BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES
DOI: 10.24425/bpasts.2026.158974

THIS IS AN EARLY ACCESS ARTICLE.
This article has been accepted for publication in a future issue of this journal,
but has not been fully edited. Content may change prior to final publication.

ES-SAC: A Hybrid Evolution Strategy and Reinforcement Learning Approach for Humanoid Locomotion Control

Mustafa Ayyıldız[1]*, **Övünç Polat**[2]

[1] The Vocational School of Technical Sciences, Department of Electronics and Automation, Akdeniz University, Antalya, Turkey
[2] Faculty of Engineering, Department of Electrical and Electronics Engineering, Akdeniz University, Antalya, Turkey

Abstract. State-of-the-art deep reinforcement learning (DRL) techniques such as Soft Actor-Critic (SAC), Twin Delayed Deep Deterministic Policy Gradient (TD3), and Deep Deterministic Policy Gradient (DDPG) have demonstrated promising results in developing control strategies. In this study, we propose ES-SAC, a hybrid learning framework that integrates Evolutionary Strategy (ES) with the SAC algorithm to enhance humanoid robot locomotion control. ES-SAC leverages the global search capabilities of evolutionary algorithms and the sample efficiency and convergence properties of DRL. The performance of the ES-SAC agent was evaluated on a bipedal robot simulation and compared to other hybrid methods employing deterministic agents, including ES-TD3 and ES-DDPG. The ES-SAC agent exhibited superior average reward performance and a more stable learning process. In contrast, the ES-TD3 agent achieved faster course completion but exhibited control instabilities. This study also highlights the importance of physical and behavioral metrics –such as torque efficiency, horizontal and vertical deflection, and Q_0 values– in assessing the reliability of DRL-based locomotion control. Our findings suggest that relying solely on cumulative reward for evaluation can be misleading, underscoring the need for a more comprehensive analysis in future research.

Keywords: actor-critic algorithms; evolutionary computation; humanoid robots; Reinforcement learning.

1. INTRODUCTION

Reinforcement learning (RL) is a machine learning paradigm in which an agent learns to maximize a numerical reward signal by mapping states to actions. Unlike supervised learning, the agent is not explicitly told which actions to take; instead, it explores the environment and identifies actions that yield higher rewards [1].

RL algorithms operate within dynamic environments, aiming to discover the optimal sequence of actions that lead to the best outcomes [2]. By continuously interacting with the environment, the agent learns and refines its policy to improve future performance [3].

In RL, an agent's policy is a function that maps observations (state inputs) to actions (outputs). Upon receiving observational data, the policy determines the most appropriate action to take. For walking humanoid robots, this observational data typically includes joint angles, body acceleration and angular velocity, foot contact information, and more. The policy processes this sensory input to generate motor commands, moving the robot's arms and legs accordingly [4, 5]. The environment then returns a numerical reward that reflects the quality of the chosen action. Based on the observations, actions taken, and the received rewards, the RL algorithm updates the policy to improve future decisions [6]. Consequently, in any given state, the agent aims to select the optimal action that maximizes the cumulative reward in the long run [1, 7].

In RL, the neural networks within the agent's architecture represent both the policy and the value function, typically using an actor-critic structure [1]. The actor network generates actions based on the current state, thereby defining the agent's policy. The critic network, in turn, evaluates these actions by estimating the Q-value of the current state-action pair. When the actor selects an action and applies it to the environment, the critic assesses the quality of the action by predicting the expected reward (Q-value) [8]. The critic then evaluates the accuracy of its value estimate using the reward received from the environment. The resulting error is the difference between the critic's previous prediction and the updated estimate based on the immediate reward and the new state. The critic updates the value function using this error, improving its future predictions. The actor updates its parameters based on feedback from the critic, refining the policy to select actions that yield higher expected rewards in the future [9]. This setup allows the policy to optimize along the reward gradient suggested by the critic, rather than relying solely on raw rewards [7].

In environmental modeling for robotic locomotion, physical components such as the dynamic and kinematic models of the robot, actuator configurations, limb lengths, and the surface

*e-mail: mustafaayyildiz@akdeniz.edu.tr

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

conditions are defined. Model-free deep RL (DRL) algorithms, by contrast, learn how to interact with the environment without relying on explicit models of these physical properties [1]. DRL has been successfully applied in various challenging domains of robotic control [10, 11], demonstrating satisfactory performance [12].

In model-free systems, the agent design can vary significantly depending on the application. In this study, we employ and compare three widely used RL algorithms for continuous action spaces [13]: Deep Deterministic Policy Gradient (DDPG), Twin Delayed Deep Deterministic Policy Gradient (TD3), and Soft Actor-Critic (SAC). Performance comparisons and evaluations of these agents are presented in the subsequent sections.

The DDPG and the TD3 algorithm are an off-policy actor-critic method designed for environments with continuous action spaces. DDPG employs a critic network to compute the Q-value, enabling it to estimate the value of the optimal policy, and it learns a deterministic policy. DDPG incorporates target actor and critic network, along with an experience replay buffer, to improve training stability [14, 15]. While sharing a similar structure with DDPG, TD3 also utilizes target networks and an experience buffer but enhances stability and performance by employing two critic networks for more robust Q-value estimation [16].

The SAC algorithm is an off-policy actor-critic method designed for environments with both continuous and discrete action spaces. SAC learns a stochastic policy that maximizes a trade-off between expected cumulative rewards and policy entropy. The entropy term measures the uncertainty of the policy's actions in a given state, with higher entropy promoting greater exploration and improving robustness during training [1, 12]. SAC employs two critic networks to enhance the accuracy of policy evaluation and also uses target critics and an experience replay buffer [17].

Over time, these foundational algorithms have been integrated with complementary strategies and algorithmic structures to capitalize on their strengths while mitigating their limitations. Specifically, hybrid approaches that combine these agents with evolutionary strategies have been proposed and shown to improve performance in various environments [18, 19].

Evolutionary algorithms are a class of search algorithms based on three fundamental operations: generating new solutions, modifying existing ones, and selecting the optimal solutions. These operations are iteratively applied to a population of candidate solutions. Throughout this process, promising solutions are preserved while new candidates are continuously introduced [19]. The selection mechanism is typically probabilistic, favoring solutions with higher fitness values, which serve as indicators of solution quality. As generations progress, the overall quality of the population is expected to improve [20, 21].

The agent developed by combining Evolutionary Strategy (ES) and RL (ES-RL) employs a hybrid training structure. ES-RL extends from the Evolutionary RL (ERL) framework, whose core component is the replay buffer system. One example of this approach, proposed by Pourchot and Sigaud [18], involves integrating the Cross-Entropy Method (CEM) with non-policy-based RL techniques. In ERL, a population of actor networks is optimized using standard evolutionary algorithms, and the resulting experiences are stored in the replay buffer. This shared experience data is used to train both the actor and critic networks, effectively transferring information from various behavioral policies to the RL model [18, 22].

The general process of the ES-RL algorithm consists of the following steps. First, a population of actor networks with randomly initialized weights is established independently of the RL system. Additionally, a separate actor–critic network is initialized [23, 24]. The population of actors interacts with the environment to generate actions and receive corresponding rewards. The resulting experience data is stored in a replay buffer, which is subsequently used to update both the actor and critic networks via DRL techniques [25, 26]. The fitness of each actor individual is computed based on the average total reward accumulated during an episode. This fitness score serves as a measure of individual performance. During the selection phase, actors are ranked according to their fitness scores, and the top-performing individuals (elites) are selected to create the next generation. Parameter variations –typically implemented as noise perturbations– are applied to these elites to form a new population. This iterative cycle of evaluation, selection, and variation continues until the algorithm converges or reaches a predefined stopping criterion [24, 27].

In this study, the evaluation goes beyond the numerical value of the reward function by incorporating performance metrics that directly affect locomotion reliability. In particular, torque efficiency, lateral and vertical deviations, forward velocity, and Q-value estimates are considered as complementary indicators. Together, these measures provide a broader perspective on controller performance in humanoid robot locomotion. In contrast to the previous study, which examined arm-swing coordination patterns within a fixed ES-SAC framework under flat and uneven terrain conditions [28], the present study addresses the problem from an algorithmic perspective. The main objective is not to compare gait modes, but to evaluate the relative performance of ES-SAC, ES-TD3, ES-DDPG, and standalone RL agents in flat-ground humanoid locomotion. Accordingly, the main contribution of the present study lies in controller-level benchmarking rather than in the analysis of arm-swing-induced gait differences.

2. RECURRENT NETWORK SYSTEM

2.1. Actor network.

The actor network, as depicted in Fig. 1a, serves as a crucial component in the policy. It receives the current state of the agent as input and outputs an action. The action selection is made by sampling from a parameterized probability distribution, utilizing deep artificial neural networks, the actor network is capable of learning complex relationships between states and actions [29]. It can be designed as either a multilayer feedforward network or a recurrent network [30]. In each layer, a linear transformation is performed using weight and bias

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

parameters, followed by a nonlinear output generated by the activation function [31].

The first layer of the network receives the state vector from the environment and transforms it into a high-dimensional hidden representation, extracting meaningful features from the raw input data. Subsequent fully connected (FC) hidden layers enhance the representational power of the network, modeling the state-action relationships of the policy. The activation function used after each FC layer is the Rectified Linear Unit (ReLU), which allows the network to learn nonlinear decision boundaries. If the input to the ReLU activation function is positive, it is transmitted directly; otherwise, it is set to zero. This property enables positive activations to pass through deep network layers without losing their derivative, which helps reduce the vanishing gradient problem compared to the sigmoidal function [32].

The actor network includes feedforward layers as well as a Long Short-Term Memory (LSTM) layer, which is particularly useful when the time dimension is critical or when complete observations are not available –typically in non-Markovian environments. The LSTM can store input and state data from previous time steps and incorporate this information into its output. This enables the actor to generate actions based on observations over time and helps mitigate vanishing-gradient issues. Owing to these features, states that are not directly observed can be indirectly represented within the LSTM layer, partially compensating for non-Markovianity [33, 34]. The parameters of the LSTM layer are updated using backpropagation through time, with the total loss of the network computed by applying the chain rule along the time dimension. This approach helps avoid gradient explosions and ensures satisfactory performance [35].

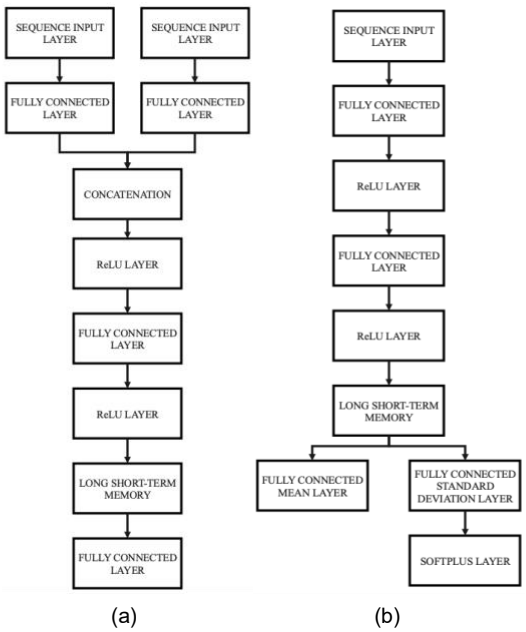

Fig.1. Actor (a) and critic (b) networks

After the LSTM layer, the actor network branches into two parts. One branch produces the mean of the action distribution, while the other computes the standard deviation (SD) value. The mean output determines the actions that the policy prefers, whereas the SD output regulates the level of uncertainty and exploration. Because the SD term must be non-negative, a Softplus layer is added after this output. The Softplus function preserves derivative information in the near-zero region, ensuring smooth gradients. This is essential, as the network's output represents a probabilistic action distribution rather than a deterministic one. A stochastic policy can be differentiated and optimized through its probability density. When the entropy value is high, the SD layer produces large values, leading to a wider distribution and promoting exploration. Conversely, when the network focuses on specific actions, the SD value becomes smaller, resulting in a narrower distribution and more stable action selections. This trade-off is managed by entropy weighting, ensuring a balance between exploration and exploitation [36].

In the learning process, the output distribution is both a result of and a factor influencing this process. Initially, the SAC agent explores extensively by following a high-entropy policy, trying different actions to gather diverse experiences. As learning progresses, the agent begins to prefer more rewarding actions. However, in the SAC algorithm, entropy does not decrease to zero over time; rather, it is dynamically adjusted based on the policy's uncertainty level [12]. When the agent begins to overfocus on specific actions and reduces its exploration, the entropy bonus encourages broader exploration again. This prevents the agent from getting stuck in local optima and ensures continued long-term exploration [37]. In other words, the SAC algorithm systematically establishes a balance between exploration and exploitation, enabling the agent to learn the best actions while still trying new options when necessary [38].

2.2. Critic network.

In the RL framework, the actor and critic networks learn together. The actor network produces an action output based on the current state, while the critic network, illustrated in Fig. 1b, evaluates this action numerically by outputting a Q-value. This value reflects how good the action is. The actor network then updates its policy parameters to maximize the Q-value, ensuring better decision-making. In the SAC agent, this process is further enhanced by entropy regularization, which prevents convergence to a single, deterministic solution and maintains high exploration capabilities [39]. The critic network processes state inputs through FC layers to generate a state feature vector. Action inputs are processed similarly by a separate subnet. These two vector structures are then combined and passed through nonlinear activation functions, resulting in a scalar Q-value. The ReLU activation function allows the model to learn complex state-action relationships by introducing non-linearities. Additionally, the LSTM layer helps the model capture temporal dependencies by retaining information from both current and past states [40].

The derivatives of the loss function with respect to the network weights are computed via backpropagation, and the critic network parameters are updated accordingly. This optimization process uses the Adaptive Moment Estimation

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

(Adam) variant [41]. The Adam optimizer combines the strengths of Stochastic Gradient Descent and RMSProp, offering a more robust and efficient update mechanism. It does so by considering the moving average of both the gradients and their squared values [41].

At each update, randomly selected samples from the replay buffer are transferred to mini-batches [12]. The difference between the output of the critic network and the target value is used to minimize the prediction error, and the system is expected to learn a Q-function that converges to the Bellman equations over time. Although this process is performed separately for each critic network, the target value is common, so the learning processes progress similarly [42]. To avoid convergence to the same values, the SAC agent uses the double Q-update strategy [43]. Additionally, when calculating the target value, the minimum of the predicted Q-values is taken. This approach reduces the overestimation bias that can arise and provides a more stable learning process by minimizing the chance of identical predictions from the critic networks with two different parameter sets [44].

Target network structures are employed during the update of the critic networks. Architecturally, the target network is identical to the critic network, but its parameters are updated more smoothly using Polyak averaging. As a result, the target Q-network lags behind the critic-Q network and acts as a stabilizing buffer, making the output values more stable. This is particularly important in the early stages of training, when the critic network lacks sufficient previous predictions and is inherently unstable. The target network mitigates this instability. Each critic network has its own target network, and these play a crucial role in the training process [44]. Critic networks are updated more rapidly than the target networks and the actor network. Meanwhile, the actor network is updated less frequently to allow the critic networks to sufficiently refine their value estimates and maintain a stable policy. Moreover, the stochastic policy-specific noise provides a natural smoothing effect, helping the actor network avoid unnecessary updates [16].

3. MATHEMATICAL MODELING

3.1. SAC Agent Structure

The SAC algorithm is a contemporary DRL method built on the principle of maximum entropy. In traditional RL approaches, the objective is to maximize the expected cumulative reward for a given policy $\pi(a_t|s_t)$, where π denotes the policy function, a_t the action taken at time t, and s_t the state at time t. However, the SAC algorithm extends this objective by additionally maximizing the entropy of the policy distribution. This promotes greater exploration, leading to a more robust and balanced learning process [12]. The overall architecture of the SAC, which embodies these principles, is illustrated in Fig. 2.

The SAC algorithm is built on a framework comprising a single actor network, two independent critic-Q networks with their target counterparts (target Q-networks), and an experience buffer. Upon receiving the current state s_t, the actor network produces an action distribution as shown in Equation (1):

$$\pi_\theta(a_t|s_t) = \mathcal{N}\big(\mu_\theta(s_t), \sigma_\theta(s_t)\big). \quad (1)$$

Here, θ denotes the learnable parameters of the actor (policy) network –specifically, the weights and biases updated during the learning process. $\mu_\theta(s_t)$ and $\sigma_\theta(s_t)$ represent the mean and SD of the action distribution for the given state s_t depending on ϕ, respectively. The SD $\sigma_\theta(s_t)$, determines the variability of actions generated by the policy, thereby influencing the exploration-exploitation balance. $\mathcal{N}(.)$ denotes the normal (Gaussian) distribution from which the policy samples its actions [42].

The mean (μ_t) and SD (σ_t) values of these output actions are learned parameters, and the action selection process is inherently stochastic. Unlike classical deterministic policies, this stochastic policy formulation naturally promotes exploration, which in turn facilitates the development of more robust policies.

In the training process, the critic-Q networks $\big(Q_k(s, a; \phi_k)\big)$ are initialized with random parameters (index k denotes each critic). The target critic networks are initially set equal to the main critics as in Equation (2):

$$\phi_{tk} = \phi_k. \quad (2)$$

where ϕ denotes the learnable parameters of the critic-Q networks, and ϕ_t represents the corresponding parameters of the target critic networks.

Simultaneously, the actor network ($\pi(a|s;\theta)$) is also initialized with random parameters. Following the initialization of these core network structures, a warm-start phase is conducted. During this phase, the agent selects actions based on a random policy, and the resulting experiences –comprising the state, action, reward, and next state– are stored in the experience replay buffer [45].

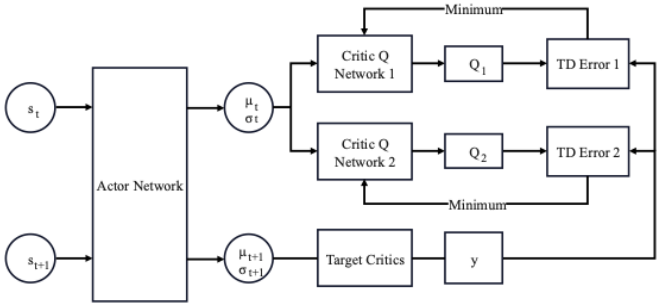

Fig.2. SAC architecture (adapted from [17])

Using $\pi(a|s;\theta)$ and the available observation data, an action a is selected. In this study, the action is constrained within continuous values to suit the continuous action space. The selected action is then applied in the environment, which yields a reward (r) and a new observation (s'). This tuple (s, a, r, s') is stored in the experience replay buffer [42]. To train the model, the mini-batch method is employed, wherein small, randomly sampled batches are drawn from the experience pool containing a large dataset. In recurrent neural network-based systems, each mini-batch comprises different

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

sequences of experiences, and each sequence contains multiple consecutive experience tuples. This approach contributes to more stable and efficient learning [46]. The training process involves updating the critic network, actor network, and the entropy coefficient for each mini-batch, as described below.

The main purpose of the twin (or binary) critic-Q networks is to mitigate the overestimation bias problem that can arise in value-based RL. Each critic network independently provides an estimated Q-value for a given state-action pair. To ensure more robust and stable learning, the target Q-value is computed by taking the minimum of the two Q-value estimates (Equation (3)):

$$y_i = r_i + \gamma \min_k \left(Q_{tk}(s'_i, a'_i; \phi_{tk})\right) - \alpha \ln \pi\,(a'_i|s'_i; \theta), \quad (3)$$

where r_i, γ and α are the immediate reward received by the agent, the discount factor (a value between 0 and 1), and the entropy coefficient, respectively. The discount factor multiplied by the minimum estimated Q-value from the critic-Q networks reflects the importance of future rewards. Selecting the minimum value helps to mitigate overestimation bias. The subscript t in Q and ϕ denotes the parameters of the target networks. The target Q-value used in policy updates enhances learning stability and helps prevent incorrect updates. The target critics network computes this target Q-value using the actions generated by the policy at the next time step. In each iteration, the next state and the corresponding action are represented by s'_i and a'_i, respectively [42]. The entropy term $(\alpha \ln \pi\,(a'_i|s'_i, \theta))$ is directly incorporated into the objective function, encouraging balanced exploration during learning [12]. Consequently, the agent not only maximizes the expected reward but also learns more reliable and accurate action policies over time.

The loss function for each critic network is defined in Equation (4):

$$L_k = \frac{1}{2M} \sum_{i=1}^{M} \left(y_i - Q_k(s_i, a_i; \phi_k)\right)^2, \quad (4)$$

where M is the mini-batch size, and $Q_k(s_i, a_i; \phi_k)$ is the Q-value estimated by the critic-Q network. The objective of this loss function is to update the Critic network parameters to make more accurate predictions by minimizing the squared difference between the target value y_i and the estimated Q-value $Q_k(s_i, a_i; \phi_k)$ [12].

The policy loss function used to update the actor network is defined in Equation (5):

$$\mathcal{J}_\pi = \frac{1}{M} \sum_{i=1}^{M} \left(-\min_k \left(Q_k(s_i, a_i; \phi_k)\right) + \alpha \ln \pi\,(a'_i|s'_i; \theta)\right). \quad (5)$$

In this equation, the policy is optimized with entropy regularization. This approach simultaneously promotes exploration and encourages the selection of actions that yield high rewards [12]. The entropy weight update loss is defined in Equation (6):

$$L_\alpha = \frac{1}{M} \sum_{i=1}^{M} (-\alpha \ln \pi\,(a_i|s_i; \theta) - \alpha\,\mathcal{H}), \quad (6)$$

where $\mathcal{H}$ is the target entropy value that balances exploration and exploitation. The objective of this equation is to dynamically update the entropy weight α to maintain an effective balance between exploration and reward optimization [12, 42].

In the SAC algorithm, the critic-Q networks are optimized by minimizing the TD error, while the actor is improved by a policy-loss with entropy regularization [17]; in parallel, the target Q-networks are updated using a slow-moving average (Polyak averaging) of the critic parameters (Equation (7)) [42]:

$$\phi_{tk} = \tau \phi_k + (1 - \tau)\phi_{tk}. \quad (7)$$

Here, τ is the Target Smooth Factor. This technique helps improve stability and ensures more accurate optimization of the loss functions used to minimize the temporal difference (TD) error [15].

3.2. ES Structure.

ES is a population-based optimization technique that belongs to the family of evolutionary algorithms [47]. It has strong performance in continuous decision spaces. The ES algorithm evaluates a population of candidate solutions (individuals) across generations, selecting the most successful ones and introducing mutations to create the next generation of individuals [48]. Initially, random policy parameters are assigned. New individuals are generated by introducing Gaussian noise to these parameters. The resulting individuals are evaluated using a reward function, and their fitness values are calculated. Elite selection is then applied to choose the best individuals, which are used to update the policy parameters. This process is iterated to form new generations, continually refining the solution [48]. The population (θ'_i), one of the core components of the ES structure, consists of randomly initialized individuals within the solution space. Mutation (σ) generates new individuals by adding Gaussian noise to the current policy parameters (θ) (Equation (8)):

$$\theta'_i = \theta + \sigma \cdot \epsilon_i. \quad (8)$$

Here, σ represents the mutation scale (SD), $\epsilon_i \sim \mathcal{N}(0, I)$ is random noise sampled from a standard normal distribution, and θ_i' denotes the parameters of the newly generated individuals. This approach ensures that new individuals are produced in each generation and that policy parameters continually improve, allowing the optimization process to progress [50].

Selection involves identifying the best-performing individuals in the population. As suggested by [49] and [51], the reward evaluation process requires calculating the average reward of

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

each individual by running multiple rollouts (different seeds). Each individual is evaluated according to the reward function, and those with the highest average rewards are selected as elite individuals for the next generation. The average reward (fitness value) of an individual is calculated using Equation (9):

$$F_i = \frac{1}{K}\sum_{k=1}^{K} R(\theta_i', k)\,, \tag{9}$$

where F_i is the average reward (fitness value) of the individual, K is the number of evaluation rollouts, $R(\theta_i', k)$, is the reward obtained in the kth run [51].

During the update step, policy parameters are adjusted for the new generation. The elite individuals with the highest rewards are selected, and new policy weights are calculated. Specifically, the policy parameters for the next generation are updated by taking a weighted average of the elite individuals (Equation (10)):

$$\theta \leftarrow \sum_{j=1}^{N_e} w_j \cdot \theta_j, \tag{10}$$

where N_e is the number of elite individuals, and w_j represents the weights assigned to each selected individual. This update is performed using Monte Carlo derivative estimation, a gradient-free optimization method [52].

The ES algorithm balances exploration and exploitation by dynamically adjusting the SD. To prevent premature convergence to local minima, the step size (σ) is gradually reduced over time according to the following update rule (Equation (11)):

$$\sigma \leftarrow (1-\lambda)\cdot\sigma + \lambda\cdot\sigma_{min}, \tag{11}$$

where λ is the SD decay rate, and σ_{min} is the minimum allowable SD. This approach is related to step-size adaptation methods proposed by [53] and [54].

4. ES-RL MODELING

ES-SAC algorithm, illustrated in Fig. 3, integrates the strong exploration abilities of ES with the sampling efficiency of SAC, facilitating more effective training of SAC agents in continuous action spaces. This approach optimizes policy parameters using both evolutionary population-based search and gradient-based updates, leveraging the global search strengths of ES alongside the fine-tuning abilities of SAC. As a result, ES-SAC facilitates the learning of more stable and robust policies, particularly in environments characterized by high-dimensional state, action spaces and complex dynamics [19].

Before starting the training process, the algorithm first defines the agent and the environment. During this phase, the observation and action spaces are determined, and the actor and critic networks are initialized for the SAC agent. The actor network parameters of the SAC agent are further optimized by integrating the ES component. As a population-based optimization method, ES generates multiple agent individuals (workers) and evaluates their performance in the environment [55]. The parameters optimized by ES include the weights and biases of the actor network, the mean and SD values of the policy outputs, as well as the entropy coefficient [19].

The population is the cornerstone of the evolutionary process, with each individual representing a distinct set of policy parameters. Initially, the population consists of randomly sampled individuals centered around a given mean and SD. The performance of each individual is evaluated in the simulation environment, where total reward values are calculated. The total reward achieved by an agent in a given generation serves as the fitness value for that individual. Subsequently, the individuals with the highest rewards are selected to create the next generation of the population [56]. This iterative process continues for a predefined number of generations.

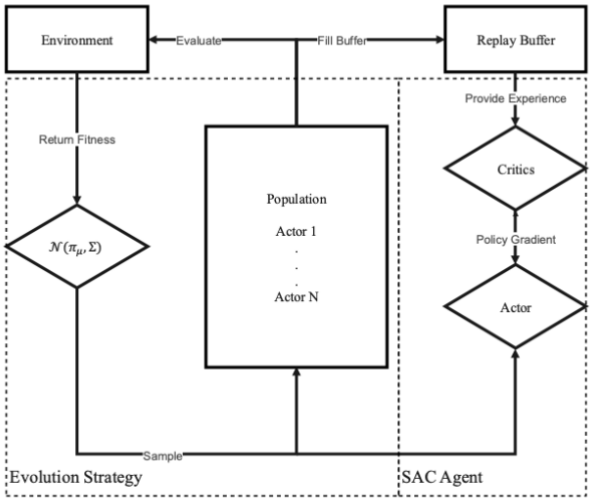

Fig.3. ES-SAC architecture (adapted from [18]).

ES updates are performed by optimizing the population's mean and variance. Successful individuals contribute to the population mean based on assigned weights. When generating new individuals, random noise sampled from a normal distribution is added to this mean to create the next generation. The magnitude of this noise plays a crucial role in balancing exploration and exploitation. Increasing the noise level allows the population to explore the solution space more broadly, but it can also lead to unstable actions from the learned policies. Therefore, the SD values used in the ES process are gradually reduced over time, guiding the population toward more stable solutions [57].

The SAC agent continues its learning process using the best policy parameters obtained from the population optimized by ES. Experiences collected by the population are stored in SAC's replay buffer, enabling the agent to learn from past interactions. The SAC agent receives feedback through its critic networks and performs policy updates by maximizing the entropy-regularized objective; stability is improved via target critic networks [18, 19].

While optimizing policies through the evolutionary process, the SAC agent drives exploration by updating the population

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

with improved individuals based on experiences gathered by SAC within ES. This hybrid approach is especially effective for high-dimensional and complex RL problems. ES performs broad exploration without relying on gradient calculations, whereas the SAC agent enhances sample efficiency and policy optimization by utilizing an experience replay buffer. Consequently, the learning process achieves a better balance between exploration and exploitation, managing both aspects effectively [14, 18, 19].

In the ES-SAC hybrid system, the ES algorithm treats the RL problem as a black-box optimization task and optimizes policy parameters accordingly. The policy population, represented by the actor network, is sampled by adding noise to the parameters using the ES method [58]. Each offspring policy (individual) is evaluated in the task environment to obtain a fitness value, calculated based on cumulative rewards. The ES algorithm then selects the top-performing elite policies according to their fitness values, which are used to generate the next population [59].

In the hybrid work model, SAC updates its parameters either on a separate timescale or in parallel through gradient descent, whereas ES does not rely on gradient-based updates. Instead, SAC and ES policies interact internally [59]. The latest policy trained by SAC is incorporated into the ES population and evaluated within the ES environment. In this way, ES enhances exploration by providing a diverse set of policy candidates [60]. When SAC becomes stuck in local optima or struggles to find high-dimensional rewards, the ES policies can help overcome these challenges [61].

Rather than mixing actions at run time, the outputs of the SAC network are influenced offline through periodic ES perturbation/elitist replacement of the SAC policy parameters. This interaction accelerates convergence and reduces the chance of SAC becoming trapped in suboptimal regions of the search space. Importantly, the core algorithmic structures of SAC and ES remain unchanged throughout this process [62].

The SAC method alone demonstrates satisfactory performance due to its off-policy learning and entropy-driven exploration capabilities. Meanwhile, the ES method offers strong scalability for high-dimensional problems thanks to its parallelization potential [62, 63]. By combining SAC and ES into a hybrid ES-SAC agent, the approach leverages the strengths of both methods to select optimal actions. Numerous studies have shown that this hybrid system outperforms DRL methods used individually [19, 56, 62].

5. SIMULATION

A simulation system was developed using the RL Toolbox, Deep Learning Toolbox, and Simscape Multibody within the MATLAB/Simulink environment. Each leg of the robot features torque-controlled rotary joints (hip, knee, and ankle) that balance and guide the gait, mimicking the way humans consciously use muscles when stepping. Meanwhile, the arms were configured with passive pivoting shoulder joints in the sagittal and frontal planes to provide a natural sense of swing and balance. Throughout the simulation, the model continuously monitored contact forces, the body's position and orientation in space, joint angles, and forward motion, enabling it to interpret its interactions with the environment. The training was conducted on a flat surface, and the agents' rewards were compared along this path. A URDF-based humanoid model in MATLAB was used; training and simulation were performed only on a flat track, as shown in Fig. 4.

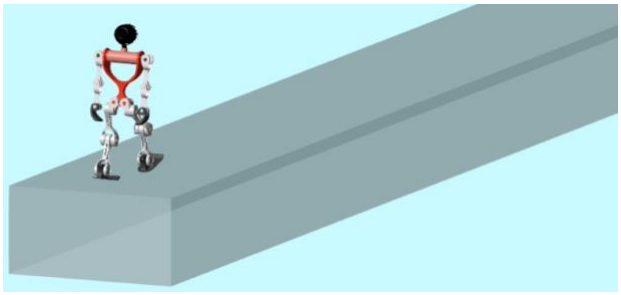

Fig.4. Humanoid robot on a 50-m flat track.

During the training and simulation processes, the forward direction (y-axis), lateral direction (x-axis), and vertical direction (z-axis) were defined with reference to the robot's body coordinate system. The agent's reward function included direct rewards for forward speed, penalties for lateral and vertical deviations (squared), and terms accounting for power consumption and standing time. This approach aimed to foster learning behaviors that maintain balance, promote energy efficiency, and encourage forward movement. This reward function structure is commonly used in the literature for humanoid robots [64–68]. The reward function employed in this study was adapted from [67], with modifications in coefficient values and specific terms to suit our simulation environment. In determining these coefficients, both the relative importance of each reward component and the numerical range of their values were considered, and the weights were normalized accordingly. Coefficient values were fixed a priori and were not tuned for individual agents, thereby focusing the comparison on the learning rules under an identical objective; accordingly, the same reward structure was applied to all agents (Equation (12)):

$$r_t = \dot{y} - 5\,x^2 - 25\,\dot{z}^2 - 0.0005\,P + 0.025. \qquad (12)$$

Here, the first term $\dot{y}$ encourages the robot to move forward. The second term x penalizes lateral deflection of the body. It denotes the lateral cross-track error of the torso center, measured in a body-fixed frame whose forward axis is aligned with the walking direction. The third term $\dot{z}$ discourages unwanted vertical movements such as jumping or jerking. These components are defined with respect to the body-fixed coordinate frame whose origin is located at the center of the torso. The fourth term represents power consumption (P), which is calculated as the sum of the products of joint torques and their corresponding angular velocities, averaged over all actuated joints. The last term is a fixed reward that incentivizes the robot's survival over time.

The humanoid model was simulated as a spatial (3D) multi-body system with a deliberately simplified joint structure. Leg

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

kinematics, based on sagittal plane motion –the most prominent component of walking– was modeled as a series chain with 3 degrees of freedom (DOF) per leg; this chain consists of three rotary joints and is driven directly by torque inputs. Joint ranges of motion were limited to remain consistent with human anatomy: the hip joint is limited between −90° and +30°, the knee joint between 5° and 90°, and the ankle joint between −20° and +20°. These constraints are intended to keep the model within realistic joint motion ranges and provide a numerically stable environment, while also enabling controllers to learn more quickly [28].

In this study, the batch size is set to 256, the number of warm-start steps to 1000, and the maximum number of training generations to 1500. The total numbers of learnable parameters are 153.7k and 169.6k for the actor and critic networks, respectively. These four hyperparameters differ from those used in the ES-SAC study reported in [28], while all other specified hyperparameters are kept identical to that work. Any hyperparameters not explicitly defined in either study are retained at their default values. Sampling time enables the robot to respond more precisely to environmental changes. Batch size promotes more stable and faster learning, helping to ensure the policy improves regularly and avoids sudden fluctuations. Experience buffer length enhances the policy's generalization capacity. Target smooth factor keeps the Q targets up to date, which improves agent adaptability, especially in dynamic environments. Warm-start steps avoid sudden drops in the initial learning curve and prevent unstable policies, ensuring the replay buffer is filled with more reliable data. Epochs per update enable multiple updates on the same data and ensure that the policy gains the maximum possible information from each mini-batch. By using low learning rates, slow but stable learning is achieved. A gradient threshold mitigates the issue of exploding gradients, while an L2 regularization factor prevents unnecessary weight growth and reduces overfitting. Within a single ES generation, 25 candidate policies are evaluated with zero-mean Gaussian noise –i.e., a mutation step size of 0.25 for weights and biases– and elites are selected; in that same generation, the agent then runs for 50 inner training cycles to consolidate these improvements via gradient-based updates, yielding a refined policy for the next generation. This procedure is repeated for 1500 generations.

The robot simulation and training in the MATLAB environment could not be executed asynchronously due to the algorithmic structure of the evolutionary process. Thus, only synchronous training was applied to the robot. All training and testing procedures were conducted on a workstation equipped with an AMD Ryzen 9 9950X3D processor, 64 GB DDR5 RAM, and an NVIDIA RTX 5090 GPU.

Table 1 summarizes performance over 1,500 episodes/generations using multiple random seeds (ES-RL: 3; vanilla RL: 5). nAUC is the area under the learning curve normalized by training length (nAUC = AUC/1500), i.e., an average reward per episode, enabling fair comparisons across horizons. Best-Win-50 is the mean reward of the best consecutive 50-episode window (sustained peak); Episodes-to-Target is the number of episodes required to reach 80% of each method's own Best-Win-50 median (time-to-competence). At this control rate (T_s = 0.025 s) and with an episode cap of T_f = 60 s, each episode comprises T_f/T_s = 2400 control steps. Across 1500 generations, this yields 1500 × 25 × 2400 = 90 M environment interactions per run (excluding the inner gradient updates— 38.4 M sample-passes). ES-SAC attains the highest average return (nAUC/AUC), with ES-TD3 following and ES-DDPG lagging well behind; under the same horizon, vanilla RL baselines yield nAUC/AUC values that are too small for a meaningful comparison. Looking at steady-state and peak performance, ES-SAC delivers the largest Final-50 and Best-Win-50 rewards, whereas ES-TD3 produces slightly lower returns but with reduced variance. In terms of reaching the 80% target derived from each method's Best-Win-50 median, ES-SAC is the fastest and most consistent across seeds, while ES-TD3 arrives later on average and exhibits wider seed-to-seed spread. Finally, on the 50-m course the robot controlled by ES-TD3 completes the track in the shortest time; by comparison, ES-SAC takes longer to finish, and robots driven by the remaining agents fail to complete the course. The robot exhibits a higher average forward velocity with ES-TD3 (1.3754 m/s) than with ES-SAC (1.0477 m/s). This higher velocity helps on the track but coincides with torque imbalance and pitch asymmetry relative to ES-SAC. Consequently, average forward velocity should be considered not only as a performance indicator but also as indirect measures of control stability and reliability.

Color scheme used throughout Fig. 5 (a–d): ES-SAC (blue), ES-TD3 (orange), ES-DDPG (yellow), SAC (purple), TD3 (green), DDPG (cyan); shaded bands (where present) denote mean ± SD. All reported quantities (e.g., lateral and vertical deviations, average torques, etc.) are presented as mean ± standard deviation across random seeds.

Table 1. Comparative performance of ES-based and vanilla RL agents (aggregated over multiple seeds) over 1500 episodes/generations

Agent	Average Reward (nAUC) (μ ± SD)	AUC (×10⁶) (μ ± SD)	Final-50 Reward (μ ± SD)	Best-Win-50 Reward (μ ± SD)	Episodes-to-Target (ep)	50 m Track Time (s)
ES-SAC	4536.50 ± 166.15	6.80 ± 0.25	6015.01 ± 216.93	6479.51 ± 318.91	454.50 ± 40.31	47.72
ES-TD3	3554.50 ± 599.98	5.33 ± 0.90	5825.82 ± 251.86	5961.64 ± 78.03	635.00 ± 455.38	36.35
ES-DDPG	830.73 ± 255.03	1.25 ± 0.38	1179.72 ± 699.98	1620.66 ± 729.57	608.00 ± 207.89	-
SAC	1.23 ± 4.85	0.002 ± 0.0073	40.02 ± 16.36	43.78 ± 11.82	515 ± 73	-
TD3	17.81 ± 4.18	0.027 ± 0.0063	70.90 ± 16.61	74.79 ± 18.98	987 ± 167	-
DDPG	−1.37 ± 8.46	−0.002 ± 0.013	5.93 ± 27.40	29.87 ± 11.76	261 ± 310	-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

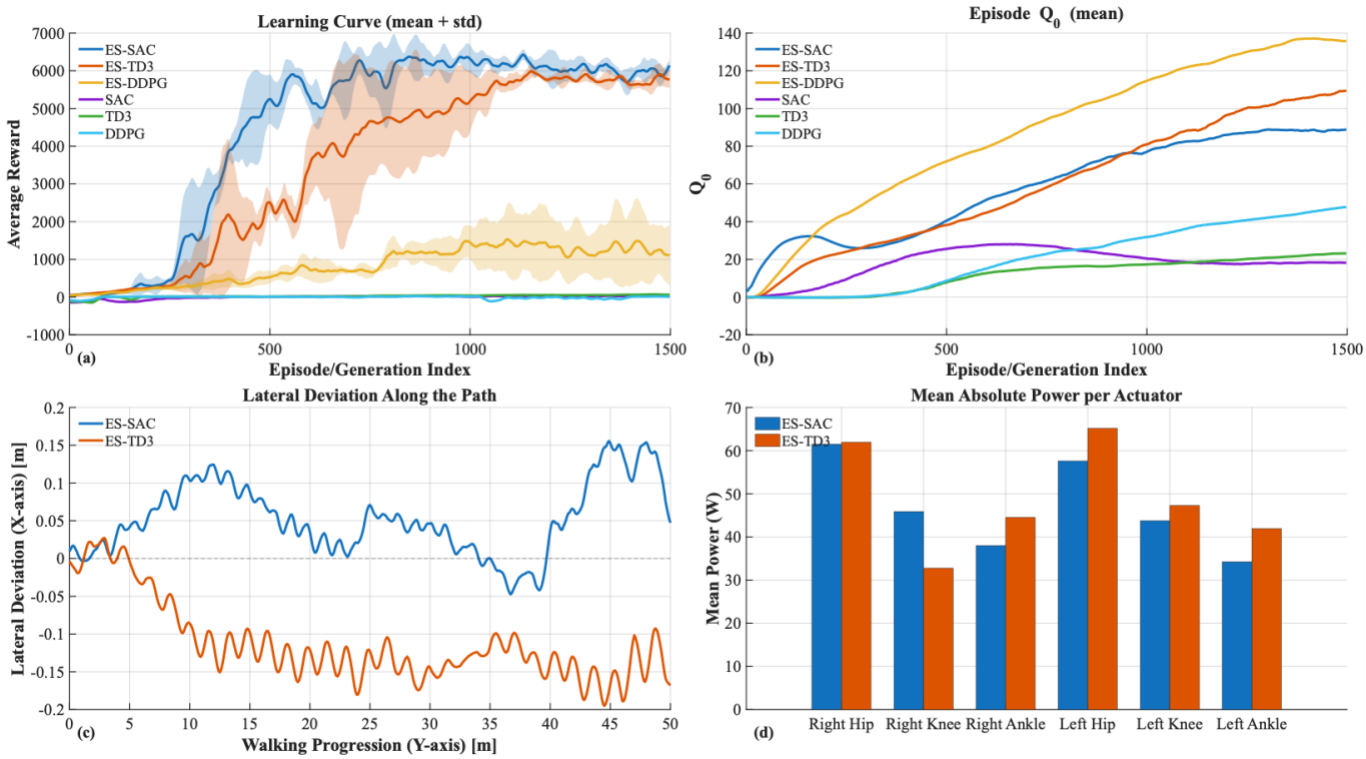

Fig.5. Comparative performance of ES-SAC, ES-TD3, ES-DDPG vs. SAC, TD3, DDPG: (a) learning curves (mean ± SD), (b) episode-wise Q0 (mean), (c) lateral deviation along the path, (d) mean absolute power per actuator. Shaded areas denote ±1 SD.

Fig. 5(a) shows the learning curves for agents trained; shaded bands denote mean ± SD. The training was conducted on a flat surface, with the robot's arms oscillating freely during the walking process. The figure clearly demonstrates that the ES-SAC agent consistently obtained a higher average reward than the others. Notably, in the 567th episode of training, the ES-SAC agent was the first to surpass the 6000 average reward level, thus achieving the high-performance threshold. Up to this point, the ES-SAC agent exhibited a higher learning acceleration than the others, highlighting the algorithm's capacity to quickly and effectively adapt to the environmental dynamics. While both the ES-SAC and ES-TD3 agents eventually reached learning saturation in later episodes, other agents failed to achieve this threshold, resulting in a much lower average reward. Also, ES-TD3 has the widest shaded band (±SD), reflecting greater across-seed variation and less stable learning. ES-SAC shows a narrower band, indicating steadier and more reliable convergence. ES-DDPG has a small SD but also a low mean reward –i.e., consistently low performance.

Fig. 5(b) shows the Q_0 values for all agents. The Q_0 value reflects the agents' estimated expected future reward at the start of an episode and indicates their confidence in the learning process. Although the ES-DDPG agent exhibited quite optimistic Q_0 forecasts, it tended to overestimate and achieved low actual rewards. In contrast, the ES-SAC and ES-TD3 agents demonstrated more cautious yet steady increases in Q_0, resulting in better actual rewards. This suggests that their learning processes are more balanced and reliable. The mean Q_0 values of the ES-SAC and ES-TD3 agents were close, measuring 58.507 and 59.217, respectively. Among the vanilla agents, DDPG shows a moderate upward trend in Q_0 despite poor returns, while SAC and TD3 remain relatively flat at low levels, consistent with limited learning.

During the training process, the agents' performances were primarily evaluated using the average reward and Q_0 values. However, to gain deeper insights, the actual gait behavior of the robot in the simulation environment was also examined. Specifically, the average torque values at the joints and the extent of horizontal and vertical body deflections were measured and analyzed. Since torque efficiency directly impacts the robot's energy consumption, it provides valuable insight into the energy efficiency of the control strategy. Similarly, horizontal deviation serves as a critical metric for assessing the robot's balance maintenance and its steadiness in progressing toward the goal. Together, these two parameters offer important clues not only about the reward values achieved but also about the quality of the agent's physical behavior. The analysis of vertical deviation reveals that both agents tend to maintain balance by keeping the robot's center of gravity close to the ground. The mean vertical deviation was calculated as –0.0285 m for ES-SAC and –0.0158 m for ES-TD3. The corresponding absolute deviation values were 0.0674 and 0.1177 m, respectively, indicating that the ES-TD3 agent exhibited slightly greater vertical variance. These data suggest that both agents produce stable gait patterns in the vertical direction; however, the ES-SAC agent maintained a slightly lower center of gravity, reflecting a more cautious control of balance.

The purpose of these graphs is to visually demonstrate the stability, efficiency, and control exhibited by each agent during the walking process. This allows for a comparative

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

evaluation not only based on numerical reward values but also in terms of physical balance and energy optimization.

Fig. 5(c) presents the horizontal deviation of the robot. A higher deviation corresponds to a greater penalty for the robot. The ideal path is indicated by a gray dashed line. Comparing the horizontal deviations of the robot's body, the ES-SAC agent clearly performs better, maintaining closer adherence to the ideal path. The mean absolute deviation values were 0.0576 m for the ES-SAC agent and 0.1146 m for the ES-TD3 agent.

Fig. 5(d) shows the torque values generated by the robot's lower limb joints during walking. Since higher torque values contribute penalty points in the reward function, the robot is incentivized to maintain power efficiency. Comparing the average absolute torque values across all joints, the ES-SAC-controlled robot recorded 46.824 Nm, while the ES-TD3-controlled robot had a higher average of 48.936 Nm. This indicates that the ES-TD3 agent applies torque more aggressively than the ES-SAC agent. Consequently, the ES-SAC agent demonstrates a more energy-efficient and smoother control strategy, achieving comparable gait performance with lower torque output. Examining the right and left knee joint torques revealed that the ES-SAC agent produced similar torque values for both knees, promoting symmetrical control. In contrast, the ES-TD3 agent showed a significantly larger torque discrepancy between the two knee joints. Such asymmetry in torque generation at these critical joints can disrupt gait stability and symmetry in humanoid robots.

6. CONCLUSION

In this study, the proposed ES-SAC hybrid method for humanoid robot gait control demonstrated remarkable results compared to deterministic agents (ES-TD3 and ES-DDPG). The ES-SAC agent achieved higher average rewards and exhibited a more balanced learning process. Although the ES-TD3 agent completed the course in a shorter time, this was accompanied by certain control instabilities. Unlike many studies that rely primarily on superficial reward value comparisons (e.g., [15, 18, 19]), this study adopted a multifaceted analysis by incorporating physical and behavioral metrics such as torque efficiency, horizontal and vertical deflection, and Q_0 values. This comprehensive approach is also consistent with recent control-oriented studies emphasizing that learning-based and hybrid control strategies should be evaluated not only by task success, but also by stability, adaptability, energy efficiency, and behavior-level reliability [69–72]. Additionally, the results indicated that integrating the ES component generally improved the performance of all agents by enabling broader exploration of the solution space and helping avoid local minima, thus generating more robust policies. Overall, the study highlights the importance of evaluating learning algorithms not only by their numerical performance but also through the quality of the resulting behaviors and control reliability.

References

[1] *R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction. Cambridge, MA, USA: MIT Press, 1998.*

[2] *A. Nagabandi et al., "Learning to adapt in dynamic, real-world environments through meta-reinforcement learning," Mar. 2018. [Online]. Available: http://arxiv.org/abs/1803.11347*

[3] *S. Padakandla, "A survey of reinforcement learning algorithms for dynamically varying environments," ACM Comput. Surv., vol. 54, no. 6, art. no. 127, pp. 1–25, Jul. 2022, doi: 10.1145/3459991.*

[4] *L. Lin and T. M. Mitchell, "Reinforcement learning with hidden states," 1993. [Online]. Available: https://api.semanticscholar.org/CorpusID:60745539*

[5] *R. A. McCallum, "Hidden state and reinforcement learning with instance-based state identification," IEEE Trans. Syst., Man, Cybern., Part B, vol. 26, no. 3, pp. 464–473, 1996, doi: 10.1109/3477.499796.*

[6] *K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," in Handbook of Reinforcement Learning and Control, K. G. Vamvoudakis, Y. Wan, F. L. Lewis, and D. Cansever, Eds. Cham: Springer Int. Publ., 2021, pp. 321–384, doi: 10.1007/978-3-030-60990-0_12.*

[7] *C.-L. Liu, C.-C. Chang, and C.-J. Tseng, "Actor-critic deep reinforcement learning for solving job shop scheduling problems," IEEE Access, vol. 8, pp. 71752–71762, 2020, doi: 10.1109/ACCESS.2020.2987820.*

[8] *L. Xi, J. Wu, Y. Xu, and H. Sun, "Automatic generation control based on multiple neural networks with actor-critic strategy," IEEE Trans. Neural Netw. Learn. Syst., vol. 32, no. 6, pp. 2483–2493, 2021, doi: 10.1109/TNNLS.2020.3006080.*

[9] *J. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," IEEE Trans. Syst., Man, Cybern., Part C, vol. 42, no. 6, pp. 1291–1307, 2012, doi: 10.1109/TSMCC.2012.2218595.*

[10] *S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," 2016. [Online]. Available: https://arxiv.org/abs/1610.00633*

[11] *T. Haarnoja, V. Pong, A. Zhou, M. Dalal, P. Abbeel, and S. Levine, "Composable deep reinforcement learning for robotic manipulation," 2018. [Online]. Available: https://arxiv.org/abs/1803.06773*

[12] *T. Haarnoja et al., "Soft actor-critic algorithms and applications," 2019. [Online]. Available: https://arxiv.org/abs/1812.05905*

[13] *X. Olaz, D. Alaez, M. Prieto, J. Villadangos, and J. J. Astrain, "Quadcopter neural controller for take-off and landing in windy environments," Expert Syst. Appl., vol. 225, p. 120146, 2023, doi: 10.1016/j.eswa.2023.120146.*

[14] *T. P. Lillicrap et al., "Continuous control with deep reinforcement learning," 2019. [Online]. Available: https://arxiv.org/abs/1509.02971*

[15] *S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," 2018. [Online]. Available: https://arxiv.org/abs/1802.09477*

[16] *X. Shen, "Comparison of DDPG and TD3 algorithms in a Walker2D scenario," in Proc. 2024 Int. Conf., 2024, pp. 148–155, doi: 10.2991/978-94-6463-370-2_17.*

[17] *Y. Zhang, J. Xie, X. Du, H. Sun, S. Wang, and K. Hashimoto, "Biped robot terrain adaptability based on improved SAC algorithm," in Proc. MSR-RoManSy 2024, J. M. Larochelle, P. Larochelle, and P. McCarthy, Eds. Cham: Springer Nature Switzerland, 2024, pp. 93–104, doi: 10.1007/978-3-031-60618-2_8.*

[18] *T. Pourchot and O. Sigaud, "CEM-RL: Combining evolutionary and gradient-based methods for policy search," 2019. [Online]. Available: https://arxiv.org/abs/1810.01222*

[19] *S. Khadka and K. Tumer, "Evolution-guided policy gradient in reinforcement learning," 2018. [Online]. Available: https://arxiv.org/abs/1805.07917*

[20] *D. B. Fogel, Evolutionary Computation: Toward a New Philosophy of Machine Intelligence. Piscataway, NJ, USA: IEEE Press, 1995.*

[21] *W. M. Spears, K. A. De Jong, T. Bäck, D. B. Fogel, and H. De Garis, "An overview of evolutionary computation," in Proc. European Conf. on Machine Learning, 1993, pp. 442–459.*

[22] *M. Callaghan, K. Mason, and P. Mannion, "Evolutionary strategy guided reinforcement learning via MultiBuffer communication," 2023. [Online]. Available: https://arxiv.org/abs/2306.11535*

[23] *Z. ul Abdeen, X. Zhang, W. Gill, and M. Jin, "Enhancing distribution system resilience: A first-order meta-RL algorithm for critical load*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

restoration," in Proc. 2024 SmartGridComm, 2024, pp. 129–134, doi: 10.1109/SmartGridComm60555.2024.10738096.

[24] *H. Bai, R. Cheng, and Y. Jin, "Evolutionary reinforcement learning: A survey," Intell. Comput., vol. 2, Jan. 2023, doi: 10.34133/icomputing.0025.*

[25] *D. Lee, B.-U. Lee, U. Shin, and I. S. Kweon, "An efficient asynchronous method for integrating evolutionary and gradient-based policy search," 2021. [Online]. Available: https://arxiv.org/abs/2012.05417*

[26] *E. S. R., S. Kolathaya, and G. Thoppe, "Improving sample efficiency in evolutionary RL using off-policy ranking," 2023. [Online]. Available: https://arxiv.org/abs/2208.10583*

[27] *N. Müller and T. Glasmachers, "Challenges in high-dimensional reinforcement learning with evolution strategies," 2018. [Online]. Available: https://arxiv.org/abs/1806.01224*

[28] *M. Ayyıldız and Ö. Polat, "Human-like arm swing strategies in ES-SAC humanoid gait: Stability and performance on flat vs rough terrain," Electron. Res. Arch., vol. 34, no. 3, pp. 1524–1545, 2026, doi: 10.3934/era.2026069.*

[29] *R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," in Advances in Neural Information Processing Systems, vol. 12, 1999.*

[30] *A. Hashemi, G. Orzechowski, A. Mikkola, and J. McPhee, "Multibody dynamics and control using machine learning," Multibody Syst. Dyn., vol. 58, no. 3, pp. 397–431, 2023, doi: 10.1007/s11044-023-09884-x.*

[31] *Y. Sun, M. Song, D. Cai, B. Zhang, S. Hong, and H. Li, "A systematic review of echo state networks from design to application," IEEE Trans. Artif. Intell., vol. 5, no. 1, pp. 23–37, 2024, doi: 10.1109/TAI.2022.3225780.*

[32] *I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. Cambridge, MA, USA: MIT Press, 2016.*

[33] *B. Bakker, "Reinforcement learning with long short-term memory," in Advances in Neural Information Processing Systems, 2001.*

[34] *R. Toro Icarte, E. Waldie, T. Klassen, R. Valenzano, M. Castro, and S. McIlraith, "Learning reward machines for partially observable reinforcement learning," in Advances in Neural Information Processing Systems, vol. 32, 2019.*

[35] *B. Bakker, "Reinforcement learning by backpropagation through an LSTM model/critic," in Proc. IEEE Int. Joint Conf. Neural Netw. (IJCNN), 2002, pp. 363–368.*

[36] *Y. Zhu, Z. Wang, Y. Zhu, C. Chen, and D. Zhao, "Discretizing continuous action space with unimodal probability distributions for on-policy reinforcement learning," 2024. [Online]. Available: https://arxiv.org/abs/2408.00309*

[37] *S. Liu, "An evaluation of DDPG, TD3, SAC, and PPO: Deep reinforcement learning algorithms for controlling continuous system," in Proc. 2023 Int. Conf. on Data Sci., Adv. Algorithm and Intell. Comput. (DAI 2023). Atlantis Press, 2024, pp. 15–24, doi: 10.2991/978-94-6463-370-2_3.*

[38] *M. Lehmann, "The definitive guide to policy gradients in deep reinforcement learning: Theory, algorithms and implementations," Jan. 2024. [Online]. Available: http://arxiv.org/abs/2401.13662*

[39] *Y. Zhang and P. Chen, "Path planning of a mobile robot for a dynamic indoor environment based on an SAC-LSTM algorithm," Sensors (Basel), vol. 23, no. 24, p. 9802, Dec. 2023, doi: 10.3390/s23249802.*

[40] *M. Reyad, A. M. Sarhan, and M. Arafa, "A modified Adam algorithm for deep neural network optimization," Neural Comput. Appl., vol. 35, no. 23, pp. 17095–17112, 2023, doi: 10.1007/s00521-023-08568-z.*

[41] *W. E. L. Ilboudo, T. Kobayashi, and K. Sugimoto, "Robust stochastic gradient descent with Student-t distribution based first-order momentum," IEEE Trans. Neural Netw. Learn. Syst., vol. 33, no. 3, pp. 1324–1337, Mar. 2022, doi: 10.1109/TNNLS.2020.3041755.*

[42] *T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," Jan. 2018. [Online]. Available: http://arxiv.org/abs/1801.01290*

[43] *T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," Dec. 2018. [Online]. Available: http://arxiv.org/abs/1812.11103*

[44] *T. Kasaura, S. Miura, T. Kozuno, R. Yonetani, K. Hoshino, and Y. Hosoe, "Benchmarking actor-critic deep reinforcement learning algorithms for robotics control with action constraints," IEEE Robot. Autom. Lett., Apr. 2023, doi: 10.1109/LRA.2023.3284378.*

[45] *A. Srinivas, M. Laskin, and P. Abbeel, "CURL: Contrastive unsupervised representations for reinforcement learning," Apr. 2020. [Online]. Available: http://arxiv.org/abs/2004.04136*

[46] *V. Mnih et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529–533, 2015.*

[47] *Z. Wang, Y. Pei, and J. Li, "A survey on search strategy of evolutionary multi-objective optimization algorithms," Appl. Sci., vol. 13, no. 7, 2023, doi: 10.3390/app13074643.*

[48] *O. M. Michael and W. H. Emmerich Shir, "Evolution strategies," in Handbook of Heuristics, P. Martí, R. Martí, and C. Pardalos, Eds. Cham: Springer International Publishing, 2018, pp. 89–119, doi: 10.1007/978-3-319-07124-4_13.*

[49] *T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever, "Evolution strategies as a scalable alternative to reinforcement learning," 2017. [Online]. Available: https://arxiv.org/abs/1703.03864*

[50] *A. Obuchowicz, Stable Mutations for Evolutionary Algorithms. Cham: Springer, 2019.*

[51] *D. Wierstra, T. Schaul, T. Glasmachers, Y. Sun, J. Peters, and J. Schmidhuber, "Natural evolution strategies," J. Mach. Learn. Res., vol. 15, pp. 949–980, 2014.*

[52] *N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," Evol. Comput., vol. 9, no. 2, pp. 159–195, Jun. 2001, doi: 10.1162/106365601750190398.*

[53] *H.-G. Beyer and H.-P. Schwefel, "Evolution strategies – a comprehensive introduction," Nat. Comput., vol. 1, no. 1, pp. 3–52, 2002, doi: 10.1023/A:1015059928466.*

[54] *C. Igel, N. Hansen, and S. Roth, "Covariance matrix adaptation for multi-objective optimization," Evol. Comput., vol. 15, no. 1, pp. 1–28, 2007, doi: 10.1162/evco.2007.15.1.1.*

[55] *S. Chethana, S. S. Charan, V. Srihitha, and J. Amudha, "Humanoid robot gait control using PPO, SAC, and ES algorithms," in Proc. 2023 4th IEEE Global Conf. for Advancement in Technology (GCAT), 2023, pp. 1–7, doi: 10.1109/GCAT59970.2023.10353490.*

[56] *X. Liu, C. Zhang, S. Li, and J. Zhu, "Data-driven VVC for unbalanced networks with tuning-friendly deep reinforcement learning," in Proc. 2024 IEEE Power & Energy Society General Meeting (PESGM), 2024, pp. 1–5, doi: 10.1109/PESGM51994.2024.10688417.*

[57] *P. Li, J. Hao, H. Tang, X. Fu, Y. Zheng, and K. Tang, "Bridging evolutionary algorithms and reinforcement learning: A comprehensive survey on hybrid algorithms," Jan. 2024. [Online]. Available: http://arxiv.org/abs/2401.11963*

[58] *C. Hu, J. Liu, and X. Yao, "Evolutionary reinforcement learning via cooperative coevolution," Apr. 2024. [Online]. Available: http://arxiv.org/abs/2404.14763*

[59] *Y. Lin, F. Lin, G. Cai, H. Chen, L. Zou, and P. Wu, "Evolutionary reinforcement learning: a systematic review and future directions," Feb. 2024. [Online]. Available: http://arxiv.org/abs/2402.13296*

[60] *E. Marchesini, D. Corsi, and A. Farinelli, "Genetic soft updates for policy evolution in deep reinforcement learning," in Proc. Int. Conf. Learn. Representations (ICLR), 2020.*

[61] *C. Dong and D. Li, "Adaptive evolutionary reinforcement learning with policy direction," Neural Process. Lett., vol. 56, no. 2, p. 69, 2024, doi: 10.1007/s11063-024-11548-6.*

[62] *K. Suri, X. Q. Shi, K. N. Plataniotis, and Y. A. Lawryshyn, "Maximum mutation reinforcement learning for scalable control," Jul. 2020. [Online]. Available: http://arxiv.org/abs/2007.13690*

[63] *K. Suri, "Off-policy evolutionary reinforcement learning with maximum mutations," in Proc. 21st Int. Conf. Autonomous Agents and Multiagent Systems (AAMAS), 2022, pp. 1237–1245.*

[64] *Z. Wu, J. Lu, Y. Chen, Y. Liu, Y. Zhuang, and L. Hu, "STRIDE: Automating reward design, deep reinforcement learning training and feedback optimization in humanoid robotics locomotion," Feb. 2025. [Online]. Available: http://arxiv.org/abs/2502.04692*

[65] *H. Jung, Z. Gu, Y. Zhao, H.-W. Park, and S. Ha, "PreCi: Pretraining and continual improvement of humanoid locomotion via model-assumption-based regularization," Apr. 2025. [Online]. Available: http://arxiv.org/abs/2504.09833*

[66] *I. Radosavovic, S. Kamat, T. Darrell, and J. Malik, "Learning humanoid locomotion over challenging terrain," 2024. [Online]. Available: https://arxiv.org/abs/2410.03654*

[67] *N. Heess et al., "Emergence of locomotion behaviours in rich environments," Jul. 2017. [Online]. Available: http://arxiv.org/abs/1707.02286*

[68] *I. Radosavovic, T. Xiao, B. Zhang, T. Darrell, J. Malik, and K. Sreenath, "Real-world humanoid locomotion with reinforcement learning," 2023. [Online]. Available: https://arxiv.org/abs/2303.03381*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.

[69] *D. Kalandyk, B. Kwiatkowski, and D. Mazur, "CNC machine control using deep reinforcement learning," Bull. Pol. Acad. Sci. Tech. Sci., vol. 72, no. 3, art. no. e148940, 2024, doi: 10.24425/bpasts.2024.148940.*

[70] *C. N. Vanitha and P. Anusuya, "Towards sustainable wireless rechargeable sensor networks: a federated multi-agent reinforcement learning approach for cooperative wireless charging," Bull. Pol. Acad. Sci. Tech. Sci., vol. 74, no. 1, art. no. e155897, 2026, doi: 10.24425/bpasts.2025.155897.*

[71] *Y. Jiang, X. Wu, W. Zhang, W. Guo, W. Yu, W. Li, and J. Li, "Human-like decision making for autonomous lane change driving: a hybrid inverse reinforcement learning with game-theoretical vehicle interaction model," Bull. Pol. Acad. Sci. Tech. Sci., art. no. e152602, 2024, doi: 10.24425/bpasts.2024.152602.*

[72] *M. O. Celik, M. Koseoglu, and F. N. Deniz, "Experimental performance analysis of an AMR controlled by a hybrid MPC-PID method," Bull. Pol. Acad. Sci. Tech. Sci., vol. 74, no. 1, art. no. e156764, 2026, doi: 10.24425/bpasts.2025.156764.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.