RRT with dynamic constraints

RRT with dynamic constraints (Part A)

0.0/5.0 points (ungraded)

In this problem, we will consider extensions of the RRT algorithm that allow us to handle dynamic constraints. In particular, we will use the swing-up problem for a torque-limited pendulum as our test-bed. The dynamics for the pendulum will be:

$$\ddot{\theta} = u - g\sin(\theta) - b\dot{\theta},$$

where $g = 9.81$, $b = 0.1$ and u **is bounded in the range [-5,5]**.

We have provided some stub code here. The basic structure of the RRT code is very similar. The only differences will be in the way we find the closest vertex in our existing tree (given a new sample) and the way we extend the tree. In particular, the extension operation must satisfy the dynamic constraints of the system. Please take a quick look at the code to make sure you understand the basic structure.

(a) As a first step, we will consider the Euclidean metric to determine the closest vertex to a new sample. One thing that you need to be careful about is that the θ variable (the first coordinate of our state) wraps around. In particular, θ will lie between $-\frac{\pi}{2}$ and $\frac{3\pi}{2}$. Thus, the angles $-\frac{\pi}{2}$ and $\frac{3\pi}{2}$ are in fact the same. Given this representation, what is the Euclidean distance between the states $[-1; 4]$ and $[4; -3]$?

Answer: 7.1166

Explanation

The difference between the angle coordinates (taking wrapping into account) is: $\mathrm{mod}\,(4-(-1)+\pi, 2\pi) - \pi \approx -1.2832$. The difference in the angular velocity coordinate is 7. Hence, the Euclidean distance is approximately 7.1166.

Submit You have used 0 of 2 attempts

Answers are displayed within the problem

RRT with dynamic constraints (Part B)

0.0/15.0 points (ungraded)

(b) Next, we will implement the RRT algorithm using the Euclidean metric as our distance function. In order to do this, you will need to fill in the parts of the code that say "FILL ME IN" by implementing the following:

- Implement the function **closest_vert = closestVertexEuclidean(rrt_verts,xy)** that takes in a $2 \times K$ vector consisting of the current vertices of the RRT, along with a state xy and returns the vertex in the tree that is closest to xy.

- Implement the function **new_vert = extendEuclidean(closest_vert,xy)** that will perform the extend operation in the following manner. Discretize the range of control inputs $[-5, 5]$ (around 20 discrete samples should be sufficient) and choose the input u_0 that will result in the system moving the most in the direction of the sample point xy when started from the state closest_vert (there are many ways to make this choice and we leave this to you to explore). Simulate the system (using ode45 for example) for a small time interval (no more than 0.1 seconds) using this constant control input to obtain new_vert. **Be sure to correctly wrap the angle coordinate. Also make sure to implement the input saturations in your simulation.**

Copy the $K \times 2$ array "rrt_verts_grade" you obtained below. The first row should be the start state and the last row should be (close to) the goal state.

```
rrt_verts_grade = [];
```

Unanswered

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function for finding closest vertex
function closest_vert = closestVertexEuclidean(rrt_verts,xy)

% Compute distances (make sure to wrap the angle coordinate)
th_dists = diff(unwrap([repmat(xy(1),1,size(rrt_verts,2));rrt_verts(1,:)]));
thdot_dists = rrt_verts(2,:) - xy(2);

% Compute distances in a vectorized way
dists = sqrt(th_dists.^2 + thdot_dists.^2);

% Find minimizer
[~,minind] = min(dists);

% Closest vertex
closest_vert = rrt_verts(:,minind);



end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function for extending tree
function xnew = extendEuclidean(closest_vert,xy)

% Find u0 that will move system towards xy
u0s = linspace(-5,5,20);

th_diff = diff(unwrap([closest_vert(1);xy(1)]));
cosangle = -Inf;
for k = 1:length(u0s)
    xdot_u0 = [closest_vert(2); (u0s(k) -9.81*1*sin(closest_vert(1)) - 0.1*closest_vert(2))];
    cosangle_u0 = dot([th_diff;xy(2)-closest_vert(2)],xdot_u0/norm(xdot_u0));
    if cosangle_u0 > cosangle
        cosangle = cosangle_u0;
        u0 = u0s(k);
    end
end

% Simulate policy
[~,Y] = ode45(@(t,x)pendulumDynamics(t,x,[0 0],xy,u0),[0 0.1],closest_vert);

% Extended state
xnew = Y(end,:)';

% Wrap state
```

```
xnew(1) = mod(xnew(1)+pi/2,2*pi) - pi/2;

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Pendulum dynamics
function xdot = pendulumDynamics(t,x,K,x0,u0)

xdot = zeros(2,1);
xdot(1) = x(2);

u = -K*(x-x0)+u0;

if u > 5
    u = 5;
end
if u < -5
    u = -5;
end

g = 9.81;
b = 0.1;
xdot(2) = (u -g*1*sin(x(1)) - b*x(2));

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% rrt_dynamic.m code

%% Code setup (Do not modify, but please read) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Bounds on world
world_bounds_th = [-pi/2,(3/2)*pi];
world_bounds_thdot = [-10,10];

% Start and goal positions
figure(1);
xy_start = [0;0]; plot(xy_start(1),xy_start(2),'bo','MarkerFaceColor','b','MarkerSize',10);
xy_goal = [pi;0]; plot(xy_goal(1),xy_goal(2),'go','MarkerFaceColor','g','MarkerSize',10); drawnc

% Initialize RRT. The RRT will be represented as a 2 x N list of points. So
% each column represents a vertex of the tree.
rrt_verts = zeros(2,1000);
rrt_verts(:,1) = xy_start;
N = 1;
nearGoal = false; % This will be set to true if goal has been reached
```

```
minDistGoal = 0.25; % This is the convergence criterion. We will declare
                    % success when the tree reaches within 0.25 in distance
                    % from the goal. DO NOT MODIFY.

% Keep track of where a node came from in order to reconstruct path
came_from = 1;

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% FILL ME IN %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Choose one of these methods
method = 'euclidean'; % Euclidean distance metric (part b of problem)
% method = 'lqr'; % LQR distance metric (part d of problem)
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


figure(1); hold on;
axis([world_bounds_th, world_bounds_thdot]);
 hxy = plot(0,0,'ro');

 iter = 1;
% RRT algorithm
while ~nearGoal
    iter = iter + 1;

    % Sample point
    rnd = rand(1);
    % With probability 0.05, sample the goal. This promotes movement to the
    % goal.
    if rnd < 0.05
        xy = xy_goal;
    else
        % Sample from space with probability 0.95
        xs = (world_bounds_th(2) - world_bounds_th(1))*rand(1) + world_bounds_th(1);
        ys = (world_bounds_thdot(2) - world_bounds_thdot(1))*rand(1) + world_bounds_thdot(1);
        xy = [xs;ys];
    end

    %% FILL ME IN %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if strcmp(method, 'euclidean')
        closest_vert = closestVertexEuclidean(rrt_verts(:,1:N),xy); % Write this function
    elseif strcmp(method, 'lqr')
        [closest_vert,K,closestInd] = closestVertexLQR(rrt_verts(:,1:N),xy); % Write this functi
        % To know where we must go, we must remember where we came from.
        came_from(iter) = closestInd;
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
    %% FILL ME IN %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if strcmp(method, 'euclidean')
        new_vert = extendEuclidean(closest_vert,xy); % Write this function
    else
        new_vert = extendLQR(closest_vert,xy,K); % Write this function
    end

    delete(hxy);
    figure(1);
    hxy = plot(xy(1),xy(2),'r.');axis([world_bounds_th, world_bounds_thdot]);


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    % Plot extension (Comment the next few lines out if you want your code to
    % run a bit quicker. The plotting is useful for debugging though.)
    figure(1)
    hold on
    plot(new_vert(1),new_vert(2),'bo','MarkerFaceColor','b','MarkerSize',5);
    % Plot line (but only if we are not wrapping to the other side of the
    % plot)
    if abs(closest_vert(1) - new_vert(1)) < 0.75*(2*pi)
        line([closest_vert(1),new_vert(1)],[closest_vert(2),new_vert(2)]);
    end
    axis([world_bounds_th, world_bounds_thdot]);


    %% DO NOT MODIFY THIS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % If it is collision free, add it to tree
    N = N+1;
    if N > size(rrt_verts,2)
        rrt_verts = [rrt_verts zeros(size(rrt_verts))];
    end
    rrt_verts(:,N) = new_vert;

    % Check if we have reached goal
    if norm(xy_goal-new_vert) < minDistGoal
        break;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


end

% Plot vertices in RRT
hold on;
```

```
plot(rrt_verts(1,:),rrt_verts(2,:),'bo','MarkerFaceColor','b','MarkerSize',5);

%% Submit rrt_verts_grade for grading %%%%
format long
rrt_verts_grade = rrt_verts(:,1:N)'
clipboard('copy',rrt_verts_grade);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem

RRT with dynamic constraints (Part C)

0.0/10.0 points (ungraded)

(c) If you run your code from part (b) a few times, you should see that although the RRT usually finds a feasible path to the goal, it can be frustratingly slow. This is in large part due to the fact that the Euclidean distance is not a very good metric for our problem. We will now consider an alternative based on LQR that typically works much better.

Given a nominal state x_0 and control input u_0, we can linearize the dynamics $\dot{x} = f(x, u)$ about this point:

$$\dot{x} \approx f(x_0, u_0) + \frac{\partial f(x_0, u_0)}{\partial x}(x - x_0) + \frac{\partial f(x_0, u_0)}{\partial u}(u - u_0).$$

We can then define a change of variables $\bar{x} = x - x_0$ and $\bar{u} = u - u_0$. Then we have:

$$\dot{x} \approx f(x_0, u_0) + A(x_0, u_0)\,\bar{x} + B(x_0, u_0)\,\bar{u},$$

where A and B are the partial derivatives of $f(x, u)$ with respect to x and u respectively.

Given a quadratic cost function defined by state and action cost matrices Q and R, we can solve a LQR problem ([K,S] = lqr(A,B,Q,R) in Matlab). Then the locally optimal policy is given by $u^\star(\bar{x}) = -K\bar{x} + u_0$. The metric we will use to determine how close a point x is to x_0 is then given by $(x - x_0)\,S\,(x - x_0)$.

For the pendulum, given $x_0 = [\pi - 0.1; 2.0]$ and $u_0 = 0$, what is the "distance" (evaluated by the metric just described) between x_0 and $x = [\pi + 0.2; -5.0]$? (Use $Q = I_{2\times2}$ (the identity matrix) and $R = 0.1$).

Answer: 26.4296

Explanation

The difference between the angle coordinates (taking wrapping into account) is 0.3. The difference in the angular velocity coordinate is -7. The matrices of partial derivatives are given by $A = [0, 1; -9.81\cos(\pi - 0.1), -0.1]$ and $B = [0; 1]$. From lqr(A,B,eye(2),0.1), we get $S \approx [7.3567, 2.0021; 2.0021, 0.6975]$. Thus, the distance is $[0.3, -7] * S * [0.3; -7] \approx 26.4296.$

Submit You have used 0 of 2 attempts

Answers are displayed within the problem

RRT with dynamic constraints (Part D)

0.0/20.0 points (ungraded)

(d) Next, we will use the LQR metric from (c) in our RRT algorithm. In order to do this, you will need to implement the following:

- Implement the function **[closest_vert,K] = closestVertexLQR(rrt_verts,xy)** that takes in a $2 \times M$ vector consisting of the current vertices of the RRT, along with a state xy and returns the vertex in the tree that is closest to xy (as measured by the LQR metric), along with the gain matrix of the LQR policy. **Make sure to handle the wrapping of the angle variable correctly.**

- Implement the function **new_vert = extendLQR(closest_vert,xy,K)** that will perform the extend operation by applying the LQR policy starting from the state closest_vert for a short time interval (no more than 0.1 seconds). Here, you should use $x_0 =$ xy and $u_0 = 0$ to find the LQR policy. You can use ode45 again for the integration. Again, make sure to correctly wrap the angle coordinate. **Also make sure to implement the input saturations in your simulation.**

- Modify your code to output the path from start to goal found by the RRT. This should be a $L \times 2$ array. Given that this array is called "xpath", you can use the code provided here to visualize the path using the PendulumVisualizer in the examples/Pendulum folder in drake (you'll need to be in this folder to run this). Once you are satisfied that the path has been correctly obtained, type in your xpath below. **Make sure that it is a $L \times 2$ array. The first row should be the start state and the last row should be (close to) the goal state. Do not type in rrt_verts_grade.**

```
xpath = [];

```

Unanswered

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function for finding closest vertex using LQR "metric"
function [closest_vert,K,minind] = closestVertexLQR(rrt_verts,xy)

% Linearize p
A = [0 1; -9.81*cos(xy(1)), -0.1];
B = [0; 1];

% LQR
Q = diag([10 1]);
R = 1;

[K,S] = lqr(full(A),full(B),Q,R);

% Compute distances (make sure to wrap the angle coordinate)
th_dists = diff(unwrap([repmat(xy(1),1,size(rrt_verts,2)); rrt_verts(1,:)]));
% th_dists = rrt_verts(1,:) - repmat(xy(1),1,size(rrt_verts,2));
thdot_dists = rrt_verts(2,:) - xy(2);

% Compute distances in a vectorized way (this is not the most efficient
% vectorization, but is good enough for our purposes).
dists = diag([th_dists;thdot_dists]'*S*[th_dists;thdot_dists]);

% Find minimizer
[~,minind] = min(dists);

% Closest vertex
closest_vert = rrt_verts(:,minind);

end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Function for performing extension
function xnew = extendLQR(closest_vert,xy,K)

% Simulate LQR policy
[~,Y] = ode45(@(t,x)pendulumDynamics(t,x,K,xy,0),[0 0.1],closest_vert);

% Extended state
xnew = Y(end,:)';

% Wrap state
xnew(1) = mod(xnew(1)+pi/2,2*pi) - pi/2;

end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Pendulum dynamics
function xdot = pendulumDynamics(t,x,K,x0,u0)

xdot = zeros(2,1);
xdot(1) = x(2);

u = -K*(x-x0)+u0;

if u > 5
    u = 5;
end
if u < -5
    u = -5;
end

g = 9.81;
b = 0.1;
xdot(2) = (u -g*1*sin(x(1)) - b*x(2));

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% rrt_dynamic.m with complete code

%% Code setup (Do not modify, but please read) %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Bounds on world
world_bounds_th = [-pi/2,(3/2)*pi];
world_bounds_thdot = [-10,10];

% Start and goal positions
figure(1);
xy_start = [0;0]; plot(xy_start(1),xy_start(2),'bo','MarkerFaceColor','b','MarkerSize',10);
xy_goal = [pi;0]; plot(xy_goal(1),xy_goal(2),'go','MarkerFaceColor','g','MarkerSize',10); drawnc

% Initialize RRT. The RRT will be represented as a 2 x N list of points. So
% each column represents a vertex of the tree.
rrt_verts = zeros(2,1000);
rrt_verts(:,1) = xy_start;
N = 1;
nearGoal = false; % This will be set to true if goal has been reached
minDistGoal = 0.25; % This is the convergence criterion. We will declare
                    % success when the tree reaches within 0.25 in distance
                    % from the goal. DO NOT MODIFY.
```

```
% Keep track of where a node came from in order to reconstruct path
came_from = 1;

%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% FILL ME IN %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Choose one of these methods
% method = 'euclidean'; % Euclidean distance metric (part b of problem)
method = 'lqr'; % LQR distance metric (part d of problem)
%% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


figure(1); hold on;
axis([world_bounds_th, world_bounds_thdot]);
 hxy = plot(0,0,'ro');

 iter = 1;
% RRT algorithm
while ~nearGoal
    iter = iter + 1;

    % Sample point
    rnd = rand(1);
    % With probability 0.05, sample the goal. This promotes movement to the
    % goal.
    if rnd < 0.05
        xy = xy_goal;
    else
        % Sample from space with probability 0.95
        xs = (world_bounds_th(2) - world_bounds_th(1))*rand(1) + world_bounds_th(1);
        ys = (world_bounds_thdot(2) - world_bounds_thdot(1))*rand(1) + world_bounds_thdot(1);
        xy = [xs;ys];
    end

    %% FILL ME IN %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if strcmp(method, 'euclidean')
        closest_vert = closestVertexEuclidean(rrt_verts(:,1:N),xy); % Write this function
    elseif strcmp(method, 'lqr')
        [closest_vert,K,closestInd] = closestVertexLQR(rrt_verts(:,1:N),xy); % Write this functi
        % To know where we must go, we must remember where we came from.
        came_from(iter) = closestInd;
    end
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

    %% FILL ME IN %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    if strcmp(method, 'euclidean')
        new_vert = extendEuclidean(closest_vert,xy); % Write this function
    else
```

```
        new_vert = extendLQR(closest_vert,xy,K); % Write this function
    end

    delete(hxy);
    figure(1);
    hxy = plot(xy(1),xy(2),'r.');axis([world_bounds_th, world_bounds_thdot]);


    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


    % Plot extension (Comment the next few lines out if you want your code to
    % run a bit quicker. The plotting is useful for debugging though.)
    figure(1)
    hold on
    plot(new_vert(1),new_vert(2),'bo','MarkerFaceColor','b','MarkerSize',5);
    % Plot line (but only if we are not wrapping to the other side of the
    % plot)
    if abs(closest_vert(1) - new_vert(1)) < 0.75*(2*pi)
        line([closest_vert(1),new_vert(1)],[closest_vert(2),new_vert(2)]);
    end
    axis([world_bounds_th, world_bounds_thdot]);


    %% DO NOT MODIFY THIS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % If it is collision free, add it to tree
    N = N+1;
    if N > size(rrt_verts,2)
        rrt_verts = [rrt_verts zeros(size(rrt_verts))];
    end
    rrt_verts(:,N) = new_vert;

    % Check if we have reached goal
    if norm(xy_goal-new_vert) < minDistGoal
        break;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


end

% Plot vertices in RRT
hold on;
plot(rrt_verts(1,:),rrt_verts(2,:),'bo','MarkerFaceColor','b','MarkerSize',5);

%% Submit rrt_verts_grade for grading %%%%
format long
```

```
rrt_verts_grade = rrt_verts(:,1:N)'
clipboard('copy',rrt_verts_grade);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Reconstruct path
reachedStart = false;
indnext = length(rrt_verts_grade);
path_inds = indnext;
while ~reachedStart
    indnext = came_from(indnext);
    path_inds = [path_inds,indnext];
    if indnext == 1
        reachedStart = true;
    end
end

xpath = rrt_verts(:,fliplr(path_inds));
xpath = xpath';

%% Visualize path (if we're in the drake/examples/Pendulum folder)
if exist('PendulumVisualizer', 'file')
  v = PendulumVisualizer();
  xp1 = unwrap(xpath(:,1)+2*pi);
  xtraj = PPTrajectory(spline(0:0.1:0.1*(length(xpath)-1),[xp1,xpath(:,2)]'));
  xtraj = xtraj.setOutputFrame(v.getInputFrame);
  v.playback(xtraj);
end
```

Run Code

Submit You have used 0 of 3 attempts

Answers are displayed within the problem