```
classdef Environment < handle
    %ENVIRONMENT This class implements the inverted pendulum batch
    %environment. The model is a simulink model that is run as a batch
    % process from this class. The state, reward, and actions of the
    % environment are defined by the class

    properties
        type = 'Continuous';
        delta = 0.5; % delta used in the action space
        reward; % reward from a single batch
        angle; % pendulum angle over the batch
        time; % pendulum time samples over the batch
        state; % environment state
        tspan = [0, 10]; % batch time-span
        Kp_bins; % Kp bins for discretization
        Ki_bins; % Ki bins for discretization
        Kd_bins; % Kd bins for discretization
        state_dimen; % state dimensions for the Q-table if discrete
        gains; % PID controller gains
    end

    methods
        function obj = Environment(sys_type, delta_step)
            %ENVIRONMENT Construct an instance of this class
            obj.delta = delta_step;
            if sys_type == "Continuous"
                obj.type = sys_type;
            elseif sys_type == "Discrete"
                obj.type = sys_type;
                obj.Kp_bins = 10:obj.delta:50;
                obj.Ki_bins = 1:obj.delta:20;
                obj.Kd_bins = 1:obj.delta:20;
                obj.state_dimen = [length(obj.Kp_bins),
 length(obj.Ki_bins), length(obj.Kd_bins)];
            else
                disp('Error: Must choose either Continuous or Discrete
 for type');
            end
            obj.gains = [10, 1, 1];
            obj.state = [10, 1, 1];
        end

        function [reward, next_state] = step(obj, A)
            % STEP apply an action to the envrionment and simulate one
            % batch of the simulink model
            global Kp; global Ki; global Kd;
            obj.gains = update_gain(A, obj.gains, obj.delta);
            Kp = obj.gains(1); Ki = obj.gains(2); Kd = obj.gains(3);
            simOut = sim('ModelInvertedPD1.mdl', 'StartTime',
 string(obj.tspan(1)), 'StopTime', string(obj.tspan(2)));
            obj.time = simOut.ScopeData.time; obj.angle =
 simOut.ScopeData.signals(2);
```

```
            obj.reward = -sum(abs(obj.angle.values));

            if obj.type == "Discrete"
                % this implementation is for Q-learning so gains are
                % discretized
                obj.state = [discretize(obj.gains(1), obj.Kp_bins),
 discretize(obj.gains(2), obj.Ki_bins), discretize(obj.gains(3),
 obj.Kd_bins)];
            elseif obj.type == "Continuous"
                obj.state = [Kp, Ki, Kd];
            else
                disp("Error: Undefined system type")
                disp("Must be Discrete or Continuous")
            end
            next_state = obj.state;
            reward = obj.reward;
        end

        function reset(obj)
            global Kp; global Ki; global Kd;
            Kp = 10; Ki = 1; Kd = 1;
            obj.gains = [10,1,1];
            obj.state = [10,1,1];
        end
    end
end



function out_gains = update_gain(action, in_gains, d)

    % UPDATE_GAINS implements the actions in the action space of the
 batch
    % method
    Kp1 = in_gains(1);
    Ki1 = in_gains(2);
    Kd1 = in_gains(3);

    delta = d;
    if action == 1
        Kp1 = in_gains(1) + delta;
    elseif action == 2
        Kp1 = in_gains(1) - delta;
    elseif action == 3
        Ki1 = in_gains(2) + delta;
    elseif action == 4
        Ki1 = in_gains(2) - delta;
    elseif action == 5
        Kd1 = in_gains(3) + delta;
    elseif action == 6
        Kd1 = in_gains(3) - delta;
    elseif action == 7
        Kp1 = in_gains(1);
        Ki1 = in_gains(2);
```

```
        Kd1 = in_gains(3);
    end

    % Check for any low values that would be unstable;
    if Kp1 < 10
        Kp1 = 10;
    elseif Ki1 < 1
        Ki1 = 1;
    elseif Kd1 < 1
        Kd1 = 1;
    end

    % check for any high values that could use too much force;
    if Kp1 > 50
        Kp1 = 50;
    elseif Ki1 > 20
        Ki1 = 20;
    elseif Kd1 > 20
        Kd1 = 20;
    end
    out_gains = [Kp1, Ki1, Kd1];
end
```

```
Not enough input arguments.

Error in Environment (line 26)
            obj.delta = delta_step;
```

Published with MATLAB® R2020b