Bachelor of Technology (B. Tech.)

Computer and Communication Engineering (CCE)

Amrita School of Engineering

Coimbatore Campus (India)

Academic Year – 2022 - 23

S. No	Name	Roll Number
1	B Ambareesh	CB.EN.U4CCE20006
2	Manoj Parthiban	CB.EN.U4CCE20032
3	V Parthiv	CB.EN.U4CCE20041
4	Santosh	CB.EN.U4CCE20053

Semester VI – Third Year

19CCE312 Cyber-Physical Systems: Design, Modelling and Simulation

Assignment – Safe Reinforcement Learning

(This assignment aims to demonstrate the effectiveness of safe reinforcement learning in training a robotic agent to navigate a grid world while avoiding unsafe states. Also, it aims to show that by incorporating safety constraints and penalties, an agent can learn about the policy that maximizes cumulative rewards while adhering to predefined safety criteria.)

Faculty In-charge – Dr Binoy B. Nair

A subfield of machine learning called reinforcement learning is concerned with teaching an agent how to maximise a cumulative reward by making successive decisions in a given environment. It draws inspiration from the way that both people and animals learn via experience and contact with their environment. By investigating and taking advantage of the dynamics of the environment, reinforcement learning methods allow an agent to learn the best possible rules.

In reinforcement learning, an agent interacts with a collection of states, actions, and rewards that serve as the environment. The agent observes the current state, chooses an action and receives a reward from the environment at each time step. To maximise the total predicted cumulative reward over time, the agent's goal is to learn a policy that maps states to actions.

Reinforcement learning may be used efficiently in robotic motion control. Take a robotic arm learning to grip items as an example. The workspace is the environment, the robotic arm is the agent, and the actions comprise adjusting the joint angles of the arm. The objective is to learn a policy that permits the arm to successfully grip items.

At first, the agent performs activities at random to investigate its surroundings, noting the states and rewards that emerge. As it investigates, it begins to understand some acts result in favourable results, like effectively gripping an object, and which actions result in unfavourable consequences, such as toppling or dropping objects. To estimate the expected cumulative reward for doing a certain action in a specific state, the agent maintains a value function or a Q-function.

The agent modifies its value function based on observed rewards and state changes using a technique known as the Q-learning algorithm. By deciding to take behaviours that have larger predicted cumulative rewards, it eventually learns the best course of action. The agent's investigation slows down over time and is likely to produce large rewards.

The idea of "safe" reinforcement learning aims to prevent catastrophic failures or destructive behaviours from occurring throughout the learning process. To avoid harm to the robot, the environment, or any associated people, safety is of the highest significance when it comes to robotic motion control. Safe reinforcement learning includes safeguards that reduce hazards and abide by predetermined safety restrictions.

The inclusion of safety restrictions or additional penalty conditions to the learning process is one method of safe reinforcement learning. To prevent going above safe limitations, the agent could, for instance, be subject to restrictions on the highest joint velocities or forces it can apply. These limitations can be integrated into the agent's policy or value function, which will motivate it to develop safe behaviours that comply with them.

Model-based reinforcement learning is a different strategy, in which the agent learns a model of the dynamics of the environment. The agent can experiment with prospective behaviours without actually influencing the real robot by simulating various actions and their results in the learnt model. The agent may learn and improve its policies through this simulation-based investigation without endangering its security.

Additionally, adding human oversight and direction can improve safe reinforcement learning. The agent may learn from and utilise as a reference during training human experts' examples of safe and desirable behaviour. Additionally, the integration of human feedback enables the

agent to actively seek advice when confronted with ambiguous circumstances or possible security issues.

In conclusion, reinforcement learning offers a strong foundation for robotic motion control, allowing machines to pick up difficult skills like object gripping. Safe reinforcement learning strategies make sure that the learning process abides by established safety restrictions and stays away from dangerous behaviour. Robots may learn efficient and safe control policies in dynamic situations by integrating exploration, exploitation, safety restrictions, and human direction.

SMALL-SCALE SITUATION EMPLOYING MATLAB

The code provided is a complete implementation of the Q-learning algorithm in MATLAB. The environment is defined as a grid world where 0 represents safe states, and 1 represents unsafe states. The rewards are set for each state with a higher reward assigned to the goal state.

The Q-table is initialized with zeros and the maximum action for each state is precomputed. The safe states are defined using a logical matrix. The training loop runs for a specified number of episodes with a maximum step limit per episode. During each step, an action is chosen based on the epsilon-greedy policy.

The Q-value of the current state-action pair is updated using the reward and discount factor. If the next state is non-safe, the agent moves back to the current state; otherwise, the corresponding reward is assigned.

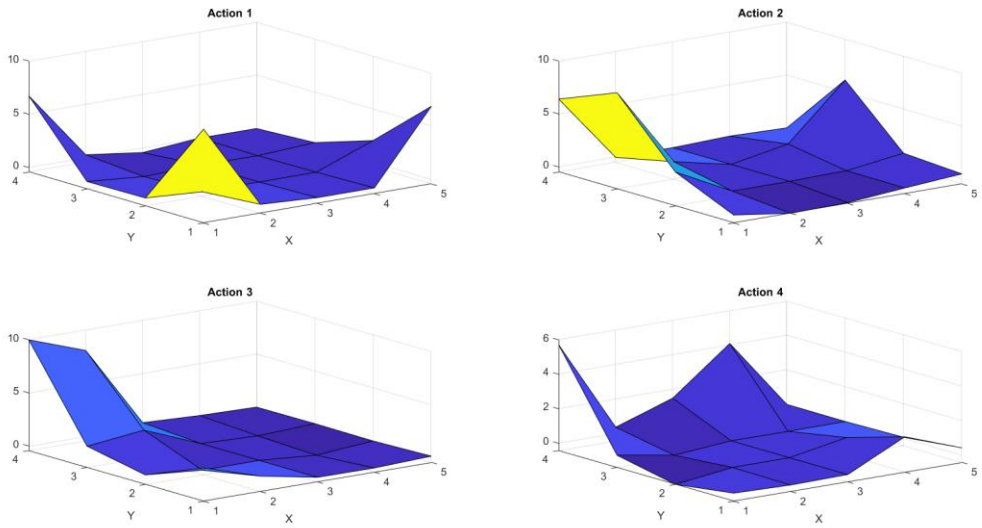
The Q-values learned for each state-action pair are plotted as a surface plot. The optimal policy is computed as a 2D matrix of x- and y-components and plotted as a vector field. The perform_action function performs the chosen action and returns the next state.

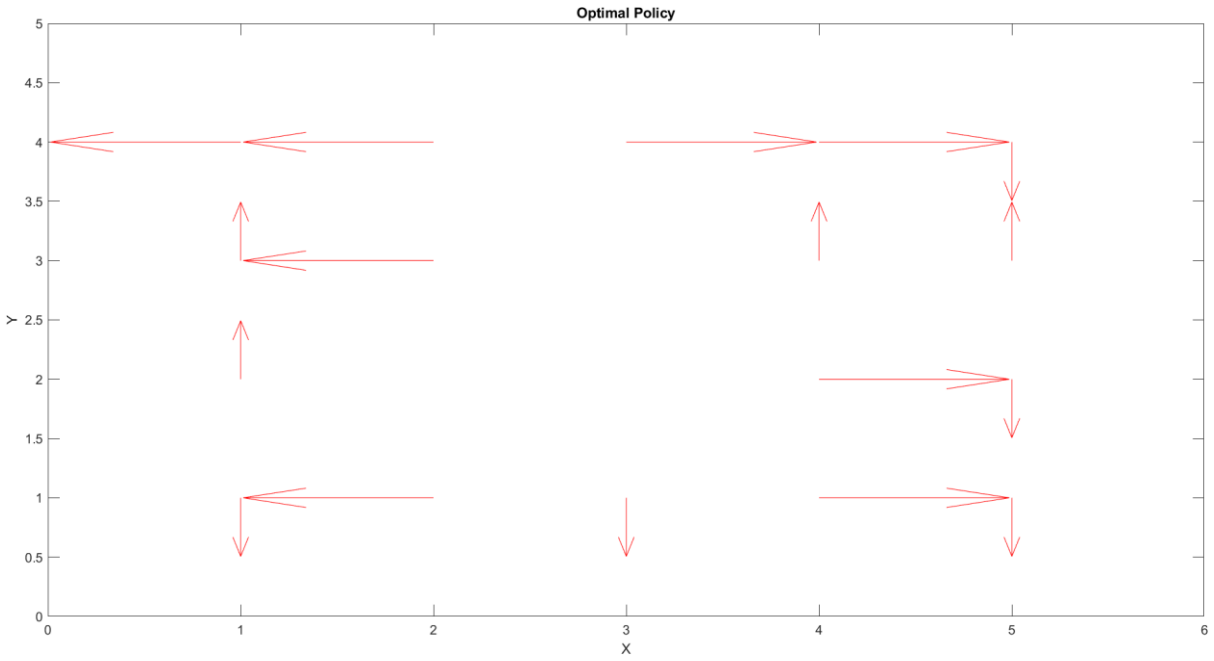
```
% Define the grid world environment
% 0 represents safe states, and 1 represents unsafe states
environment = [0, 0, 0, 0, 0;
               0, 1, 1, 0, 0;
               0, 0, 1, 0, 0;
               0, 0, 0, 0, 0];

% Define the rewards for each state
rewards = ones(size(environment)) * -0.1;
rewards(end, end) = 10; % reaching the goal state has higher reward

% Set the parameters
num_episodes = 1000; % number of training episodes
max_steps = 100; % maximum number of steps per episode
learning_rate = 0.1; % learning rate
discount_factor = 0.9; % discount factor

epsilon_initial = 1.0; % initial exploration rate
```

```
epsilon_decay = 0.99; % decay rate for exploration rate
epsilon_min = 0.05; % minimum exploration rate

max_velocity = 2; % maximum allowed velocity in each direction

% Initialize the Q-table with zeros
Q = zeros(size(environment,1), size(environment,2), 4);

% Precompute the index of the action with the maximum Q-value for each state
[~, max_action] = max(Q,[],3);

% Define a logical matrix to represent safe states
safe_states = environment == 0;

% Start the training loop
for episode = 1:num_episodes
    % Reset the robot's position to the starting state
    current_state = [size(environment,1), 1];

    % Decay the exploration rate
    epsilon = max(epsilon_initial * epsilon_decay^episode, epsilon_min);

    % Loop through each step of the episode
    for step = 1:max_steps
        % Choose an action based on epsilon-greedy policy
        if rand < epsilon
            % Exploration: choose a random action
            action = randi(4);
        else
            % Exploitation: choose the action with the maximum Q-value
            action = max_action(current_state(1), current_state(2));
        end

        % Take the chosen action and observe the next state and reward
        next_state = perform_action(current_state, action);

        % Wrap around indices that go out of bounds
        next_state = mod(next_state-1,size(environment))+1;

        % Assign negative reward to non-safe states
        reward = -1;
        if ~safe_states(next_state(1),next_state(2))
            % If next_state is non-safe, move back to the current state
            next_state = current_state;
        else
            % If next_state is safe, assign the corresponding reward
            reward = rewards(next_state(1), next_state(2));
        end
```

```
        % Update the Q-value of the current state-action pair
        Q(current_state(1), current_state(2), action) = ...
            (1 - learning_rate) * Q(current_state(1), current_state(2),
action) ...
            + learning_rate * (reward + discount_factor * max(Q(next_state(1),
next_state(2), :)));

        % Update the index of the action with the maximum Q-value for the new
state
        [~, max_action(next_state(1), next_state(2))] = max(Q(next_state(1),
next_state(2), :));

        % Move to the next stage
        current_state = next_state;

        % Check if the goal state is reached
        if reward == 10
            disp('Goal reached!');
            break;
        end
    end
end

% Plot the learned Q-values for each state-action pair
figure;
for action = 1:4
    subplot(2,2,action);
    surf(squeeze(Q(:,:,action)));
    title(['Action ' num2str(action)]);
    xlabel('X');
    ylabel('Y');
end

% Compute the optimal policy as a 2D matrix of x- and y-components
optimal_policy = zeros(size(environment,1), size(environment,2), 2);
for i = 1:size(environment,1)
    for j = 1:size(environment,2)
        if environment(i,j) == 0 % Only consider safe states
            [~,idx] = max(Q(i,j,:));
            if idx == 1 % Up
                optimal_policy(i,j,:) = [-1, 0];
            elseif idx == 2 % Down
                optimal_policy(i,j,:) = [1, 0];
            elseif idx == 3 % Left
                optimal_policy(i,j,:) = [0, -2];
            else % Right
                optimal_policy(i,j,:) = [0, 2];
```

```
            end
        end
    end
end

% Plot the optimal policy as a vector field
figure;
[X,Y] = meshgrid(1:size(environment,2),1:size(environment,1));
quiver(X,Y,optimal_policy(:,:,2),optimal_policy(:,:,1),'r');
title('Optimal Policy');
xlabel('X');
ylabel('Y');
xlim([0 size(environment,2)+1]);
ylim([0 size(environment,1)+1]);

% Function to perform an action and return the next state
function next_state = perform_action(current_state, action)
    % Define the possible actions: 1 = up, 2 = down, 3 = left, 4 = right
    % Update the state based on the chosen action
    if action == 1
        next_state = [current_state(1)-1, current_state(2)];
    elseif action == 2
        next_state = [current_state(1)+1, current_state(2)];
    elseif action == 3
        next_state = [current_state(1), current_state(2)-1];
    else
        next_state = [current_state(1), current_state(2)+1];
    end
end
```





CONCLUSION

Based on the assignment, it can be inferred that safe reinforcement learning with Q-learning is a viable approach for training robotic agents to make decisions in dynamic environments.

The results of the experiment demonstrate that by incorporating safety constraints and penalties, the agent successfully learns to avoid unsafe states and takes actions that lead to higher cumulative rewards.

This implies that safe reinforcement learning techniques can enhance the safety and reliability of robotic motion control systems, allowing them to operate in real-world scenarios while minimizing the risk of accidents or harmful actions.