Table of Contents


```
% This script runs a steepest descent based controller for
 stabilization of
% a linearized (DLTI) approximation of a CNL system about its
 equilibrium
% trajectory.

% Let us first start off by doing an LQR for two time steps:
```

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          = 0.02;          % 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); % Terminal cost.
```

SIMULATION: DT-LQR:

```
% 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) = [-20,-20,10,-10,-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); % State-Cost time series.
R = zeros(m,m,N); % Input-Cost time series.
R22 = zeros(m-1,m-1,N); % 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.
```

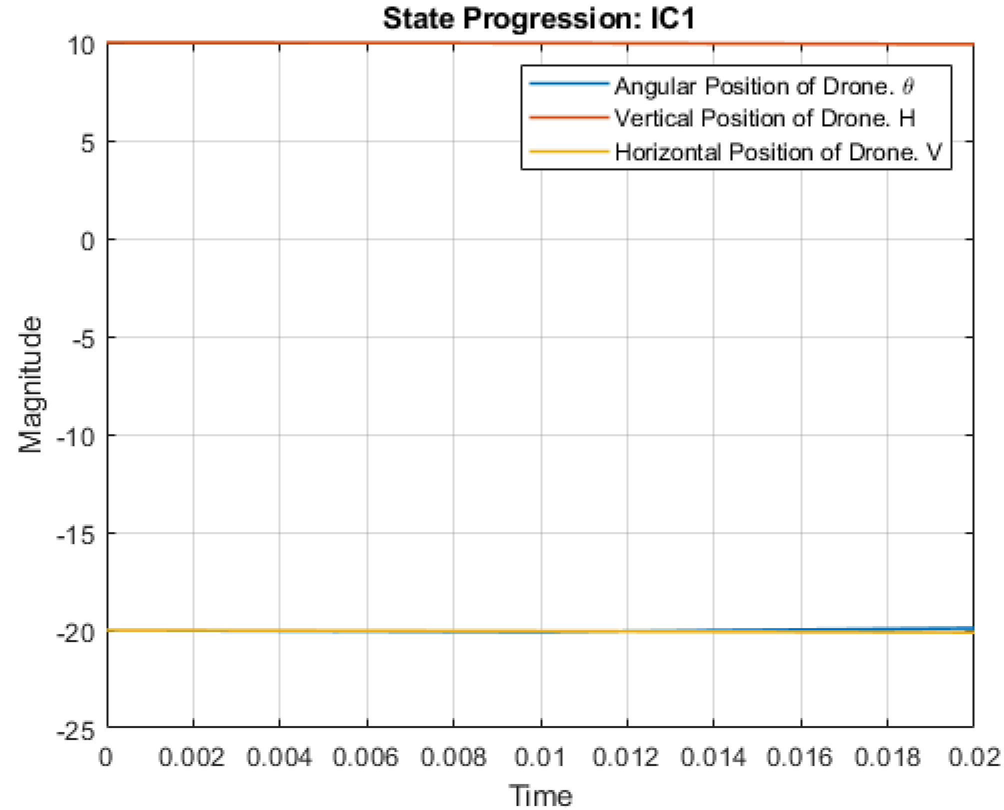
```
        u(1:m-1,1,i) = -K(:,:,i)*x(:,1,i); % Calculating intial
 optimal input.
        % Here we know exactly what my initial states is.
        u(m,1,i)     = 9.81; % Uncontrolled g!
        J(1,i)   = 0.5*transpose(x(:,1,i))*P(:,:,i)*x(:,1,i);
        % I will be zeroing out my gravity if the deviation is smaller
 than
        % 1cm from my nominal trajectory. We 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
```

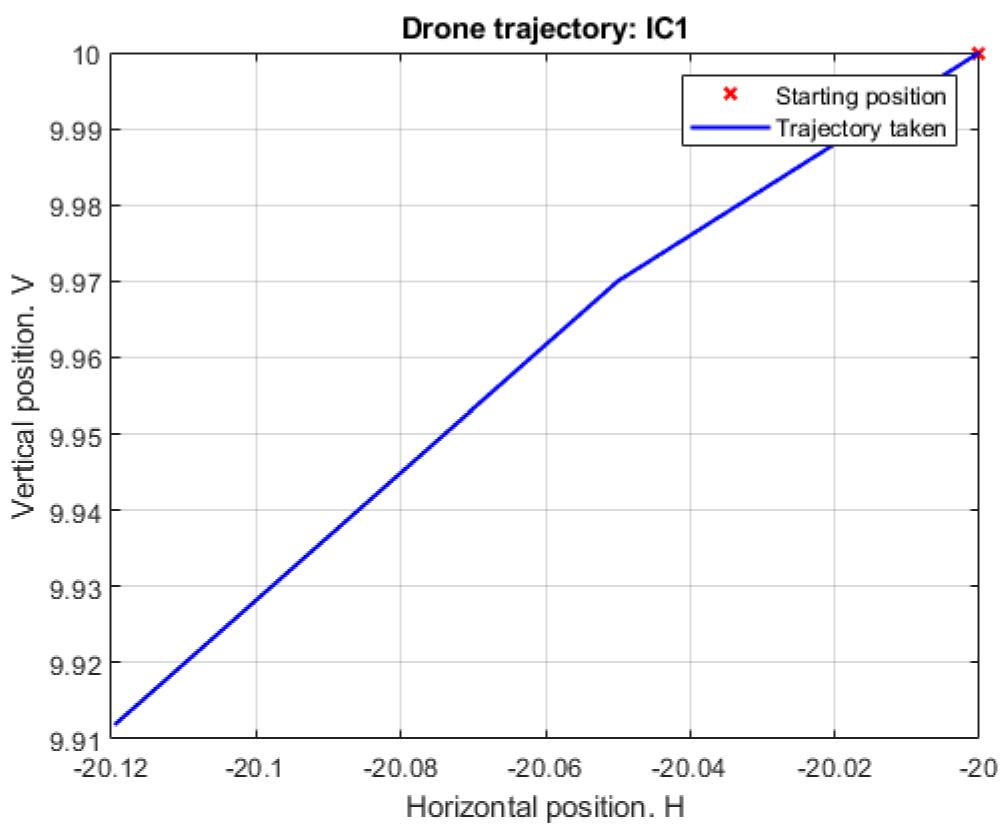
PLOTTING RESULTS: PART (B): IC1:

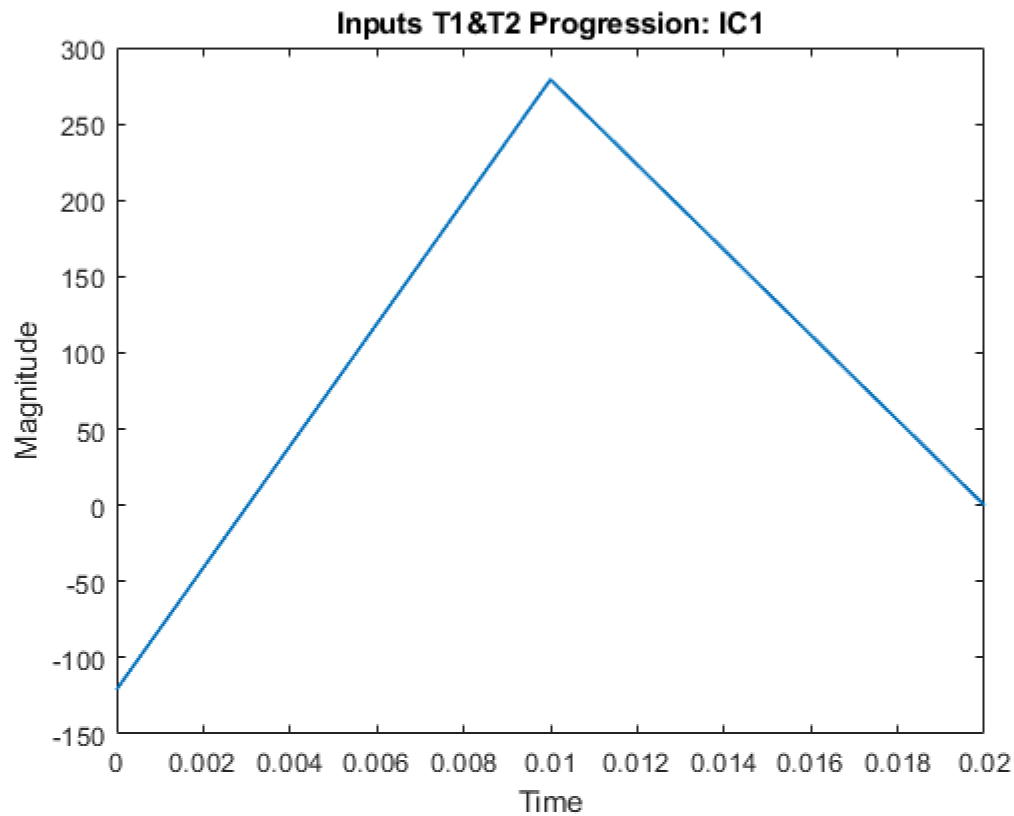
Now let us plot the results we just obtained.

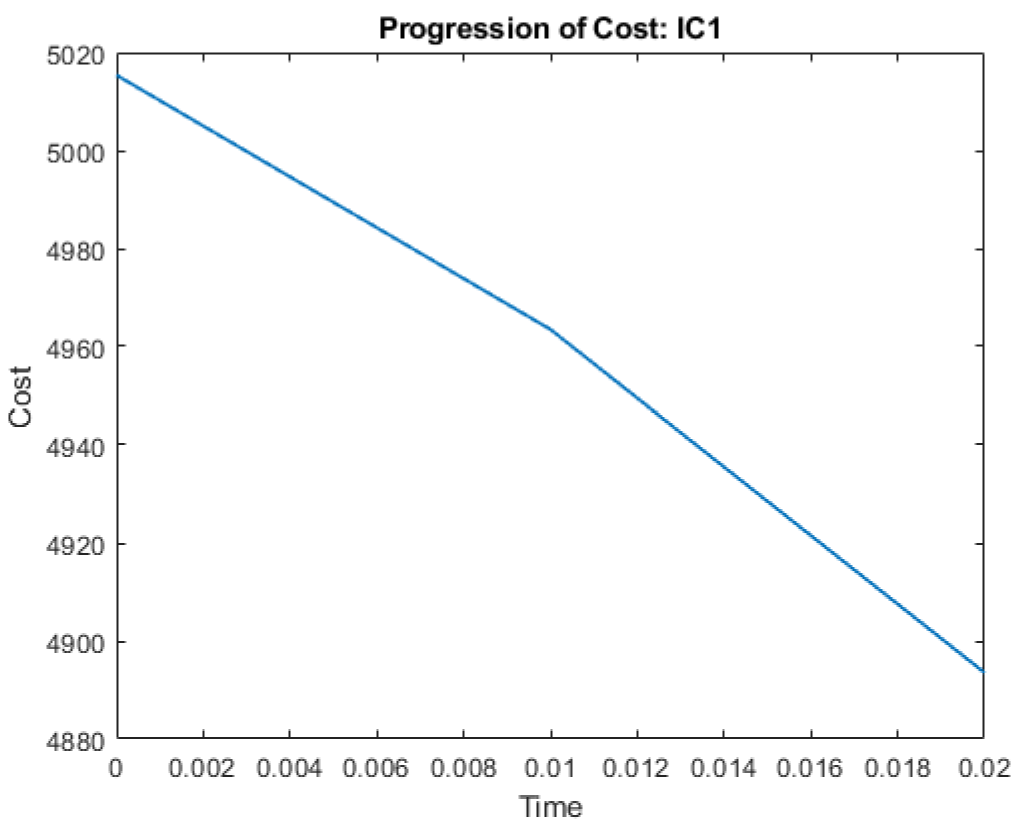
```
theta = zeros(1,N);
H = zeros(1,N);
V = zeros(1,N);
T1= zeros(1,N); % Input 1 - Thrust 1.
T2= zeros(1,N); % Input 2 - Thrust 2.

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');
```


State Progression: IC1
Angular Position of Drone. θ
Vertical Position of Drone. H
Horizontal Position of Drone. V
Magnitude
Time
10
5
0
-5
-10
-15
-20
-25
0
0.002
0.004
0.006
0.008
0.01
0.012
0.014
0.016
0.018
0.02


Drone trajectory: IC1
Starting position
Trajectory taken
Vertical position. V
Horizontal position. H
10
9.99
9.98
9.97
9.96
9.95
9.94
9.93
9.92
9.91
-20.12
-20.1
-20.08
-20.06
-20.04
-20.02
-20


Inputs T1&T2 Progression: IC1
Magnitude
Time
300
250
200
150
100
50
0
-50
-100
-150
0
0.002
0.004
0.006
0.008
0.01
0.012
0.014
0.016
0.018
0.02


Progression of Cost: IC1
Cost
Time
5020
5000
4980
4960
4940
4920
4900
4880
0
0.002
0.004
0.006
0.008
0.01
0.012
0.014
0.016
0.018
0.02

STEEPEST DESCENT:

```
% State space definition/initialization:
n = 6;            % Number of states.
m = 3;            % Number of inputs.
x2 = zeros(n,1,N);   % Time-series of states.
l  = zeros(1,m,N);   % Co-state/Lagrange multiplier.
J2  = zeros(1,N);     % Re-defining our cost.

x2(:,1,1) = [-20,-20,10,-10,-5,-3]'; % Matching the initial condition
 with
                            % the IC of previous method.
                            % Let us attempt without considering the
 final
                            % state for now.
                            % Literally dropping my drone from a
 height of
                            % 10 meters.

u2 = rand(m,1,N);   % Time-series of randomly applied inputs.
u2(m,1,:) = 9.81;   % Uncontrolled input.
% The system and input matrices are the same as last case.
A = Abar;
B = Bbar;

% Cost definitions used from previous LQR formulation.

% Input perturbation magnitude:
a = 0.0001;

% Descent iteration variable:
iter = 1;
titer = 5; % Total descent iterations.

% Start simulation:
for i = 1:N
    if i == 1
        iter = 1; % Resetting the descent counter.
        while iter < titer
            DuJ = Jacobiu(x2,u2,i,N,Q,R,J2,P,A,B);
            u2(:,1,i) = u2(:,1,i) - a*transpose(DuJ);
            iter = iter + 1;
        end
        iter = 1; % Resetting the descent counter.
        % Checking if our drone's V and H position is close to zero:
        if x2(3,1,i) < 10^-2 && x2(3,1,i) > -1*(10^-2) && x2(2,1,i) <
 10^-2 && x2(2,1,i) > -1*(10^-2)
            u2(m,1,i) = 0;
        else
            u2(m,1,i) = 9.81;
        end
        J2(1,i) = 0.5*transpose(x2(:,1,i))*Q(:,:,i)*x2(:,1,i)...
            + 0.5*transpose(u2(:,1,i))*R(:,:,i)*u2(:,1,i);
```

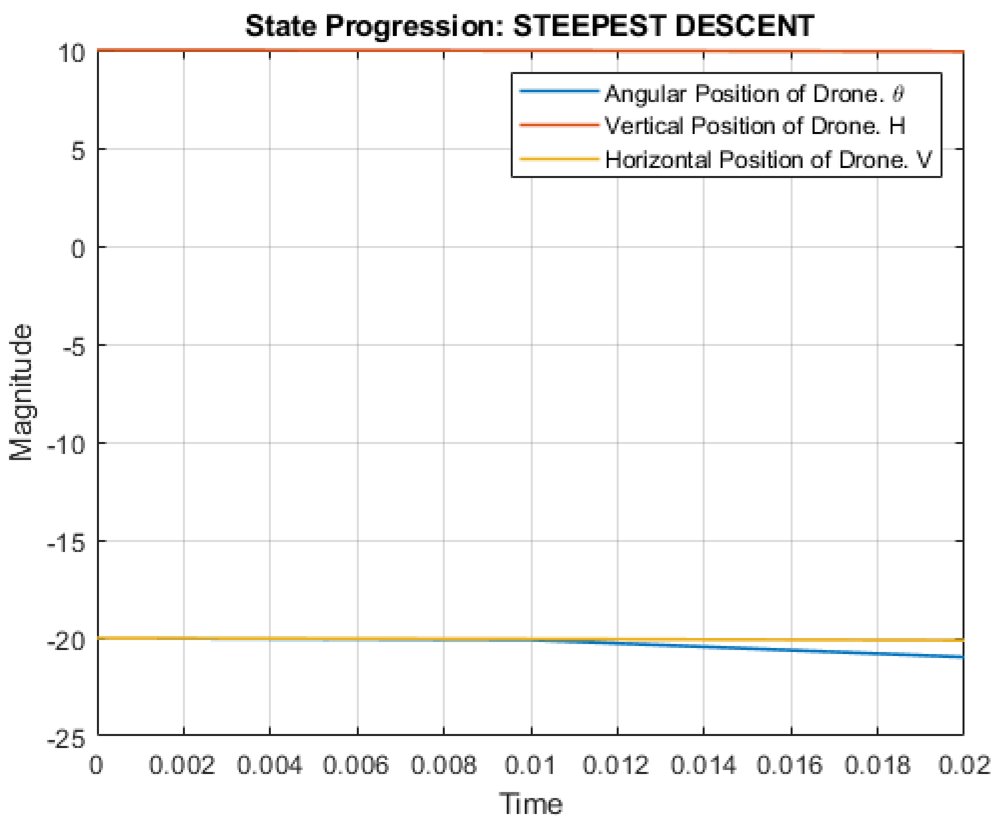
```
    elseif i == N
        x2(:,1,i) = A*x2(:,1,i-1) + B*u2(:,1,i-1);
        u2(:,1,i) = 0;
        if x2(3,1,i) < 10^-2 && x2(3,1,i) > -1*(10^-2) && x2(2,1,i) <
 10^-2 && x2(2,1,i) > -1*(10^-2)
            u2(m,1,i) = 0;
        else
            u2(m,1,i) = 9.81;
        end
        J2(1,i) = 0.5*transpose(x2(:,1,i))*P(:,:,i)*x2(:,1,i) +
 sum(J2);
    else
        x2(:,1,i) = A*x2(:,1,i-1) + B*u2(:,1,i-1);
        % Iteratively finding the optimal input:
        while iter < titer
            DuJ = Jacobiu(x2,u2,i,N,Q,R,J2,P,A,B);
            u2(:,1,i) = u2(:,1,i) - a*transpose(DuJ);
            iter = iter + 1;
        end
        iter = 1; % Resetting the descent counter.
        if x2(3,1,i) < 10^-16 && x2(3,1,i) > -1*(10^-16)
            u2(m,1,i) = 0;
        else
            u2(m,1,i) = 9.81;
        end
        J2(1,i) = 0.5*transpose(x2(:,1,i))*Q(:,:,i)*x2(:,1,i)...
            + 0.5*transpose(u2(:,1,i))*R(:,:,i)*u2(:,1,i) + sum(J2);
    end
end
```

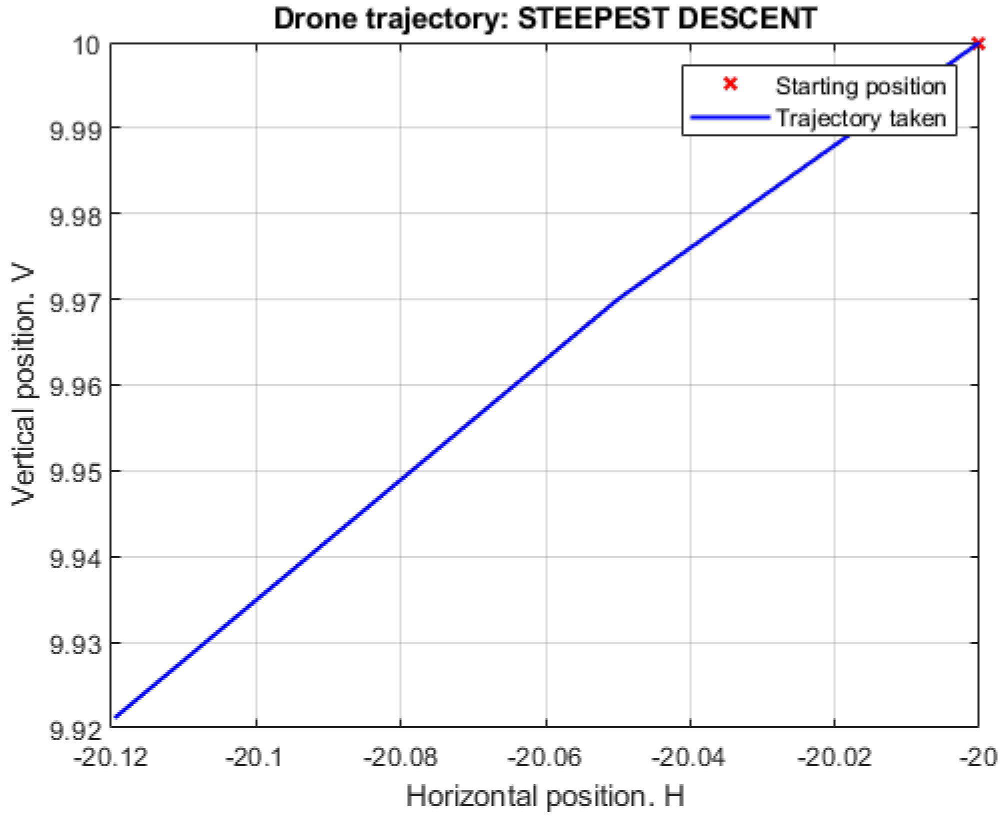
PLOTTING RESULTS: STEEPEST DESCENT:

Now let us plot the results we just obtained.

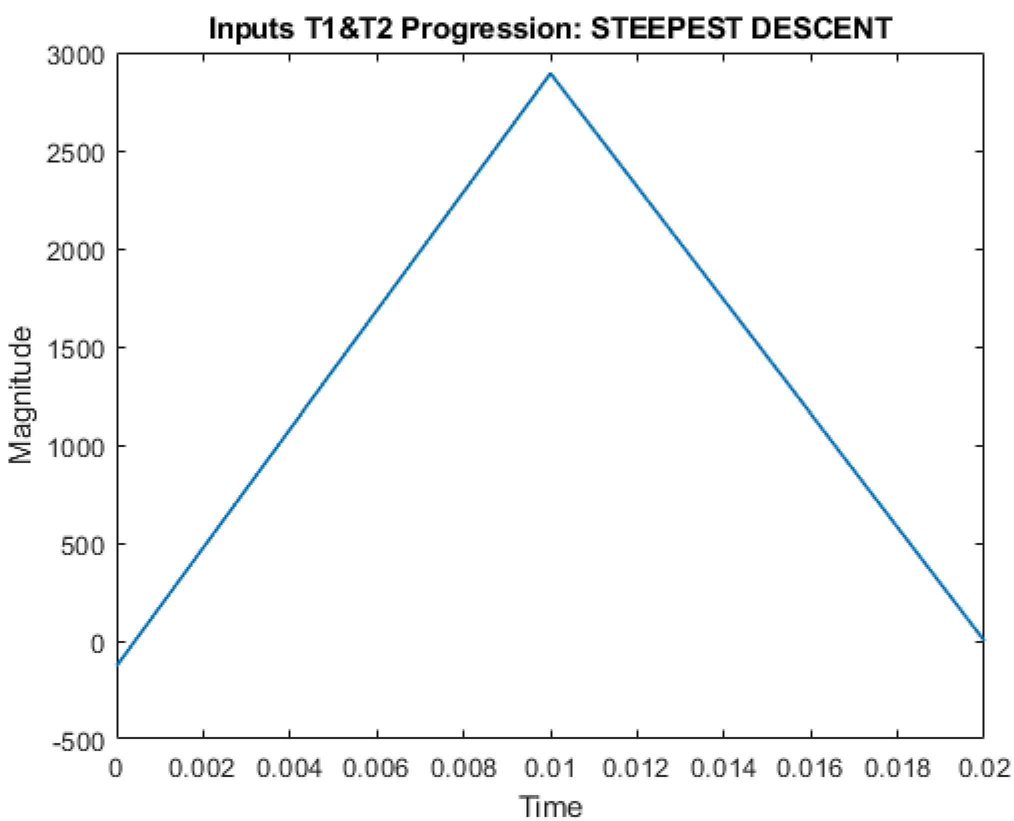
```
theta1 = zeros(1,N);
H1 = zeros(1,N);
V1 = zeros(1,N);
T1= zeros(1,N); % Input 1 - Thrust 1.
T2= zeros(1,N); % Input 2 - Thrust 2.

for j = 1:N
    theta1(1,j) = x2(1,1,j); % Angular position of drone.
    H1(1,j) = x2(2,1,j); % Horizontal position of drone.
    V1(1,j) = x2(3,1,j); % Vertical position of drone.
    T1(1,j)= u2(1,1,j); % Total drone Left thrust.
    T2(1,j)= u2(2,1,j); % Total drone Right thrust.
end

% Progression of states:
figure()
plot(T,theta1,'LineWidth',1.2);
hold on; grid on;
plot(T,V1,'LineWidth',1.2);
```

```
plot(T,H1,'LineWidth',1.2);
xlabel('Time');
ylabel('Magnitude');
title('State Progression: STEEPEST DESCENT');
legend('Angular Position of Drone. \theta','Vertical Position of
 Drone. H','Horizontal Position of Drone. V');
% THE ANGULAR POSITION IS APPROXIMATELY 0. HENCE NOT VISIBLE IN THE
 PLOT.

% Trajectory taken by the drone:
figure()
plot(H1(1),V1(1),'rx','LineWidth',1.5);
hold on; grid on;
plot(H1,V1,'b','LineWidth',1.5);
xlabel('Horizontal position. H');
ylabel('Vertical position. V');
title('Drone trajectory: STEEPEST DESCENT');
legend('Starting position','Trajectory taken');

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





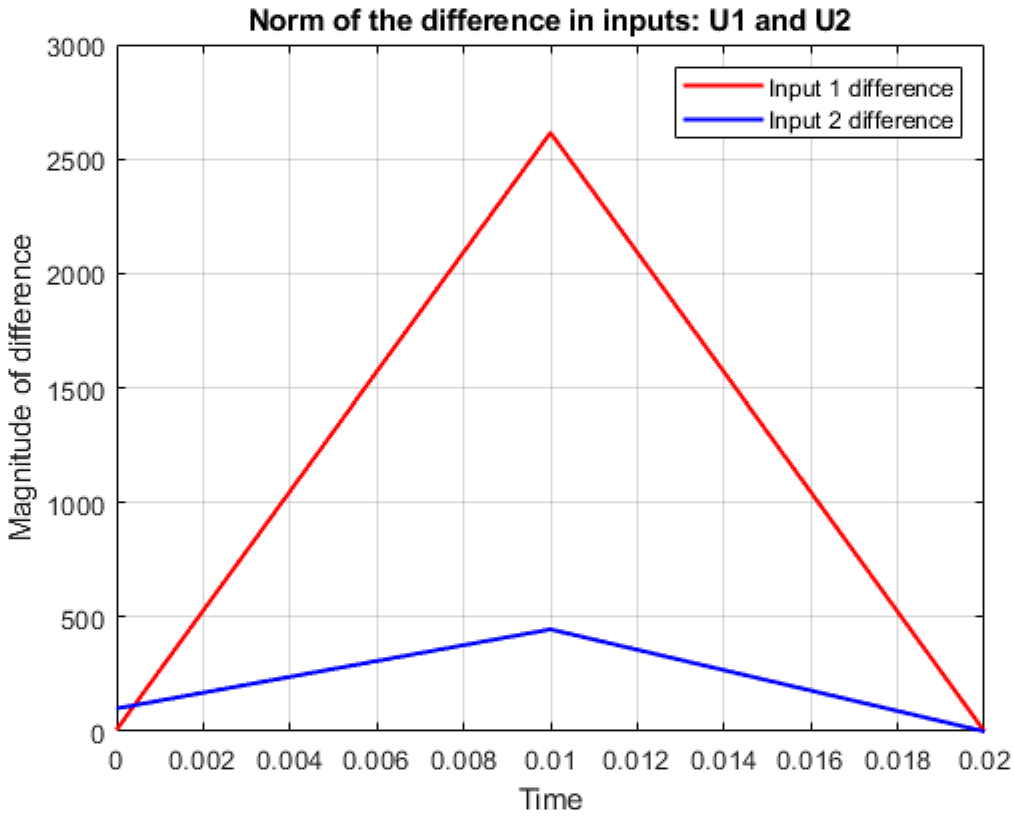
Drone trajectory: STEEPEST DESCENT
Starting position
Trajectory taken
Vertical position. V
Horizontal position. H
10
9.99
9.98
9.97
9.96
9.95
9.94
9.93
9.92
-20.12
-20.1
-20.08
-20.06
-20.04
-20.02
-20



Inputs T1&T2 Progression: STEEPEST DESCENT
Magnitude
Time
3000
2500
2000
1500
1000
500
0
-500
0
0.002
0.004
0.006
0.008
0.01
0.012
0.014
0.016
0.018
0.02

Calculating the difference between the inputs: SD:

```
U = zeros(1,N); % To store the norm of the difference between the
 inputs.

for c = 1:2
    for b = 1:N
        U(c,b) = abs(u(c,1,b) - u2(c,1,b));
    end
end

figure()
plot(T,U(1,:),'r','LineWidth',1.5);
grid on; hold on;
plot(T,U(2,:),'b','LineWidth',1.5);
xlabel('Time');
ylabel('Magnitude of difference');
title('Norm of the difference in inputs: U1 and U2');
legend('Input 1 difference','Input 2 difference');
```



EXTERNAL FUNCTION DEFINITION:

We will now compute the Jacobian of cost with respect to the two inputs:

WRONG NEED TO CHANGE!!! %%%

```
function DuJ = Jacobiu(x2,u2,i,N,Q,R,J2,P,A,B)
DuJ = zeros(1,3); % Initializing the Jacobian.
% Changing inputs so that we don't lose data:
ii = i;
x22 = x2;
u22 = u2;
J22 = J2;
% Get our initial final cost for the first set of inputs:
while ii < N+1
    if ii == 1
        J22(1,ii) = 0.5*transpose(x22(:,1,ii))*Q(:,:,i)*x22(:,1,ii)...
            + 0.5*transpose(u22(:,1,ii))*R(:,:,i)*u22(:,1,ii);
    else
        if ii == N
            J22(1,ii)   =
 0.5*transpose(x22(:,1,ii))*P(:,:,N)*x22(:,1,ii) + sum(J22);
            u22(:,1,ii) = 0;
        end
        x22(:,1,ii) = A*x22(:,1,ii-1) + B*u22(:,1,ii-1);
        J22(1,ii) = 0.5*transpose(x22(:,1,ii))*Q(:,:,i)*x22(:,1,ii)...
            + 0.5*transpose(u22(:,1,ii))*R(:,:,i)*u22(:,1,ii) +
 sum(J22);
    end
    ii = ii+1;
end
Jf1 = J22(1,N);
ii = i; % Reset step number.
uf1 = u22(:,1,ii); % Initial input.
% Now let us get our perturbed set of initial cost:
u22(:,1,ii) = u22(:,1,ii) + 0.1*randn(3,1);
u22(3,1,ii) = 9.81;
uf2 = u22(:,1,ii); % Perturbed input.
% Repeat the process:
while ii < N+1
    if ii == 1
        J22(1,ii) = 0.5*transpose(x22(:,1,ii))*Q(:,:,i)*x22(:,1,ii)...
            + 0.5*transpose(u22(:,1,ii))*R(:,:,i)*u22(:,1,ii);
    else
        if ii == N
            J22(1,ii)   =
 0.5*transpose(x22(:,1,ii))*P(:,:,N)*x22(:,1,ii) + sum(J22);
            u22(:,1,ii) = 0;
        end
        x22(:,1,ii) = A*x22(:,1,ii-1) + B*u22(:,1,ii-1);
        J22(1,ii) = 0.5*transpose(x22(:,1,ii))*Q(:,:,i)*x22(:,1,ii)...
            + 0.5*transpose(u22(:,1,ii))*R(:,:,i)*u22(:,1,ii) +
 sum(J22);
    end
    ii = ii+1;
end
Jf2 = J22(1,N);
```

```
% Let us take the gradient of our final cost:
Jf = Jf2 - Jf1;

% Change in input:
uf = uf2 - uf1;

% Overall effect on Final cost:
DuJ(1)= Jf/uf(1);
DuJ(2)= Jf/uf(2);
DuJ(3)= 0; % Uncontrolled input g.
end
```

Published with MATLAB® R2018b