SC645-Project

Dhrumil Lotiya, 21d070026

April 2024

1 Aim

The project's goal is to train a MATLAB-based reinforcement learning (RL) agent to stabilize a plant. To meet the specified performance requirements, the RL agent needs to be trained to work in the feedback loop and combine the necessary control commands. The challenge is to bring a pendulum back to its upright position and make it stabilize there. The non-linear pendulum dynamics is as follows,

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -\frac{g}{L}\sin(x_1) - \frac{b}{mL^2}x_2 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{mL^2} \end{bmatrix} \tau$$

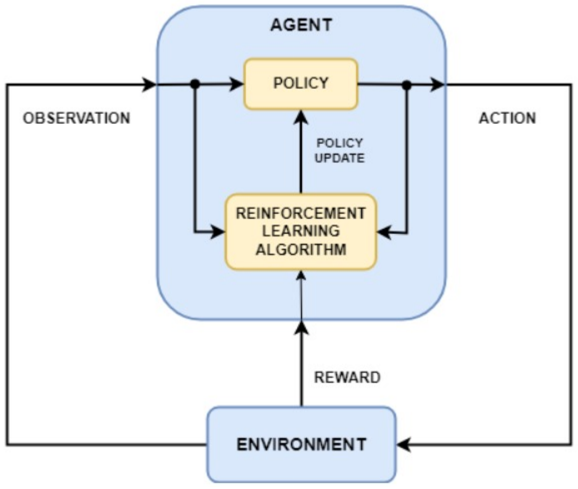
where $x_1 = \theta$, $x_2 = \dot{\theta}$, and the control input is τ.

The RL agent assigned to me according to my roll number is the Policy Gradient Agent(PG Agent).

Agent's Learning Process

- Initialization: The RL agent initializes its policy, value function, and any other necessary parameters. The environment is also initialized, including setting up the state space, action space, and dynamics of the system The RL agent sets up its strategy for making decisions (policy), estimates of how valuable different actions are (value function), and other necessary parameters. Meanwhile, the environment gets ready too, defining what states and actions are possible, and how the pendulum behaves.
- Exploration and Exploitation: During training, the agent explores the environment by trying out different actions and observing the outcomes. This allows it to gather information about which actions are better in different situations. At the same time, it exploits its current knowledge by favoring actions that it believes will lead to higher rewards based on its value function.

- Feedback and Learning: After taking each action, the agent receives feedback from the environment in the form of a reward signal. It uses this feedback to update its value function and adjust its policy. The goal is to learn from experience and improve its decision-making over time.
- Iterative Process: The learning process is iterative, meaning it repeats over and over again. The agent continues to interact with the environment, learn from its experiences, and update its strategy accordingly. Over time, it refines its policy and value function to become more effective at achieving its goals.
- Convergence: With enough training iterations, the agent's policy and value function should converge to optimal or near-optimal values, meaning it has learned the best way to behave in the given environment to maximize its rewards.



Observation Space

The observation vector consists of two elements: the angle of the pendulum (x_1) and its angular velocity (x_2).

- Angle (x_1): Since the angle of the pendulum directly affects its present position, it is an important parameter. The agent must take corrective action to return the angle to the upright position if it deviates too much from it.
- Angular Velocity (x_2): The pendulum's angular velocity tells us how quickly it is moving. This is significant because it shows how quickly the pendulum is descending or shifting away from its upright position.

An agent may need to take more drastic corrective action in the case of a high angular velocity.

Action Space

The action space represents the control inputs that the RL agent can select to influence the behavior of the inverted pendulum system. Specifically, the action space consists of a finite set of discrete actions, which are integer values ranging from -20 to 20.

- Explicit Constraints on Control Inputs: The explicit constraints on the control inputs are imposed by defining the action space using rlFiniteSetSpec. This ensures that the RL agent can only choose actions within the specified range (-20 to 20). If the agent attempts to select an action outside of this range, an error will be raised, enforcing the constraints on the control inputs.
- Tuning Considerations: In PG algorithms, the policy is typically represented directly as a parameterized function (e.g., a neural network) that maps states to action probabilities. Tuning the architecture and complexity of this policy network is crucial.

  Learning Rate: The learning rate determines how quickly the agent updates its policy based on new experiences. A high learning rate can lead to faster learning but may result in instability, while a low learning rate may lead to slow learning.

  Reward Function: The reward function is carefully tuned to provide meaningful feedback to the agent. It penalizes actions that deviate the pendulum from the upright position and reward actions that help maintain balance.

Reward Function

The reward function = -($angle - error^2 + 0.1w^2 + 0.001last\_action^2$)
where w= angular velocity

This reward design is a weighted sum of three components:

- Angle Error Term = -($angle - error^2$) This part penalizes the angle deviation, which is the pendulum's angle from vertical. Greater deviations will be punished more harshly if the angle inaccuracy is squared.I am making angle-error square term negative and thus for large error term will encourage the pendulum to maintain equilibrium near the vertical position.
- Angular Velocity Term= -(0.1* w^2) This element penalizes the angle's rate of change, or angular velocity. Larger velocities are penalized more

severely if the angular velocity is squared once more. High angular velocities are penalized when error term is made negative, which may aid in stabilizing the pendulum's motion and preventing it from swinging too quickly.

- Action (Torque) Term = (-0.001 * $(\mathit{this.LastAction})^2$) This element penalizes the agent for the last action taken. Larger acts are penalized more heavily when the action is squared. Large torque inputs are effectively penalized when the term is made negative. This will urge the agent to take smaller, more deliberate movements, which may result in the pendulum being controlled more smoothly and steadily.

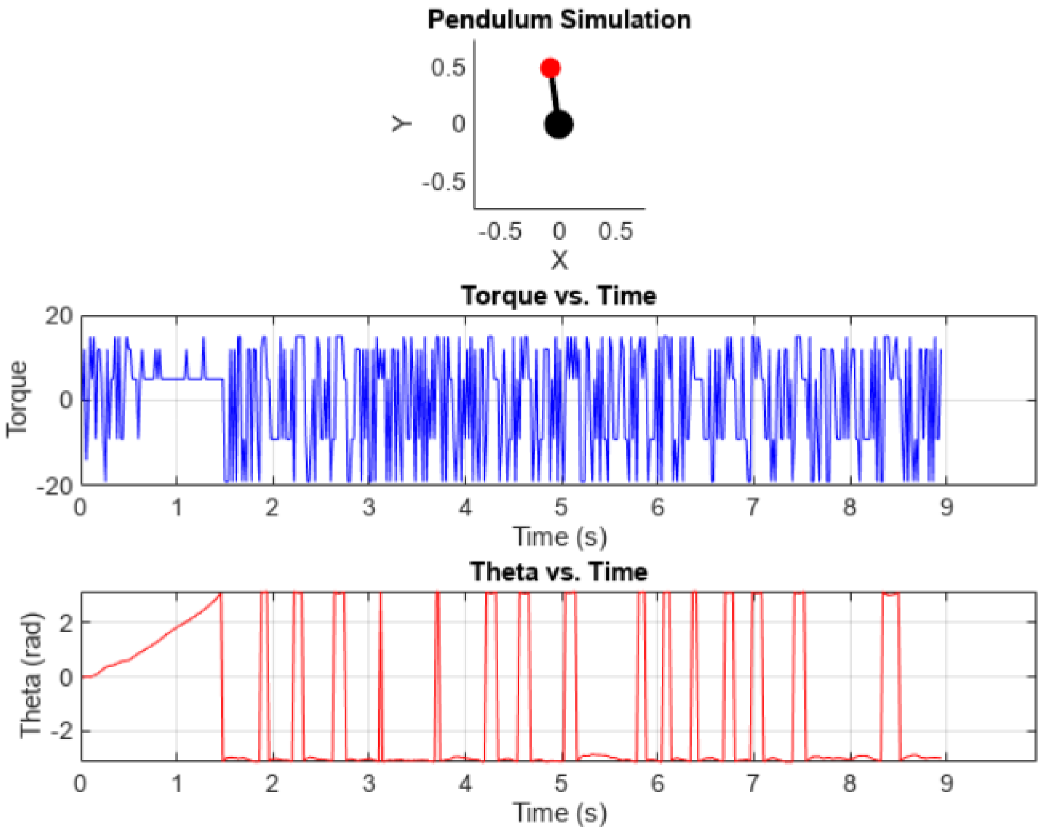


Figure 1: Output plots and Simulation