**The PID controller which is tuned using TD3 reinforcement learning algorithm.**

|  |  |  |
| --- | --- | --- |
| **Candidate** | **：** | **Anton Kolonin** |
| **Supervisor** | **：** | **Huang Qianyi** |

**Novosibirsk State University**

**1, Pirogova str., Novosibirsk, 630090, Russia**

**December, 2023**

Abstract

The TD3 algorithm is also a deterministic deep reinforcement learning algorithm under the Actor-Critic (AC) framework. It combines a deep deterministic policy gradient algorithm and dual Q-learning, and has achieved good performance on many continuous control tasks.

The idea refers to the DDPG algorithm I will introduce these two algorithms and the simulation application of TD3 algorithm respectively below.

Content

[Abstract II](#_Toc452297565)

[1 DDPG algorithm 1](#_Toc452297567)

[1.1 Introduction 1](#_Toc452297568)

[1.2 Algorithm principle 2](#_Toc452297569)

[1.3 Experience replay 3](#_Toc452297570)

[1.4 Target network 6](#_Toc452297571)

[1.5 Noise exploration 7](#_Toc452297572)

[2 TD3 algorithm 12](#_Toc452297575)

[2.1 Principle of algorithm 12](#_Toc452297576)

[2.2 dual network 14](#_Toc452297577)

[2.3 Target policy smoothing regularization 16](#_Toc452297578)

[2.4 Delay update 17](#_Toc452297580)

[2.5 Process of updating 17](#_Toc452297580)

[3 Application of TD3 alg. 17](#_Toc452297579)

[3.1 ENVIROMENT 17](#_Toc452297580)

[3.2 Tune PI Controller Using Control System Tuner 18](#_Toc452297581)

[3.3 Create Environment for Training Agent 21](#_Toc452297582)

[3.4 Create TD3 Agent 23](#_Toc452297583)

[3.5 Train TD3 26](#_Toc452297584)

[3.6 Validate Trained Agent 28](#_Toc452297585)

[3.7 Compare Controller Performance 30](#_Toc452297586)

REFERNCE

# DDPG algorithm

## Introduction

Before the DDPG algorithm, there were two main ways to solve continuous action space problems: one was to discretize the continuous action, and then use reinforcement learning algorithms (such as DQN) to solve it. The second is to use the Policy Gradient (PG) algorithm (such as Reinforce) to solve directly. However, for the first method, the discretization process is divorced from engineering reality to a certain extent; for the second method, the PG algorithm is often unsatisfactory in solving continuous control problems. For this reason, the DDPG algorithm was born and has achieved very good results in many continuous control problems.

The DDPG algorithm is an online deep reinforcement learning algorithm under the Actor-Critic (AC) framework. Therefore, the algorithm includes an Actor network and a Critic network. Each network is updated according to its own update rules, thereby maximizing the cumulative expected return

## Algorithm principle

DDPG algorithm mainly includes the following three key technologies:

(1) Experience replay: The agent puts the obtained experience data into the Replay Buffer, and samples in batches when updating network parameters.

(2) Target network: In addition to the Actor network and Critic network, a set of Target Actor network and Target Critic network are used to estimate the target. When updating the target network, in order to avoid parameter updates too fast, a soft update method is used.

(3) Noise exploration: The actions output by the deterministic strategy are deterministic actions and lack exploration of the environment. In the training phase, noise is added to the actions output by the Actor network, so that the agent has certain exploration capabilities.

## Experience replay

Experience replay is a technology that stabilizes the empirical probability distribution, which can improve the stability of training. Experience playback mainly has two key steps: "storage" and "playback":

Storage: Store experience in the experience pool in the form of 

Playback: Sampling one or more pieces of experience data from the experience pool according to certain rules.

From a storage perspective, experience playback can be divided into centralized playback and distributed playback:

Centralized playback: The agent runs in an environment and stores experience uniformly in the experience pool.

Distributed playback: Multiple agents run in multiple environments at the same time, and their experiences are uniformly stored in the experience pool. Since multiple agents generate experience simultaneously, experience can be collected faster while using more resources.

From a sampling perspective, experience playback can be divided into uniform playback and priority playback:

Uniform replay: Sampling experience from the experience pool with equal probability.

Prioritize playback: assign a priority to each experience in the experience pool, and prefer experience with higher priority when sampling experience. The general approach is that if the priority of a certain experience (such as experience) is, then the probability of selecting the experience is:



Advantages of experience replay:

1. When training the Q network, the correlation between data can be broken so that the data satisfies independent and identical distribution, thereby reducing the variance of parameter updates and improving the convergence speed.

2. Able to reuse experience, high data utilization rate, especially useful for situations where data acquisition is difficult.

Disadvantages of experience replay:

Cannot be applied to round update and multi-step learning algorithms. However, applying experience replay to Q learning avoids this shortcoming.

## Target network

Since the DDPG algorithm is based on the AC framework, the algorithm must contain Actor and Critic networks. In addition, each network has its corresponding target network, so the DDPG algorithm includes four networks, namely Actor network, Critic network, Target Actor network and Target Critic network. This section mainly introduces the update process of the DDPG algorithm, the update method of the target network and the purpose of introducing the target network.

### Algorithm update process

The algorithm update mainly updates the parameters of the Actor and Critic networks. The Actor network is updated by maximizing the cumulative expected return, and the Critic network is updated by minimizing the error between the evaluation value and the target value. In the training phase, we sample a batch of data from the Replay Buffer. Assuming that the sampled data is, the Actor and Critic network update process is as follows.

Critic network update process: Use the Target Actor network to calculate the actions in the state:



Then use the Target Critic network to calculate the state-action pair (s, a)

target value:



Then use the Critic network to calculate the state-action pair (s, a)

Evaluated value:



Finally, the gradient descent algorithm is used to minimize the difference Lc between the evaluation value and the expected value.

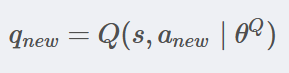
, thereby updating the parameters in the Critic network



Actor network update process: Use the Actor network to calculate the action in state s:



Then use the Critic network to calculate the evaluation value (i.e., cumulative expected return) of the state-action pair



Finally, the gradient ascent algorithm is used to maximize the cumulative expected return ,thereby updating the parameters in the Actor network.

### Updates to the target network

For the update of the target network, the DDPG algorithm uses a soft update method, which can also be called exponential moving average (EMA). That is, introduce a learning rate (or momentum) τ

, do a weighted average of the old target network parameters and the new corresponding network parameters, and then assign them to the target network:

Target Actor network update process:



Target Critic network update process:



## Noise exploration

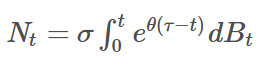
Exploration is crucial for an intelligent agent, and deterministic strategies "naturally" lack the ability to explore. Therefore, we need to artificially add noise to the output actions so that the agent has the ability to explore. In the DDPG algorithm, the author uses the Ornstein Uhlenbeck process as the action noise. The Ornstein Uhlenbeck process is defined by the following stochastic differential equation (taking the one-dimensional case as an example):



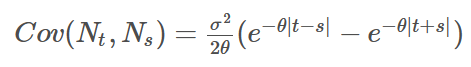
are parameters. is standard Brownain action.

When the initial perturbation is a single point distribution at the origin and :

The solution to the above equation is:



The covariance is:



Because we always have ,so the covariance always greater than 0.

The Ornstein Uhlenbeck process is used to make adjacent perturbations positively correlated, thereby causing the actions to shift in similar directions.

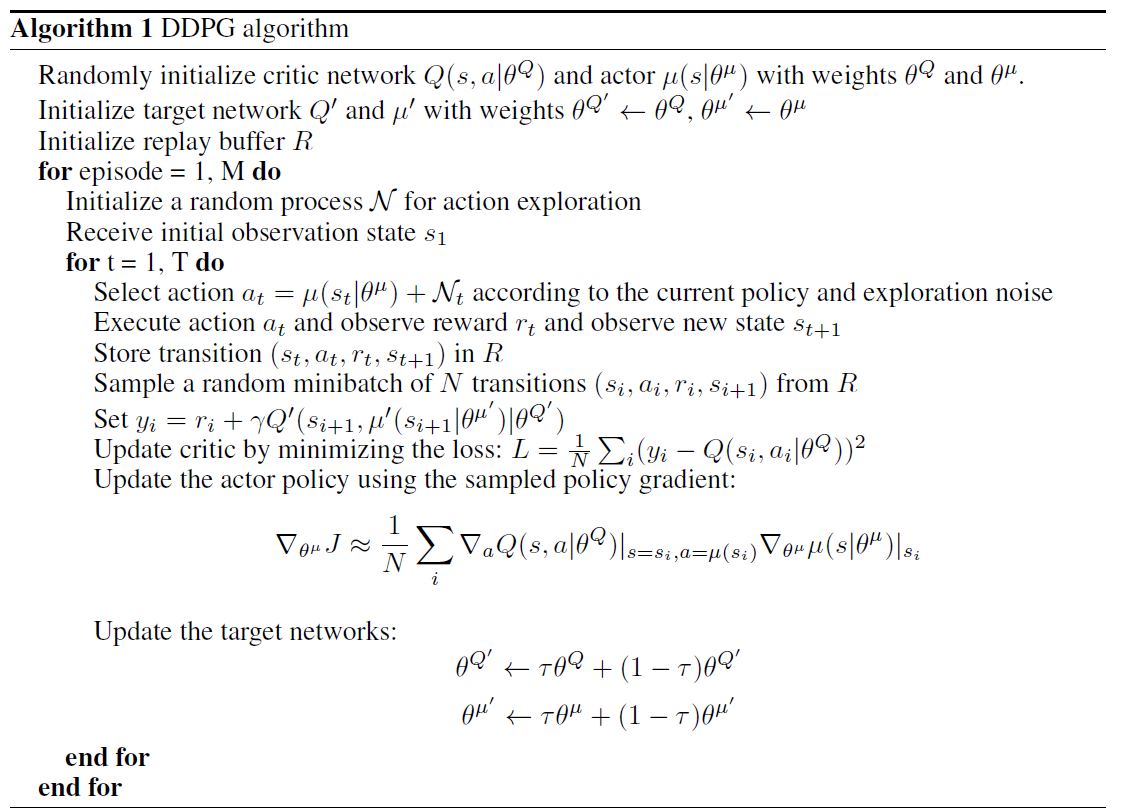


Figure. 1‑1The fake code of alg.

# TD3 algorithm

## Principle of algorithm

Based on the DDPG algorithm, the TD3 algorithm proposes three key technologies:

(1) Double network: Two sets of Critic networks are used, and the smaller of the two is used when calculating the target value, thereby suppressing the problem of network overestimation.

(2) Target policy smoothing regularization: When calculating the target value, perturbations are added to the action in the next state, thereby making the value assessment more accurate.

(3) Delayed update: After the Critic network is updated multiple times, the Actor network is updated to ensure that the training of the Actor network is more stable.

### Dual network

The TD3 algorithm includes six networks, namely Actor network , Critic1 network , Critic2 network, Target Actor network, Target Critic1 Network ​​​​​​​, Target Critic2 Network. Compared with the DDPG algorithm, the TD3 algorithm has an additional critic network, which is the origin of the dual network.

### Causes of network overestimation

The overestimation of the DDPG algorithm mainly comes from two aspects: bootstrapping and maximization. If the overestimation is uniform, it will not have an impact on the agent’s final decision; if it is non-uniform, it will have no impact on the agent’s final decision. Decisions have significant consequences. However, in fact, the overestimation of the network is usually non-uniform. Here is a brief analysis of the reasons.

When updating the Critic network, assume that the data sampled from the experience pool is . First we will calculate the target y



Because of overestimation:



Among them, represents the real optimal state-action State action value.

Then we will make approach y, so that  will be overestimated, that is



Each time the state-action \left ( s,a \right )is sampled to update the Critic network, the network will overestimate the value of the state-action of \left ( s,a \right ), while \left ( s,a \right )The frequency in the experience pool is obviously uneven. The higher the frequency, the greater the overestimation. Therefore, the network's overestimation is non-uniform, and non-uniform overestimation is harmful to the agent's decision-making, so we need to avoid network overestimation.

## Why we use dual network

Assume that the true state-action value of each state-action pair is . There will be some noise in the estimation of the Q network. Assuming it is an unbiased estimation, then the estimated state action value is. Since the mean value of noise is 0, it satisfies



 is a typical overestimation, that is



When we calculate the target value, we will execute. Due to overestimation of , the target value will also be overestimated. When the network is updated we will approximate  to y and due to the overestimation of y, there will be an overestimation.

To sum up, the maximization operation will make the estimated value of the network larger than the true value, causing the network to overestimate.

Dual networks are an efficient way to solve maximization problems. In the TD3 algorithm, the author introduces two sets of critic networks with the same network architecture. When calculating the target value, the smaller value between the two is used to estimate the state action value of the next state action pair, that is



## Target policy smoothing regularization

There is a problem with the deterministic strategy: it overfits to shrink the peaks in value estimates. When updating the Critic network, the learning target using a deterministic strategy is extremely susceptible to function approximation errors, resulting in large variance in target estimation and inaccurate estimation values. This induced variance can be reduced through regularization, so the author imitates SARSA's learning update and introduces a regularization strategy for deep value learning - target policy smoothing.

This approach mainly emphasizes that similar actions should have similar values. While the function approximation achieves this implicitly, the relationship between similar actions can be emphasized explicitly by modifying the training process. The specific implementation is to use the area around the target action to calculate the target value, which is beneficial to smoothing the estimated value.

y=r+E\left [ Q^{'}\left ( s^{'}, \mu^{'}\left ( s^{'} \mid \theta^{'} \right ) +\epsilon \mid \theta^{Q^{'}} \right ) \right ]

In practice, we can approximate the expectation of an action by adding a small amount of random noise to the target action and averaging it over a mini-batch. Therefore, the above formula can be modified as





## Delay update

The delayed update here refers to the delayed update of the Actor network, that is, the Actor network is updated after the Critic network is updated multiple times. This idea is actually very intuitive, because the Actor network is updated by maximizing the cumulative expected return, and it needs to be evaluated using the Critic network. If the Critic network is very unstable, then the Actor network will naturally fluctuate as well.

Therefore, we can make the update frequency of the Critic network higher than that of the Actor network, that is, wait for the Critic network to become more stable before helping the Actor network to update.

## Process of updating

The update process of the TD3 algorithm is not much different from that of the DDPG algorithm. The main difference lies in the calculation method of the target value. The Actor network is updated by maximizing the cumulative expected return (deterministic policy gradient), the Critic1 and Critic2 networks are updated by minimizing the error between the evaluation value and the target value (MSE), and all target networks use soft updates. to update (Exponential Moving Average, EMA). In the training phase, we sample a batch (Batch size) of data from the Replay Buffer. Assuming that the sampled piece of data is, the update process of all networks is as follows.

Critic1 and Critic2 network update process: using the Target Actor network to calculate the actions in the state



Then based on the smooth regularization of the target strategy, noise is added to the target action



\epsilon \sim clip\left ( N\left ( 0,\sigma \right ),-c,c \right )

Target value:



Finally, the gradient descent algorithm is used to minimize the error  between the evaluation value and the target value, thereby updating the parameters in the Critic1 and Critic2 networks.

L_{c_{i}}=\left ( Q_{i}\left ( s,a \mid \theta^{Q_{i}} \right )-y \right )^{2} \left ( i=1,2 \right )

Actor network update process: (After the Ctitic1 and Critic2 network update step d, start the Actor network update) Use the Actor network to calculate the action in state s



Then use the Critic1 or Critic2 network to calculate the evaluation value of the state action . Here we assume that the Critic1 network is used



Finally, the gradient ascent algorithm is used to maximize , thereby completing the update of the Actor network.

The update process of the target network: Use soft update method to update the target network. Introduce a learning rate (or become momentum) , do a weighted average of the old target network parameters and the new corresponding network parameters, and then assign them to the target network





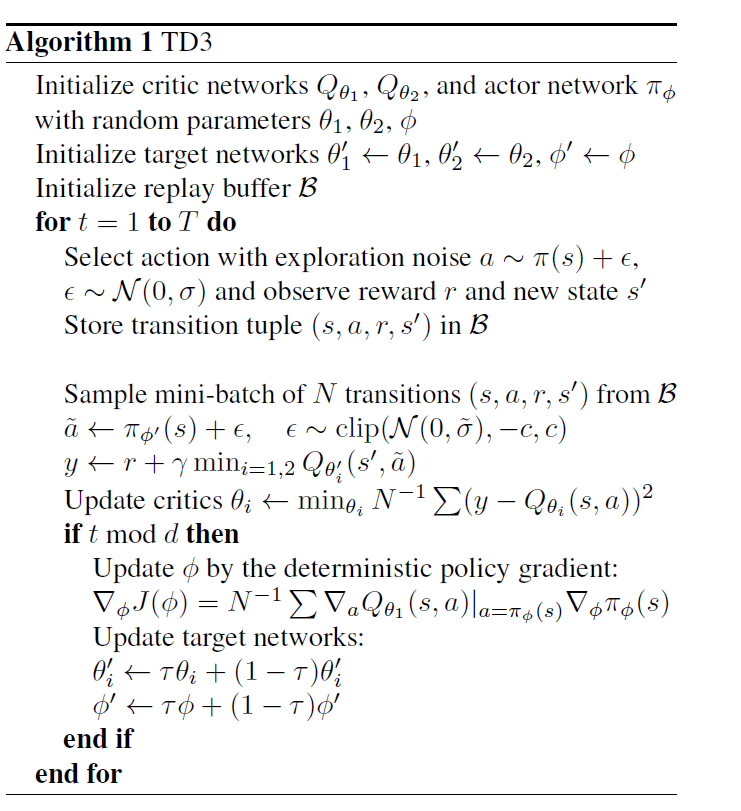


Figure. 2‑1The fake code of alg.

# Application of TD3 ALG.

## ENVIROMENT

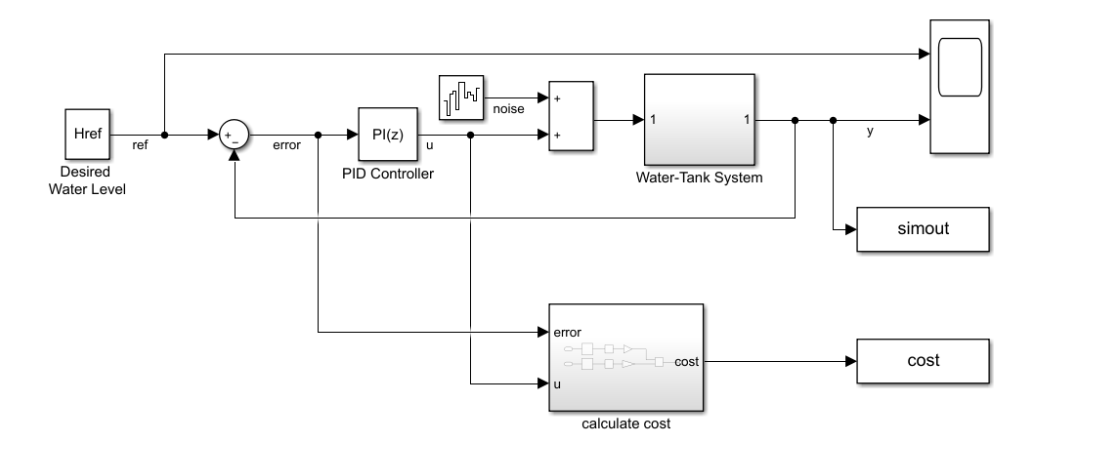


Figure. 3‑1The model.

The model includes process noise with variance .

To maintain the water level while minimizing control effort u, the controllers use the following LQG criterion.



To simulate the controller in this model, you must specify the simulation time Tf and the controller sample time Ts in seconds.

To simulate the controller in this model, you must specify the simulation time Tf and the controller sample time Ts in seconds.

1. open\_system('watertankLQG');
2. Ts = 0.1;
3. Tf = 10;

## Tune PI Controller Using Control System Tuner

1. controlSystemTuner("ControlSystemTunerSession")
2. Kp\_CST = 9.80199999804512;
3. Ki\_CST = 1.00019996230706e-06;

The tuned proportional and integral gains are approximately 9.8 and 1e-6, respectively.

## Create Environment for Training Agent

To define the model for training the RL agent, modify the water tank model using the following steps.

1. Delete the PID Controller.
2. Insert an RL Agent block.
3. Create the observation vector [∫*e* dt*e*]*T*  where *e*=*Href*−*h*, *h* is the height of the water in the tank, and *Href* is the reference water height. Connect the observation signal to the RL Agent block.
4. Define the reward function for the RL agent as the **negative** of the LQG cost , that is, Reward=−((*Href*−*h*(*t*))2+0.01*u*2(*t*)). The RL agent maximizes this reward, thus minimizing the LQG cost.
5. mdl = 'rlwatertankPIDTune';
6. open\_system(mdl)
7. [env,obsInfo,actInfo] = localCreatePIDEnv(mdl);
8. numObservations = obsInfo.Dimension(1);
9. numActions = prod(actInfo.Dimension);
10. rng(0)

## Create TD3 Agent

To create the actor, first create a deep neural network with the observation input and the action output. For more information, see [rlContinuousDeterministicActor](https://www.mathworks.com/help/reinforcement-learning/ref/rl.function.rlcontinuousdeterministicactor.html).

You can model a PI controller as a neural network with one fully-connected layer with error and error integral observations.

*u*= [∫*e* dt*e*]∗[*KiKp*]*T*

Here:

* u is the output of the actor neural network.
* Kp and Ki are the absolute values of the neural network weights.
* *e*=*Href*−*h*(*t*), *h*(*t*) is the height of the tank, and *Href* is the reference height.

1. initialGain = single([1e-3 2]);
2. actorNetwork = [
3. featureInputLayer(numObservations,'Normalization','none','Name','state')
4. fullyConnectedPILayer(initialGain, 'Action')];
5. actorOptions = rlRepresentationOptions('LearnRate',1e-3,'GradientThreshold',1);
6. actor = rlDeterministicActorRepresentation(actorNetwork,obsInfo,actInfo,...
7. 'Observation',{'state'},'Action',{'Action'},actorOptions);
8. criticNetwork = localCreateCriticNetwork(numObservations,numActions);
9. criticOpts = rlRepresentationOptions('LearnRate',1e-3,'GradientThreshold',1);
10. critic1 = rlQValueRepresentation(criticNetwork,obsInfo,actInfo,...
11. 'Observation','state','Action','action',criticOpts);
12. critic2 = rlQValueRepresentation(criticNetwork,obsInfo,actInfo,...
13. 'Observation','state','Action','action',criticOpts);
14. critic = [critic1 critic2];
15. agentOpts = rlTD3AgentOptions(...
16. 'SampleTime',Ts,...
17. 'MiniBatchSize',128, ...
18. 'ExperienceBufferLength',1e6);
19. agentOpts.ExplorationModel.Variance = 0.1;
20. agentOpts.TargetPolicySmoothModel.Variance = 0.1;
21. agent = rlTD3Agent(actor,critic,agentOpts);

## Train TD3

To train the agent, first specify the following training options.

* Run each training for at most 1000 episodes, with each episode lasting at most 100 time steps.
* Display the training progress in the Episode Manager (set the Plots option) and disable the command-line display (set the Verbose option).
* Stop training when the agent receives an average cumulative reward greater than -355 over 100 consecutive episodes. At this point, the agent can control the level of water in the tank
* maxepisodes = 1000;
* maxsteps = ceil(Tf/Ts);
* trainOpts = rlTrainingOptions(...
* 'MaxEpisodes',maxepisodes, ...
* 'MaxStepsPerEpisode',maxsteps, ...
* 'ScoreAveragingWindowLength',100, ...
* 'Verbose',false, ...
* 'Plots','training-progress',...
* 'StopTrainingCriteria','AverageReward',...
* 'StopTrainingValue',-355);
* if doTraining
* % Train the agent.
* traintrainingStats = train(agent,env,trainOpts);
* else
* % Load pretrained agent for the example.
* load('WaterTankPIDtd3.mat','agent')
* end

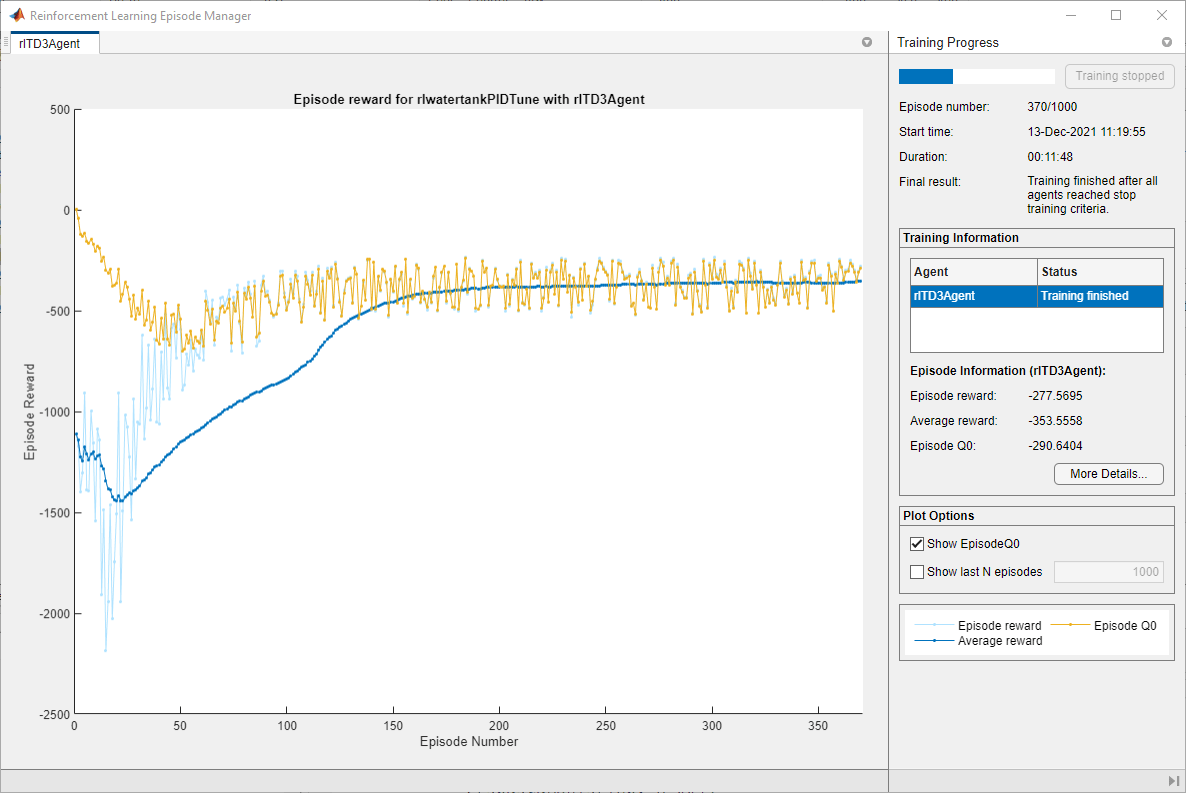


Figure. 3‑2 The result-1.

## Validate Trained Agent

Validate the learned agent against the model by simulation.

1. simOpts = rlSimulationOptions('MaxSteps',maxsteps);
2. experiences = sim(env,agent,simOpts);

The integral and proportional gains of the PI controller are the absolute weights of the actor representation. To obtain the weights, first extract the learnable parameters from the actor.

1. actor = getActor(agent);
2. parameters = getLearnableParameters(actor);

Obtain the controller gains.

1. Ki = abs(parameters{1}(1));
2. Kp = abs(parameters{1}(2));

Apply the gains obtained from the RL agent to the original PI controller block and run a step-response simulation.

1. mdlTest = 'watertankLQG';
2. open\_system(mdlTest);
3. set\_param([mdlTest '/PID Controller'],'P',num2str(Kp))
4. set\_param([mdlTest '/PID Controller'],'I',num2str(Ki))
5. sim(mdlTest)

Apply the gains obtained from the RL agent to the original PI controller block and run a step-response simulation.

1. mdlTest = 'watertankLQG';
2. open\_system(mdlTest);
3. set\_param([mdlTest '/PID Controller'],'P',num2str(Kp))
4. set\_param([mdlTest '/PID Controller'],'I',num2str(Ki))
5. sim(mdlTest)

Extract the step response information, LQG cost, and stability margin for the simulation.

1. rlStep = simout;
2. rlCost = cost;
3. rlStabilityMargin = localStabilityAnalysis(mdlTest);

Apply the gains obtained using Control System Tuner to the original PI controller block and run a step-response simulation.

1. set\_param([mdlTest '/PID Controller'],'P',num2str(Kp\_CST))
2. set\_param([mdlTest '/PID Controller'],'I',num2str(Ki\_CST))
3. sim(mdlTest)
4. cstStep = simout;
5. cstCost = cost;
6. cstStabilityMargin = localStabilityAnalysis(mdlTest);

## Compare Controller Performance

Plot the step response for each system.

1. figure
2. plot(cstStep)
3. hold on
4. plot(rlStep)
5. grid on
6. legend('Control System Tuner','RL','Location','southeast')
7. title('Step Response')

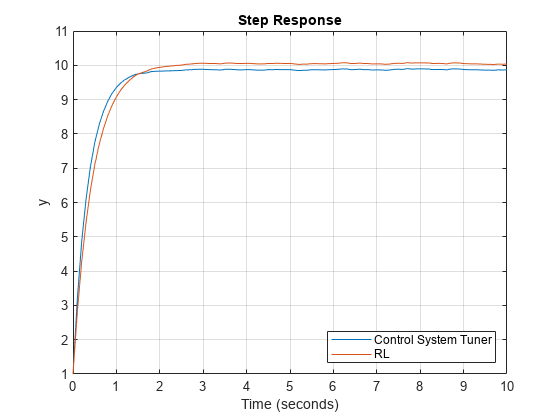


Figure. 3‑3 The result-2

Analyze the step response for both simulations.

1. rlStepInfo = stepinfo(rlStep.Data,rlStep.Time);
2. cstStepInfo = stepinfo(cstStep.Data,cstStep.Time);
3. stepInfoTable = struct2table([cstStepInfo rlStepInfo]);
4. stepInfoTable = removevars(stepInfoTable,{...
5. 'SettlingMin','SettlingMax','Undershoot','PeakTime'});
6. stepInfoTable.Properties.RowNames = {'Control System Tuner','RL'};
7. stepInfoTable

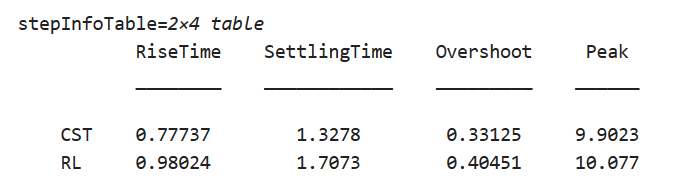


Figure. 3‑4 The result-3

Analyze the stability for both simulations.

1. stabilityMarginTable = struct2table([cstStabilityMargin rlStabilityMargin]);
2. stabilityMarginTable = removevars(stabilityMarginTable,{...
3. 'GMFrequency','PMFrequency','DelayMargin','DMFrequency'});
4. stabilityMarginTable.Properties.RowNames = {'Control System Tuner','RL'};
5. stabilityMarginTable

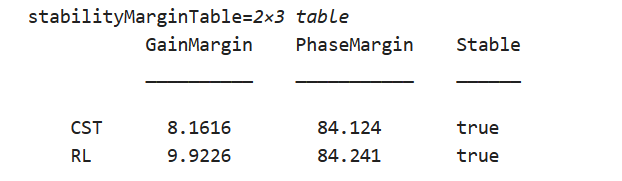


Figure. 3‑5 The result-4

Reference

[1] Fujimoto S, Hoof H, Meger D. Addressing function approximation error in actor-critic methods[C]//International conference on machine learning. PMLR, 2018: 1587-1596.

[2] Lillicrap T P, Hunt J J, Pritzel A, et al. Continuous control with deep reinforcement learning[J]. arXiv preprint arXiv:1509.02971, 2015.

[3] [以编程方式运行仿真 - MATLAB & Simulink (mathworks.com)](https://www.mathworks.com/help/simulink/ug/using-the-sim-command_zh_CN.html)