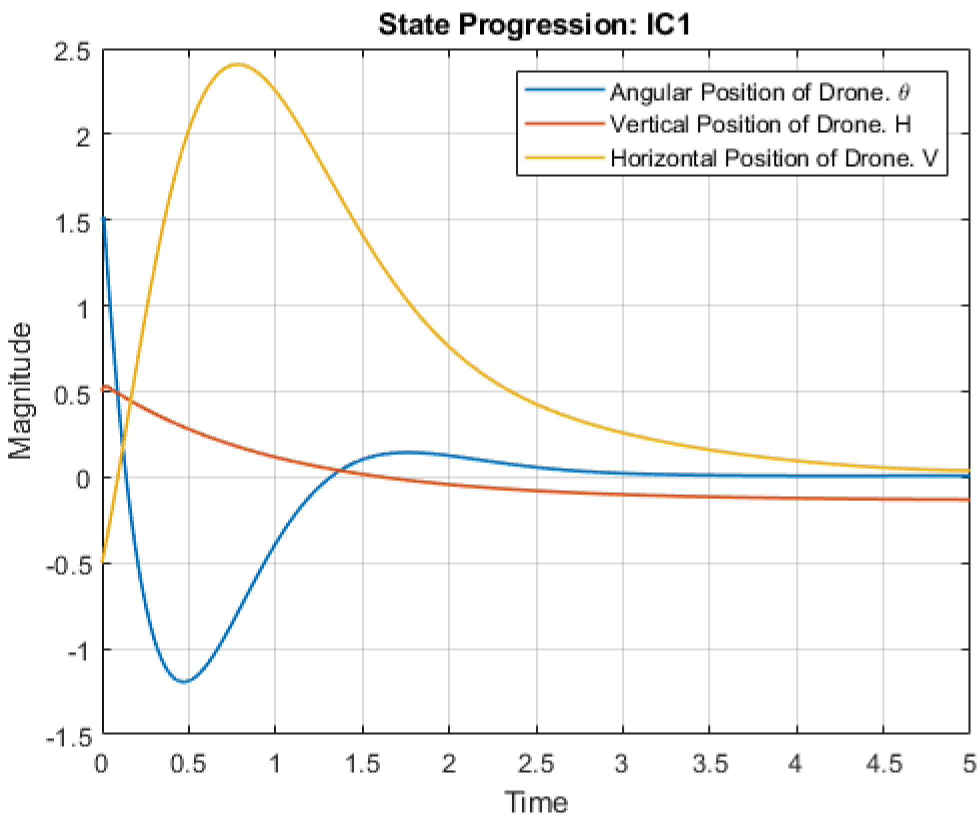
```
        % 1cm from my nominal trajectory. I don't want the
 stabilization
        % inputs in that range.
        if ((x(3,1,i) > -10^(-2)) && (x(3,1,i) < 10^(-2))) &&
 (x(2,1,i) > -10^(-2)) && (x(2,1,i) < 10^(-2))
            u(m,1,i) = 0;
        end
    else
        x(:,1,i) = A(:,:,i-1)*x(:,1,i-1) + B(:,:,i-1)*u(:,1,i-1); %DT
 system.
        u(1:m-1,1,i) = -K(:,:,i)*x(:,1,i);
        u(m,1,i)     = 9.81; % Uncontrolled g!
        if ((x(3,1,i) > -10^(-2)) && (x(3,1,i) < 10^(-2))) &&
 (x(2,1,i) > -10^(-2)) && (x(2,1,i) < 10^(-2))
            u(m,1,i) = 0;
        end
        J(1,i)   = 0.5*transpose(x(:,1,i))*P(:,:,i)*x(:,1,i);
    end
end
```

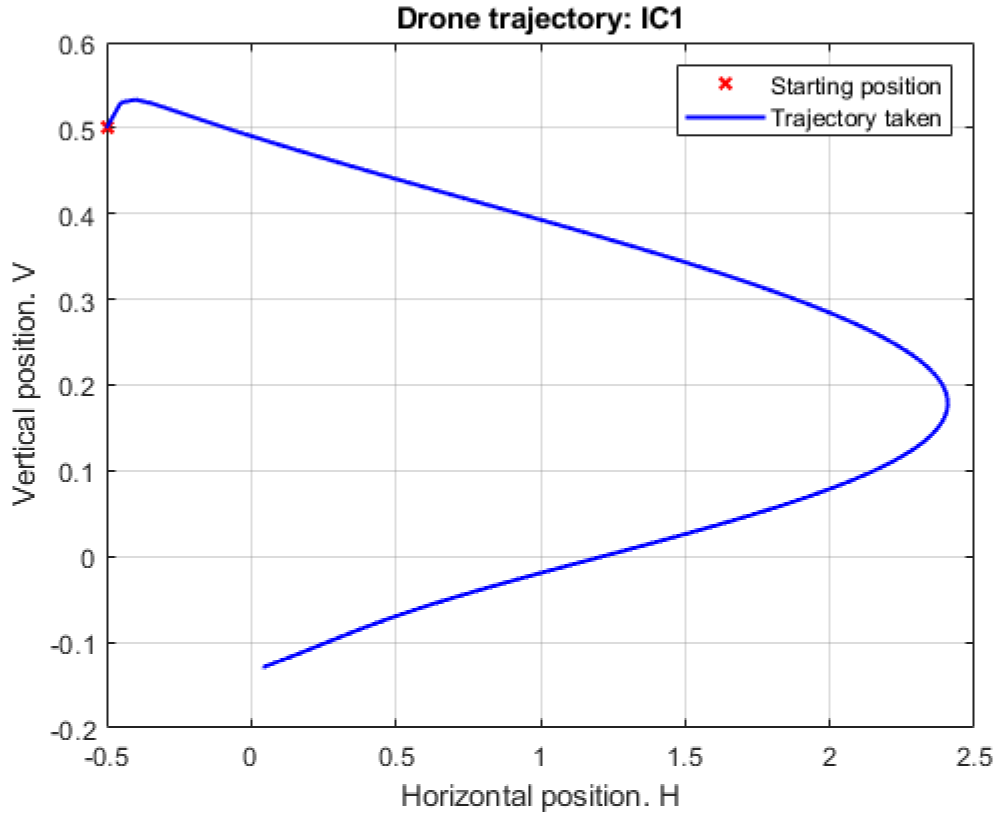
PLOTTING RESULTS: PART (B): IC1:

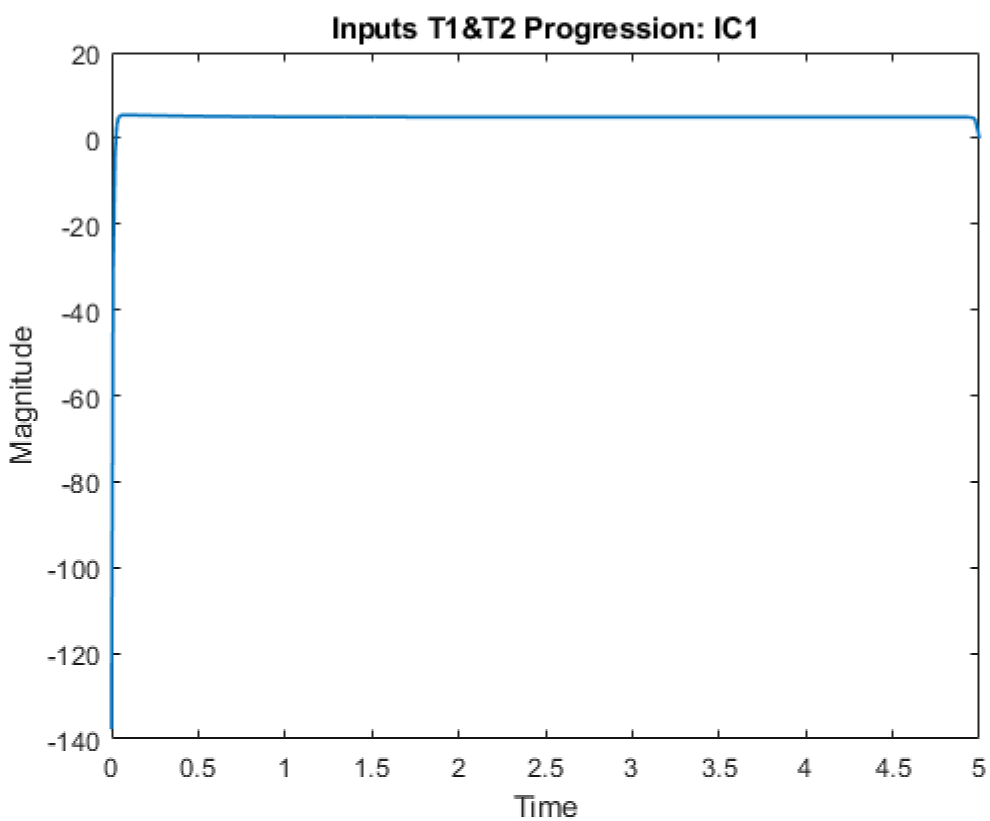
Now let us plot the results we just obtained.

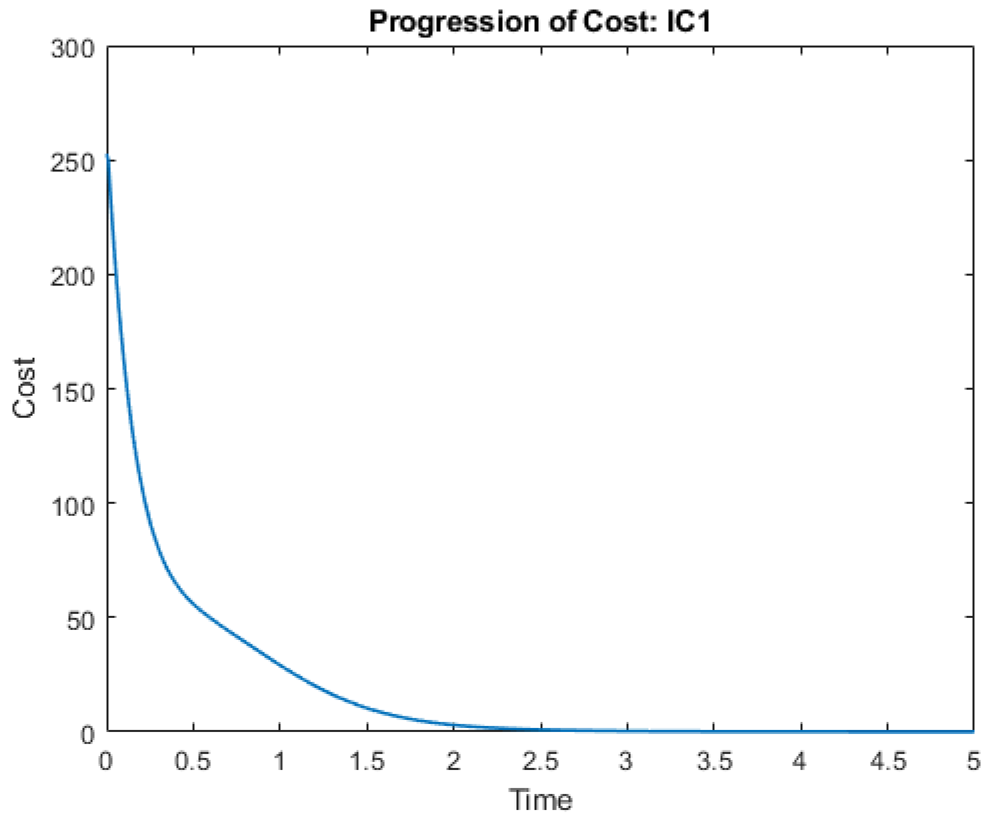
```
theta = zeros(1,N);
H = zeros(1,N);
V = zeros(1,N);
T1= zeros(1,N); % Input 1 - Thrust 1.
T2= zeros(1,N); % Input 2 - Thrust 2.

for j = 1:N
    theta(1,j) = x(1,1,j); % Angular position fo drone.
    H(1,j) = x(2,1,j); % Horizontal position of drone.
    V(1,j) = x(3,1,j); % Vertical position of drone.
    T1(1,j)= u(1,1,j); % Total drone Left thrust.
    T2(1,j)= u(2,1,j); % Total drone Right thrust.
end

% Progression of states:
figure()
plot(T,theta,'LineWidth',1.2);
hold on; grid on;
plot(T,V,'LineWidth',1.2);
plot(T,H,'LineWidth',1.2);
xlabel('Time');
ylabel('Magnitude');
title('State Progression: IC1');
legend('Angular Position of Drone. \theta','Vertical Position of
 Drone. H','Horizontal Position of Drone. V');

% Trajectory taken by the drone:
figure()
plot(H(1),V(1),'rx','LineWidth',1.5);
```

```
hold on; grid on;
plot(H,V,'b','LineWidth',1.5);
xlabel('Horizontal position. H');
ylabel('Vertical position. V');
title('Drone trajectory: IC1');
legend('Starting position','Trajectory taken');

% Progression of inputs:
figure()
plot(T,T1,'LineWidth',1.2);
xlabel('Time');
ylabel('Magnitude');
title('Inputs T1&T2 Progression: IC1');

% Progression of Cost:
figure()
plot(T,J,'LineWidth',1.2);
xlabel('Time');
ylabel('Cost');
title('Progression of Cost: IC1');
```




Drone trajectory: IC1
Starting position
Trajectory taken
Vertical position. V
Horizontal position. H
0.6
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.5
0
0.5
1
1.5
2
2.5


Inputs T1&T2 Progression: IC1
Magnitude
Time
20
0
-20
-40
-60
-80
-100
-120
-140
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5



Published with MATLAB® R2018b