First order system control with Artificial intelligent

Reinforcement Learning algorithm

Deep Deterministic Policy Gradient algorithm (DDPG)

Using MATLAB 2019b

Script:

open\_system('FirstOrderSystemControl')

obsInfo = rlNumericSpec([3 1],...

'LowerLimit',[-inf -inf 0 ]',...

'UpperLimit',[ inf inf inf]');

obsInfo.Name = 'observations';

obsInfo.Description = 'integrated error, error, and respons';

numObservations = obsInfo.Dimension(1);

actInfo = rlNumericSpec([1 1]);

actInfo.Name = 'flow';

numActions = actInfo.Dimension(1);

env = rlSimulinkEnv('FirstOrderSystemControl','FirstOrderSystemControl/RL Agent',...

obsInfo,actInfo);

Ts = 1.0;

Tf = 200;

rng(0)

statePath = [

imageInputLayer([numObservations 1 1],'Normalization','none','Name','State')

fullyConnectedLayer(50,'Name','CriticStateFC1')

reluLayer('Name','CriticRelu1')

fullyConnectedLayer(25,'Name','CriticStateFC2')];

actionPath = [

imageInputLayer([numActions 1 1],'Normalization','none','Name','Action')

fullyConnectedLayer(25,'Name','CriticActionFC1')];

commonPath = [

additionLayer(2,'Name','add')

reluLayer('Name','CriticCommonRelu')

fullyConnectedLayer(1,'Name','CriticOutput')];

criticNetwork = layerGraph();

criticNetwork = addLayers(criticNetwork,statePath);

criticNetwork = addLayers(criticNetwork,actionPath);

criticNetwork = addLayers(criticNetwork,commonPath);

criticNetwork = connectLayers(criticNetwork,'CriticStateFC2','add/in1');

criticNetwork = connectLayers(criticNetwork,'CriticActionFC1','add/in2');

figure

plot(criticNetwork)

criticOpts = rlRepresentationOptions('LearnRate',1e-03,'GradientThreshold',1);

critic = rlRepresentation(criticNetwork,obsInfo,actInfo,'Observation',{'State'},'Action',{'Action'},criticOpts);

actorNetwork = [

imageInputLayer([numObservations 1 1],'Normalization','none','Name','State')

fullyConnectedLayer(3, 'Name','actorFC')

tanhLayer('Name','actorTanh')

fullyConnectedLayer(numActions,'Name','Action')

];

actorOptions = rlRepresentationOptions('LearnRate',1e-04,'GradientThreshold',1);

actor = rlRepresentation(actorNetwork,obsInfo,actInfo,'Observation',{'State'},'Action',{'Action'},actorOptions);

agentOpts = rlDDPGAgentOptions(...

'SampleTime',Ts,...

'TargetSmoothFactor',1e-3,...

'DiscountFactor',1.0, ...

'MiniBatchSize',64, ...

'ExperienceBufferLength',1e6);

agentOpts.NoiseOptions.Variance = 0.3;

agentOpts.NoiseOptions.VarianceDecayRate = 1e-5;

agent = rlDDPGAgent(actor,critic,agentOpts);

maxepisodes = 500;

maxsteps = ceil(Tf/Ts);

trainOpts = rlTrainingOptions(...

'MaxEpisodes',1000,...

'MaxStepsPerEpisode',500,...

'ScoreAveragingWindowLength',10,...

'Verbose',false,...

'Plots','training-progress',...

'StopTrainingCriteria','AverageReward',...

'StopTrainingValue',10000,...

'SaveAgentCriteria','EpisodeReward',...

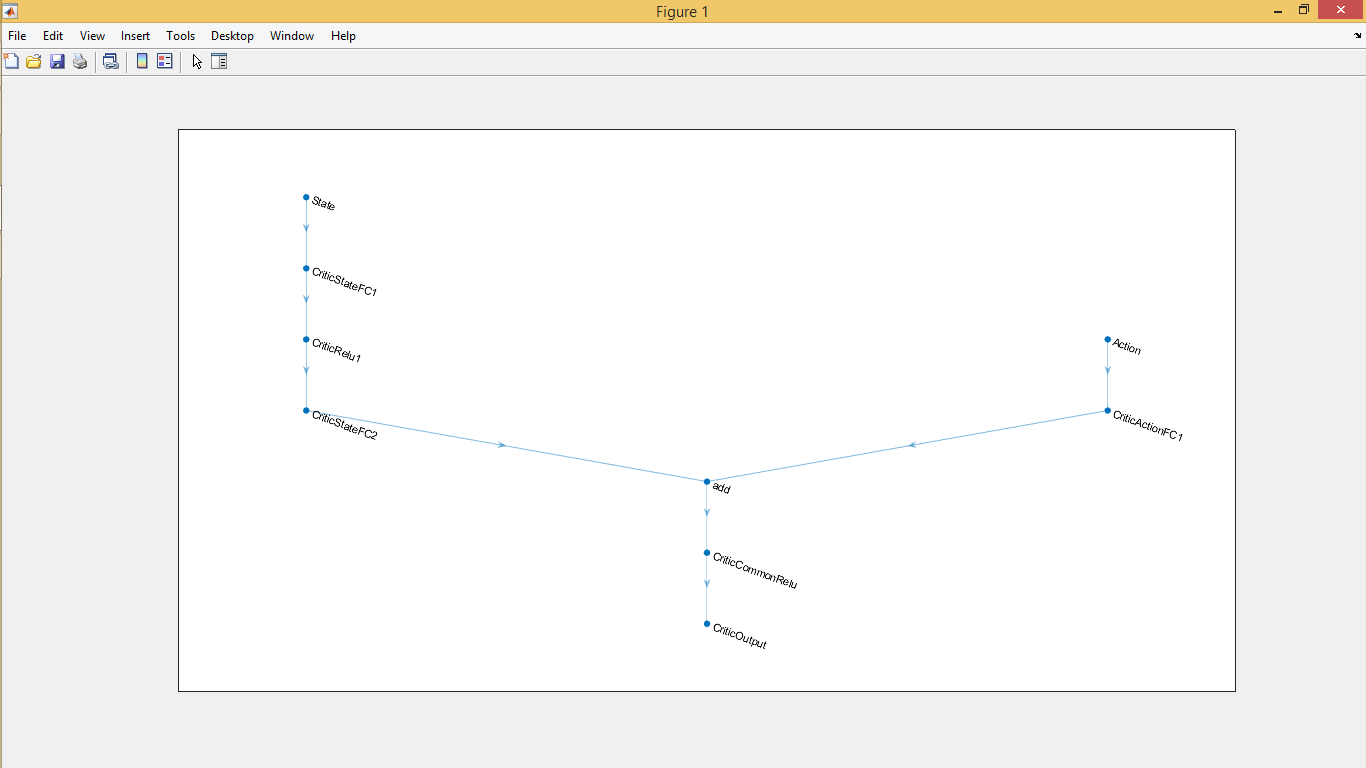
'SaveAgentValue',10000);

trainingStats = train(agent,env,trainOpts);

simOpts = rlSimulationOptions('MaxSteps',maxsteps,'StopOnError','on');

experiences = sim(env,agent,simOpts);

Neural Network graphic representation:



Simulink control:

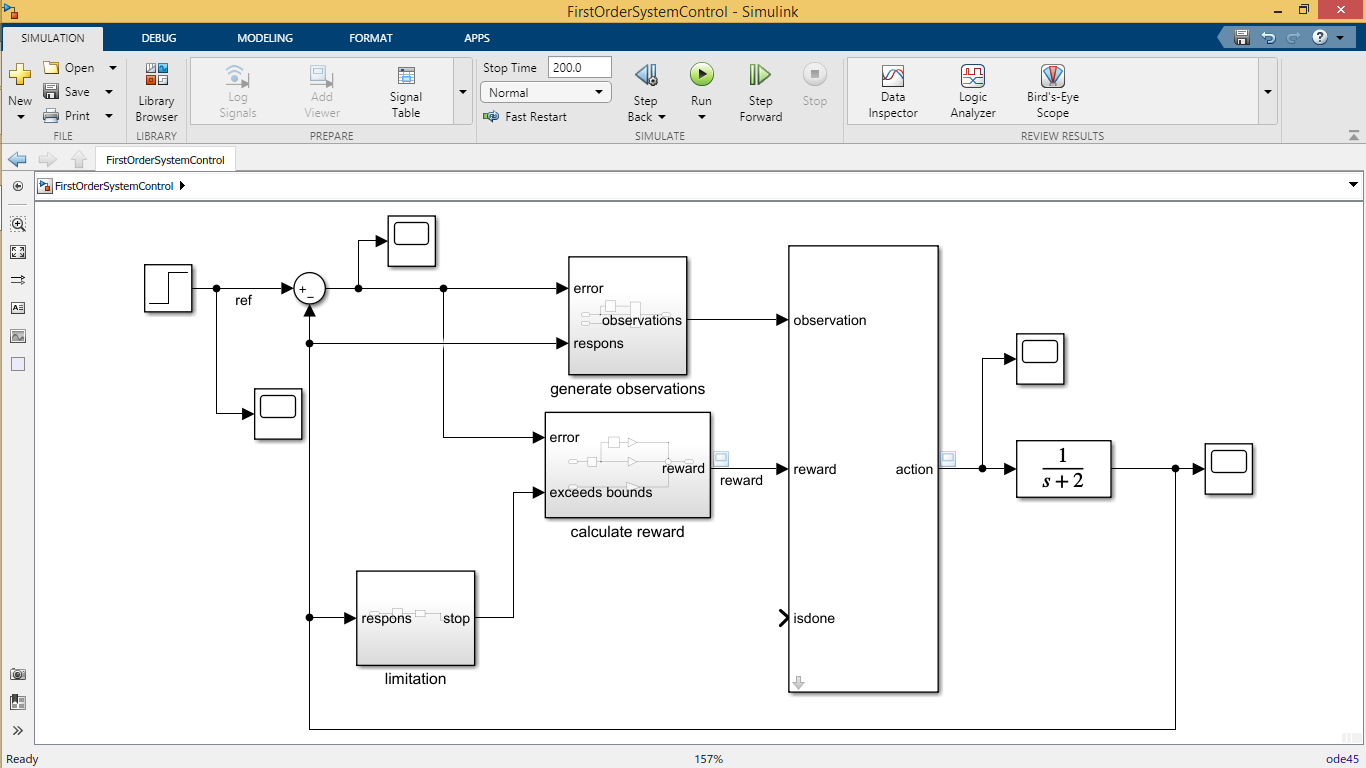


Figure1: Simulink simulation

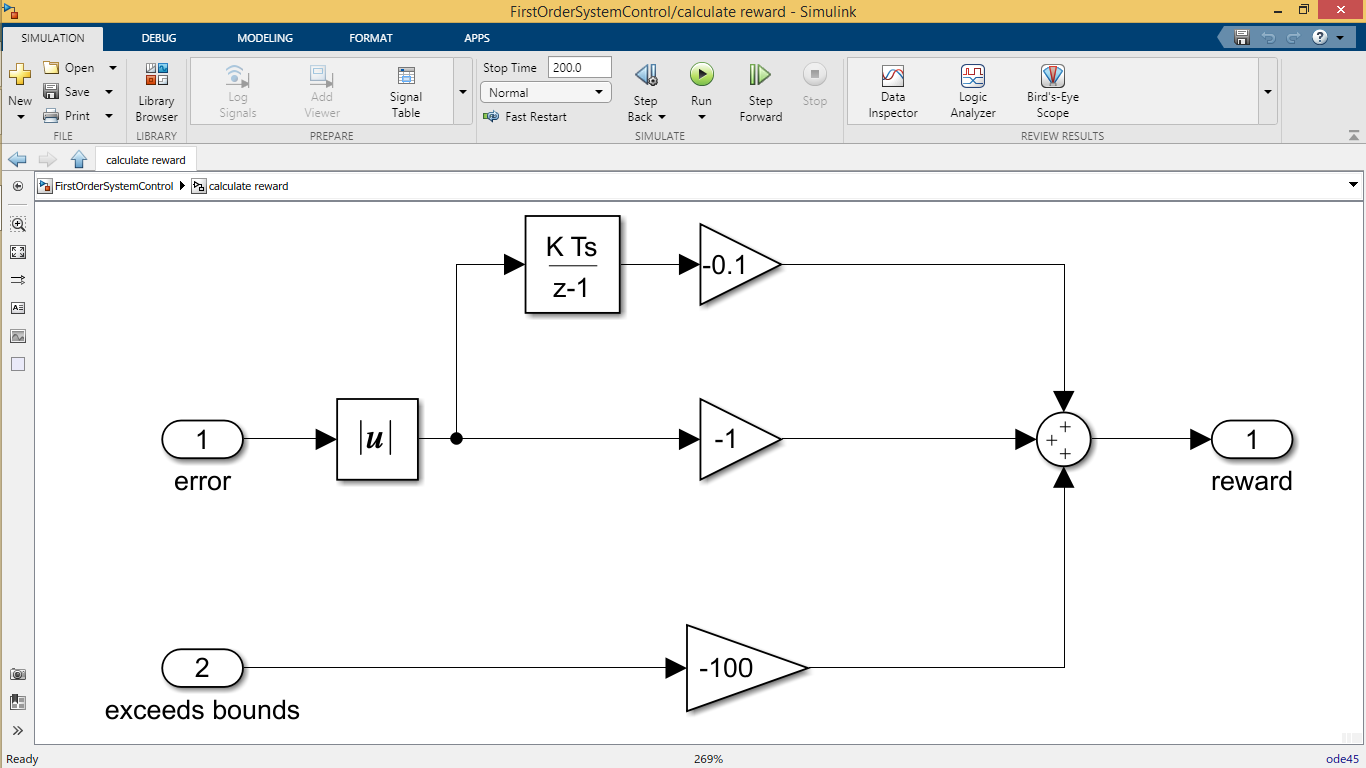
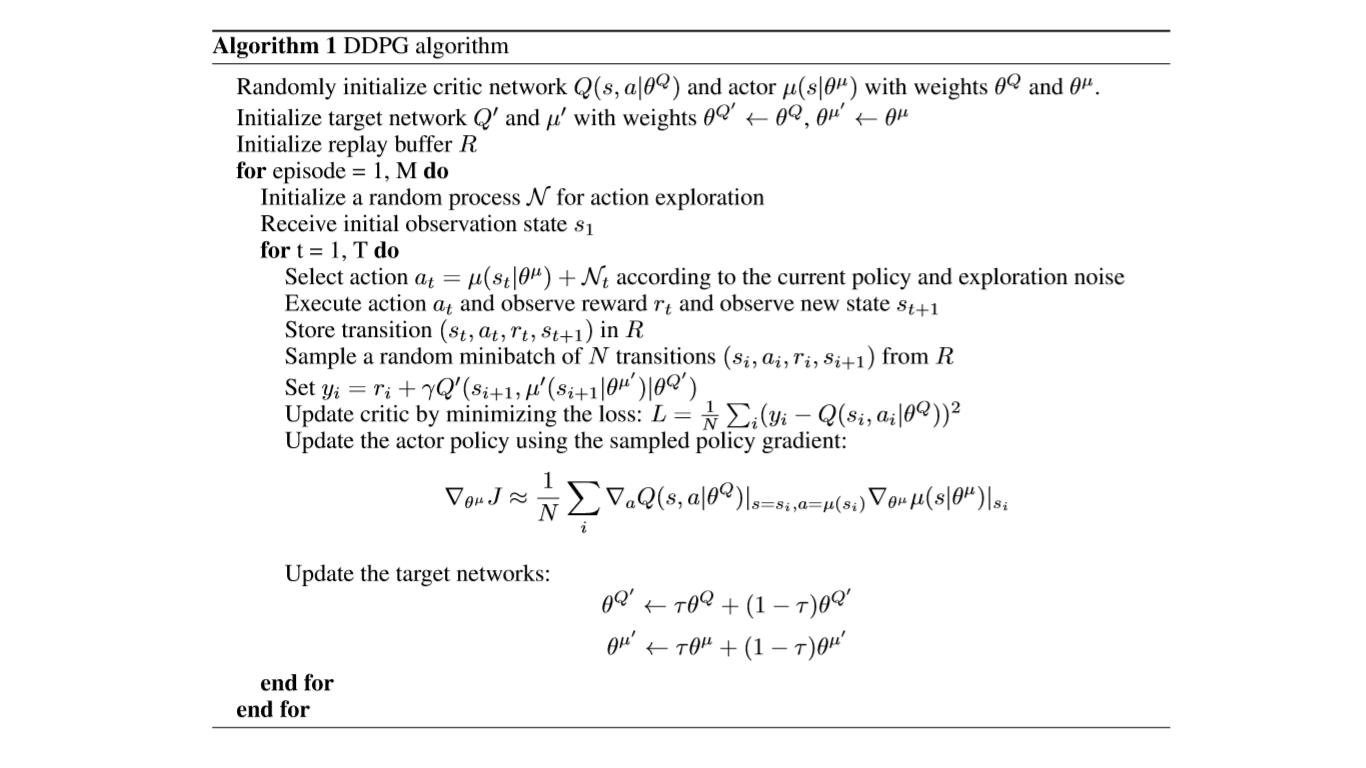


Figure2: Simulink reward function

\_Algorithm:

******

* **Training:**

We trained this model on a random Inputs between zero and one randomly and for 200 episodes.

The model trained by Deep Deterministic policy Gradient algorithm takes two networks for training (Critic(Q-learning) & Actor (Policy Optimization) with continuous actions space.

* **Reward function:**

The reward function represents the goal of the experience. It takes the derivative of error as negative (-0.1) in each step it helps on elimination of the variation and vibration of the system response. The second function is the error with a negative (-1) in each step to guaranteed the precision of system response. The last reward is a negative (-100) to avoid any exceeding of error (>1.5).

* **Observations:**

For more inputs training information, we toke the error and the derivative of error and response of the system as input for the RL model.

* **Neural network:**

Medium size of Neural Network, because of simplicity of the environment (system), for complex systems we need larger Neural Network.

* **Conclusion:**
* Control without having a model for the system.
* Train a model to achieve our goal. (Maximizing the reward function)
* the downsides of RL that we don’t a mathematical or physical representation of the RL-Model (controller), so we can’t do any prediction or studies of its performances.

**After 1000 episode of Training**

Test01: Step input (Final value = 1)

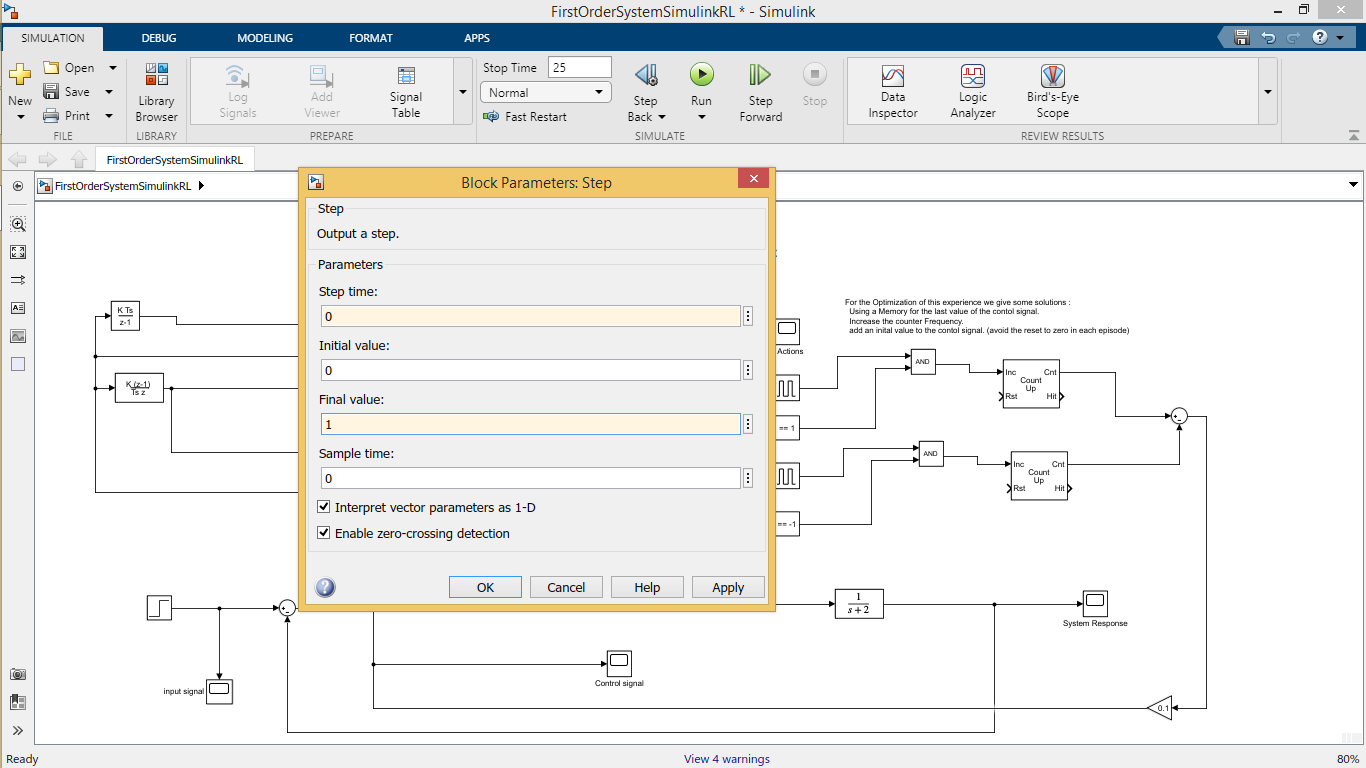


Figure1: Step Block option



Figure2: input signal

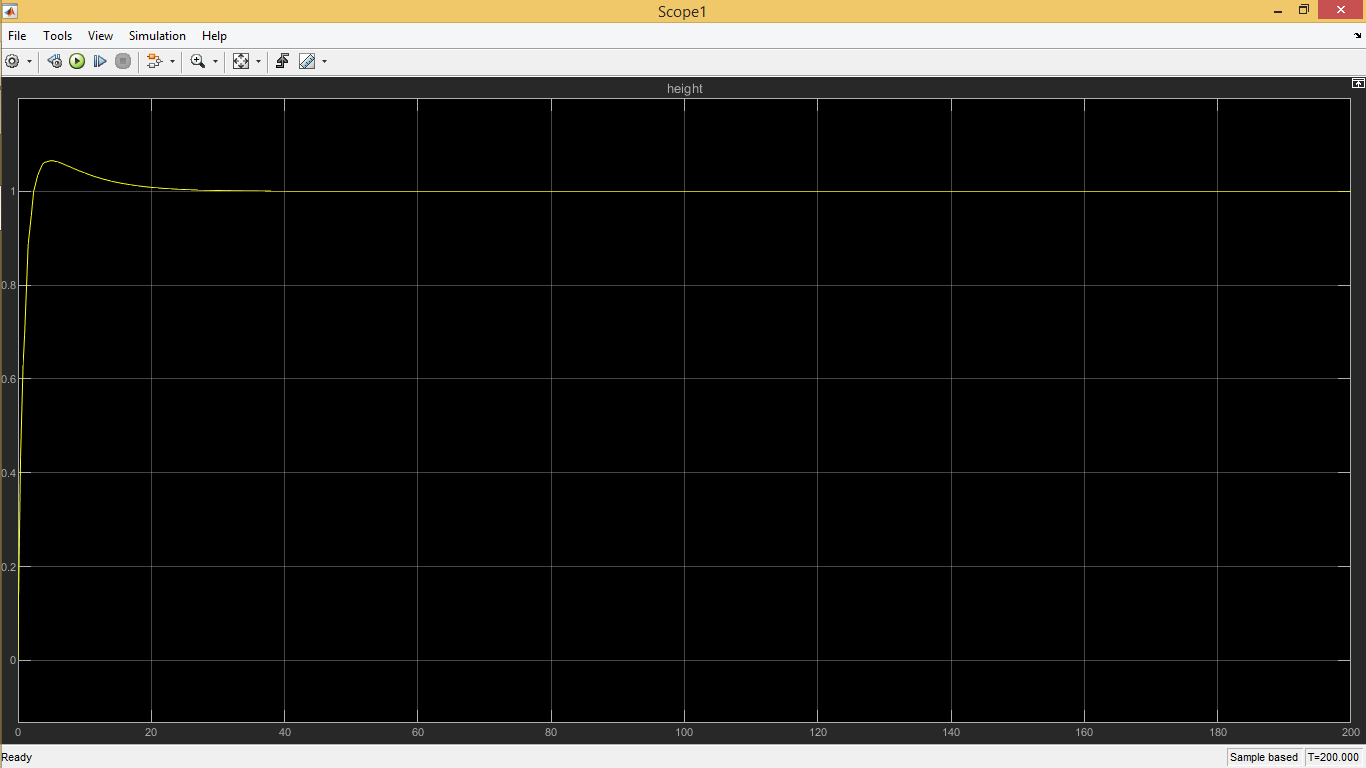


Figure3: system response

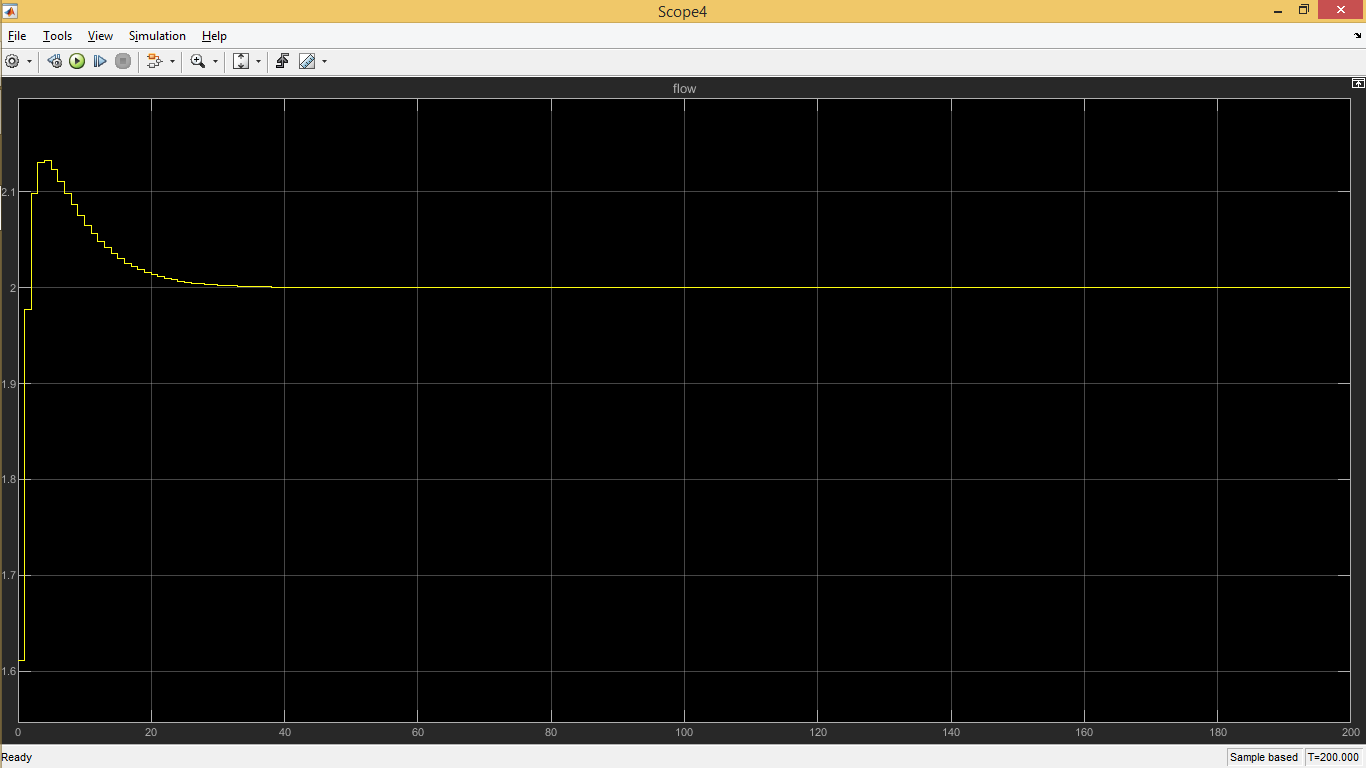


Figure4: control signal

Test02: Step input (Final value = 0.5)

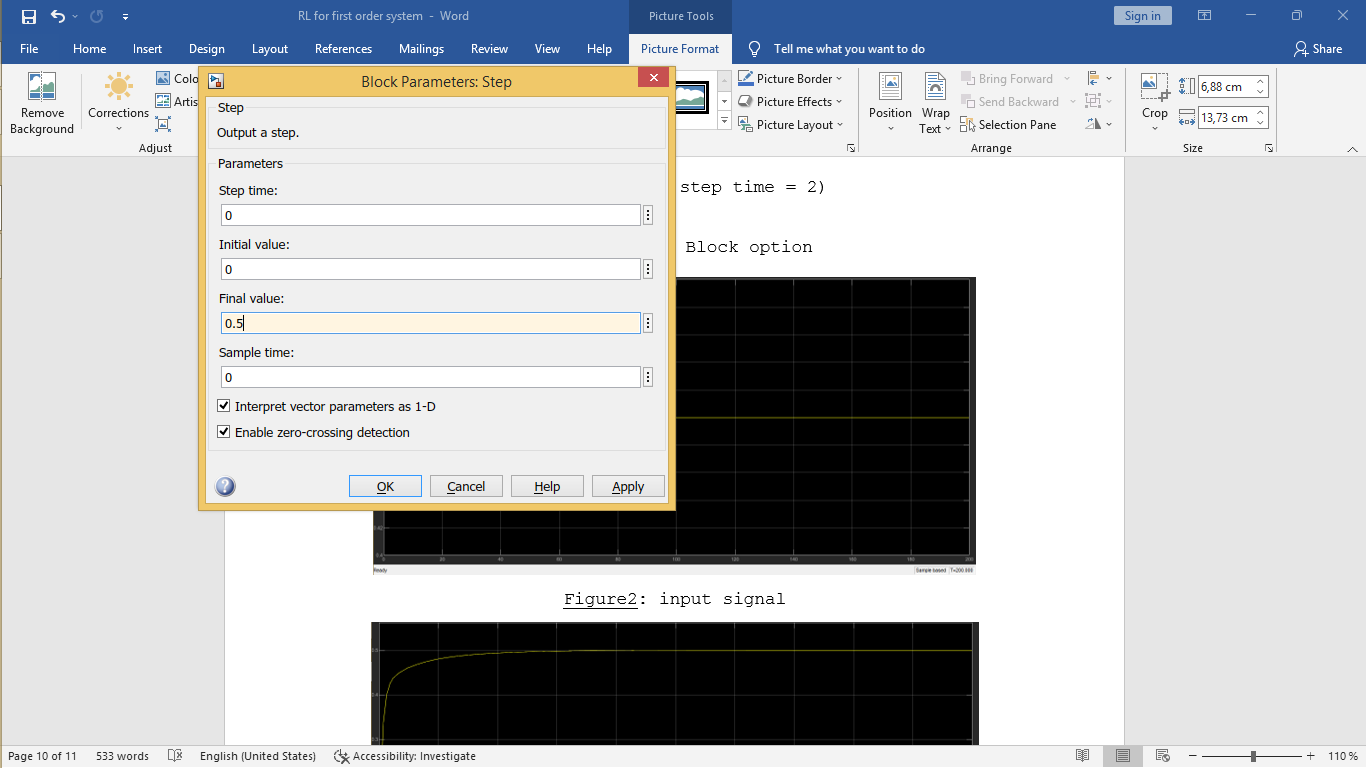


Figure1: Step Block option

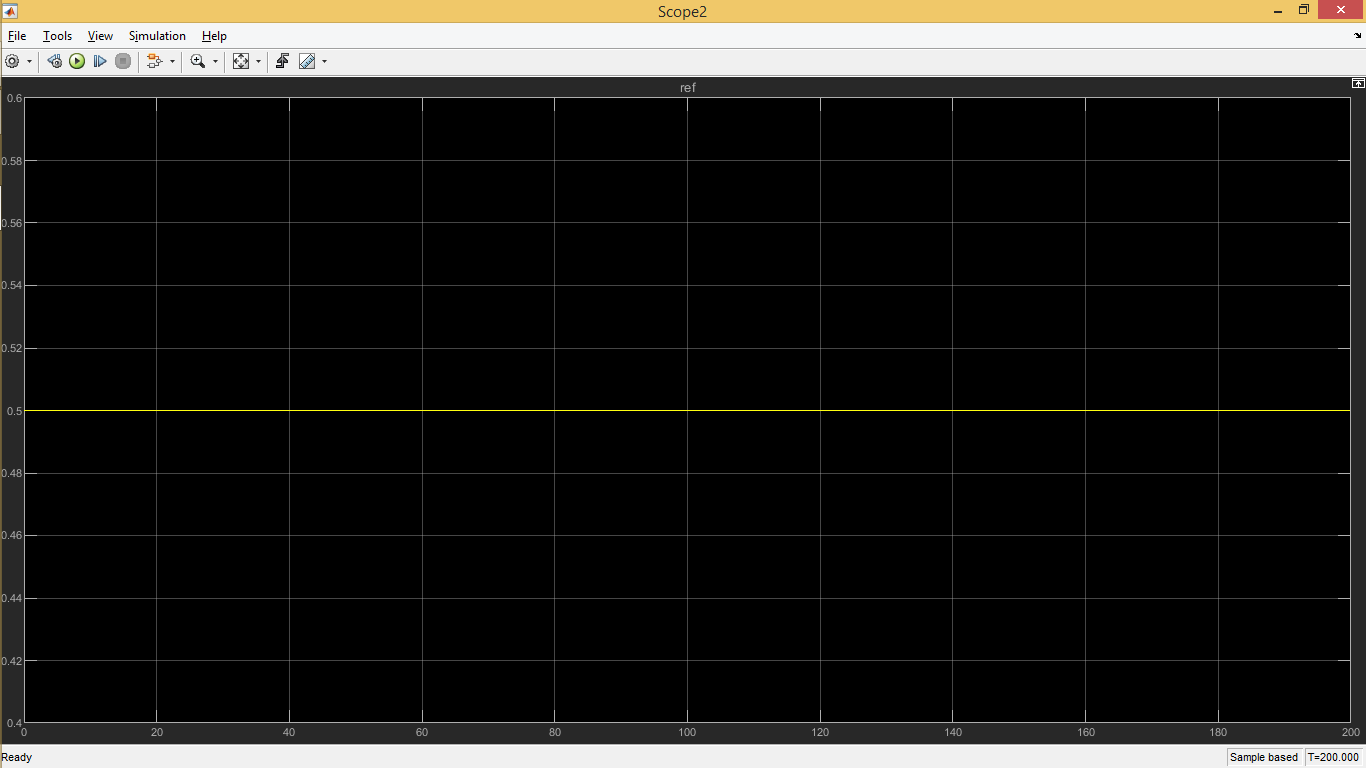


Figure2: input signal

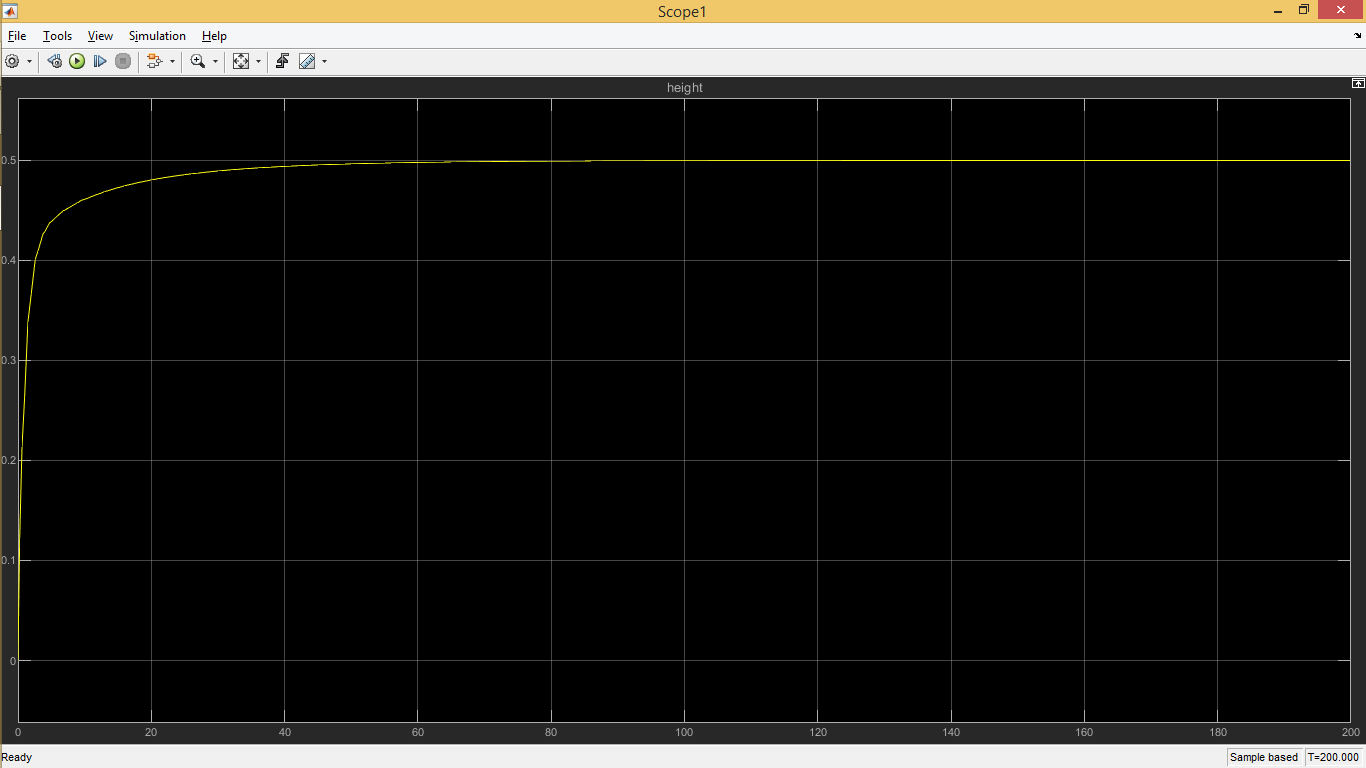


Figure3: system response

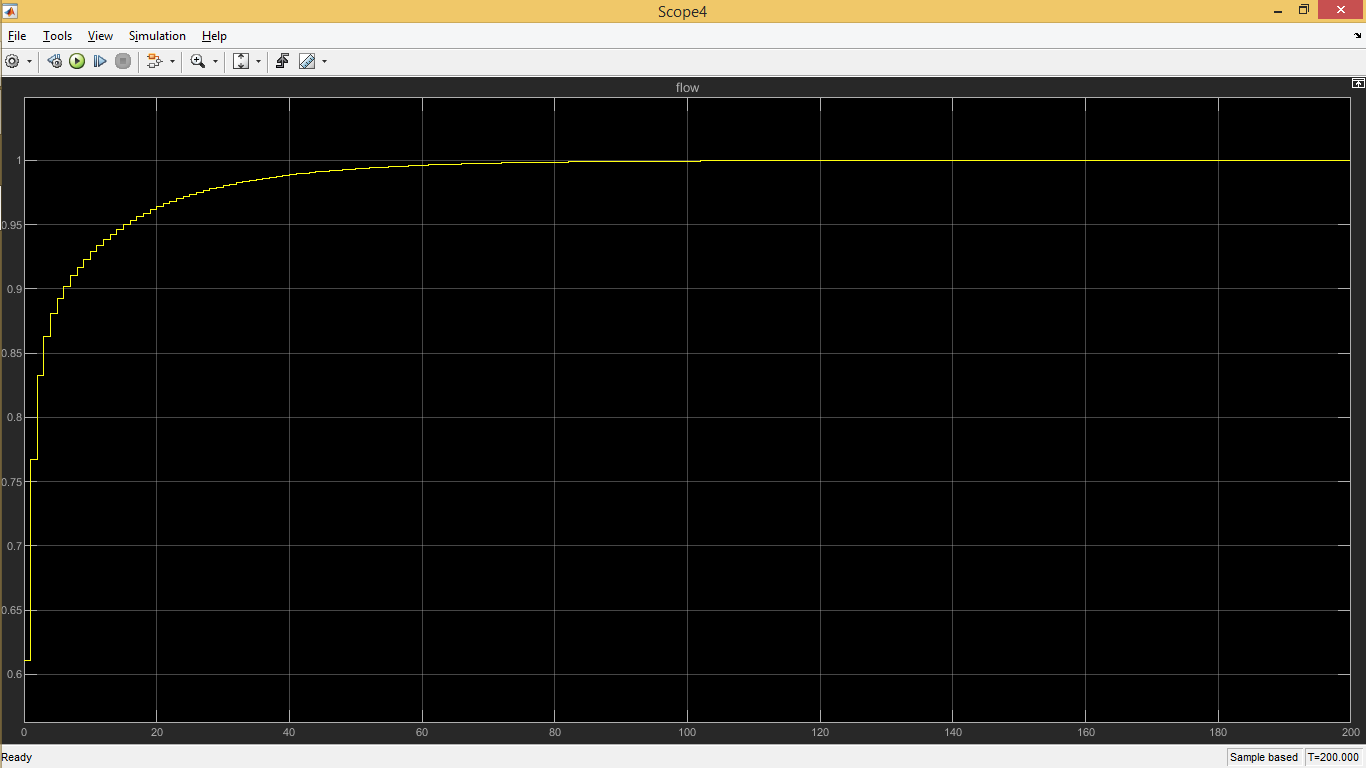


Figure4: control signal

Test02: Step input (Final value = 0.5 & Initial value = 1)

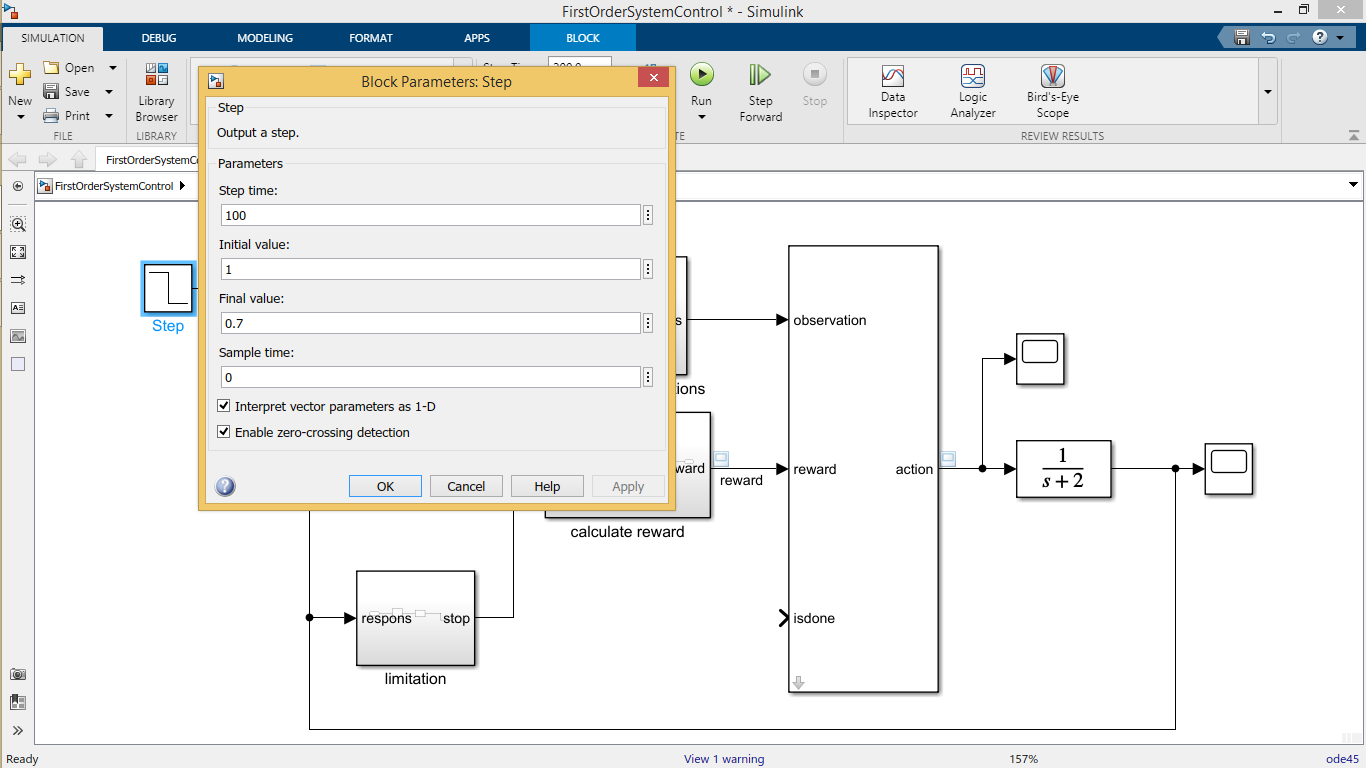


Figure1: Step Block option

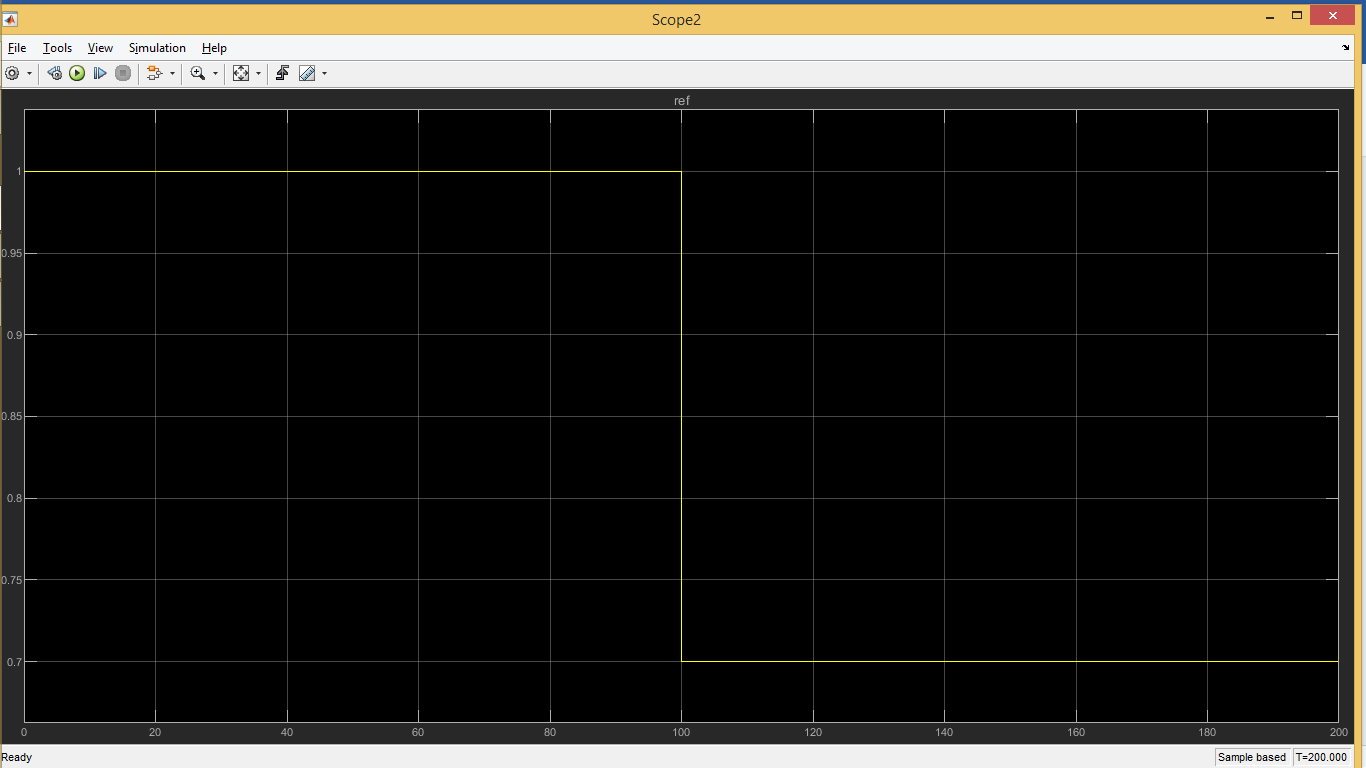


Figure2: input signal

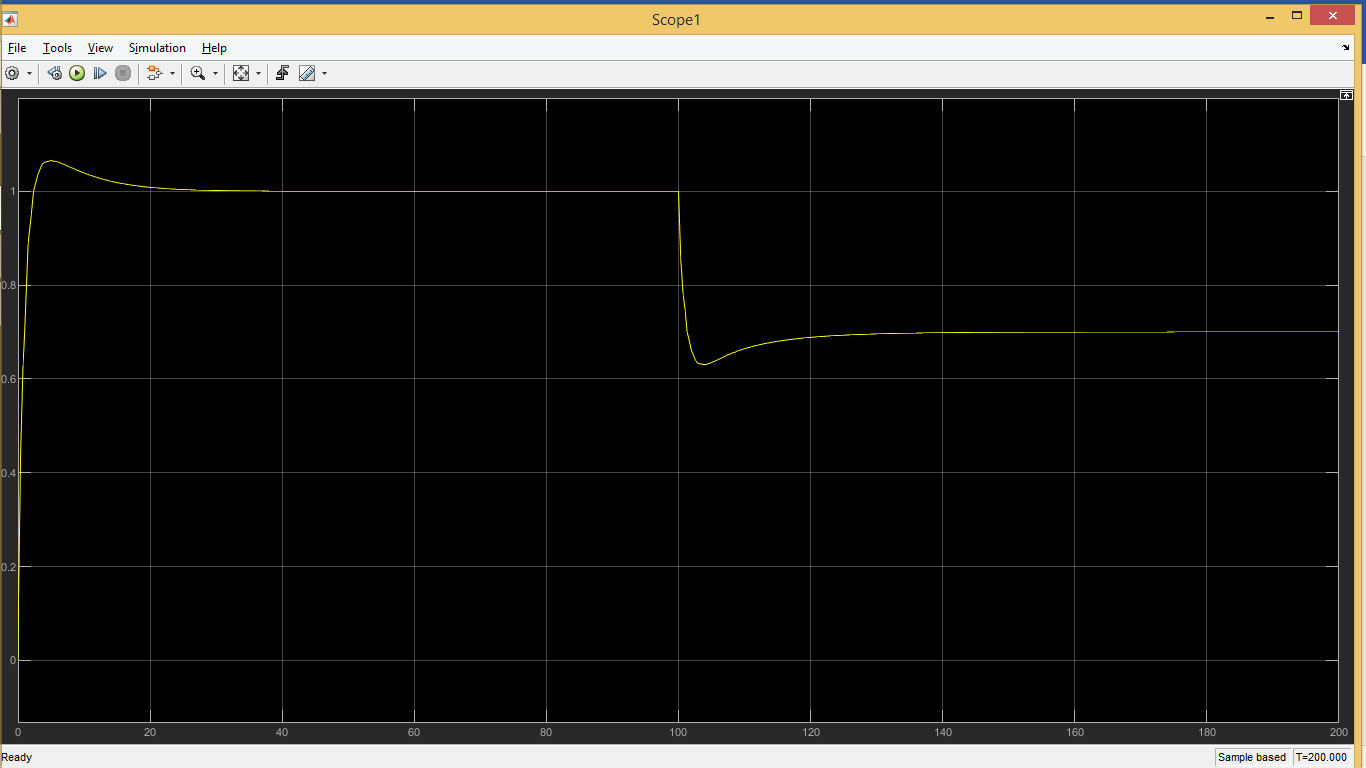


Figure3: system response

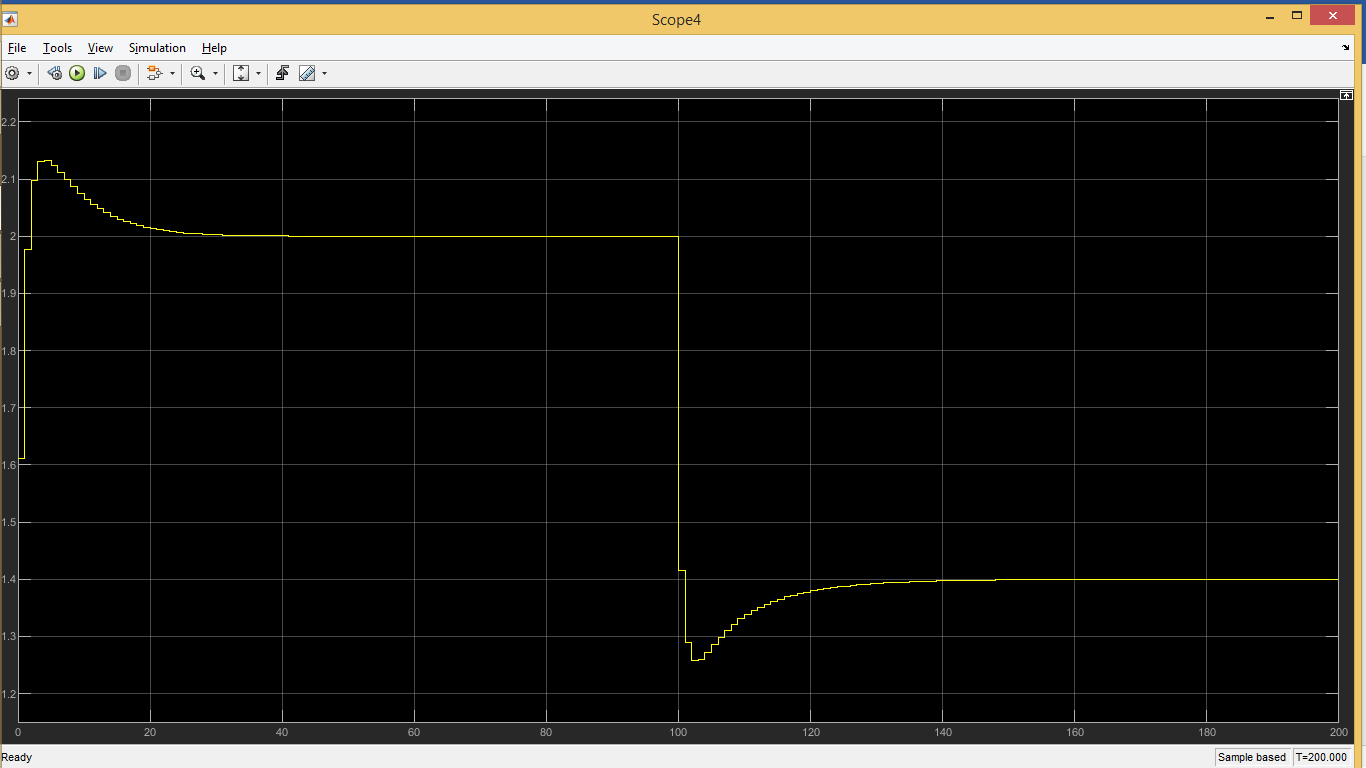


Figure4: control signal

Optimization Options:

\_Train the model for more long Time and more episodes.

\_Decrease the discount factor to make the system faster

\_Increase the range of training inputs and the neural network hidden layer’s size