POLISH-JAPANESE ACADEMY OF INFORMATION TECHNOLOGY

Faculty of Information Technology

Department of Mechanics, Robotics and computer science
Robotics

Maciej Lizuraj, s25032

Bipedal walker reinforcement learning experiments

Bacherlor's degree thesis written under the supervision of:

Msc. Eng. Piotr Gnyś

Warsaw, February 2025

Abstract

This thesis presents experiments on addressing the challenge of bipedal walker movement using different reinforcement learning algorithms. It explores the distinct walking behaviors that emerge from training model-free, off-policy algorithms, including PPO, DDPG, and SAC. The following libraries were utilized:

- **StableBaselines3**: for algorithm implementations,
- **Gymnasium**: for the bipedal walker and environment it traverses,
- **Optuna**: for hyperparameter optimization.

The experiment involved several stages. Initially, 250 trials of hyperparameter optimization were conducted for each algorithm, leading to the identification of optimal hyperparameters. Models were then trained using these parameters, and the best-performing models were saved. Finally, the performance of these models was compared to evaluate their effectiveness.

Keywords

Rinforcement learning (RL) · Bipedal Walker · Robotic Simulations

POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Wydział Informatyki

Katedra mechaniki, informatyki i robotyki
Robotyka

Maciej Lizuraj, s25032

Eksperymenty z uczeniem przez wzmacnianie na dwunożnym robocie kroczącym

Praca inżynierska napisana pod kierunkiem:

Mgr Inż. Piotr Gnyś

Warszawa, luty 2025

Streszczenie

Ta praca prezentuje eksperymenty związane z tematem ruchu dwunożnego robota kroczącego korzystając z różnych algorytmów uczenia przez wzmacnianie. Eksploruje ona unikalne stategie chodzenia, które wyłaniają się w wyniku trenowania model-free, off-policy algorytmów takich jak PPO, DDPG i SAC. Do eksperymentów wykorzystano następujące biblioteki.

- **StableBaselines3**: dla zaimplementowanych algorytmów,
- **Gymnasium**: dla dwunożnego robota i środowiska po którym chodzi,
- **Optuna**: do optymalizacji hiperparametrów.

Eksperyment składał się z kilku etapów. Na początku przeprowadzono 250 prób optymalizacji hiperparametów dla każdego z algorytmów, co pozwoliło na identyfikację optymalnych ustawień. Następnie modele zostały przetrenowane z wykorzystaniem tych parametrów, a najlepiej działające modele zostały zapisane. Na koniec ich wydajność została porównana celem określenia ich skuteczności.

Słowa kluczowe

Uczenie przez wzmacnianie · Dwunożny robot kroczący · Symulacje robotów

Generative AI Tools Usage Disclosure

This thesis employed generative AI tools to enhance its content and structure. Specifically, ChatGPT was used to:

- Analyze existing content and suggest changes for readability and structural enhancements,
- Assist in coding by generating skeletons for certain files, which were subsequently adjusted by the author to meet the specific requirements of the thesis.

All AI-generated suggestions were reviewed before being incorporated by the author.

Table of Contents

1. Introduction

This thesis explores the performance of Proximal Policy Optimization (PPO), Deep Deterministic Policy Gradient (DDPG), and Soft Actor-Critic (SAC) algorithms in the context of bipedal walker reinforcement learning. It serves as a foundational study that can be extended with more realistic and advanced environments. The study involves three primary stages: hyperparameter optimization, model training, and performance comparison. To achieve these objectives, the tools **Optuna**, **StableBaselines3**, and **Gymnasium** were employed.

1.1 Goals

The primary goal of this thesis is to produce tangible results that compare the performance and strategies of the selected algorithms. This includes identifying strengths and weaknesses in their approaches to solving the bipedal walker movement problem.

1.2 Results

Among the tested algorithms, **DDPG** demonstrated the best performance. Detailed results, including performance metrics and walking strategies, are presented in the chapter **6 Conclusions**.

2. Robot Movement Modeling

Robots have become indispensable for automation, excelling in tasks such as managing warehouses, packaging objects, and transporting goods with far greater efficiency than humans. However, before robots can perform these tasks effectively, they must undergo training to learn the required behaviors and techniques.

Reinforcement learning provides a framework for robots to experiment with different approaches over extended periods to achieve the desired outcomes. However, testing in the real world is often impractical due to the time required and the challenges associated with resetting experiments. A more efficient and safer alternative is to simulate the environment and the robot itself, conducting all tests within this controlled virtual setting.

This simulation-based approach offers several advantages:

- **Enhanced Documentation:** All trials can be easily documented, and all variables can be precisely controlled and explained.
- **Automatic Resetting:** Simulated environments allow for automatic resets, saving time and effort.
- **Accelerated Testing:** Simulations can be run significantly faster than real-world experiments. For example, a robot moving at 0.5 m/s would take 20 seconds to traverse a 10-meter track in reality. In simulations, the only limitation is the computational power of the system, enabling faster execution.
- **Parallelization and Flexibility:** Simulations can be parallelized, paused, or resumed at any time, providing flexibility during experimentation.
- **Customisable Environments:** Simulated environments are easily modifiable. For instance, terrain features such as flatness or unevenness can be adjusted with minimal effort, whereas modifying real-world terrains requires substantial physical work.

Simulations also mitigate risks associated with real-world testing. When robots begin learning to walk, they often stumble or exhibit unnatural movements. Constant falls could cause minor damage to both the robot and its surroundings. Additionally, risks such as engine overheating or mechanical stress from attempting to move against obstacles can compromise the integrity of robotic components. By conducting these tests in a simulated environment, these risks are eliminated, and the development process becomes safer and more streamlined.

Tracks used for robot movement testing typically feature a variety of edge cases that the robot must navigate. The rationale is that if a robot can successfully traverse both common terrains and challenging edge cases, it will be well-equipped to handle any intermediate scenarios encountered in practical applications.

3. Reinforcement Learning Theory - General

Machine learning can be broadly categorized into three approaches: supervised learning, unsupervised learning, and reinforcement learning (RL). Each approach has unique advantages and is suited to solving specific types of problems.

Reinforcement learning involves two primary components: an environment and an agent. The agent interacts with the environment by performing actions, while the environment responds with states and rewards. The agent's objective is to identify patterns in its actions that maximize the cumulative reward, ultimately forming a policy.

3.1 Key Components of Reinforcement Learning

The fundamental elements of RL are as follows:

- **Actions:** The ways an agent can influence the environment, typically predefined.
- **Rewards:** Signals used to guide the agent toward desired outcomes. Rewards can be positive or negative, helping the agent differentiate between desirable and undesirable outcomes. Designing an optimal reward system is one of the most challenging aspects of RL.
- **State:** A representation of the environment provided to the agent. The agent uses this information to decide which action to take to maximize its reward.
- **Policy:** A set of rules generated during learning that maps states to actions. Policies can be categorized as:
  - **Deterministic:** Maps a single action to each state.
  - **Stochastic:** Maps a set of actions, each with an associated probability, to each state.
- **Exploration-Exploitation Dilemma:** A trade-off between exploiting known actions that yield good rewards and exploring new actions that might lead to even better outcomes.

3.2 Model-Free Algorithms

The algorithms discussed in this thesis are all **model-free**, meaning they do not attempt to predict future states or rewards. Instead, they rely on trial-and-error methods to explore different policies. While model-free algorithms are generally less efficient than model-based algorithms, they are easier to apply to complex environments.

Model-free algorithms can be classified as:

- **Value-Based:** Focus on estimating the value of different actions.
- **Policy-Based:** Directly optimize the policy to maximize rewards.
- **Actor-Critic:** Combine both approaches, where:
  - The **actor** explores the action space to maximize rewards.
  - The **critic** evaluates the quality of actions performed by the actor, providing feedback to guide the actor's decisions.

The critic's role is limited to assessing the correctness of actions based on cumulative rewards. It does not understand the reasons behind the quality of an action but simply evaluates outcomes based on predefined criteria.

3.3 Illustrative Example: Chess

To simplify the explanation of RL, consider the example of chess:

- The **agent** is one of the players.
- The **environment** is the chessboard and its pieces.
- The **actions** are all legal moves available at any given moment.
- The **state** is the arrangement of pieces on the board.
- The **rewards** are tied to achieving checkmate (positive reward for mating the opponent and negative reward for being mated). Additional rewards could be assigned for capturing pieces, but this might discourage strategies like gambits, which involve sacrificing pieces for long-term positional advantages. Balancing short-term and long-term rewards is a significant challenge in RL.

After training, the agent uses the learned policy to make optimal moves, maximizing its chances of winning.

3.4 On-Policy vs. Off-Policy Algorithms

RL algorithms can be categorized based on how policies are evaluated and learned:

- **On-Policy:** The same policy is used for both environment interaction and learning. This approach is straightforward but may limit exploration.
- **Off-Policy:** A separate policy is used for evaluation and learning, distinct from the one used for interaction with the environment. This allows the agent to learn from other, potentially superior policies, enabling more aggressive exploration and faster convergence to optimal solutions.

3.5 Proximal Policy Optimization (PPO)

Proximal Policy Optimization (PPO) is an on-policy reinforcement learning algorithm that builds upon stochastic gradient descent (SGD). In SGD, solutions are iteratively refined by probing nearby candidates, moving progressively closer to an optimal solution. However, SGD often suffers from convergence to local optima, resulting in inconsistent outcomes. Furthermore, the inherent randomness in this method can prolong the time required to achieve satisfactory results.

PPO addresses these issues using a clipping mechanism to regulate the magnitude of policy updates. Specifically, PPO limits the size of a single update to the policy within a predefined range $[1 - \epsilon, 1 + \epsilon]$, where ϵ is a hyperparameter. This clipping strikes a balance between small steps, which would require many iterations to converge, and large steps, which risk destabilizing the policy. Selecting an appropriate value for ϵ is crucial to ensuring both training stability and efficiency.

PPO is often compared to Trust Region Policy Optimization (TRPO), which also aims to limit drastic policy updates but does so using trust regions. While TRPO is theoretically sound, it is computationally expensive. PPO's simpler clipping-based approach makes it more practical and widely used.

In addressing the exploration-exploitation dilemma, PPO initially encourages exploration. However, as training progresses, the algorithm increasingly favors exploitation of known solutions, often resulting in convergence to local optima. This gradual reduction in exploration stems from the update rule, which prioritizes actions with demonstrated reward potential. While effective, this behavior can limit the algorithm's ability to discover global optima in environments with numerous local optima.

In summary, PPO is a robust and efficient algorithm that balances stability and progress during training. Its simplicity and effectiveness have made it a popular choice in reinforcement learning applications. [1][2][3][4][5]

3.6 Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradient (DDPG) is a model-free reinforcement learning algorithm designed for continuous action spaces. This makes it particularly well-suited for tasks like bipedal walking, where actions, such as the force applied to a joint, are continuous variables.

DDPG extends Q-learning by utilizing gradient-based methods to estimate Q-values. In continuous action spaces, directly calculating Q-values for all possible actions is computationally unfeasible. Instead, DDPG uses gradients to approximate Q-values based on neighboring calculations.

The algorithm leverages a **replay buffer** to store past experiences. This buffer enables the algorithm to sample and learn from previous interactions, improving stability. A buffer that is too small limits the diversity of experiences, while an excessively large buffer may introduce computational inefficiencies.

DDPG employs two neural networks:

- **Main Network:** Responsible for learning and predictions during training. It is updated at every step based on the sampled experiences.
- **Target Network:** Used to compute target values for the critic's loss function. The target network is updated using soft updates (Polyak averaging), where the target network is a weighted average of the current main and target networks.

By incorporating replay buffers and target networks, DDPG maintains stability despite being an off-policy algorithm. This stability allows the algorithm to efficiently learn from diverse experiences without destabilizing the learning process. [6][7][8][9]

3.7 Soft Actor-Critic (SAC)

Soft Actor-Critic (SAC) is an advanced reinforcement learning algorithm that combines elements of stochastic policy optimization and deterministic approaches like DDPG. The key feature of SAC is **entropy regularization**, which balances exploration and exploitation by maximizing a trade-off between expected return and entropy.

Entropy, a measure of randomness in a variable, is incorporated into the reward function. In entropy-regularized reinforcement learning, the reward is augmented by the product of the entropy and its associated coefficient. This coefficient can either be static or dynamically adjusted during training. While static coefficients are simpler to implement, dynamic coefficients are generally preferred for their adaptability. Higher coefficients encourage exploration, while lower coefficients prioritize exploitation.

SAC shares several characteristics with DDPG:

- It uses the Bellman equation to minimize the mean squared error of value estimates.
- It employs Polyak averaging to update the two networks.

The main distinction between SAC and DDPG lies in their approaches to policy and exploration:

- **Policy Type:** SAC uses a stochastic policy, while DDPG employs a deterministic policy.
- **Exploration Mechanism:** SAC achieves exploration through entropy regularization, whereas DDPG relies on noise added to its deterministic policy.
- **Action Choice:** Actions are chosen by main policy, whereas DDPG uses target policy.

By directly incorporating entropy into the optimization process, SAC maintains a good balance between exploration and exploitation throughout training. This makes SAC particularly effective in environments with high uncertainty or complex dynamics. [10][11][12][13][14]

4. Reinforcement Learning Theory - Hyperparameters

Hyperparameters are predefined parameters that are set before training a machine learning algorithm. Unlike model parameters, which are learned during training, hyperparameters govern the learning process and are fixed during training. While hyperparameters do not fundamentally alter the underlying mechanism of the algorithm, their values can significantly influence the efficiency and effectiveness of the training process. Common hyperparameters across different algorithms include the learning rate, batch size, and discount factor, though each algorithm may have its own set of unique hyperparameters.

In this study, all algorithms were optimized using Optuna by maximizing the mean reward over the course of training. Optimization was performed for 250,000 timesteps (one-quarter of the timesteps used for real training). Evaluations were conducted five times. PPO and DDPG hyperparameters were optimized over 250 different combinations, while SAC was evaluated on 103 combinations due to the high computational cost associated with the algorithm. Despite the reduced number of trials, SAC took considerably longer to test different values and train compared to PPO and DDPG, highlighting the greater computational demands of SAC.

4.1 PPO Hyperparameters

The hyperparameters tested for PPO and their corresponding value ranges are outlined in Table 4.1.[1][2][3][4][5]

Name	Range	Description
Learning rate	1e-5 – 3e-4	Controls the magnitude of policy updates. Smaller rates lead to stability but slower learning.
n steps	[2048, 4096, 8192]	Number of steps taken before each policy update. Larger values improve sample efficiency but require more memory.
Batch size	[32, 64, 128, 256]	Number of samples used per gradient update. Smaller batch sizes improve generalization but increase noise.
n epochs	5 – 20	Number of passes through the data during optimization. Larger values might result in overfitting.
Gamma	0.9 – 0.999	Discount factor for future rewards. Higher values emphasize long-term rewards.
Clip range	0.8 – 0.999	Limits the extent to which the policy can change in a single update, providing stability.

Table 4.1: PPO Hyperparameters

4.2 DDPG hyperparameters

The hyperparameters tested for DDPG and their value ranges are listed in Table 4.2.[6][7][8][9]

Name	Range	Description
Learning rate	1e-6 – 1e-3	Controls the magnitude of policy updates. Smaller rates lead to stability but slower learning.
Buffer size	100k – 1M	Size of the replay buffer that stores past experiences. Larger buffers improve the diversity of experiences but require more memory.
Batch size	[512, 1024]	Number of samples from the replay buffer used for a single update. Larger sizes improve stability but increase memory usage.
Tau	0.001 – 0.05	Coefficient for soft updates of the target network. Smaller values lead to smoother updates.
Gamma	0.95 – 0.999	Discount factor for future rewards. Higher values encourage long-term rewards.

Table 4.2: DDPG Hyperparameters

4.3 SAC Hyperparameters

The hyperparameters tested for SAC and their corresponding value ranges are summarized in Table 4.3.[10][11][12][13][14]

Name	Range	Description
Learning rate	1e-6 – 1e-3	Controls the magnitude of policy updates. Smaller rates lead to stability but slower learning.
Buffer size	100k – 1M	Size of the replay buffer that stores past experiences. Larger buffers improve the diversity of experiences but require more memory.
Batch size	[512, 1024, 2048]	Number of samples from the replay buffer used for a single update. Larger sizes improve stability but increase memory usage.
Tau	0.001 – 0.05	Coefficient for soft updates of the target network. Smaller values lead to smoother updates.
Gamma	0.95 – 0.999	Discount factor for future rewards. Higher values encourage long-term rewards.
Train freq	1 – 20	Number of steps after which the model should be updated.
Gradient steps	1 – 50	Number of gradient steps taken after every update. Larger values speed up training but increase resource demands.

Table 4.3: SAC Hyperparameters

The entropy coefficient in SAC can be set either statically or dynamically. This study used the default dynamic setting, which adjusts the coefficient throughout the training process to balance exploration and exploitation effectively.

5. Bipedal Walker Movement

The BipedalWalker-v3 environment from the Gymnasium library was used for training the bipedal walker. The walker consists of two legs and a hull, with an anatomy resembling that of a human, albeit only from the hips down. Similar to humans, the walker has knee and hip joints. Both legs are symmetrical, mimicking human biomechanics.

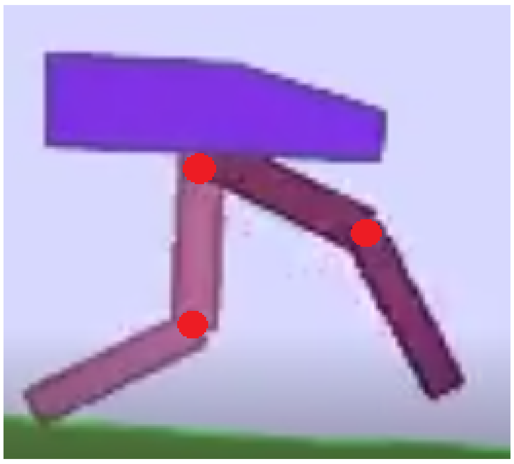

Figure 5.1: Walker joints and anatomy

Given the humanlike anatomy of the robot, it is expected that the learned walking policy will resemble the strategies developed by humans for efficient locomotion, as humans have had millennia to master walking.

All algorithms were trained for 1 million timesteps, with evaluations conducted after every update. Each evaluation consisted of 10 trials, and both the mean reward and standard deviation were monitored to select the best-performing model. While the best model typically emerged towards the end of training, this was not always the case.

The cumulative distribution graphs shown below represent the results from 1,000 episodes of the best-performing model from each algorithm. It is more practical to scrutinize a single model for each algorithm once the training has concluded, rather than continuously testing it during training.

The environment itself is relatively simple, consisting of varying terrain types.

Figure 5.2: Flat terrain environment

Some areas feature slightly hilly terrain. While these sections pose a bit more of a challenge, they are not particularly difficult to navigate.

Figure 5.3: Moderate terrain environment

Other parts of the environment are more hilly, presenting a greater challenge for the robot. These sections are more difficult to traverse, but not insurmountable.

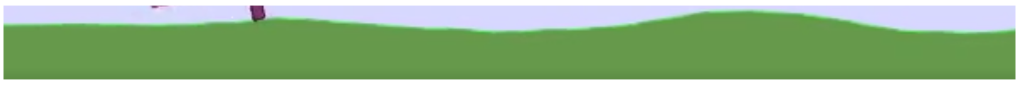

Figure 5.4: Hilly environment

The environment is designed to resemble real-world terrains that robots might encounter in everyday scenarios. The goal is not to create a robot capable of navigating all environments, but one that can effectively walk in typical everyday environments.

Although the track surface is consistent across trials, it incorporates a mix of different terrain types, encouraging the model to become proficient at navigating various surfaces. This approach is commonly used in robot navigation tasks to ensure robustness across different environments.

The scoring system is designed to balance the robot's ability to maintain stability and its velocity. Both factors are crucial for successful movement, and neither can be neglected. The scoring system can be adjusted based on the specific use case. For example, in scenarios where the robot is tasked with carrying fragile materials, maintaining stability would be more important than speed, as the risk of damaging the materials increases with each fall.

- Completing the entire track awards 300+ points, with points awarded incrementally based on the distance traveled. The value does not increase with greater distance beyond the finish line.
- To encourage efficiency and speed, the robot incurs slight costs for applying torque.
- To discourage falling, if the robot's hull touches the ground, it loses 100 points, and the episode ends. This ensures the robot does not prioritize speed over safety.

Algorithms that achieve rewards of 300 or higher are considered successful.

It is crucial to perform multiple evaluations and monitor both the mean reward and the standard deviation to ensure that the selected model is not a result of random chance. A single evaluation can lead to misleading results, so using multiple evaluations and ensuring a low standard deviation are important for reliable model selection.

Hyperparameter	Value
Learning rate	1.2e-4
n steps	4096
batch size	32
n epochs	14
gamma	0.97
clip range	0.9

Table 5.1: PPO Best Hyperparameters

5.1 PPO Results:

The optimal hyperparameters after tuning are as follows:

The best-performing model was obtained at step 815104 with a mean reward of 291.75 and a standard deviation of 0.41, as shown in the following training rewards plot:

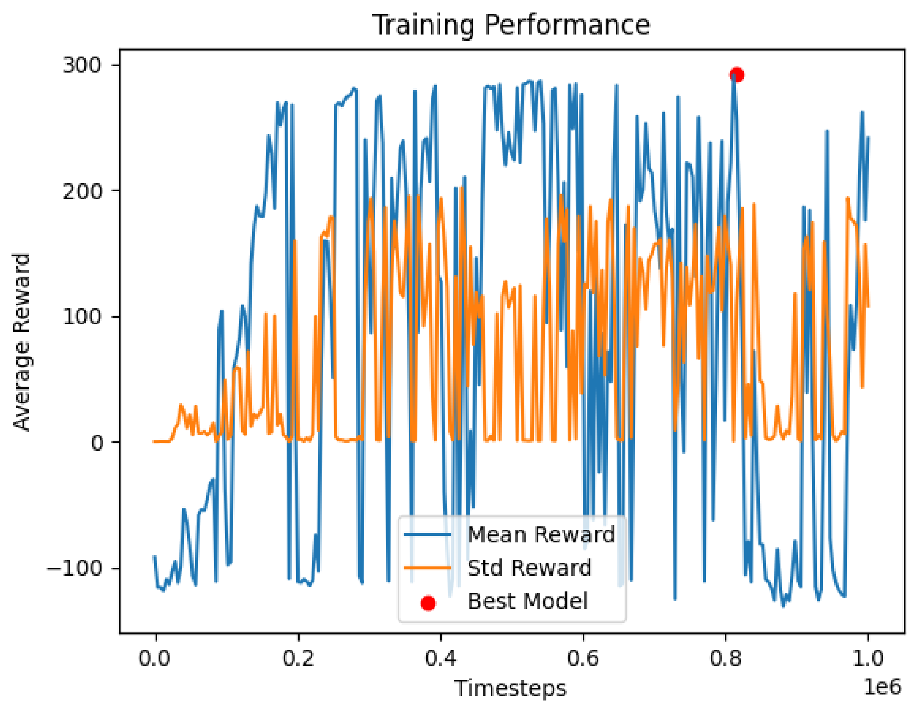

Figure 5.5: PPO Training Rewards Plot

In terms of movement, the agent designates one leg as the front leg and the other as the rear. However, the movement is inefficient, with only two joints utilized. The rear leg has a constant knee angle, transforming anything below it into a single, large foot. This increases friction and provides stability but significantly impairs movement due to the reliance on only the hip joint for motor functions. The front leg maintains a constant hip angle, with the knee joint constantly moving to keep the walker in motion. The movement alternates between two distinct phases: stabilization and forward movement.

In the stabilization phase, the front leg stays close to the hull, while the rear leg's knee-foot part fully contacts the ground:

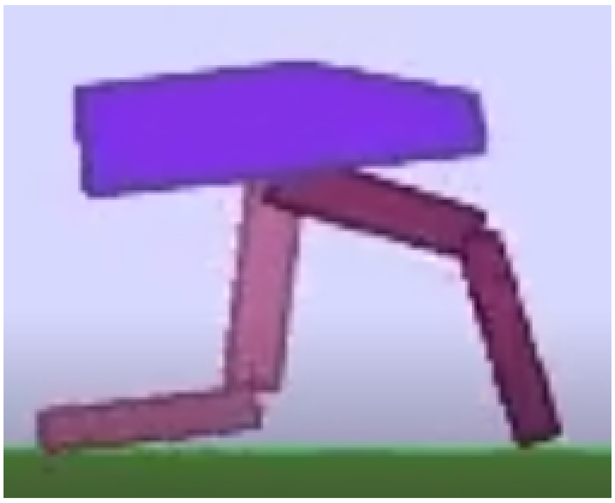

Figure 5.6: PPO Movement - Stable Phase

In the forward movement phase, the front leg leaps forward while the rear leg is angled differently, with the difference primarily visible in the position of the front leg:

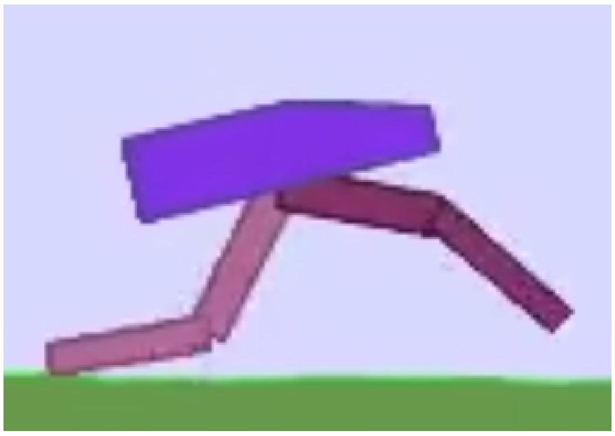

Figure 5.7: PPO Movement - Forward Phase

Rewards below 280 are considered failures for this algorithm, and the cumulative distribution plot shows where these failures occurred. Episodes above 200 are inefficient but do not result in falls.

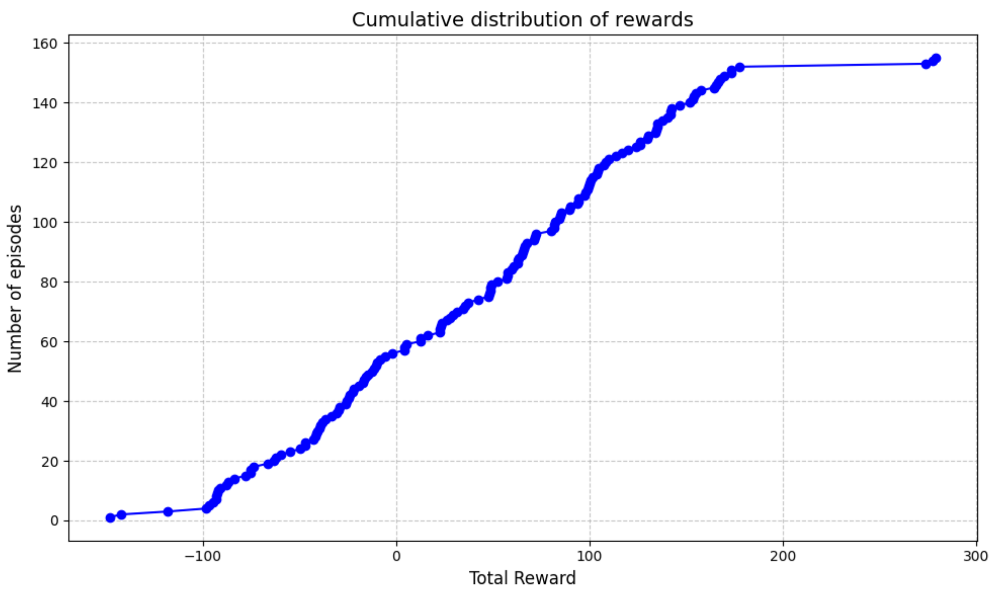

Figure 5.8: PPO Cumulative Distribution Plot

The distribution of failures is fairly constant across trials. The main cause of failures is poor coordination between the legs. If the front leg attempts to move forward before the rear leg has fully stabilized, the walker loses balance. Once the walker tips beyond a certain threshold, it cannot recover. This tipping point is easily triggered, leading to a failure rate of approximately 15%.

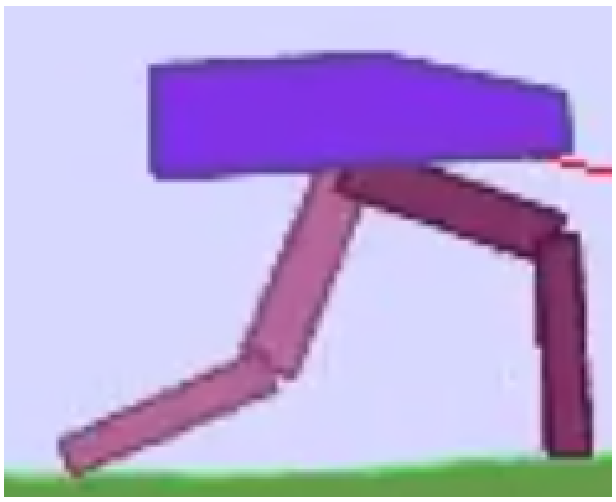

Figure 5.9: PPO Movement - Frame After Two Consecutive Forward Phases

This hinders the ability to go into Stable Phase, causing more consecutive forward phases without stabilization, resulting in a fall.

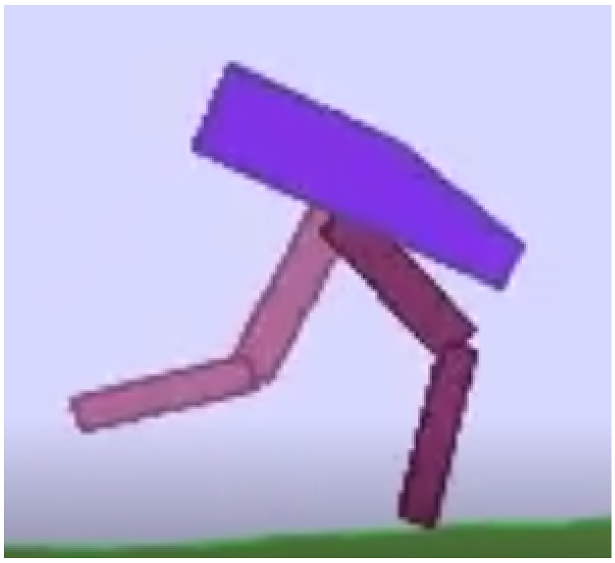

Figure 5.10: PPO Movement - Few Frames After Two Consecutive Forward Phases

5.2 DDPG Results

The optimal hyperparameters after tuning are as follows:

Hyperparameter	Value
Learning rate	8.8e-4
gamma	0.99
Batch size	1024
Buffer size	400000
Tau	0.022

Table 5.2: DDPG Best Hyperparameters

The best-performing model was obtained at step 990208 with a mean reward of 312.80 and a standard deviation of 1.45, as shown in the following training rewards plot:

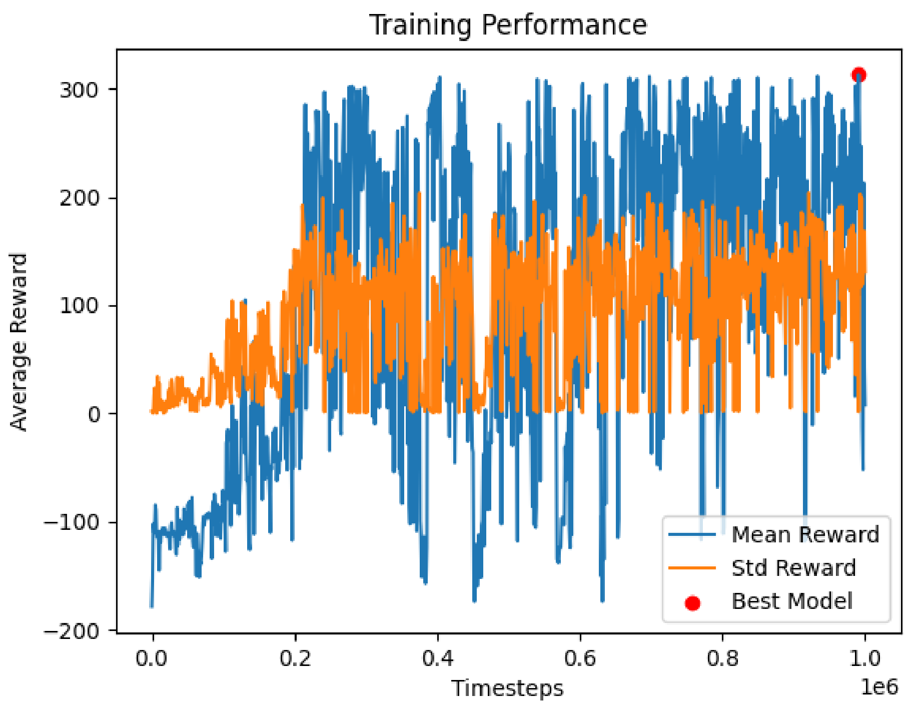

Figure 5.11: DDPG Training Rewards Plot

This agent, rather than running, spends most of its time skipping. Its knees are constantly held high. It alternates which leg powers the movement, mostly relying on the hip joints, though it does slightly utilize the knee joints as well. Unlike the PPO model, there are no glaring issues, such as a primitive foot.

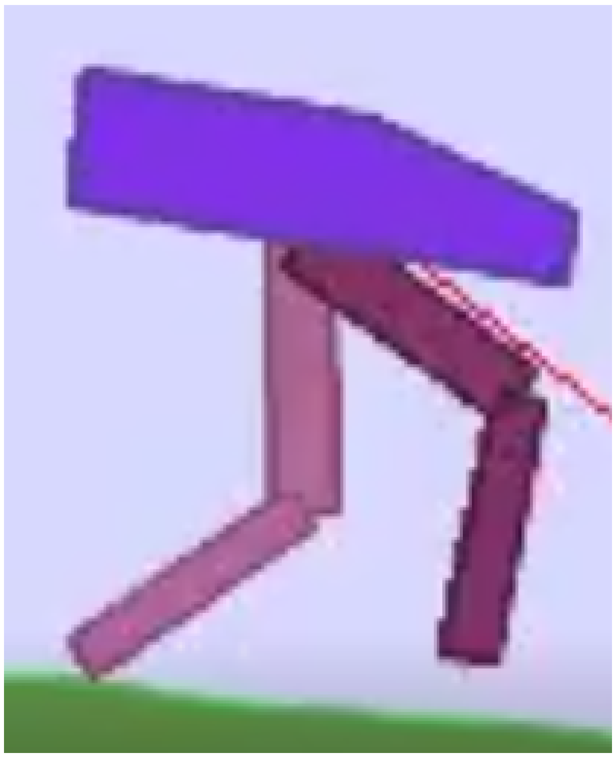

Figure 5.12: DDPG Movement - Regular Skip

Skipping can sometimes lead to dangerous behaviors, such as skipping to dangerous heights. While this seems likely to result in failure, it does not always do so.

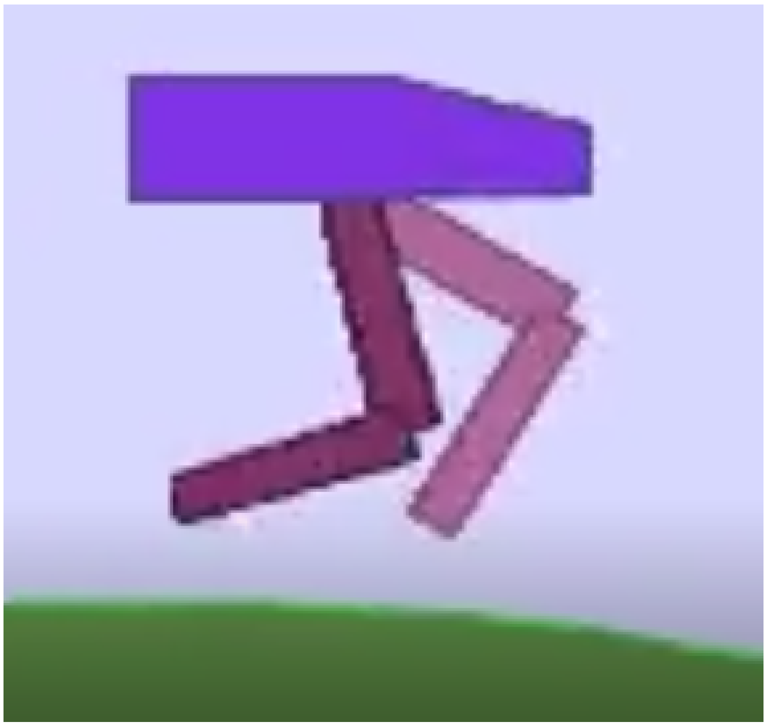

Figure 5.13: DDPG Movement - High Skip

Rewards below 280 are considered failures for this algorithm, and the cumulative distribution plot shows where these failures occurred. Episodes above 200 are inefficient but do not result in falls.

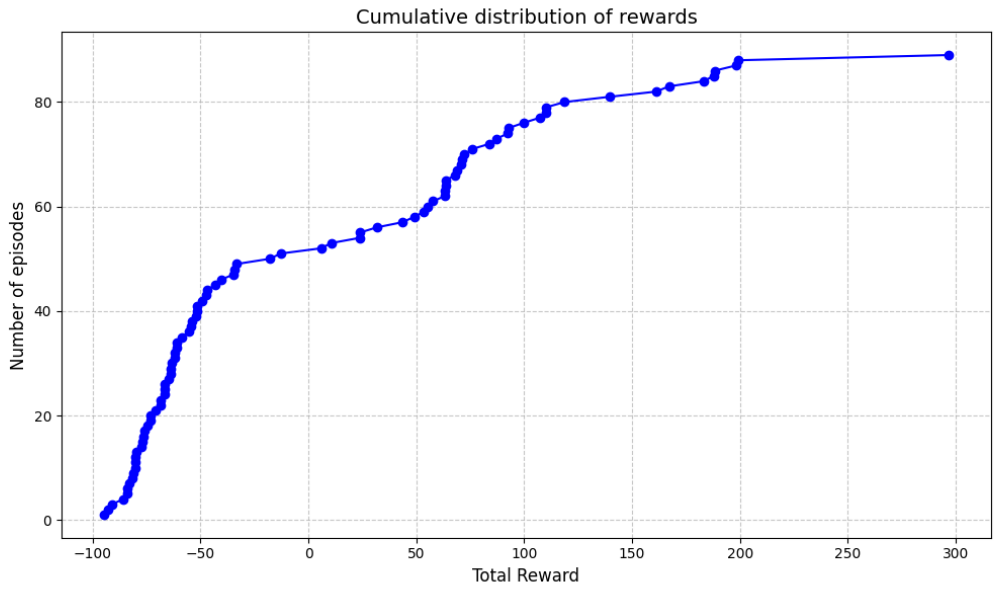

Figure 5.14: DDPG Cumulative Distribution Plot

The distribution reveals two spikes in the number of failures. The first occurs early on and is caused by an initial faulty skip. While skipping is an efficient movement, it is prone to errors. If the agent fails at the beginning, it often struggles to recover, as the right angle is necessary to start skipping. A failed skip leads to a chain reaction, causing the walker to struggle to balance on one leg.

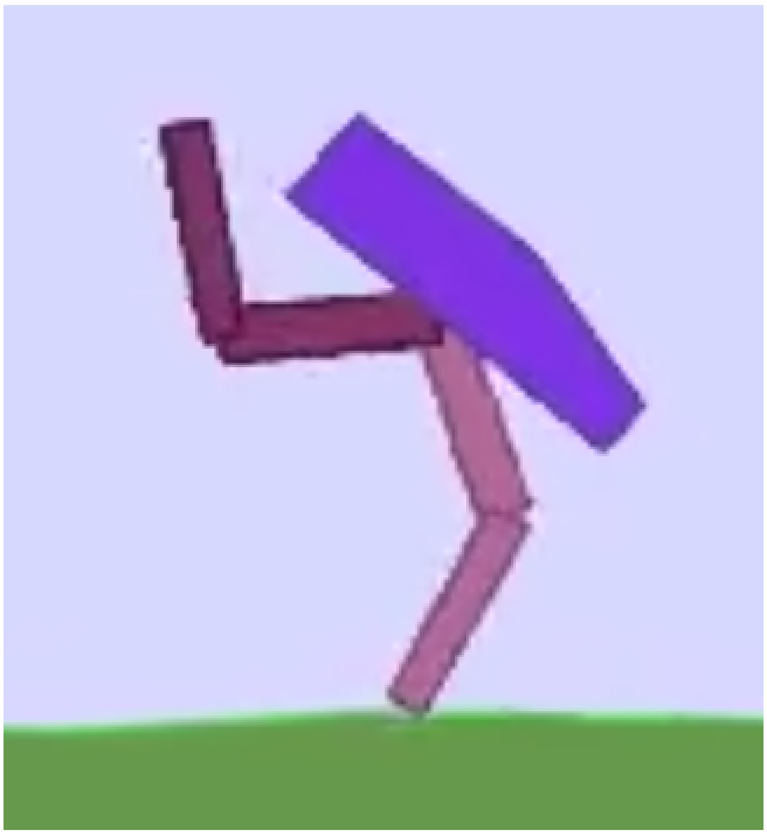
Figure 5.15: DDPG Movement - Failed Skip Balancing

The second spike occurs due to terrain challenges around the middle of the map. Both uphill and downhill skipping pose significant challenges for the agent. Since the agent lacks a mechanism to predict future steps, it cannot adjust its skipping angle to avoid losing balance.

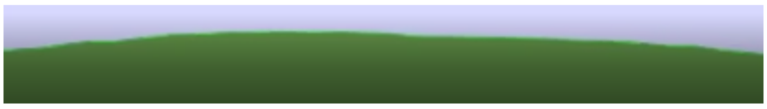
Figure 5.16: DDPG Hill of failures

Apart from the initial and terrain-related problems, the DDPG algorithm performs well, with a failure rate of less than 10%.

5.3 SAC Results

The optimal hyperparameters after tuning are as follows:

Name	Range
Learning rate	7.7e-4
Buffer size	100k.
Batch size	1024
Tau	0.037
Gamma	0.96
Train freq	3
Gradient steps	36

Table 5.3: SAC Best Hyperparameters

The best-performing model was obtained at step 507904 with a mean reward of 324.38 and a standard deviation of 0.6. It is quite surprising that the best performance was found in the middle of the training.

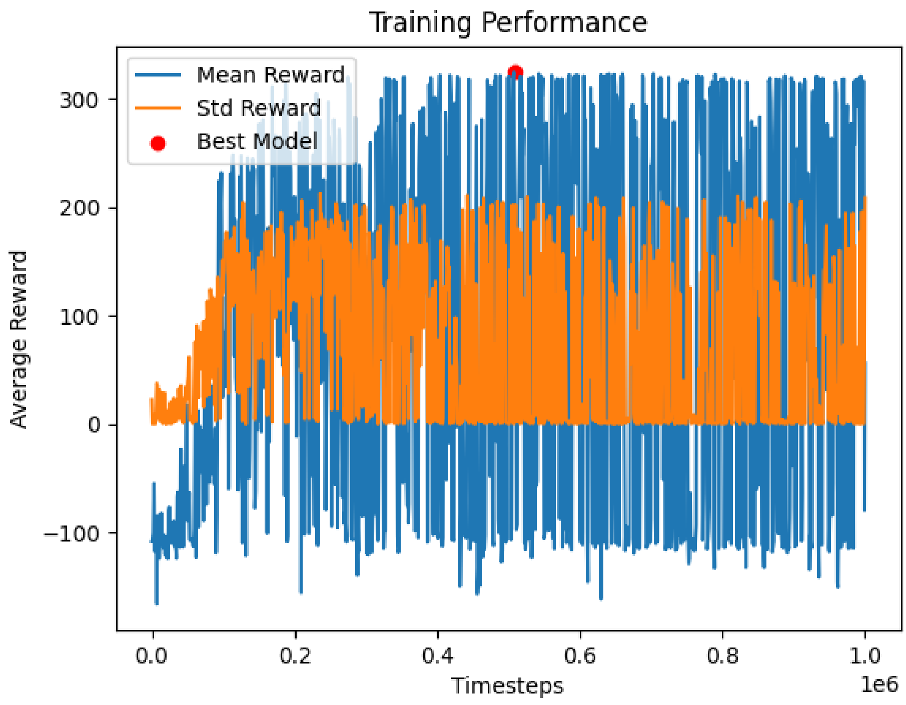

Figure 5.17: SAC Training Rewards Plot

This agent, like the PPO agent, designates one leg as the front leg and the other as the rear leg. It moves with very small steps and keeps its knees low. Unlike the PPO walker, it uses both feet more effectively. The legs are not as rigid, although the joints that were underused in PPO are still not fully utilized here. It exhibits similar issues but moves at a significantly better speed.

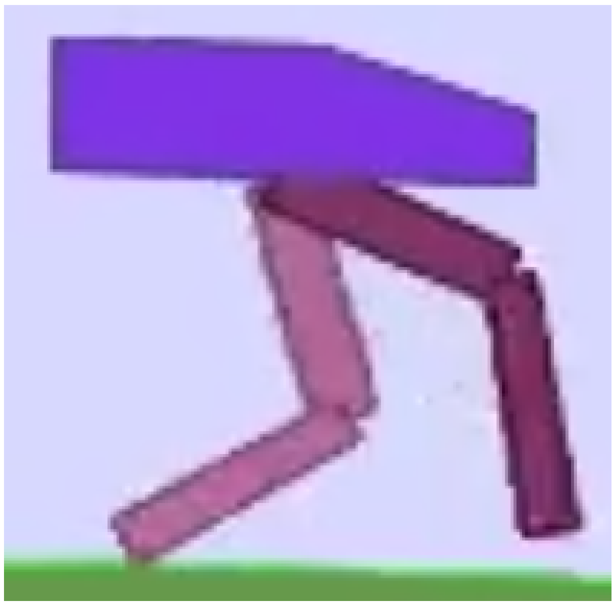

Figure 5.18: SAC Movement

Rewards below 315 are considered failures for this algorithm, and the cumulative distribution plot shows where these failures occurred. Unlike in case of previous algorithms, the failure with biggest reward is around 200 points. This means that all failures ended up in falling, rather than managing to get to the end in a very inefficient manner. However it is worth noting that the failure rate reaches 17.5%.

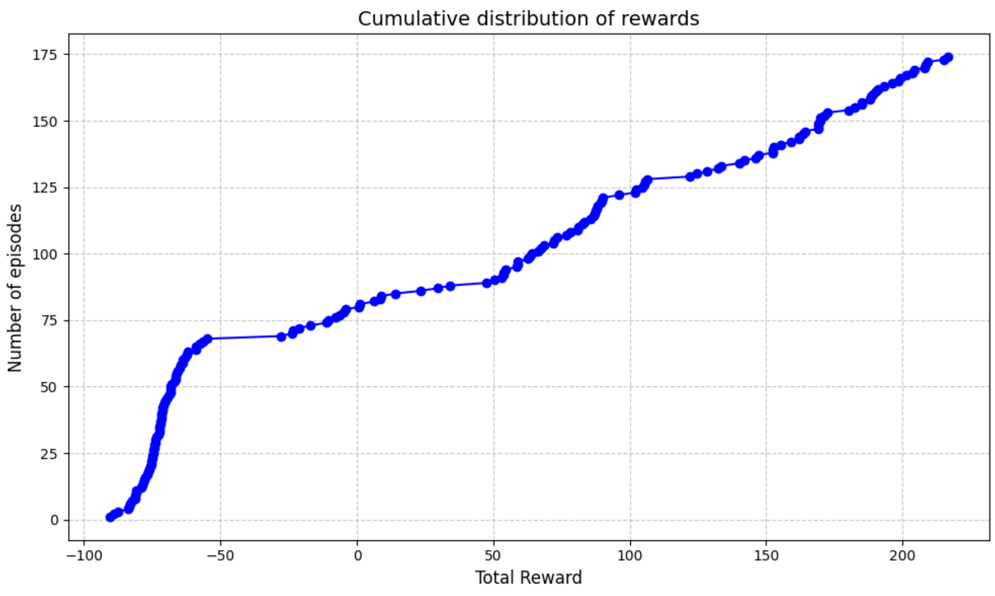

Figure 5.19: SAC Cumulative Distribution Plot

One common failure occurs during the first few steps when the rear leg is initially designated as the front leg. The leg assignments switch at the very beginning, which sometimes leads to failure. This can be seen in the -100 to -50 reward range in the graph. Other failures are caused by the agent attempting to ascend a hill. The small step strategy struggles in areas where the terrain becomes uneven.

6. Conclusions

Each algorithm developed a distinct walking strategy. It is worth noting that the initial 10-episode tests during training showed better results than the later 1k-episode evaluations used to test the best models.

The weakest algorithm was PPO. It failed to reach the 300-reward threshold, which is considered the standard for a good walking algorithm. The shortcomings of its strategy are immediately apparent. From inefficient use of the knee-foot segment of one leg, through the rigid designation of one leg as the front and the other as the rear, to utilizing only half of the available joints, its movement is highly suboptimal. Even without examining the data, merely observing the algorithm's motion reveals its severe deficiencies. Compared to the other algorithms, PPO performed poorly, receiving a mean reward of only 253.1, far below the 291.8 points achieved during training. Its failure rate of 15% further underscores its weaknesses. Successful attempts required around 21 seconds to reach the end.

SAC, while also designating legs as front and rear, showed significant improvements over PPO. None of the walker's components were visibly misused, though joint usage remained suboptimal. The rear leg's contact with the ground was minimal, and its angle was inefficient. Despite achieving the highest training reward of 324, SAC recorded a much lower mean reward of 274.3 during the 1k-episode evaluation, highlighting the agent's volatility. Additionally, it exhibited the highest failure rate among the tested algorithms at 17.5%. Successful attempts took approximately 14 seconds to complete the course.

DDPG adopted a unique skipping strategy, which proved to be the most effective. It demonstrated the best balance between velocity and stability. Unlike the other algorithms, DDPG alternated between legs during movement, making its motion more efficient and smoother. This alternating strategy utilized inertia effectively and allowed the agent to better manage its center of mass, reducing the failure rate to just 9%. During training, DDPG achieved a mean reward of 312.8, while the 1k-episode evaluation resulted in a slightly lower mean reward of 285.2. This showed the stability of the model. Its lower failure rate and efficient movement made DDPG the best-performing algorithm. While skipping may not be the ideal form of locomotion, the ability to alternate legs provided DDPG with a distinct advantage over the other algorithms.

Rank	Algorithm	Mean reward	Failure rate	Time to finish
1	DDPG	285.2	9%	14s
2	SAC	274.3	17.5%	14s
3	PPO	253.1	15%	21s

Table 6.1: Algorithms comparison on 1k episodes

Bibliography

[1] OpenAI, “Proximal policy optimization documentation,” https://spinningup.openai.com/en/latest/algorithms/ppo.html, [Online; accessed 21 January 2025].

[2] S. Baselines3, “Ppo documentation,” https://stable-baselines3.readthedocs.io/en/master/modules/ppo.html, [Online; accessed 21 January 2025].

[3] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” https://arxiv.org/abs/1707.06347.

[4] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, “High-dimensional continuous control using generalized advantage estimation,” https://arxiv.org/abs/1506.02438.

[5] N. Heess, D. TB, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. Riedmiller, and D. Silver, “Emergence of locomotion behaviours in rich environments,” https://arxiv.org/abs/1707.02286.

[6] OpenAI, “Deep deterministic policy gradient documentation,” https://spinningup.openai.com/en/latest/algorithms/ddpg.html, [Online; accessed 21 January 2025].

[7] S. Baselines3, “Ddpg documentation,” https://stable-baselines3.readthedocs.io/en/master/modules/ddpg.html, [Online; accessed 21 January 2025].

[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, “Continuous control with deep reinforcement learning,” https://arxiv.org/abs/1509.02971.

[9] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmille, “Deterministic policy gradient algorithms,” http://proceedings.mlr.press/v32/silver14.pdf.

[10] OpenAI, “Soft actor-critic documentation,” https://spinningup.openai.com/en/latest/algorithms/sac.html, [Online; accessed 21 January 2025].

[11] S. Baselines3, “Sac documentation,” https://stable-baselines3.readthedocs.io/en/v1.0/modules/sac.html, [Online; accessed 21 January 2025].

[12] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, “Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor,” https://arxiv.org/abs/1801.01290.

[13] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, “Learning to walk via deep reinforcement learning,” https://arxiv.org/abs/1812.11103.

[14] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine, “Soft actor-critic algorithms and applications,” https://arxiv.org/abs/1801.01290.