Train SAC Agent for Ball Balance Control Using Reinforcement Learning Agents

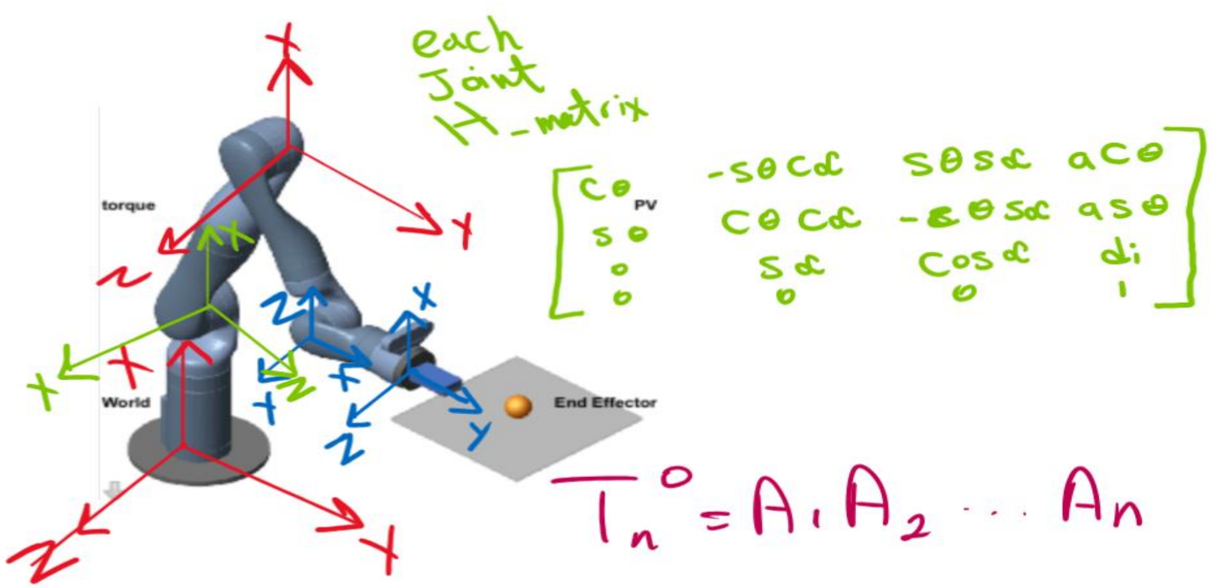
Team Members:

1-	Youssef Faisal Abdelkader Nassef : 2021170894
2-	Mahmoud Sayed Mahmoud Abdein : 2021170832
3-	Mahmoud Tarek Mahfouz Hamada : 20201701752
4-	Ahmed Mohammed Ramadan Mohammed Eid : 2021170853
5-	Ahmed Mostafa Ayman Abdelfattah Aboalesaad : 2021170843
6-	Nouran Magdy abdallah elsabahy : 2021170886
7-	Abdelaziz Ihab Mohamed: 2021170871

---

1. Kinematics of the Robot

The Kinova Gen3 is a 7-DOF robotic arm, but for this project, only the last two joints (wrist and hand) are actively used to control a plate for ball balancing. The remaining joints are f ixed. The two actuated joints control the plate's pitch and roll angles, enabling 2D motion of a ping-pong ball placed on it. Forward kinematics translates joint angles into platform orientation. We approximate dynamics and use reinforcement learning to learn torque control.



2. Environment Interface

The Simulink model rlKinovaBallBalance serves as the physical simulation, capturing both robot dynamics and contact interactions. By using rlSimulinkEnv, this model is encapsulated as a reinforcement learning environment with specified observation and action spaces:

- **Observation:** a 22-element vector (such as ball position, velocities, and joint states)
- **Action:** a pair of torque commands, each limited to the range [-1, 1]

```
nObs = 22;
nAct = 2;
obsInfo = rlNumericSpec([nObs 1]);
actInfo = rlNumericSpec([nAct 1]);
actInfo.LowerLimit = -1;
actInfo.UpperLimit = 1;

mdl = "rlKinovaBallBalance";
blk = mdl + "/RL Agent";
env = rlSimulinkEnv(mdl,blk,obsInfo,actInfo);  % Creates RL environment interface
```

I want to join Rabbit because I love fast-paced teamwork, solving real problems, and I'm excited to help make shopping easier for everyone—while learning from the best!

I want to join Rabbit because I love solving real problems and I am excited to help make shopping easier for everyone while learning from the best

3. Reward Function Design

The Default Reward Function Calcualted using the Follig formula:

Reward = R_Ball + R_Plate + R_Action, Where:

- R_Ball = e^(-000.1*(x^2 + y^2))
- R_Plate = -0.1 * (ϕ ^2 + θ ^2 + ψ ^2)
- R_Action = -0.05 * (T1^2 + T2^2)

Key Changes and Reasoning in the reward function

Robel = 1 / (1 + (x^2 + y^2)^2)

- The inverse function provides a steeper penalty for moderate-to-large displacements, discouraging unstable oscillations.

R_Plate = -0.2(ϕ^2 + θ^2)

- Simplified to penalize only pitch (ϕ) and roll (θ), which directly affect plate stability, ignoring redundant degrees of freedom (e.g., yaw, ψ).
- Increased coefficient (-0.1 → -0.2) emphasizes plate-leveling to prevent excessive tilting.

R_Action = -0.1(T_1^2 + T_2^2)

- Reduces mechanical stress on actuators and promotes energy-efficient policies.

---

4. Policy and Value Function Design

Two types of networks are created:

• Actor (Policy): Maps observations to a Gaussian distribution over actions

• Critics (Value): Estimate Q-values for given observation-action pairs

Default Actor NN:

[obsLayer -> FC(128) -> ReLU -> FC(64) -> ReLU -> [meanFC, stdFC]]]

Enhanced Actor NN (used) add new FC with 32 units followed by ReLU:

[obsLayer -> FC(128) -> ReLU -> FC(64) -> ReLU -> FC(32) -> ReLU -> [meanFC, stdFC]]]

Default Critic NN used:

[obs + action -> concat -> FC(128) -> ReLU -> FC(64) -> ReLU -> FC(32) -> ReLU -> Q output]

Enhanced Critic NN (used) add new FC with 16 units followed by ReLU:

[obs + action -> concat -> FC(128) -> ReLU -> FC(64) -> ReLU -> FC(32) -> ReLU -> FC(16) -> ReLU -> Q output]

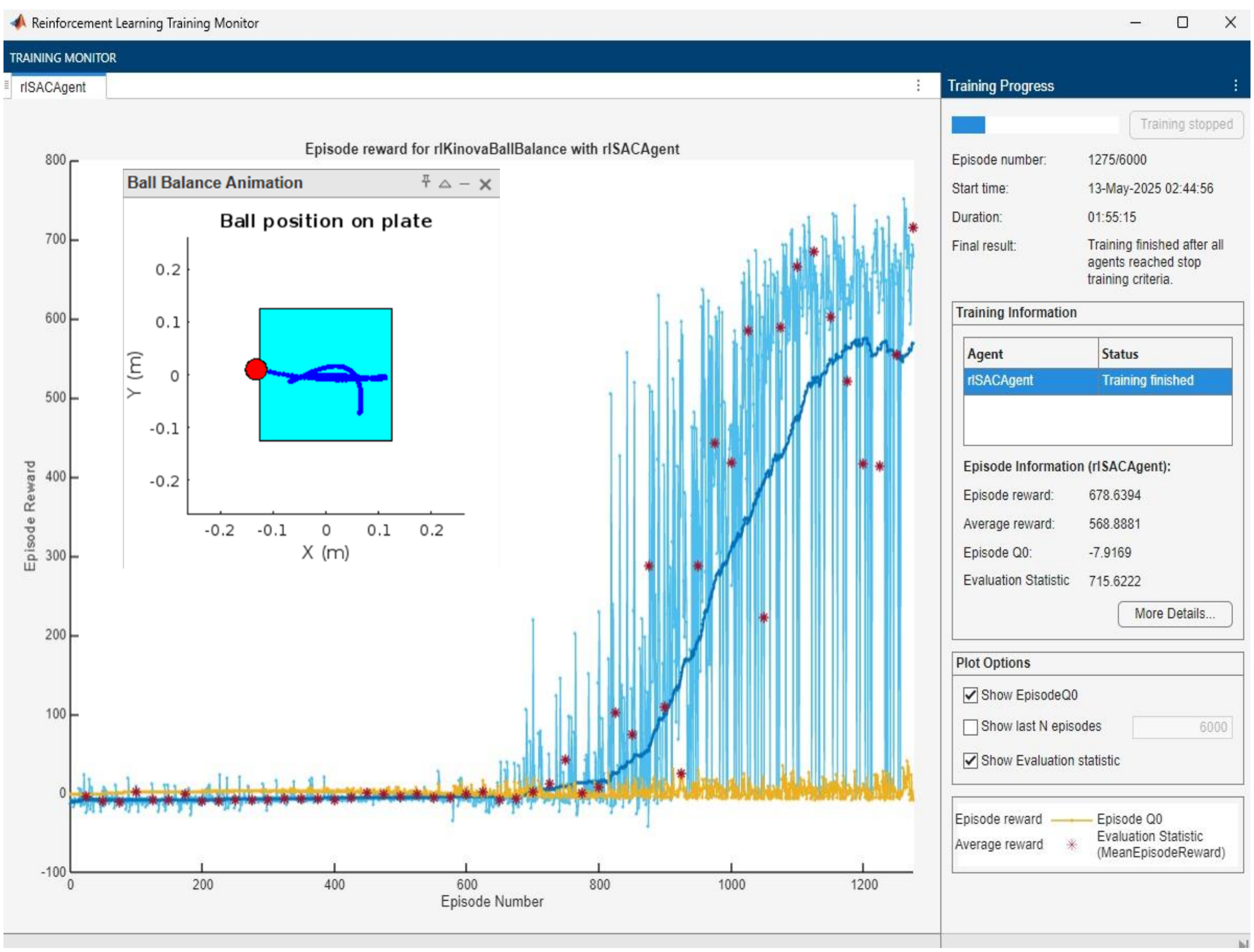
---

5. SAC Agent Training

Agent 1 Training: Default Reward Function + Default Network

Episode Number: 1275

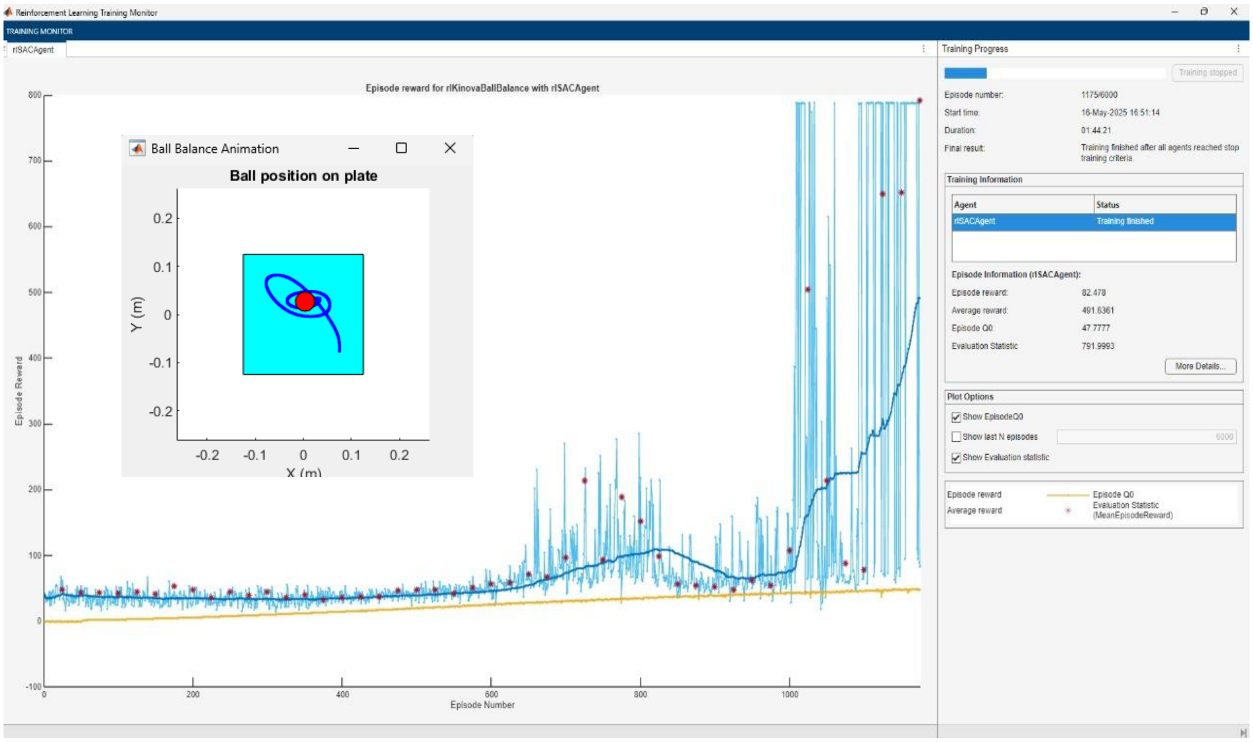
Evaluation Statistic: 715



Agent 2 Training: New Reward Function + Default Network

Episode Number: 1175

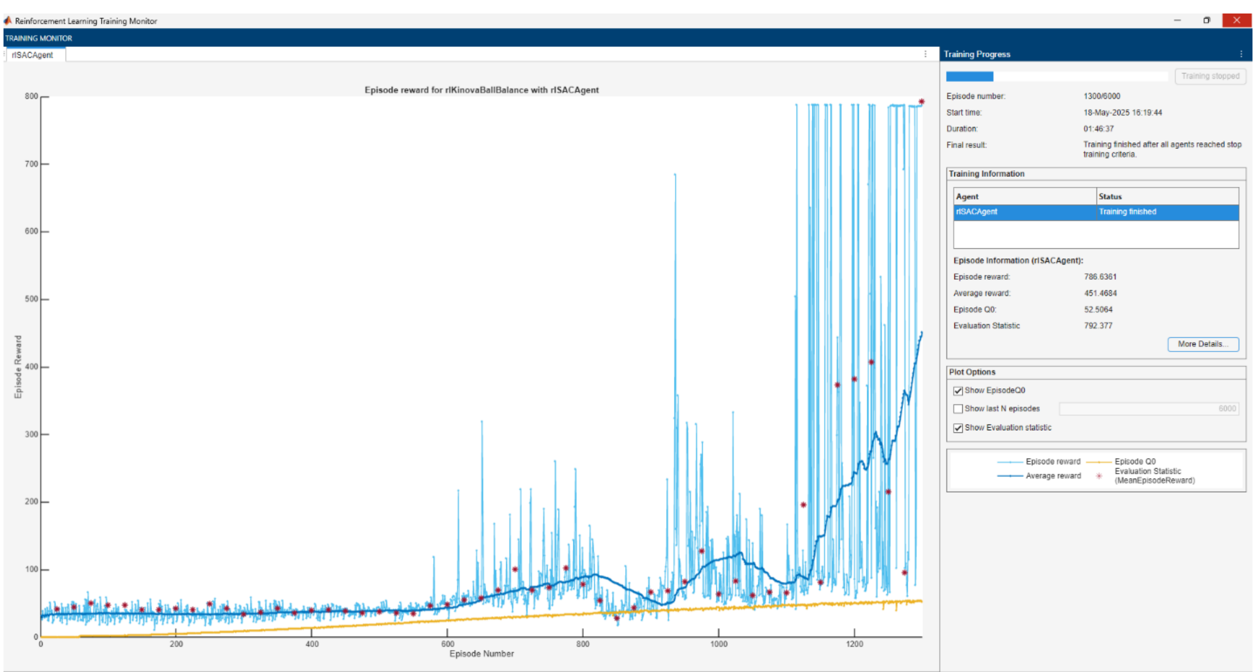
Evaluation Statistic: 791



Agent 3 Training: New

Episode Number: 1300

Evaluation Statistic: 792



6. Evaluation and Comparison

Each agent was evaluated without exploration noise. And both agents stopped training at the same evaluation threshold, but Agent 2 reached higher peak rewards in fewer episodes.

Property	Agent 1	Agent 2	Agent 3
Episode number	1275	1175	1300
Episode reward	678.6394	878.6394	786.6361
Total agent steps	429,523	359,523	168,871
Average reward	568.8881	491.2281	451.4684
Average steps	863.75	763.75	573.83

---

7. MATLAB Code

```
open_system("rlKinovaBallBalance")

open_system("rlKinovaBallBalance/Kinova Ball Balance")

kinova_params

nObs = 22;  % Number of dimensions in the observation space (e.g., joint angles, velocities, ball/plate state)[1]

nAct = 2;   % Number of dimensions in the action space (e.g., two control signals for the agent)


% Define the observation specification as a continuous numeric vector of size [22, 1]

obsInfo = rlNumericSpec([nObs 1]);


% Define the action specification as a continuous numeric vector of size [2, 1]

actInfo = rlNumericSpec([nAct 1]);
```

```
% Set the lower and upper limits for each action dimension to -1 and 1, respectively
% This constrains the agent's actions to the range [-1, 1] for each action component[2][3]
actInfo.LowerLimit = -1;
actInfo.UpperLimit = 1;
mdl = "rlKinovaBallBalance";
blk = mdl + "/RL Agent";
env = rlSimulinkEnv(mdl,blk,obsInfo,actInfo);  % Creates RL environment interface
env.ResetFcn = @kinovaResetFcn;
Ts = 0.01;
Tf = 10;
rng(0)
% Define the network paths.
observationPath = [
  featureInputLayer(nObs, Name="observation")
  concatenationLayer(1,2,Name="concat")
  fullyConnectedLayer(128)
  reluLayer
  fullyConnectedLayer(64)
  reluLayer
  fullyConnectedLayer(32)
  reluLayer
  fullyConnectedLayer(16)
  reluLayer
  fullyConnectedLayer(1, Name="QValueOutLyr")
];
actionPath = featureInputLayer(nAct,Name="action");
criticNet = dlnetwork;
criticNet = addLayers(criticNet, observationPath);
criticNet = addLayers(criticNet, actionPath);
```

```
criticNet = connectLayers(criticNet,"action","concat/in2");

plot(criticNet)

summary(initialize(criticNet))

critic1 = rlQValueFunction(initialize(criticNet),obsInfo,actInfo, ...

   ObservationInputNames="observation");

critic2 = rlQValueFunction(initialize(criticNet),obsInfo,actInfo, ...

   ObservationInputNames="observation");

% Shared path

commonPath = [

   featureInputLayer(nObs, Name="observation")

   fullyConnectedLayer(128)

   reluLayer

   fullyConnectedLayer(64)

   reluLayer

   fullyConnectedLayer(32)

   reluLayer(Name="commonPath")

];


% Mean path

meanPath = [

   fullyConnectedLayer(32, Name="meanFC")

   reluLayer

   fullyConnectedLayer(nAct, Name="actionMean")

];


% Std path

stdPath = [

  fullyConnectedLayer(nAct, Name="stdFC")

  reluLayer
```

```
  softplusLayer(Name="actionStd")
];
actorNet = dlnetwork;
actorNet = addLayers(actorNet,commonPath);
actorNet = addLayers(actorNet,meanPath);
actorNet = addLayers(actorNet,stdPath);
actorNet = connectLayers(actorNet,"commonPath","meanFC/in");
actorNet = connectLayers(actorNet,"commonPath","stdFC/in");
plot(actorNet)
actorNet = initialize(actorNet);
summary(actorNet)
actor = rlContinuousGaussianActor(actorNet, obsInfo, actInfo, ...
    ObservationInputNames="observation", ...
    ActionMeanOutputNames="actionMean", ...
    ActionStandardDeviationOutputNames="actionStd");
agentOpts = rlSACAgentOptions( ...
    SampleTime=Ts, ...
    TargetSmoothFactor=1e-3, ...
    ExperienceBufferLength=1e6, ...
    MiniBatchSize=256, ...
    NumWarmStartSteps=256*10, ...
    DiscountFactor=0.99);
agentOpts.ActorOptimizerOptions.Algorithm = "adam";
agentOpts.ActorOptimizerOptions.LearnRate = 1e-3;
agentOpts.ActorOptimizerOptions.GradientThreshold = 1;

for ct = 1:2
    agentOpts.CriticOptimizerOptions(ct).Algorithm = "adam";
    agentOpts.CriticOptimizerOptions(ct).LearnRate = 1e-3;
```

```
    agentOpts.CriticOptimizerOptions(ct).GradientThreshold = 1;
end
agent = rlSACAgent(actor,[critic1,critic2],agentOpts);
trainOpts = rlTrainingOptions(...
    MaxEpisodes=6000, ...
    MaxStepsPerEpisode=floor(Tf/Ts), ...
    ScoreAveragingWindowLength=100, ...
    Plots="training-progress", ...
    SimulationStorageType="file",...
    StopTrainingCriteria="EvaluationStatistic", ...
    StopTrainingValue=700, ...
    UseParallel=true);
if trainOpts.UseParallel
    % Disable visualization in Simscape Mechanics Explorer
    set_param(mdl, SimMechanicsOpenEditorOnUpdate="off");
    set_param(mdl+"/Kinova Ball Balance/7 DOF Manipulator", ...
        "VChoice", "None");
    % Disable animation in MATLAB figure
    doViz = false;
    save_system(mdl);
else
    % Enable visualization in Simscape Mechanics Explorer
    set_param(mdl, SimMechanicsOpenEditorOnUpdate="on");
    % Enable animation in MATLAB figure
    doViz = true;
end
logger = rlDataLogger();
logger.AgentLearnFinishedFcn = @logAgentLearnData;
logger.EpisodeFinishedFcn    = @(data) logEpisodeData(data, doViz);
```

```
doTraining = true;
if doTraining
   % Evaluate the performance of the greedy policy every 25 training
   % episodes, averaging the cumulative reward of 5 simulations.
   evaluator = rlEvaluator(EvaluationFrequency=25,NumEpisodes=5);
   % train
   trainResult = train(agent,env,trainOpts,Logger=logger,Evaluator=evaluator);
else
   load("kinovaBallBalanceAgent.mat")
end
userSpecifiedConditions = true;
if userSpecifiedConditions
   ball.x0 = 0.075;
   ball.y0 = -0.075;
   env.ResetFcn = [];
else
   env.ResetFcn = @kinovaResetFcn;
end
simOpts = rlSimulationOptions(MaxSteps=floor(Tf/Ts));
set_param(mdl, SimMechanicsOpenEditorOnUpdate="on");
doViz = true;
agent.UseExplorationPolicy = false;
experiences = sim(agent,env,simOpts);
function dataToLog = logAgentLearnData(data)
% This function is executed after completion
% of the agent's learning subroutine

dataToLog.ActorLoss = data.ActorLoss;
dataToLog.CriticLoss = data.CriticLoss;
```

```
end


function dataToLog = logEpisodeData(data, doViz)
% This function is executed after the completion of an episode


dataToLog.Experience = data.Experience;


% Show an animation after episode completion
if doViz
   animatedPath(data.Experience);
end
end
```