RS-HL-12: Multi User Code

Hongseok Kim

UT Austin Oden Institute

07/20/24

I. Load the Dataset

```
% clear;clc;
% % Load the Satellite Contat Dataset
% addpath('~/Desktop/Redstone_Project/RS_HL/RS_HL_10_TV_MDP_Functions')
% load('/workspace/RS_Dataset/RS_HL_3_dataset.mat')
```

II. Destination Setting and Time Index Vector Setting

```
% time_index_vector = 100:130;
%
% destination_1 = 10;
% destination_2 = 12;
% destination_3 = 18;
% destination_4 = 27;
% destination_5 = 40;
```

II.1 Run the MDP simulation for each destination

```
% MDP.(['MDP', num2str(18)]) = runMDP(sat_to_sat_contact_3d_matrix,
time_index_vector,18);
```

III. Configure each Agent's Setting

```
a_start = 31;
a_destination = 18;
a_time = 100;


b_start = 31;
b_destination= 18;
b_time = 100;
```

```
c_start = 31;
c_destination = 18;
c_time = 100;

d_start = 31;
d_destination = 18;
d_time = 100;

e_start = 31;
e_destination = 18;
e_time = 118;

agents_input = [a_time, a_start, a_destination;
                b_time, b_start, b_destination;
                c_time, c_start, c_destination;
                d_time, d_start, d_destination;
                e_time, e_start, e_destination];
```

IV. Configure the simulation structure setting

```
% Level 1: Initialize simulation structure
sim = struct();

for time_index = time_index_vector
    sim.(['time' num2str(time_index)]) = {};
end

% Level 2/3: Initialize Agent (Level 2) with States and Destination (Level 3)

number_of_agents = length(agents_input(:,1));

for agent_index = 1:number_of_agents
   sim.(['time' num2str(agents_input(agent_index,1))]).(['agent'
num2str(agent_index)]).('state') = agents_input(agent_index,2);
   sim.(['time' num2str(agents_input(agent_index,1))]).(['agent'
num2str(agent_index)]).('destination') = agents_input(agent_index,3);
end
```

V. Propagation of agents' state

```
collision_flag = false;
```

```
for time_index = time_index_vector
    % Break when time index reaches the end time
    if time_index == max(time_index_vector)
       break;
    end


    % If there's no active agent, continue to next time step
    if isempty(sim.(['time' num2str(time_index)]))
       continue;
    end


    % Parse the number of active agents
    number_of_active_agents = length(fieldnames(sim.(['time'
num2str(time_index)])));

    % Make the status matrix represents current and next
    status_matrix = zeros(3, number_of_active_agents);

    agents_list = fieldnames(sim.(['time' num2str(time_index)]));

    % Find the Next state from Current Agent-State
    for active_agent_index = 1:number_of_active_agents
        % Find the Current State and Destination of given agent
        current_state = sim.(['time' num2str(time_index)]).
(agents_list{active_agent_index}).('state');
        destination = sim.(['time' num2str(time_index)]).
(agents_list{active_agent_index}).('destination');

        % Find the Next state from given MDP pi distribution
        pi_dist = MDP.(['MDP' num2str(destination)]).(['time'
num2str(time_index)]).('policy_distribution');
        action_number = find(pi_dist(current_state,:));

        if length(action_number) > 1
        action_number = randsample(action_number,1);
        end

        next_state = MDP.(['MDP' num2str(destination)]).(['time'
num2str(time_index)]).(['state' num2str(current_state)]).(['action'
num2str(action_number)]).('success').('next_state');

        status_matrix(1,active_agent_index) = current_state;
        status_matrix(2,active_agent_index) = next_state;
        status_matrix(3,active_agent_index) = destination;
    end

    for active_agent_index = 1:number_of_active_agents
```

```
        % Don't update the agent already arrived to destination
        if status_matrix(1,active_agent_index) ==
status_matrix(3,active_agent_index)
            continue;
        end

        % Update the time+1 for next state
        sim.(['time' num2str(time_index+1)]).
(agents_list{active_agent_index}).('state') =
status_matrix(2,active_agent_index);
        sim.(['time' num2str(time_index+1)]).
(agents_list{active_agent_index}).('destination') =
status_matrix(3,active_agent_index);
    end


    % Avoid Collision Avoidance Algorithm


    % If next state Agent info is empty, continue
    if isempty(sim.(['time' num2str(time_index+1)]))
        continue;
    end

    % Parse the number of next state active agents
    next_state_number_of_active_agents = length(fieldnames(sim.(['time'
num2str(time_index+1)])));
    next_state_status_matrix = zeros(1, next_state_number_of_active_agents);
    next_state_agents_list = fieldnames(sim.(['time'
num2str(time_index+1)]));

    for next_state_agent_index = 1:next_state_number_of_active_agents
        next_state_status_matrix(next_state_agent_index)
= sim.(['time' num2str(time_index+1)]).
(next_state_agents_list{next_state_agent_index}).('state');
    end

    unique_elements = unique(next_state_status_matrix);

    % If there is no collision -> contintue
    if length(next_state_status_matrix) == length(unique_elements)
        continue;
    end
```

VI. If there exists Collision -> Activate Collision Avoidance Algorithm

```
    fprintf('collision occured at time index %d\n', time_index);
```

```
    % If there exist collision -> Start the infinite loop until the
    % propblem resolved
    collision_flag = true;


    while collision_flag == true
     % Collect Action value vector of each agent's state

     action_value_struct = struct();
     vector_length_information = zeros(number_of_active_agents,1);

     for active_agent_index = 1:number_of_active_agents
        % Find the Current State and Destination of given agent
        current_state = sim.(['time' num2str(time_index)]).
(agents_list{active_agent_index}).('state');
        destination = sim.(['time' num2str(time_index)]).
(agents_list{active_agent_index}).('destination');

        % Get the Action value Matrix from given state in given MDP
        action_value_vector = MDP.(['MDP' num2str(destination)]).
(['time' num2str(time_index)]).(['state' num2str(current_state)]).
('action_value_vector');
        vector_length = length(action_value_vector);
        action_value_matrix = [(1:vector_length)',action_value_vector];
        action_value_matrix = sortrows(action_value_matrix,2, 'descend');

        if vector_length > 4
            action_value_matrix = action_value_matrix(1:4,:);
        end


        action_value_struct.(agents_list{active_agent_index}).
('action_value_matrix') = action_value_matrix;
        vector_length_information(active_agent_index) =
length(action_value_matrix(:,1));
     end


     % Generate Cases For Each Action Value
     combination_matrix = [];

     for active_agent_index = 1:number_of_active_agents-1
        new_matrix = [];
        if active_agent_index == 1
           pre_matrix = (1:vector_length_information(active_agent_index))';
        else
           pre_matrix = combination_matrix;
        end

        length_of_pre_matrix = length(pre_matrix(:,1));
```

```
        for next_agent_index =
1:vector_length_information(active_agent_index+1)

        adding_vector = ones(length_of_pre_matrix,1)*next_agent_index;
        new_matrix_segment = [pre_matrix,adding_vector];
        new_matrix = [new_matrix;new_matrix_segment];

        end
        combination_matrix = new_matrix;
     end


     if number_of_active_agents == 1
        combination_matrix =  (1:vector_length_information(1))';
     end

     action_number_matrix =
zeros(length(combination_matrix(:,1)),number_of_active_agents);
     action_value_matrix =
zeros(length(combination_matrix(:,1)),number_of_active_agents);
     action_value_sum_vector = zeros(length(combination_matrix(:,1)),1);
     next_state_matrix =
zeros(length(combination_matrix(:,1)),number_of_active_agents);




     for case_index = 1:length(combination_matrix(:,1))
         for active_agent_index = 1:number_of_active_agents
             action_index =
combination_matrix(case_index,active_agent_index);

             action_value_info =  action_value_struct.
(agents_list{active_agent_index}).('action_value_matrix');

             action_number = action_value_info(action_index,1);
             action_value = action_value_info(action_index,2);

             action_number_matrix(case_index,active_agent_index) =
action_number;
             action_value_matrix(case_index,active_agent_index) =
action_value;


             current_state = sim.(['time' num2str(time_index)]).
(agents_list{active_agent_index}).('state');
             destination = sim.(['time' num2str(time_index)]).
(agents_list{active_agent_index}).('destination');
```

```
            next_state_info =  MDP.(['MDP' num2str(destination)]).
(['time' num2str(time_index)]).(['state' num2str(current_state)]).(['action'
num2str(action_number)]).('success').('next_state');
            next_state_matrix(case_index,active_agent_index) =
next_state_info;

         end
         action_value_sum_vector(case_index) =
sum(action_value_matrix(case_index,:));
     end

     case_evaluation_matrix =
[next_state_matrix,action_number_matrix,action_value_sum_vector];
     case_evaluation_matrix =
sortrows(case_evaluation_matrix,length(case_evaluation_matrix(1,:)),
'descend');

     for  case_index = 1:length(combination_matrix(:,1))

        next_state =
case_evaluation_matrix(case_index,1:number_of_active_agents);
        % Modify Status Matrix for corresponding next state vector
        status_matrix(2,:) = next_state;

        for active_agent_index = 1:number_of_active_agents

            % Don't update the agent already arrived to destination
            if status_matrix(1,active_agent_index) ==
status_matrix(3,active_agent_index)
                continue;
            end

            % Update the time+1 for next state
            sim.(['time' num2str(time_index+1)]).
(agents_list{active_agent_index}).('state') =
status_matrix(2,active_agent_index);

        end

        % Parse the number of next state active agents
        next_state_number_of_active_agents = length(fieldnames(sim.(['time'
num2str(time_index+1)])));
        next_state_status_matrix = zeros(1,
next_state_number_of_active_agents);

        for next_state_agent_index = 1:next_state_number_of_active_agents
            next_state_status_matrix(next_state_agent_index)
= sim.(['time' num2str(time_index+1)]).
(next_state_agents_list{next_state_agent_index}).('state');
```

```
        end

        unique_elements = unique(next_state_status_matrix);

        % If there is no collision -> contintue
        if length(next_state_status_matrix) == length(unique_elements)
           fprintf('collision resolved at time index %d\n', time_index);
           collision_flag = false;
           break;
        end

     end


     if collision_flag == true
            fprintf('we could not resolve the collision at time index %d\n',
time_index);
            break;
     end

    end

    if collision_flag == true

    fprintf('Simulation Terminated with fail');
    break;
    end
end
```

```
collision occured at time index 100
collision resolved at time index 100
collision occured at time index 108
collision resolved at time index 108
```

```
if collision_flag == false
fprintf('Simulation Terminated with Success')
end
```

```
Simulation Terminated with Success
```

VI. result display

```
result_matrix = zeros(length(time_index_vector),number_of_agents);

for time_index = time_index_vector

    if isempty(sim.(['time' num2str(time_index)]))
     continue;
    end
```

```
    agents_list = fieldnames(sim.(['time' num2str(time_index)]));
    number_of_agents = length(agents_list);

    for agent_index = 1:number_of_agents
        agent_name = cell2mat(agents_list(agent_index));
        agent_no = regexp(agent_name, '\d+','match');
        agent_number = str2double(agent_no{1});

        result_matrix(time_index - min(time_index_vector) + 1,agent_number)
= sim.(['time' num2str(time_index)]).(agent_name).('state');


    end

end

result_matrix
```

```
result_matrix = 31×5
    31    31    31    31     0
     5    30     9    32     0
    26    29    10    11     0
    24    28    11    12     0
    45    27    12    13     0
    19    26    13    14     0
    18    24    14    15     0
     0    45    15    16     0
     0    19    16    17     0
     0    20    17    18     0
     ⋮
```