MOSEAC: Streamlined Variable Time Step Reinforcement Learning

Dong Wang
dong-1.wang@polymtl.ca
Department of Computer Engineering
Polytechnique Montréal

Giovanni Beltrame
giovanni.beltrame@polymtl.ca
Department of Computer Engineering
Polytechnique Montréal

Abstract

Traditional reinforcement learning (RL) methods typically employ a fixed control loop, where each cycle corresponds to an action. This rigidity poses challenges in practical applications, as the optimal control frequency is task-dependent. A suboptimal choice can lead to high computational demands and reduced exploration efficiency. Variable Time Step Reinforcement Learning (VTS-RL) addresses these issues by using adaptive frequencies for the control loop, executing actions only when necessary. This approach, rooted in reactive programming principles, reduces computational load and extends the action space by including action durations. However, VTS-RL's implementation is often complicated by the need to tune multiple hyperparameters that govern exploration in the multi-objective action-duration space (i.e., balancing task performance and number of time steps to achieve a goal). To overcome these challenges, we introduce the Multi-Objective Soft Elastic Actor-Critic (MOSEAC) method. This method features an adaptive reward scheme that adjusts hyperparameters based on observed trends in task rewards during training. This scheme reduces the complexity of hyperparameter tuning, requiring a single hyperparameter to guide exploration, thereby simplifying the learning process and lowering deployment costs. We validate the MOSEAC method through simulations in a Newtonian kinematics environment, demonstrating high task and training performance with fewer time steps, ultimately lowering energy consumption. This validation shows that MOSEAC streamlines RL algorithm deployment by automatically tuning the agent control loop frequency using a single parameter. Its principles can be applied to enhance any RL algorithm, making it a versatile solution for various applications.

1 Introduction

Model-free deep reinforcement learning (RL) algorithms have achieved significant success in various domains, including gaming (Liu et al., 2021; Ibarz et al., 2021) and robotic control (Akalin & Loutfi, 2021; Singh et al., 2022). Traditional RL methods typically rely on a fixed control loop where the agent takes actions at predetermined intervals, e.g. every 0.1 seconds (10 Hz). However, this approach introduces significant limitations: fixed-rate control can lead to stability issues and high computational demands, especially in dynamic environments where the optimal action frequency may vary over time.

To address these limitations, Variable Time Step Reinforcement Learning (VTS-RL) was recently introduced, based on reactive programming principles (Majumdar et al., 2021; Bregu et al., 2016). The key idea behind VTS-RL is to *execute control actions only when necessary*, thereby reducing computational load, as well as expanding the action space including variable action durations. For instance, in an autonomous driving scenario, VTS-RL enables the agent to adapt its control strategy

based on the situation, using a low frequency in stable conditions and a high frequency in more complex situations (Wang & Beltrame, 2024a).

Two notable VTS-RL algorithms are Soft Elastic Actor-Critic (SEAC) (Wang & Beltrame, 2024a) and Continuous-Time Continuous-Options (CTCO) (Karimi et al., 2023), which simultaneously learn actions and their durations.

CTCO employs continuous-time decision-making and flexible option durations, improving exploration and robustness in continuous control tasks. However, its effectiveness can be hampered by the need to tune several hyperparameters, and the fact that tasks can take a long time to complete as CTCO does not optimize for task duration.

In contrast, SEAC adds reward terms for task energy (i.e. the numbers of actions performed) and task time (i.e. the time needed to complete a task), showing effectiveness in time-restricted environments like a racing video game (Wang & Beltrame, 2024b). However, SEAC requires careful tuning of its hyperparameters (balancing task, energy, and time costs) to avoid performance degradation.

SEAC and CTCO's brittleness to hyperparameter settings poses a challenge for users aiming to fully leverage their potential. To mitigate this issue, we propose the Multi-Objective Soft Elastic Actor-Critic (MOSEAC) algorithm based on SEAC (Wang & Beltrame, 2024a). MOSEAC adds action durations to the action space and adjusts hyperparameters based on observed trends in task rewards during training, leaving a single hyperparameter to be set to guide the exploration. We evaluate MOSEAC in a Newtonian kinematics simulation environment, comparing it CTCO, SEAC, and fixed-frequency SAC, showing that MOSEAC is stable, accelerates training, and has high final task performance. Additionally, our hyperparameter setting approach can be broadly applied to any continuous action reinforcement learning algorithm, such as Twin Delayed Deep Deterministic (TD3) (Fujimoto et al., 2018) or Proximal Policy Optimization (PPO) (Schulman et al., 2017). This adaptability facilitates the transition from fixed-time step to variable-time step reinforcement learning, significantly expanding the scope of its practical application.

2 Related Work

The importance of action duration in reinforcement learning algorithms has been severely underestimated for a long time. In fact, if reinforcement learning algorithms are to be applied to the real world, it is an essential factor, impacting an agent's exploration capabilities (Amin et al., 2020; Park et al., 2021). High frequencies may reduce exploration efficiency in some cases, but are needed in others such as environments with delays (Bouteiller et al., 2021). Wang & Beltrame (2024b); Karimi et al. (2023) show how various frequencies impact learning, with high frequencies potentially impeding convergence. Thus, a dynamically optimal control frequency, which adjusts in real time based on reactive principles, could enhance performance and adaptability.

Sharma et al. (2017) introduced a concept similar to variable control frequencies, developing a framework for repetitive action reinforcement learning. This model allows an agent to perform identical actions over successive states, combining them into a larger action. This method has intrigued researchers, especially in gaming (Metelli et al., 2020; Lee et al., 2020), but fails to account for physical properties or reduce computational demands, making its practical application challenging.

Additionally, Chen et al. (2021) attempted to modify the traditional "control rate" by integrating actions such as "sleep" to reduce activity periods ostensibly. However, this approach still mandates fixed-frequency checks of system status, which does not effectively diminish the computational load as anticipated. These instances underscore the ongoing challenges and the nascent stage of effectively integrating variable control frequencies and repetitive behaviors into real-world applications, highlighting a critical gap between theoretical innovation and practical efficacy.

3 Algorithm Framework

To reduce hyperparameter dimensions, we combine SEAC's (Wang & Beltrame, 2024a) hyperparameters balancing task, energy, and time rewards using a simple multiplication method. We then apply an adaptive adjustment method on the remaining hyperparameters. Like SEAC, our reward equation includes components for:

- Quality of task execution (the standard RL reward),
- Time required to complete a task (important for varying action durations), and
- Energy (the number of time steps, a.k.a. the number of actions taken, which we aim to minimize).

Definition 1

$$R = \alpha_m R_t R_\tau - \alpha_\varepsilon$$

where R_t is the task reward; R_τ is a time dependent term.

α_m is a weighting factor used to modulate the magnitude of the reward. Its primary function is to prevent the reward from being too small, which could lead to task failure, or too large, which could cause reward explosion, ensuring stable learning.

α_ε is a penalty parameter applied at each time step to impose a cost on the agent's actions. This parameter gives a fixed cost to the execution of an action, thereby discouraging unnecessary ones. In practice α_ε promotes the completion of a task using fewer time steps (remember that time steps have variable duration), i.e. it reduces the energy used by the control loop of the agent.

We determine the optimal policy π^*, which maximizes the reward R.

The reward as designed, minimizes both energy cost (number of steps) and the total time to complete the task through R_τ. By scaling the task-specific reward based on action duration with α_m, agents are motivated to complete tasks using fewer actions:

Definition 2 *The remap relationship between action duration and reward*

$$R_\tau = t_{min}/t, \quad R_\tau \in [t_{min}/t_{max}, 1]$$

where t is the duration of the current action, t_{min} is the minimum duration of an action (strictly greater than 0), and t_{max} is the maximum duration of an action.

To automatically set α_m and α_ε to optimal values, we adjust them dynamically during training. Based on Wang & Beltrame (2024b)'s experience, we increase α_m and decrease α_ε to mitigate convergence issues, specifically the problem of sparse rewards caused by a suboptimal set of a large α_ε and a small α_m. This adjustment ensures that rewards are appropriately balanced, facilitating learning and convergence.

These parameters are adjusted based on observed trends in task rewards: if the average reward is declining (see Definition 4), we increase α_m and decrease α_ε, linking them with a sigmoid function, ensuring a balanced reward structure that promotes convergence and efficient learning. To guarantee stability, we ensure the change is monotonic, forcing a uniform sweep of the parameter space.

Definition 3 *The relationship between α_ε and α_m is*

$$\alpha_\varepsilon = 0.2 \cdot \left(1 - \frac{1}{1 + e^{-\alpha_m + 1}}\right)$$

Based on Wang & Beltrame (2024a)'s experience, we establish a mapping relationship between the two parameters: when the initial value of α_m is 1.0, the initial value of α_ε is 0.1. As α_m increases, α_ε decreases, but never falls below 0.

To determine the trend on the average reward, we perform a linear regression across the current training episode and compute the slope of the resulting line: if it is negative, the reward is declining.

Definition 4 *The slope of the average reward (k_R) is:*

$$k_R(R_a) = \frac{n\sum_{i=1}^{n}(i \cdot R_{ai}) - (\sum_{i=1}^{n} i)\,(\sum_{i=1}^{n} R_{ai})}{n\sum_{i=1}^{n} i^2 - (\sum_{i=1}^{n} i)^2}$$

where n is the total number of data points collected across the update interval (k_{update} in Algorithm 1). The update interval is a hyperparameter that determines the frequency of updates for these neural networks used in the actor and critic policies, occurring after every n episode (Sutton & Barto, 2018). R_a represents the list of average rewards $(R_{a1}, R_{a2}, ..., R_{an})$ during training. Here, an average reward R_{ai} is calculated across one episode.

After observing a downward trend in the average reward during the training process, we introduced the hyperparameter ψ to dynamically adjust α_m as defined in Definition 5.

Definition 5 *We adaptively adjust the reward every k_{update} episodes. When $k_R < 0$:*

$$\alpha_m = \alpha_m + \psi$$

where, ψ is the only additional hyperparameter required by our MOSEAC to tune the reward equation during training. Additionally, α_ε is adjusted as described in Definition 3.

Algorithm 1 shows the pseudocode of MOSEAC. Convergence analysis is provided in Appendix A.

Algorithm 1: Multi-Objective Soft Elastic Actor and Critic

Require: a policy π with a set of parameters θ, $\theta^{'}$, critic parameters ϕ, $\phi^{'}$, variable time step environment model Ω, learning-rate λ_p, λ_q, reward buffer β_r, replay buffer β.

Initialization $i = 0$, $t_i = 0$, $\beta_r = 0$, observe S_0
while $t_i \leq t_{max}$ **do**
  for $i \leq k_{length} \vee Not\ Done$ **do**
    $A_i, D_i = \pi_\theta(S_i)$ → simple action and its duration
    $S_{i+1}, R_i = \Omega(A_i, D_i)$ → compute reward with Definition 1 and 2
    $i \leftarrow i + 1$
  end
  $\beta_r \leftarrow 1/i \times \sum_0^i R_i$ → collect the average reward for one episode
  $\beta \leftarrow S_{0\sim i}$, $A_{0\sim i}$, $D_{0\sim i}$, $R_{0\sim i}$, $S_{1\sim i+1}$
  $i = 0$
  $t_i \leftarrow t_i + 1$
  if $t_i \geq k_{init}$ & $t_i \mid k_{update}$ **then**
    $Sample\, S,\ A,\ D,\ R,\ S^{'}\ from(\beta)$
    $\phi \leftarrow \phi - \lambda_q \nabla_\delta \mathcal{L}_Q(\phi,\ S,\ A,\ D,\ R,\ S^{'})$ → critic update
    $\theta \leftarrow \theta - \lambda_p \nabla_\theta \mathcal{L}_\pi(\theta,\ S,\ A,\ D,\ \phi)$ → actor update
    if $k_R(\beta_r)$ **then**
      $\alpha_m \leftarrow \alpha_m + \psi$ → see Definition 4 for k_R
      $\alpha_\varepsilon \leftarrow F_{update}(\alpha_m)$ → update $\alpha_m, \alpha_\varepsilon$ followed Definition 3
    end
    $\beta_r = 0$ → Re-record average reward values under new hyperparameters
  end
  Perform soft-update of $\phi^{'}$ and $\theta^{'}$
end

Here, t_{max} represents maximum training steps (Sutton & Barto, 2018); k_{length} is maximum exploration steps per episode (Sutton & Barto, 2018); k_{init} is steps in the initial random exploration

phase (Sutton & Barto, 2018). The reward R_i is calculated as $R(S_i, A_i, D_i)$, where D_i lies within $[T_{min}, T_{max}]$, representing action duration.

Our algorithm optimizes multiple objectives using a streamlined, weighted strategy. Unlike Hierarchical Reinforcement Learning (HRL) (Dietterich, 2000; Li et al., 2019), which seeks Pareto optimality (Kacem et al., 2002; Monfared et al., 2021) with layered reward policies, our method simplifies the approach. We emphasize user-friendliness and computational efficiency, making our strategy easily adaptable to various algorithms.

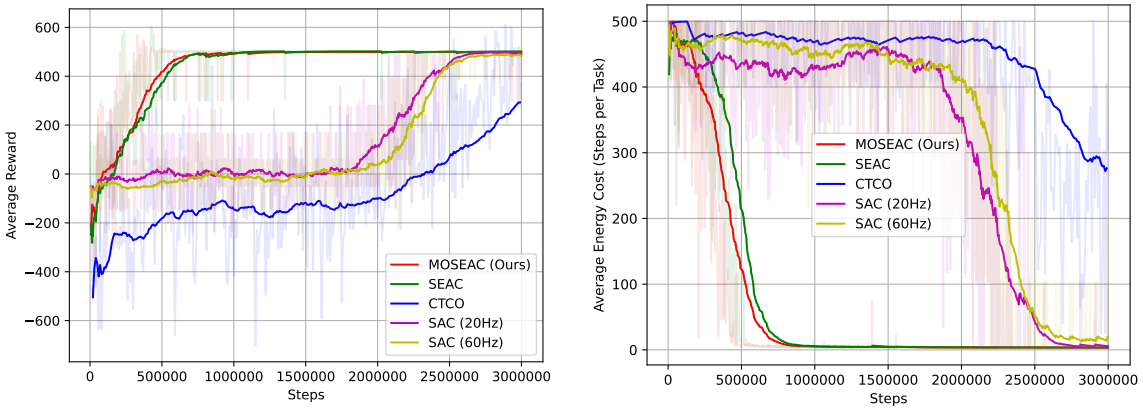
Apart from a series of hyperparameters inherent to RL that need adjustment, such as learning rate, t_{max}, k_{update}, etc., ψ is the only hyperparameter that requires tuning in MOSEAC.

However, one limitation of our approach is that high ψ values can lead to issues such as reward explosion. While a small ψ value generally avoids significant problems and guides α_m to its optimal value, it requires extended training periods. Determining the appropriate ψ value remains a critical optimization point in our method. We recommend using the pre-set ψ value provided in our implementation, thus reducing the need for additional parameter adjustments. If users encounter problems such as reward explosion, it is advisable to reduce the ψ value appropriately. Our ongoing research aims to mitigate further the risks associated with ψ, enhancing the algorithm's reliability and robustness.

4 Empirical Analysis

We conduct six experiments involving MOSEAC, SEAC, CTCO, and SAC algorithms (at 20 Hz and 60 Hz) within a Newtonian kinematics environment. A detailed description of the environment appears in Appendix B. These tests are performed on a computer equipped with an Intel Core i5-13600K and a Nvidia RTX 4070 GPU. The operating system is Ubuntu 22.04 LTS. For the hyperparameter settings of MOSEAC, please refer to Appendix C[1]

Figure 1 illustrates the result of training process. In Figure 1a, the average returns align with our expectations, showing that CTCO is sensitive to the choice of action duration range, which affects its discount factor γ. During the training process γ tended to be very small, compromising CTCO's long-term planning. Overall, CTCO performs worse than the two fixed-control frequency SAC baselines in this environment. In contrast, MOSEAC trains slightly faster than SEAC, as shown in Figure 1b.



(a) Average returns of 5 reinforcement learning algorithms in 3M steps during the training.

(b) Average Energy cost of 5 reinforcement learning algorithms in 3M steps during the training.

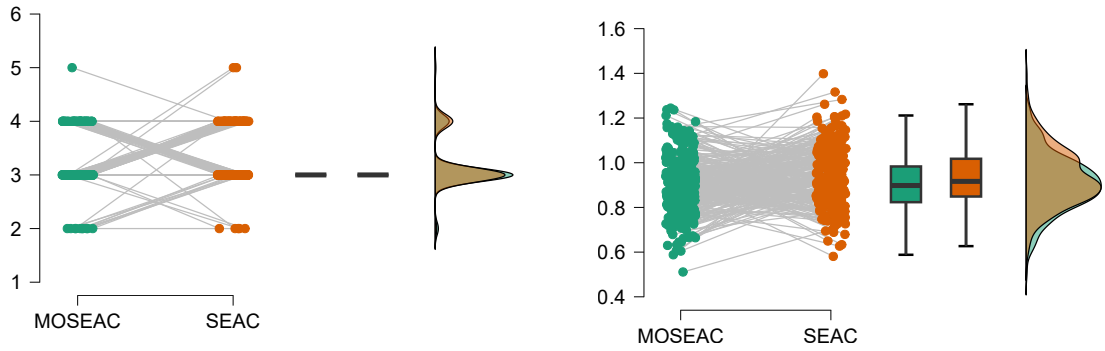
Figure 1: Result graphs of five reinforcement learning algorithms.

[1]Our code is public available.

We compared MOSEAC's task performance with the best-performing SEAC after training. Due to the poor performance of CTCO and SAC, we excluded them from the analysis. Figure 2 presents performance metrics related to energy consumption and task duration. Specifically, Figure 2a shows the average energy consumption (measured in steps) for 300 tasks, and Figure 2b compares average task durations. Data show that MOSEAC's average energy and time costs are lower than SEAC's.

Wilcoxon signed-rank test (chosen due to the lack of normality in the data distribution) indicates that MOSEAC's energy cost is lower than SEAC's ($z = -1.823$, $p = 0.020$), where MOSEAC has a mean energy cost of 3.120 (SD = 0.424), compared to SEAC's mean of 3.193 (SD = 0.451). Detailed statistical results are provided in Appendix D. Similarly, MOSEAC's time cost is significantly lower than SEAC's ($z = -2.669$, $p = 0.004$), with MOSEAC having a mean time cost of 0.905 (SD = 0.125), compared to SEAC's mean of 0.935 (SD = 0.127).

The improved performance of MOSEAC over SEAC can be attributed to its reward function. While SEAC's reward function is linear, combining task reward, energy penalty, and time penalty independently, MOSEAC introduces a multiplicative relationship between task reward and time-related reward. This non-linear interaction enhances the reward signal, particularly when both task performance and time efficiency are high, and naturally balances these factors. By keeping the energy penalty separate, MOSEAC maintains flexibility in tuning without complicating the relationship between time and task rewards. This design allows MOSEAC to more effectively guide the agent's decisions, resulting in better energy efficiency and task completion speed in practical applications.



(a) The energy cost (counted by numbers of steps) figure for 300 different tasks.

(b) The time cost figure for 300 different tasks.

Figure 2: Comparison figure of energy consumption and time performance of 300 different tasks. We use the same random seed to ensure that both have the same tasks.

5 Conclusion and Future Work

In this paper, we present MOSEAC, a VTS-RL algorithm with adaptive hyperparameters that respond to observed reward trends during training. This approach significantly reduces hyperparameter sensitivity and increases robustness, improving data efficiency. Unlike other VTS-RL algorithms, our method does not requires a single hyperparameter, thereby reducing learning and tuning costs when transitioning from fixed to variable time steps. This establishes a strong foundation for real-world applications.

Our next objective is to implement our method in real-world robotics applications, such as smart cars and robotic arms.

References

Neziha Akalin and Amy Loutfi. Reinforcement learning approaches in social robotics. *Sensors*, 21 (4):1292, 2021.

Susan Amin, Maziar Gomrokchi, Hossein Aboutalebi, Harsh Satija, and Doina Precup. Locally persistent exploration in continuous control tasks with sparse rewards. *arXiv preprint arXiv:2012.13658*, 2020.

Yann Bouteiller, Simon Ramstedt, Giovanni Beltrame, Christopher Pal, and Jonathan Binas. Reinforcement learning with random delays. In *International conference on learning representations*, 2021.

Endri Bregu, Nicola Casamassima, Daniel Cantoni, Luca Mottola, and Kamin Whitehouse. Reactive control of autonomous drones. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, pp. 207–219, 2016.

Yuxin Chen, Hejun Wu, Yongheng Liang, and Guoming Lai. Varlenmarl: A framework of variable-length time-step multi-agent reinforcement learning for cooperative charging in sensor networks. In *2021 18th Annual IEEE International Conference on Sensing, Communication, and Networking (SECON)*, pp. 1–9. IEEE, 2021.

Thomas G Dietterich. Hierarchical reinforcement learning with the maxq value function decomposition. *Journal of artificial intelligence research*, 13:227–303, 2000.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018a.

Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018b.

Julian Ibarz, Jie Tan, Chelsea Finn, Mrinal Kalakrishnan, Peter Pastor, and Sergey Levine. How to train your robot with deep reinforcement learning: lessons we have learned. *The International Journal of Robotics Research*, 40(4-5):698–721, 2021.

Imed Kacem, Slim Hammadi, and Pierre Borne. Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic. *Mathematics and computers in simulation*, 60(3-5):245–276, 2002.

Amirmohammad Karimi, Jun Jin, Jun Luo, A Rupam Mahmood, Martin Jagersand, and Samuele Tosatto. Dynamic decision frequency with continuous options. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7545–7552. IEEE, 2023.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

Jongmin Lee, Byung-Jun Lee, and Kee-Eung Kim. Reinforcement learning for control with multiple frequencies. *Advances in Neural Information Processing Systems*, 33:3254–3264, 2020.

Siyuan Li, Rui Wang, Minxue Tang, and Chongjie Zhang. Hierarchical reinforcement learning with advantage-based auxiliary rewards. *Advances in Neural Information Processing Systems*, 32, 2019.

Rongrong Liu, Florent Nageotte, Philippe Zanne, Michel de Mathelin, and Birgitta Dresp-Langley. Deep reinforcement learning for the control of robotic manipulation: a focussed mini-review. *Robotics*, 10(1):22, 2021.

Rupak Majumdar, Aman Mathur, Marcus Pirron, Laura Stegner, and Damien Zufferey. Paracosm: A test framework for autonomous driving simulations. In *International Conference on Fundamental Approaches to Software Engineering*, pp. 172–195. Springer International Publishing Cham, 2021.

Alberto Maria Metelli, Flavio Mazzolini, Lorenzo Bisi, Luca Sabbioni, and Marcello Restelli. Control frequency adaptation via action persistence in batch reinforcement learning. In *International Conference on Machine Learning*, pp. 6862–6873. PMLR, 2020.

Mohammadali Saniee Monfared, Sayyed Ehsan Monabbati, and Atefeh Rajabi Kafshgar. Pareto-optimal equilibrium points in non-cooperative multi-objective optimization problems. *Expert Systems with Applications*, 178:114995, 2021.

Seohong Park, Jaekyeom Kim, and Gunhee Kim. Time discretization-invariant safe action repetition for policy gradient methods. *Advances in Neural Information Processing Systems*, 34:267–279, 2021.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Sahil Sharma, Aravind Srinivas, and Balaraman Ravindran. Learning to repeat: Fine grained action repetition for deep reinforcement learning. *arXiv preprint arXiv:1702.06054*, 2017.

Bharat Singh, Rajesh Kumar, and Vinay Pratap Singh. Reinforcement learning in robotic applications: a comprehensive survey. *Artificial Intelligence Review*, 55(2):945–990, 2022.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

Dong Wang and Giovanni Beltrame. Deployable reinforcement learning with variable control rate. *arXiv preprint arXiv:2401.09286*, 2024a.

Dong Wang and Giovanni Beltrame. Reinforcement learning with elastic time steps. *arXiv preprint arXiv:2402.14961*, 2024b.

A Convergence Analysis of MOSEAC

This appendix aims to analyze the convergence of the Multi-Objective Soft Elastic Actor-Critic (MOSEAC) algorithm. Its action space includes a time dimension D and the reward function is $R = \alpha_m R_t R_\tau - \alpha_\varepsilon$, with α_m and α_ε dynamically adjusted during training, details can be found in section 3.

A.1 SAC Algorithm Overview

In the standard SAC algorithm (Haarnoja et al., 2018a), the policy $\pi_\theta(a|s)$ selects action a, and updates the policy parameters θ and value function parameters ϕ. The objective function is:

$$J(\pi_\theta) = \mathbb{E}_{(s,a)\sim\pi_\theta}\left[Q^\pi(s,a) + \alpha\mathcal{H}(\pi_\theta(\cdot|s))\right]$$

where $J(\pi_\theta)$ is the objective function, $Q^\pi(s,a)$ is the state-action value function, α is a temperature parameter controlling the entropy term, and $\mathcal{H}(\pi_\theta(\cdot|s))$ is the entropy of the policy.

A.2 Incorporating Time Dimension and Our Reward Function

When the action space is extended to include D and the reward function is modified to $R = \alpha_m R_t R_\tau - \alpha_\varepsilon$, where $\alpha_m \geq 0$, $0 < R_\tau \leq 1$, and α_ε is a small positive constant, the new objective function becomes:

$$J(\pi_\theta) = \mathbb{E}_{(s,a,D)\sim\pi_\theta}\left[Q^\pi(s,a,D) + \alpha\mathcal{H}(\pi_\theta(\cdot|s))\right]$$

where $Q^\pi(s,a,D)$ is the extended state-action value function with time dimension D, and R_t and R_τ are components of the reward function, see Definition 1.

A.3 Policy Gradient

With our reward function, the policy gradient is:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log \pi_\theta(a,D|s)\left(Q^\pi(s,a,D)\cdot(\alpha_m\cdot R_\tau) - \alpha_\varepsilon\right)\right]$$

where $\nabla_\theta J(\pi_\theta)$ is the gradient of the objective function with respect to the policy parameters θ.

A.4 Value Function Update

The value function update, incorporating the time dimension D and our reward function, is:

$$L(\phi) = \mathbb{E}_{(s,a,D,r,s')}\left[\left(Q_\phi(s,a,D) - \left(r + \gamma\mathbb{E}_{(a',D')\sim\pi_\theta}[V_{\bar{\phi}}(s') - \alpha\log\pi_\theta(a',D'|s')]\right)\right)^2\right]$$

where $L(\phi)$ is the loss function for the value function update, r is the reward, γ is the discount factor, and $V_{\bar{\phi}}(s')$ is the target value function.

A.5 Policy Parameter Update

The new policy parameter θ update rule is:

$$\theta_{k+1} = \theta_k + \beta_k\mathbb{E}_{s\sim D,(a,D)\sim\pi_\theta}\left[\nabla_\theta\log\pi_\theta(a,D|s)\left(Q_\phi(s,a,D)\cdot(\alpha_m\cdot R_\tau) - \alpha_\varepsilon - V_{\bar{\phi}}(s) + \alpha\log\pi_\theta(a,D|s)\right)\right]$$

where β_k is the learning rate at step k.

A.6 Dynamic Adjustment and Convergence Analysis

To analyze the impact of dynamically adjusting α_m and α_ε, we assume:

1. **Dynamic Adjustment Rules:**
   - α_m increases monotonically by a small increment ψ if the reward trend decreases over consecutive episodes (See Definition 4).
   - α_ε decreases with the Definition 3.

2. **Learning Rate Conditions:**
   The learning rates α_k and β_k must satisfy the following conditions (Konda & Tsitsiklis, 1999):

$$\sum_{k=0}^{\infty} \alpha_k = \infty, \quad \sum_{k=0}^{\infty} \alpha_k^2 < \infty$$
$$\sum_{k=0}^{\infty} \beta_k = \infty, \quad \sum_{k=0}^{\infty} \beta_k^2 < \infty$$

A.7 Unbiased Estimation

Assuming the critic estimates are unbiased:

$$\mathbb{E}[Q_\phi(s, a, D) \cdot (\alpha_m \cdot R_\tau) - \alpha_\varepsilon] = Q^\pi(s, a, D) \cdot (\alpha_m \cdot R_\tau) - \alpha_\varepsilon$$

Since R_τ is a positive number within [0, 1], its effect on $Q^\pi(s, a, D)$ is linear and does not affect the consistency of the policy gradient.

A.8 Impact Evaluation

1. **Positive Scaling:**
   As R_τ is always a positive number less than or equal to 1, and $\alpha_m \geq 0$, it only scales the reward, not altering its sign. This scaling does not change the direction of the policy gradient but affects its magnitude.

2. **Small Offset:**
   α_ε is a small constant used to accelerate training. This small offset does not affect the direction of the policy gradient but introduces a minor shift in the value function, which does not alter the overall policy update direction.

A.9 Conclusion

Under the aforementioned conditions, the modified SAC algorithm with the time dimension D and the new reward function $R = \alpha_m R_t R_\tau - \alpha_\varepsilon$, which we called MOSEAC, will converge to a local optimum, i.e. (Sutton & Barto, 2018),

$$\lim_{k \to \infty} \nabla_\theta J(\pi_\theta) = 0$$

This implies that over time, the policy parameters will stabilize at a local optimum, maximizing the policy performance.

B Details of Newtonian kinematics environment

To validate our algorithm, we set up an autonomous driving environment based on Newtonian dynamics, including a random birth point, a random endpoint, and a random obstacle. Check Definition 7 for its state value information, Definition 8 for its action value information, and Definition 6 for the physical formula it follows.

As Definition 1, the precise reward configuration for our environment are outlined in Table 1.

Definition 6 *The Newton Kinematics formulas are:*

$$D_{aim} = 1/2 \cdot (V_{aim} + V_{current}) \cdot T;$$

$$V_{aim} = V_{current} + AT;$$

$$F_{aim} = mA;$$

$$F_{true} = F_{aim} - f_{friction};$$

$$f_{friction} = \mu mg, \ \ if\ F_{aim} > f_{friction} \wedge V_{agent} \neq 0$$
$$f_{friction} = F_{aim}, \ \ if\ F_{aim} \leq f_{friction} \wedge V_{agent} = 0$$

where D_{aim} is the distance generated by the policy that the agent needs to move. V_{agent} is the speed of the agent. T is the time to complete the movement generated by the policy, F_{aim} is the traction force necessary for the agent to change to the aimed speed. m is the mass of the agent, μ is the friction coefficient, and g is the acceleration of gravity.

Table 1: Reward Settings for The Newtonian Kinematics Environment

Reward Settings		
Name	Value	Annotation
	500.0	Reach the goal
r	-500.0	Crash on an obstacle
	$D_{origin} - 1.0 \times D_{goal}$	D_{origin}: distance from start to goal; D_{goal}: distance to goal
α_m	1.0	

The state dimensions are shown in Definition 7:

Definition 7 *The positions of the endpoint and obstacle are randomized in each episode.*

$$S_t = (Pos, Obs, Goal, Speed, Time, Force)$$

Where:
Pos = Position Data of The Agent in X and Y Direction,
Obs = Position Data of The Obstacle in X and Y Direction,
$Goal$ = Position Data of The Goal in X and Y Direction,
$Speed$ = Spped Data of The Agent for Current Step,
$Time$ = The Duration data of The Agent to Execute The Last Time Step,
$Force$ = Force Data of The Agent for Last Time Step in X and Y Direction.

The Spatial Information of our environment are shown in Table 2:

The action dimensions are shown in Definition 8:

Table 2: Details of The Simple Newtonian Kinematics Gymnasium Environment

Environment details		
Name	Value	Annotation
Action dimension	3	
Range of speed	$[-2, 2]$	m/s
Action Space	$[-100.0, 100.0]$	Newton
Range of time	$[0.01, 1.0]$	second
State dimension	11	Task gain factor
World size	$(2.0, 2.0)$	in meters
Obstacle shape	Round	Radius: 5cm
Agent weight	20	in Kg
Gravity factor	9.80665	in m/s^2
Static friction coeddicient	0.6	

Definition 8 *Action*

Each action should have a corresponding execution duration.

$$a_t = (D_t, A_{fx}, A_{fy})$$

Where:
D_t *= The duration of the Agent to implement current action,*
A_{fx} *= The Force of the Agent in X Coordinate,*
A_{fy} *= The Force of the Agent in Y Coordinate.*

C Hyperparameters of MOSEAC

Hyperparameter sheet of MOSEAC		
Name	Value	Annotation
Total steps	$3e6$	
γ	0.99	Discount factor
Net shape	$(256, 256)$	
batch_size	256	
a_lr	$3e-5$	Learning rate of Actor Network
c_lr	$3e-5$	Learning rate of Critic Network
max_steps	500	Maximum steps for one episode
α	0.12	
η	-3	Refer to SAC (Haarnoja et al., 2018b)
min_time	0.01	Minimum control duration, in seconds
max_time	1.0	Maximum control duration, in seconds
α_m	1.0	Init value of α_m (Definition 1)
ψ	$1e-4$	Monotonically increasing H-parameter (Algorithm 1)
Optimizer	Adam	Refer to Adam (Kingma & Ba, 2014)
environment steps	1	
Replaybuffer size	$1e6$	
Number of samples before training start	$5 \cdot max\_steps$	
Number of critics	2	

D Statistic Results

Table 3: Paired Samples T-Test

Measure 1		Measure 2	W	z	p
MOSEAC Energy Cost	-	SEAC Energy Cost	2080.500	-1.823	0.020
MOSEAC Time Cost	-	SEAC Time Cost	18561.000	-2.669	0.004

Table 4: Test of Normality (Shapiro-Wilk)

			W	p
MOSEAC Energy Cost	-	SEAC Energy Cost	0.775	< 0.001
MOSEAC Time Cost	-	SEAC Time Cost	0.996	0.714

Table 5: Descriptives

	N	Mean	SD	SE	Coefficient of variation
MOSEAC Energy Cost	300	3.120	0.424	0.024	0.136
SEAC Energy Cost	300	3.193	0.451	0.026	0.141
MOSEAC Time Cost	300	0.905	0.125	0.007	0.139
SEAC Time Cost	300	0.935	0.127	0.007	0.136