Soft Actor–Critic Based Bipedal Walker Control Using Reinforcement Learning

Manas Mondal (Team Member 1)
Arkaprava Jas(Team Member 2)
Department of Computer Science,RKMVERI

December 17, 2025

1 Introduction

Bipedal locomotion is a fundamental yet challenging problem in artificial intelligence and robotics due to nonlinear dynamics, continuous control requirements, and stability constraints. Designing controllers that enable stable and efficient walking behavior remains a significant research problem.

Reinforcement Learning (RL) offers a data-driven framework where agents learn optimal behavior through interaction with an environment. In this project, we solve the *BipedalWalker-v3* task using the **Soft Actor–Critic (SAC)** algorithm, a state-of-the-art off-policy deep reinforcement learning method designed for continuous action spaces.

The objectives of this project are:

- To model bipedal locomotion as a Markov Decision Process (MDP)
- To train an agent using the Soft Actor–Critic algorithm
- To evaluate the learned walking policy
- To analyze the effectiveness of entropy-regularized reinforcement learning

2 Problem Formulation and Representation

2.1 Problem Domain

The problem domain is simulated robotic locomotion using the *BipedalWalker-v3* environment from the Gymnasium framework. The agent controls a two-legged robot in a 2D physics-based environment with uneven terrain.

The goal of the agent is to learn stable and efficient walking behavior by applying appropriate joint torques while maintaining balance and minimizing energy consumption.

2.2 State, Actions, and Constraints

2.2.1 State Representation

The state represents the complete observable information available to the agent at each timestep. It is modeled as a continuous-valued vector that captures both the internal dynamics of the bipedal robot and its interaction with the environment.

The observation vector includes hull orientation and angular velocity, linear velocities, joint angles and angular velocities, ground contact indicators, and terrain sensor readings. This representation enables the agent to maintain balance, coordinate leg motion, and adapt to uneven terrain.

Implementation Assumptions: In the implemented system, the state vector is directly obtained from the Gymnasium environment without manual preprocessing or feature engineering. The state is assumed to be fully observable, satisfying the Markov property. The SAC neural networks learn internal representations automatically from raw observations.

2.2.2 Action Space

The action space is continuous and four-dimensional:

$$a = [a_1, a_2, a_3, a_4]$$

where each action component corresponds to torque applied at the hip and knee joints of both legs.

Implementation Assumptions: The SAC actor network outputs the parameters of a Gaussian distribution over actions. During training, actions are sampled stochastically, while during evaluation deterministic actions are used. Action bounds are enforced internally using a tanh squashing function provided by the Stable-Baselines3 implementation.

2.2.3 Constraints

Constraints arise from physical realism and environment design. Joint torques are bounded, excessive energy usage is penalized, and the robot must remain upright to continue an episode. Episodes terminate when the robot falls or when the maximum time limit is reached.

Implementation Assumptions: All constraints are implicitly enforced by the BipedalWalker-v3 environment and the Box2D physics engine. No explicit constraint-handling logic is implemented in code. Instead, constraint satisfaction is learned through the reward function and environment termination conditions.

2.2.4 Goal Condition

The objective of the agent is to maximize cumulative reward over an episode. This corresponds to walking forward efficiently while maintaining balance and avoiding early termination due to falling.

3 Data / Environment

This project is formulated as a reinforcement learning task and does not rely on any external dataset. Instead, training data is generated dynamically through interaction between the agent and the environment.

3.1 Nature of Data in Reinforcement Learning

In reinforcement learning, data is generated online in the form of transition tuples:

$$(s_t, a_t, r_t, s_{t+1})$$

where s_t is the current state, a_t is the action taken, r_t is the reward received, and s_{t+1} is the next state.

Implementation Assumptions: Transition tuples are automatically stored in a replay buffer. No offline dataset or manual data loading is used. Data is sampled randomly from the replay buffer during training.

3.2 Reinforcement Learning Environment

The environment used in this project is the *BipedalWalker-v3* environment provided by the Gymnasium framework. It simulates a two-dimensional bipedal robot operating in a physics-based environment with uneven terrain.

Implementation Details: The environment is initialized using the Gymnasium API. Rendering is disabled during training for computational efficiency and enabled during evaluation for visualization.

3.3 Observation Space

The observation space is a continuous-valued vector that includes information about the robot's physical state and its interaction with the terrain, such as body orientation, velocities, joint angles, joint angular velocities, ground contact indicators, and terrain sensor readings.

Implementation Assumptions: Observations are used directly without preprocessing or normalization. The state is assumed to be fully observable, satisfying the Markov property.

3.4 Action Space

The action space is continuous and four-dimensional, representing torques applied to the hip and knee joints of both legs.

Implementation Assumptions: Actions are sampled from a stochastic policy during training and selected deterministically during evaluation. Action bounds are enforced internally by the environment and the SAC implementation.

3.5 Reward Structure

The reward function encourages forward motion, stability, and energy-efficient movement, while penalizing falling and unstable behavior.

Implementation Assumptions: The reward function is predefined by the environment, and no additional reward shaping is introduced in the code.

3.6 Episode Structure and Termination

Episodes terminate when the robot falls or when the maximum time step limit is reached. Episode termination is handled using the environment's termination signals.

3.7 Data Usage During Training

Training data is stored in a replay buffer, enabling random mini-batch sampling and off-policy learning. This improves sample efficiency and stabilizes training.

3.8 Summary

The environment generates all training data online through agent interaction. No static dataset is used, and learning is entirely model-free, consistent with the Soft Actor–Critic framework.

4 Methodology

The proposed approach models bipedal locomotion as a continuous control problem and solves it using deep reinforcement learning. The Soft Actor–Critic (SAC) algorithm is employed due to its stability and effectiveness in continuous action spaces.

4.1 Algorithm Used: Soft Actor–Critic

Soft Actor–Critic (SAC) is an off-policy actor–critic algorithm that optimizes a stochastic policy while maximizing both expected cumulative reward and policy entropy. This entropy regularization encourages exploration and improves robustness.

SAC consists of the following components:

- Actor network that outputs a stochastic policy
- Two critic networks to estimate state–action values

- Target critic networks for stable learning
- Replay buffer for experience storage

4.2 Training Pipeline

During training, the agent interacts with the environment by sampling actions from the policy. Each interaction generates a transition tuple (s, a, r, s'), which is stored in a replay buffer. Mini-batches sampled from the buffer are used to update the actor and critic networks. Target networks are updated using soft updates.

4.3 Justification

SAC is chosen because it supports continuous action spaces, provides stable learning through twin critics, encourages efficient exploration via entropy maximization, and enables sample-efficient off-policy learning.

4.4 Mathematical Formulation

The environment is modeled as a Markov Decision Process:

$$(S, A, T, R, \gamma)$$

4.4.1 Objective Function

$$J(\pi) = \mathbb{E}\left[\sum_t R(s_t, a_t) + \alpha \mathcal{H}(\pi(\cdot|s_t))\right]$$

4.4.2 Critic Update

$$L(\phi_i) = \mathbb{E}\left[\left(Q_{\phi_i}(s, a) - \left(r + \gamma\left(\min_{j=1,2} Q_{\bar{\phi}_j}(s', a') - \alpha \log \pi(a'|s')\right)\right)\right)^2\right]$$

4.4.3 Actor Update

$$L(\theta) = \mathbb{E}\left[\alpha \log \pi_\theta(a|s) - \min_{i=1,2} Q_{\phi_i}(s, a)\right]$$

4.4.4 Entropy Coefficient Update

$$L(\alpha) = \mathbb{E}\left[-\alpha\left(\log \pi(a|s) + \mathcal{H}_{target}\right)\right]$$

4.5 Methodology Summary

The proposed methodology integrates entropy-regularized reinforcement learning, twin critic networks, and off-policy training to effectively solve the bipedal locomotion problem.

5 Implementation

This section describes the practical realization of the proposed Soft Actor–Critic based bipedal locomotion system. The implementation converts the theoretical reinforcement learning framework into an executable training and evaluation pipeline.

5.1 Software Tools and Libraries

The system is implemented using the following tools:

- Python programming language
- Gymnasium framework with Box2D physics engine
- Stable-Baselines3 reinforcement learning library
- PyTorch for neural network computations
- NumPy for numerical operations

5.2 Environment Setup

The BipedalWalker-v3 environment is created using the Gymnasium API. During training, rendering is disabled to improve computational efficiency. Rendering is enabled only during testing for visual inspection of the learned policy. Each episode begins with a reset of the environment, generating randomized terrain conditions.

5.3 SAC Model Architecture

The SAC agent uses a multilayer perceptron architecture internally.

5.3.1 Actor Network

The actor network takes the environment state as input and outputs the parameters of a Gaussian distribution over actions. Actions are sampled from this distribution and passed through a tanh function to enforce action bounds.

5.3.2 Critic Networks

Two independent critic networks estimate state–action value functions. The use of twin critics reduces overestimation bias and improves training stability.

5.4 Hyperparameter Configuration

Table 1: SAC Hyperparameters

Parameter	Value
Learning Rate	3×10^{-4}
Discount Factor (γ)	0.99
Replay Buffer Size	300,000
Batch Size	256
Soft Update (τ)	0.005
Entropy Coefficient	Automatic
Training Timesteps	500,000

5.5 Training Procedure

The training process consists of repeated interaction with the environment. At each timestep, actions are sampled from the stochastic policy and executed in the environment. Transitions are stored in a replay buffer. Mini-batches sampled from the buffer are used to update the actor and critic networks. Target networks are softly updated after each learning step.

5.6 Evaluation and Testing

After training, the learned policy is evaluated in a separate testing phase. Rendering is enabled, and actions are selected deterministically to assess walking stability and performance without exploration noise.

5.7 Hardware and Execution Details

Training is conducted on a CPU-based system using a single environment instance. The implementation is modular, separating training and evaluation scripts to ensure reproducibility and ease of extension.

6 Results

This section presents the experimental results obtained by training the Soft Actor–Critic (SAC) agent on the BipedalWalker-v3 environment. Both quantitative trends and qualitative observations are used to evaluate the performance of the trained policy.

6.1 Training Performance

The SAC agent was trained for 500,000 timesteps. During early training stages, the agent exhibited unstable behavior with frequent falls and low cumulative rewards. As training progressed, the agent gradually learned coordinated joint control and balance, resulting in improved walking behavior and higher rewards.

A steady increase in cumulative episode rewards was observed, along with a reduction in reward variance, indicating improved policy stability and convergence.

6.2 Reward Curve Analysis

The reward curve shows three distinct phases: an initial exploration phase with low rewards and high variance, a learning phase with rapid reward improvement, and a convergence phase where rewards stabilize at consistently higher values. This progression confirms effective policy learning and successful entropy-regularized exploration.

6.3 Policy Evaluation and Behavioral Analysis

The trained policy was evaluated using deterministic action selection with environment rendering enabled. The agent demonstrated stable upright posture, smooth alternating leg movements, and effective forward locomotion. Episodes were completed without premature termination due to falling, indicating successful learning of balance and coordination.

6.4 Stability and Robustness

The learned policy exhibited robustness to small perturbations in terrain and variations in initial conditions. Stable walking behavior was maintained across multiple evaluation episodes, demonstrating reliable policy execution.

6.5 Energy Efficiency

The trained agent exhibited smoother joint trajectories and reduced oscillatory motion compared to early training behavior. This indicates improved energy efficiency, as the agent avoided excessive or unnecessary torque application.

6.6 Comparison with Untrained Policy

Compared to an untrained or random policy, the SAC-trained agent achieved significantly higher cumulative rewards, longer episode durations, and a dramatic reduction in falls. The learned behavior transitioned from erratic movements to coordinated bipedal walking.

6.7 Summary of Results

Overall, the results demonstrate that the Soft Actor–Critic algorithm successfully learns stable and efficient bipedal locomotion. The trained policy generalizes across episodes and achieves reliable performance in a complex continuous control environment.

7 Discussion

- **Interpretation of Results**
  - The SAC agent successfully learned stable bipedal locomotion in a continuous control setting.
  - Cumulative rewards increased steadily, indicating effective policy learning.
  - Reduced reward variance in later training stages suggests convergence.
- **Effectiveness of Soft Actor–Critic**
  - Entropy regularization promoted efficient exploration.
  - Automatic entropy tuning reduced manual hyperparameter effort.
  - Twin critic networks minimized Q-value overestimation.
  - Off-policy learning improved sample efficiency.
- **Behavioral Observations**
  - Learned policy exhibits smooth and coordinated leg motion.
  - Stable upright posture maintained across episodes.
  - Robustness observed against terrain and initial state variations.
- **Limitations**
  - High computational cost due to long training duration.
  - Sensitivity to reward design and hyperparameters.
  - Evaluation limited to simulated environment.
- **Challenges**
  - Initial instability during early exploration.
  - Long training time on CPU-based systems.
  - Difficulty in quantitative visualization without plots.

8 Conclusion

- **Summary**
  - Bipedal locomotion was modeled as a Markov Decision Process.
  - The Soft Actor–Critic algorithm was successfully implemented.
  - Stable and efficient walking behavior was achieved.
- **Achievement of Objectives**

  - Continuous control problem solved using deep reinforcement learning.
  - No explicit motion planning or dynamics modeling was required.
  - Demonstrated effectiveness of entropy-regularized RL.

- **Key Takeaways**
  - SAC is effective for high-dimensional continuous control tasks.
  - Entropy maximization improves learning robustness.
  - Off-policy methods enhance sample efficiency.

- **Future Work**
  - Extension to more complex environments.
  - Comparative analysis with other RL algorithms.
  - Incorporation of energy efficiency metrics.
  - Real-world robotic deployment.

9 References

- Haarnoja et al., “Soft Actor-Critic Algorithms and Applications,” ICML, 2018.
- Brockman et al., “OpenAI Gym,” arXiv, 2016.
- Stable-Baselines3 Documentation.
- Gymnasium Documentation.

Appendix (Optional)

Additional plots, reward curves, and code snippets can be included here if required.