Lab6: Create Simulink Environment and Train Agent - RL based Water Level Control of Water Tank

Abhishek M J - CS21B2018

30-03-2024

Water Tank Model

The original model for this lab is the water tank model. The goal is to control the level of the water in the tank.

```
open_system("rlwatertank")
```

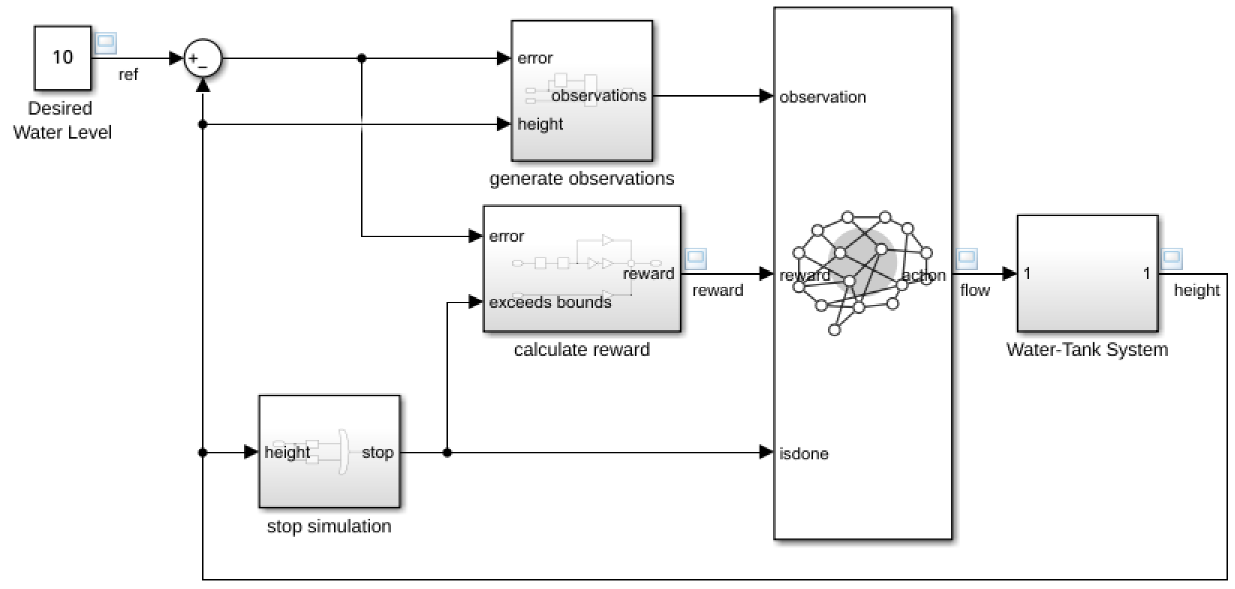


Figure 1: Water Tank Model

Create the Environment

Creating an environment model includes defining the following:

- Action and observation signals that the agent uses to interact with the environment.
- Reward signal that the agent uses to measure its success.

```
% Observation info
obsInfo = rlNumericSpec([3 1],...
    LowerLimit=[-inf -inf 0  ]',...
    UpperLimit=[ inf  inf inf]');
```

```
% Name and description are optional and not used by the software
obsInfo.Name = "observations";
obsInfo.Description = "integrated error, error, and measured height";

% Action info
actInfo = rlNumericSpec([1 1]);
actInfo.Name = "flow";
```

Create the environment object.

```
env = rlSimulinkEnv("rlwatertank","rlwatertank/RL Agent",...
    obsInfo,actInfo);
```

Set a custom reset function that randomizes the reference values for the model.

```
env.ResetFcn = @(in)localResetFcn(in);
```

Specify the simulation time Tf and the agent sample time Ts in seconds.

```
Ts = 1.0;
Tf = 200;
```

Fix the random generator seed for reproducibility.

```
rng(0)
```

Create the Critic

DDPG agents use a parametrized Q-value function approximator to estimate the value of the policy. A Q-value function critic takes the current observation and an action as inputs and returns a single scalar as output (the estimated discounted cumulative long-term reward for which receives the action from the state corresponding to the current observation, and following the policy thereafter).

To model the parametrized Q-value function within the critic, use a neural network with two input layers (one for the observation channel, as specified by obsInfo, and the other for the action channel, as specified by actInfo) and one output layer (which returns the scalar value).

Define each network path as an array of layer objects. Assign names to the input and output layers of each path. These names allow you to connect the paths and then later explicitly associate the network input and output layers with the appropriate environment channel. Obtain the dimension of the observation and action spaces from the obsInfo and actInfo specifications.

```
% Observation path
obsPath = [
    featureInputLayer(obsInfo.Dimension(1),Name="obsInLyr")
    fullyConnectedLayer(50)
    reluLayer
    fullyConnectedLayer(25,Name="obsPathOutLyr")
    ];

% Action path
actPath = [
    featureInputLayer(actInfo.Dimension(1),Name="actInLyr")
    fullyConnectedLayer(25,Name="actPathOutLyr")
    ];

% Common path
commonPath = [
    additionLayer(2,Name="add")
    reluLayer
    fullyConnectedLayer(1,Name="QValue")
    ];

% Create the network object and add the layers
criticNet = dlnetwork();
criticNet = addLayers(criticNet,obsPath);
criticNet = addLayers(criticNet,actPath);
criticNet = addLayers(criticNet,commonPath);

% Connect the layers
criticNet = connectLayers(criticNet, ...
    "obsPathOutLyr","add/in1");
criticNet = connectLayers(criticNet, ...
    "actPathOutLyr","add/in2");
```

View the critic network configuration.

```
plot(criticNet)
```

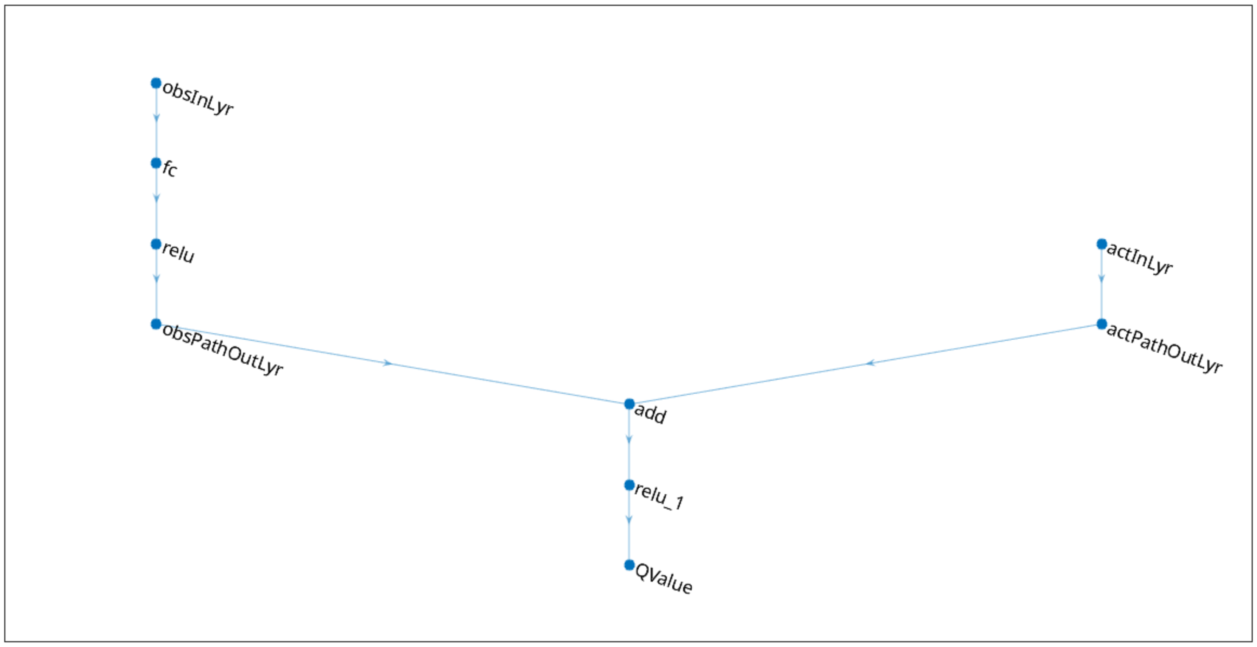


Figure 2: Critic Network

Initialize the dlnetwork object and summarize its properties.

```
criticNet = initialize(criticNet);
summary(criticNet)
```

Create the critic approximator object using the specified deep neural network, the environment specification objects, and the names if the network inputs to be associated with the observation and action channels.

```
critic = rlQValueFunction(criticNet, ...
    obsInfo,actInfo, ...
    ObservationInputNames="obsInLyr", ...
    ActionInputNames="actInLyr");
```

Check the critic with a random input observation and action.

```
getValue(critic, ...
    {rand(obsInfo.Dimension)}, ...
    {rand(actInfo.Dimension)})
```

Create the Actor

DDPG agents use a parametrized deterministic policy over continuous action spaces, which is learned by a continuous deterministic actor.

A continuous deterministic actor implements a parametrized deterministic policy for a continuous action space. This actor takes the current observation as input and returns as output an action that is a deterministic function of the observation.

To model the parametrized policy within the actor, use a neural network with one input layer (which receives the content of the environment observation channel, as specified by obsInfo) and one output layer (which returns the action to the environment action channel, as specified by actInfo).

Define the network as an array of layer objects.

```
actorNet = [
    featureInputLayer(obsInfo.Dimension(1))
    fullyConnectedLayer(3)
    tanhLayer
    fullyConnectedLayer(actInfo.Dimension(1))
    ];
```

Convert the network to a dlnetwork object and summarize its properties.

```
actorNet = dlnetwork(actorNet);
summary(actorNet)
```

Create the actor approximator object using the specified deep neural network, the environment specification objects, and the name if the network input to be associated with the observation channel.

```
actor = rlContinuousDeterministicActor(actorNet,obsInfo,actInfo);
```

Check the actor with a random input observation.

```
getAction(actor,rand(obsInfo.Dimension))
```

Create the DDPG Agent

Create the DDPG agent using the specified actor and critic approximator objects.

```
agent = rlDDPGAgent(actor,critic);
```

Specify options for the agent, the actor, and the critic using dot notation.

```
agent.SampleTime = Ts;

agent.AgentOptions.TargetSmoothFactor = 1e-3;
agent.AgentOptions.DiscountFactor = 1.0;
agent.AgentOptions.MiniBatchSize = 64;
agent.AgentOptions.ExperienceBufferLength = 1e6;

agent.AgentOptions.NoiseOptions.Variance = 0.3;
agent.AgentOptions.NoiseOptions.VarianceDecayRate = 1e-5;
agent.AgentOptions.CriticOptimizerOptions.LearnRate = 1e-03;
agent.AgentOptions.CriticOptimizerOptions.GradientThreshold = 1;
agent.AgentOptions.ActorOptimizerOptions.LearnRate = 1e-04;
agent.AgentOptions.ActorOptimizerOptions.GradientThreshold = 1;
```

Check the agent with a random input observation.

```
getAction(agent,{rand(obsInfo.Dimension)})
```

Train Agent

To train the agent, first specify the training options. For this example, use the following options:

- Run each training for at most 5000 episodes. Specify that each episode lasts for at most ceil(Tf/Ts) (that is 200) time steps.
- Display the training progress in the Episode Manager dialog box (set the Plots option) and disable the command line display (set the Verbose option to false).
- Stop training when the agent receives an average cumulative reward greater than 800 over 20 consecutive episodes. At this point, the agent can control the level of water in the tank.

```
trainOpts = rlTrainingOptions(...
    MaxEpisodes=5000, ...
    MaxStepsPerEpisode=ceil(Tf/Ts), ...
    ScoreAveragingWindowLength=20, ...
    Verbose=false, ...
    Plots="training-progress",...
    StopTrainingCriteria="AverageReward",...
    StopTrainingValue=800);
```

Train the agent using the train function. Training is a computationally intensive process that takes several minutes to complete.

```
doTraining = true;

if doTraining
    % Train the agent.
    trainingStats = train(agent,env,trainOpts);
else
    % Load the pretrained agent for the example.
    load("WaterTankDDPG.mat","agent")
end
```

Validate Trained Agent

Validate the learned agent against the model by simulation. Since the reset function randomizes the reference values, fix the random generator seed to ensure simulation reproducibility.

```
rng(1)
```

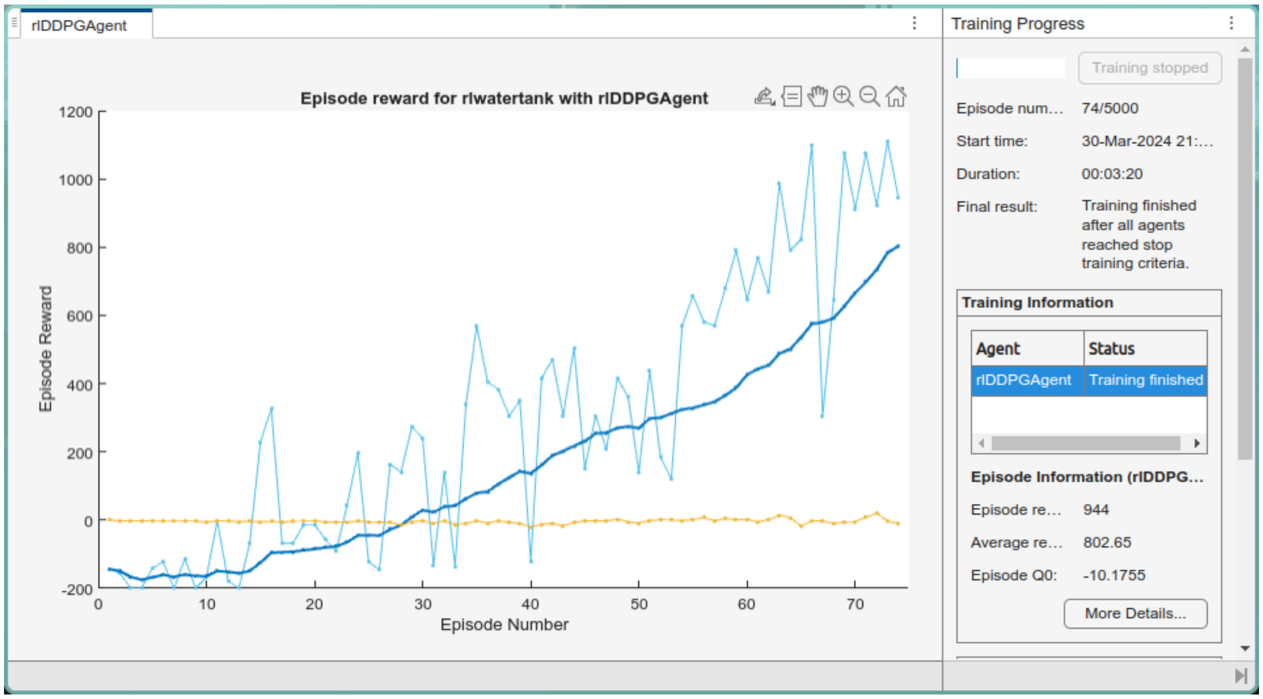
Simulate the agent within the environment, and return the experiences as output.

```
simOpts = rlSimulationOptions(MaxSteps=ceil(Tf/Ts),StopOnError="on");
experiences = sim(env,agent,simOpts);
```

Local Reset Function

```
function in = localResetFcn(in)

% Randomize reference signal
```



Episode num…	74/5000
Start time:	30-Mar-2024 21:…
Duration:	00:03:20
Final result:	Training finished after all agents reached stop training criteria.

Training Information

Agent	Status
rlDDPGAgent	Training finished

Episode Information (rlDDPG…

Episode re…	944
Average re…	802.65
Episode Q0:	-10.1755

Figure 3: Training Progress

```
blk = sprintf("rlwatertank/Desired \nWater Level");
h = 3*randn + 10;
while h <= 0 || h >= 20
    h = 3*randn + 10;
end
in = setBlockParameter(in,blk,Value=num2str(h));

% Randomize initial height
h = 3*randn + 10;
while h <= 0 || h >= 20
    h = 3*randn + 10;
end
blk = "rlwatertank/Water-Tank System/H";
in = setBlockParameter(in,blk,InitialCondition=num2str(h));

end
```

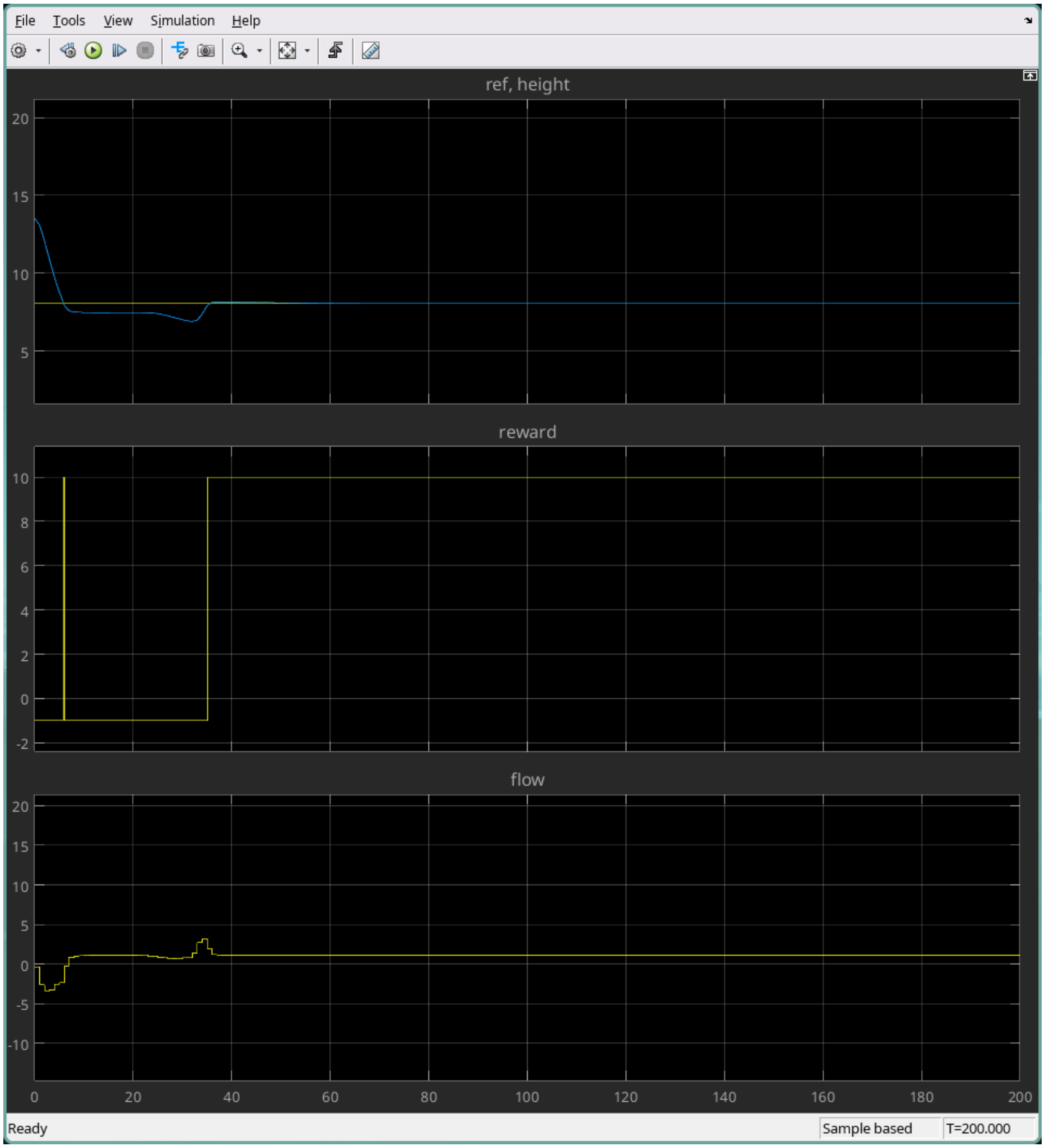
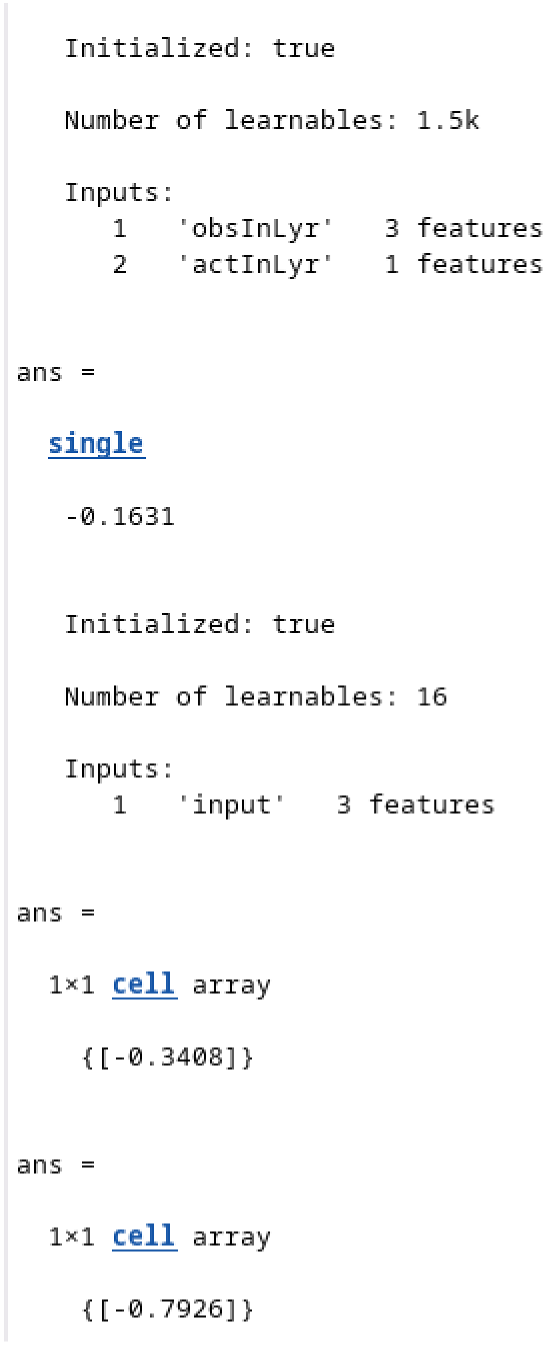


Figure 4: Simulation



```
   Initialized: true

   Number of learnables: 1.5k

   Inputs:
      1   'obsInLyr'   3 features
      2   'actInLyr'   1 features


ans =

  single

   -0.1631


   Initialized: true

   Number of learnables: 16

   Inputs:
      1   'input'   3 features


ans =

  1×1 cell array

    {[-0.3408]}


ans =

  1×1 cell array

    {[-0.7926]}
```

Figure 5: Output