helicopter_learn_loop.m

Table of Contents

Summary: Script to learn a controller for the helicopter problem

High-level steps

1. Initialize variables and compile RL-Glue trainer
2. Generate random trajectories to speed up learning
3. Apply learning loop (model learning, policy learning, policy application)

Code

```
% 1. Initialization
clear all; close all;
settings_hc;                    % Load scenario-specific settings

use_reward_model = false;
use_previous_history = true;

% Modify the trainer source files and compile for the desired MDP
whichMDP = 0;
setenv('tMDP', num2str(whichMDP));
cd([trainerDir 'consoleTrainerJavaHelicopter']);
!sed "s/whichTrainingMDP = [0-9]/whichTrainingMDP = ${tMDP}/" <src/consoleTrainerB
!make

% Get a suitable name for the history file
cd([pilcoDir 'scenarios/helicopter']);
getHistoryFilename;


% 2. Collect data with random policy to start learning, or start from the
% latest history file
if ~use_previous_history
    cd([trainerDir 'testTrainerJavaHelicopter']);
    !sed "s/whichTrainingMDP = [0-9]/whichTrainingMDP = ${tMDP}/" <src/testTrainer
    !make
    !bash run.bash &
    cd([agentDir 'randomAgentMatlab']);
    delete(['randomDataMDP' num2str(whichMDP) '.mat']);
    theRandomAgent = random_agent_matlab(whichMDP, codecDir, agentDir);
    runAgent(theRandomAgent);
```

```
        load([agentDir 'randomAgentMatlab/randomDataMDP' num2str(whichMDP) '.mat']);

        delta_n = min(n_init, size(random_data.x, 1));
        x = random_data.x(1:delta_n,:);
        y = random_data.y(1:delta_n,:);
        r_x = random_data.r_x(1:delta_n,:);
        r_y = random_data.r_y(1:delta_n,:);
        H = 5;
    else
        load(historyFilename);
        x = history{end}.x;
        y = history{end}.y;
        r_x = history{end}.r_x;
        r_y = history{end}.r_y;
        dynmodel = history{end}.dynmodel;
        if use_reward_model
            if ~isfield(history{end}, 'reward_dynmodel')
                disp('Requested reward model, but no reward model in previous history.
                return
            else
                reward_dynmodel = history{end}.reward_dynmodel;
            end
        end
        policy = history{end}.policy;
        H = history{end}.H;
        last_size = history{end}.steps;
    end


    % Start learning loop. Feel free to Ctrl-C at any time, history file is
    % saved automatically and you can resume later using
    % |use_previous_history = true;|
    cd([pilcoDir 'scenarios/helicopter']);
    j = 1;
    while true

        % 3. Train GP's and specify cost function

        % 3.1. Train GP for the helicopter dynamics
        trainHelicopterModel
        if exist('history', 'var'); history{end+1}.dynmodel = dynmodel; else history{1

        % 3.2. Train GP model to predict reward to use as cost function, or
        % set parameters of user-defined cost function
        if use_reward_model
            trainRewardModel;
            cost.fcn = @Rbased_loss_hc;
            cost.rewardgpmodel = reward_dynmodel;
            history{end}.reward_dynmodel = reward_dynmodel;
        else
            cost.width = 1;
            cost.fcn = @loss_hc;
        end
```

```
% 4. Learn policy and save structure
H = H + 5;
learnPolicy;
policy.date = clock;

history{end}.policy = policy;
history{end}.H = H;

% 5. Run the simulator with the latest policy until trajectory is
% longer than lower limit. If it isn't aggregate more datasets.
last_size = 0;
min_last_size = H;
run_counter = 0;
while last_size < min_last_size && run_counter < 10
 cd([trainerDir 'consoleTrainerJavaHelicopter']);
 !bash run.bash &
 cd([pilcoDir 'scenarios/helicopter']);
 theAgent = helicopter_agent(policy, codecDir, pilcoDir);
 runAgent(theAgent);

    aux_newdata = load('GPHistory.mat');
 newdata = [newdata; newdata.helicopter_agent_struct];
 last_size = size(newdata,1) - 1;

    run_counter = run_counter + 1;
end

% 6. Get new data from the last trajectory. If the last trajectory
% was long enough, select data
if last_size < 5990
    delta_n = min(last_size, max_last_size) + 1;
    x = [x; newdata(1:delta_n-1,1:16)];
    y = [y; newdata(2:delta_n,1:12)];
    r_x = [r_x; newdata(2:delta_n,1:12)];
    r_y = [r_y; newdata(2:delta_n, 17)];
else
    [dynIdx, rewardIdx] = selectData(history{end}, max_last_size);
    x = [x; history{end}.trajectory(dynIdx, 1:16)];
    y = [y; history{end}.trajectory(dynIdx+1, 1:12)];
    r_x = [r_x; history{end}.trajectory(rewardIdx, 1:12)];
    r_y = [r_y; history{end}.trajectory(rewardIdx, 17)];
end


history{end}.steps = last_size;
history{end}.x = x;
history{end}.y = y;
history{end}.r_x = r_x;
history{end}.r_y = r_y;
history{end}.trajectory = newdata;

save(historyFilename, 'history');

disp(['Length of the last trajectory is ' num2str(last_size)]);
```

```
    % 7. Plot NLPD of model when following the trajectory
    drawNLPDplots;

    j = j + 1;

end
```

Published with MATLAB® R2014a