Titre: Title:	Elastic Time Step Reinforcement Learning
Auteur: Author:	Dong Wang
Date:	2024
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Wang, D. (2024). Elastic Time Step Reinforcement Learning [Thèse de doctorat, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/59033/

URL de PolyPublie: PolyPublie URL:	https://publications.polymtl.ca/59033/
Directeurs de recherche: Advisors:	Giovanni Beltrame
Programme: Program:	Génie informatique

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Elastic Time Step Reinforcement Learning

DONG WANG
Département de génie informatique et génie logiciel

Thèse présentée en vue de l'obtention du diplôme de *Philosophiæ Doctor*
Génie informatique

Juillet 2024

POLYTECHNIQUE MONTRÉAL
affiliée à l'Université de Montréal

Cette thèse intitulée :

Elastic Time Step Reinforcement Learning

présentée par **Dong WANG**
en vue de l'obtention du diplôme de *Philosophiæ Doctor*
a été dûment acceptée par le jury d'examen constitué de :

DEDICATION

To my wife (Jiaying Tang),
and my parents . . .

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest and most heartfelt gratitude to my supervisor, Prof. Giovanni Beltrame. His unwavering support and belief in me during my darkest times of severe depression were nothing short of life-saving. His compassionate guidance and steadfast encouragement pulled me out of the depths of despair and gave me the strength to stand tall once more. Without his support and understanding, I would not have been able to persevere and complete my Ph.D. studies. Thank you for believing in me when I struggled to believe in myself. Throughout my Ph.D. journey, I faced significant physical and emotional challenges, including the broke to my right foot and the passing of my grandparents. In that dark year, he kept believing that I could get out of hell even if I didn't believe in myself.

Pursuing a Ph.D. is a journey marked by a constant oscillation between self-doubt and self-assurance, often accompanied by both physical and mental hardships. Throughout this journey, I faced significant challenges, including the break of my right foot, which left sequelae throughout my whole life, the loss of my loved grandparents, and various technical and theoretical difficulties. Despite these trials, perseverance, good communication with my colleagues, and steadfast dedication to my chosen path have ultimately led to triumph.

Working with Prof. Beltrame during these years has been a true pleasure. He consistently provided me with excellent technical advice and guidance, always standing by me, especially when reviewers offered harsh critiques and appropriate guidance and support. When I found it difficult to reintegrate into the academic environment, he encouraged and guided me, helping me find my footing again. His immense patience, exceptional communication skills, and genuine passion for his work set a perfect example of true leadership. I have learned a great deal from his exemplary display of leadership and am eager to emulate these qualities in the future.

Second, I thank all my friends and colleagues in MISTLab, especially Yann Bouteller. My interactions with them over the years have been everything I could have hoped for. The numerous fruitful discussions and the consistent, often selfless help I received from many of my colleagues significantly shaped the evolution of my Ph.D., making it what it is today.

Next, I would also like to express my gratitude to the China Scholarship Council (CSC) Scholarship for providing the financial support that allowed me to complete my Ph.D. studies without worrying about my quality of life. This scholarship not only alleviated my financial burdens but also enabled me to focus on my research and strive for academic excellence.

Then, I would like to thank my strong and beautiful wife, Jiaying Tang, for her selfless help and support during the darkest periods of my life. She gave me the courage and confidence to complete my Ph.D. studies. We started our long-distance relationship in 2013. After eleven years of ups and downs, we are finally bringing it to a close this year. Thank you for your unwavering support, enduring the challenges of time differences, and even seasonal differences every day. But it has all been worth it. I hope that our future life together will be smooth sailing, and from this moment on, our eleven-year long-distance journey will turn into a journey where we walk hand in hand, never to be separated again.

I would like to thank the authors of all the references. Their work guided me to explore the academic frontier.

I would also like to express my gratitude to the friends who supported me during my Ph.D. studies. Your presence helped me manage my mindset and alleviate pressure at all times. Thank you for your encouragement and support; my achievements are a testament to your contributions and unwavering assistance.

Last but not least, I would like to thank my family for all their unwavering support and motivation throughout the years. My family's sacrifices during my Ph.D. journey cannot be overstated, and their dedication gave me the motivation and resolve to persevere. Quitting was never an option because of their unrelenting support through the highest and lowest highs. I am privileged to have such a supportive family; no words can fully express my gratitude towards them.

RÉSUMÉ

Les avancées en intelligence artificielle et en technologie de la robotique ont accéléré l'intégration de l'apprentissage par renforcement (RL) en robotique, permettant aux systèmes d'accomplir des tâches complexes dans des environnements difficiles avec une efficacité et une sécurité accrues.

Les méthodes RL traditionnelles utilisent des fréquences de contrôle fixes, ce qui entraîne une utilisation inefficace des ressources informatiques. Par exemple, dans la conduite autonome sur des routes dégagées, des mises à jour fréquentes du contrôle sont inutiles, mais des fréquences fixes imposent des demandes informatiques constantes, entravant le déploiement de RL sur des ordinateurs embarqués aux ressources limitées. Cette inefficacité impacte également les fonctions de détection environnementale et de communication. Des fréquences de contrôle sous-optimales peuvent dégrader les performances, provoquant des réactions lentes aux changements ou un gaspillage de ressources par des actions excessives, entraînant une consommation d'énergie plus élevée et des échecs potentiels dans les applications sensibles au temps. Par conséquent, l'optimisation des fréquences de contrôle est cruciale pour les modèles RL dans des environnements complexes.

Pour relever ces défis, nous proposons le cadre d'apprentissage par renforcement à pas de temps variable (VTS-RL), qui ajuste dynamiquement la fréquence de contrôle en fonction des exigences de la tâche. Cette approche réduit l'utilisation des ressources informatiques et garantit que les actions se produisent uniquement lorsque cela est nécessaire. La mise en œuvre et l'évaluation de VTS-RL dans les systèmes robotiques sont difficiles en raison du manque de cadres spécifiques et d'implémentations de référence, car les processus de décision de Markov traditionnels (MDP) ne tiennent pas compte des intervalles de temps entre les actions, ce qui limite la recherche dans ce domaine.

Nous introduisons deux algorithmes : Soft Elastic Actor-Critic (SEAC) et Multi-Objective Soft Elastic Actor-Critic (MOSEAC). SEAC améliore sa politique d'action en intégrant la durée des actions et en utilisant une politique de récompense novatrice pour minimiser la consommation d'énergie et le temps. Cette approche établit VTS-RL dans les systèmes robotiques, améliorant l'attribution de crédit et réduisant les frais généraux informatiques, ce qui le rend adapté aux déploiements à ressources limitées. Les tests de simulation valident l'efficacité de SEAC, démontrant des performances élevées et une charge informatique réduite par rapport à d'autres algorithmes RL à fréquence de contrôle fixe.

MOSEAC, une version améliorée de SEAC, offre des vitesses d'entraînement plus rapides et une simplification du réglage des hyperparamètres tout en suivant les principes de la programmation réactive pour minimiser la consommation d'énergie et le temps. Il présente un schéma de récompense adaptatif. Nous validons théoriquement son efficacité avec des garanties de performance, des analyses de convergence et de complexité. La formule de récompense adaptative équilibre l'efficacité énergétique et le temps d'accomplissement des tâches, explorant le front de Pareto en optimisation multi-objectif pour trouver des compromis optimaux. Les expériences avec Trackmania, un jeu de course vidéo, et AgileX Limo dans un environnement de test en intérieur démontrent l'efficacité de notre cadre à réduire les besoins informatiques. Enfin, nous passons en revue les caractéristiques clés, discutons des limitations et suggérons des directions de recherche futures.

ABSTRACT

Artificial intelligence and robotics technology advancements have accelerated the integration of reinforcement learning (RL) in robotics, enabling systems to perform complex tasks in challenging environments with enhanced efficiency and safety.

Traditional RL methods utilize fixed control frequencies, resulting in inefficient use of computational resources. For instance, in autonomous driving on clear roads, frequent control updates are unnecessary, yet fixed frequencies impose constant computing demands, hindering RL deployment on resource-constrained onboard computers. This inefficiency also impacts environmental sensing and communication functions. Suboptimal control frequencies can degrade performance, causing slow reactions to changes or resource wastage through excessive actions, leading to higher energy consumption and potential failures in time-sensitive applications. Therefore, optimizing control frequencies is crucial for RL models in complex environments.

To address these challenges, we propose the Variable Time-Step Reinforcement Learning (VTS-RL) framework, which dynamically adjusts the control frequency based on task requirements. This approach reduces computational resource usage and ensures actions occur only when necessary. Implementing and evaluating VTS-RL in robotic systems is challenging due to the lack of specific frameworks and reference implementations, as traditional Markov Decision Processes (MDPs) do not account for time intervals between actions, leading to limited research focus in this area.

We introduce two algorithms: Soft Elastic Actor-Critic (SEAC) and Multi-Objective Soft Elastic Actor-Critic (MOSEAC). SEAC enhances its action policy by incorporating action duration and employing a novel reward policy to minimize energy consumption and time. This approach establishes VTS-RL in robotic systems, improving credit assignment and reducing computational overhead, making it suitable for resource-constrained deployments. Simulation tests validate SEAC's efficacy, demonstrating high performance and reduced computational load compared with other fixed control frequency RL algorithms.

MOSEAC, an enhanced version of SEAC, offers faster training speeds and simplified hyperparameter tuning while following reactive programming principles to minimize energy and time consumption. It features an adaptive reward scheme. We theoretically validate its effectiveness with performance guarantees, convergence, and complexity analysis. The adaptive reward formula balances energy efficiency and task completion time, exploring the Pareto front in multi-objective optimization to find optimal trade-offs. Experiments with Trackmania, a video racing game, and AgileX Limo in an indoor test environment demonstrate our framework's effectiveness in reducing computational requirements. Finally, we review key features and discuss limitations and future research directions.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF SYMBOLS AND ACRONYMS

RL	Reinforcement Learning
DRL	Deep Reinforcement Learning
VTS-RL	Variable Time Step Reinforcement Learning
SAC	Soft Actor-Critic
PPO	Proximal Policy Optimization
TD3	Twin Delayed Deep Deterministic Policy Gradient
SEAC	Soft Elastic Actor-Critic
MOSEAC	Multi-Objective Soft Elastic Actor-Critic
$\mathcal{S}$	Set of states
$\mathcal{A}$	Set of actions
D	Action durations set
π_θ	Policy parameters θ
$\|\theta\|$	Policy parameter count
$V^\pi(s)$	Value function
$Q^\pi(s, a, D)$	Q-value function with D
α	Entropy temperature param.
α_m	Reward scaling param.
α_ϵ	Reward offset param.
R_t	Reward at time t
R_τ	Time-based reward
γ	Discount factor
ϕ	Value function params.
β_k	Learning rate at step k
$\mathcal{T}$	Soft Bellman operator
$\mathcal{H}(\pi(\cdot\|s))$	Policy entropy
ρ_π	State-action distribution
O	Complexity upper bound

LIST OF APPENDICES

CHAPTER 1 INTRODUCTION

1.1 Context and Motivation

Deep reinforcement learning (DRL) [3, 4] has demonstrated substantial potential in robotic applications, particularly in robot control [5, 6, 7] and video games [8, 9, 10]. The versatility and adaptability of RL algorithms have enabled significant advancements in these fields, allowing agents to perform complex tasks with minimal human intervention. This progress underscores the transformative impact of DRL in automating operations across various domains.

Despite these advancements, traditional Markov Decision Process (MDP) frameworks [11] fall short of addressing the temporal dynamics of action duration. The conventional approach to implementing RL in real-world scenarios often involves discretely simulated time steps with fixed action durations. This simplification, while practical, can lead to significant inefficiencies when applied to dynamic environments [12, 13].

In particular, using fixed action durations can result in two major issues: Firstly, robots may react too slowly to rapidly changing conditions, compromising performance and responsiveness. Secondly, the system might act too frequently, leading to unnecessary computational overhead and increased energy consumption. These inefficiencies are especially problematic in areas such as autonomous navigation and robotic manipulation, where adapting swiftly to environmental changes is critical.

The implications of these inefficiencies are far-reaching. In autonomous navigation, for example, delayed reactions can cause navigation errors or collisions, while excessive actions can drain battery life and reduce operational time. In robotic manipulation, the lack of timely responses can impair the robot's ability to handle objects accurately, affecting the overall task performance.

1.2 Problem Statement

One potential solution to the inefficiencies caused by fixed control frequencies is to develop a framework that dynamically adjusts action durations based on task demands. This approach could enhance performance, reduce energy consumption, and extend operational time. However, implementing such a solution presents several challenges.

Firstly, ensuring that control frequency adjustments are systematic and practical requires a structured approach. Changes could become arbitrary without specific rules, leading to instability and unpredictable performance.

Secondly, ensuring that algorithms adapted to our framework maintain convergence and performance guarantees while not introducing significant computational complexity is crucial. The framework must also be user-friendly to facilitate broader adoption.

Thirdly, deploying the VTS-RL framework in real-world robotic systems is challenging due to limited computational resources. Many existing RL frameworks are designed with fixed action durations in mind, making it difficult to adapt them for variable time steps. This complicates simulation training and real-world implementation, where computational efficiency and reliability are paramount.

Given these complexities, our primary research objectives are:

1. Developing a structured approach for adjusting control frequencies, ensuring changes are rule-based.
2. Ensuring that algorithms adapted to our framework maintain convergence and performance guarantees without significant computational overhead.
3. Overcoming the challenges of deploying VTS-RL in resource-constrained robotic systems, including adapting existing RL frameworks to support variable time steps.

Through these objectives, we aim to provide a comprehensive solution to the inefficiencies inherent in traditional RL methods, enabling more adaptable and efficient robotic systems.

1.3 Research Objectives

The *primary research objective* of this thesis is to develop and validate a Variable Time Step Reinforcement Learning (VTS-RL) framework to enhance RL's performance on robotic system in dynamic and resource-constrained environments. The VTS-RL framework aims to adapt control frequencies based on task demands, optimizing performance and energy efficiency. By incorporating control frequency into the action policy, the action and its execution duration are determined simultaneously.

Research Objective 1: Development of VTS-RL Algorithms

The development of VTS-RL algorithms should address the following points:

1.1 Establish a principle where control actions are executed only when necessary, minimizing the number of control updates to optimize performance in dynamic environments.

1.2 Create a reward function that balances task performance, energy consumption, and time efficiency. The goal is to minimize the number of steps and the overall time to achieve optimal performance.

1.3 Design the framework for general applicability, extending it to various RL algorithms, thereby increasing user-friendliness and facilitating further development and application.

Research Objective 2: Theoretical Foundations of VTS-RL

After developing the algorithms, the next step is to establish the theoretical foundations to ensure reproducibility and sustainable optimization:

2.1 Prove system convergence to an optimal policy and stability, demonstrating that the system will reliably reach a stable state under varying control frequencies and dynamic task demands.

2.2 Establish performance guarantees under varying action duration conditions, ensuring that the VTS-RL framework maintains high performance and adapts efficiently.

2.3 Analyze the computational complexity to ensure feasibility and efficiency, assessing the algorithm's scalability and practical applicability in real-world scenarios.

Research Objective 3: Practical Validation of VTS-RL

Finally, validating the practical applicability and efficiency of the VTS-RL framework in real-world robotic systems is crucial:

3.1 Implement and test VTS-RL on a video game and a rover robot platform, providing diverse and challenging environments to test the framework's capabilities and limitations.

3.2 Evaluate adaptability and performance in various real-world environments, assessing key metrics such as task completion, energy efficiency, and responsiveness to demonstrate the framework's robustness and versatility.

These research objectives aim to systematically address the challenges associated with fixed control frequencies in traditional RL methods, paving the way for more adaptable and efficient robotic systems.

1.4 Research Contributions

Our research works (RW) address the challenges and objectives outlined in the previous section, providing solutions through the development and validation of the Variable Time Step Reinforcement Learning (VTS-RL) framework.

RW 1: Development and Enhancement of VTS-RL Algorithms

To address the inefficiency and instability issues caused by fixed control frequencies (Objective 1), we developed the Soft Elastic Actor Critic (SEAC) algorithm, which integrates action durations into the action policy and formulates a reward function balancing task performance, energy consumption, and time efficiency. Building on SEAC, we introduced the Multi-Objective Soft Elastic Actor Critic (MOSEAC) algorithm, incorporating an adaptive reward framework that dynamically adjusts based on reward trends during training. Both algorithms adhere to the principle of executing control actions only when necessary, optimizing performance in dynamic environments.

These contributions were published in:

> Wang, Dong, and Giovanni Beltrame. *"Deployable reinforcement learning with variable control frequency."* In Proceedings of the Association for the Advancement of Artificial Intelligence Conference, Deployable AI Workshop, 2024.
>
> Wang, Dong, and Giovanni Beltrame. *"MOSEAC: Streamlined Variable Time Step Reinforcement Learning."* In Proceedings of the Reinforcement Learning Conference, Finding the Frame Workshop, 2024

RW 2: Theoretical Foundations and Convergence Analysis

To ensure the MOSEAC maintains convergence (Objective 2), we introduced an upper limitation on the MOSEAC's hyperparameter α_m as $\alpha_{\max}$ to ensure its convergence. A convergence proof was made by establishing a Lyapunov stability model [14]. This work validates the theoretical foundations of MOSEAC and illustrates its effectiveness in a sparse reward environment, Trackmania, a video racing game.

This work was submitted to:

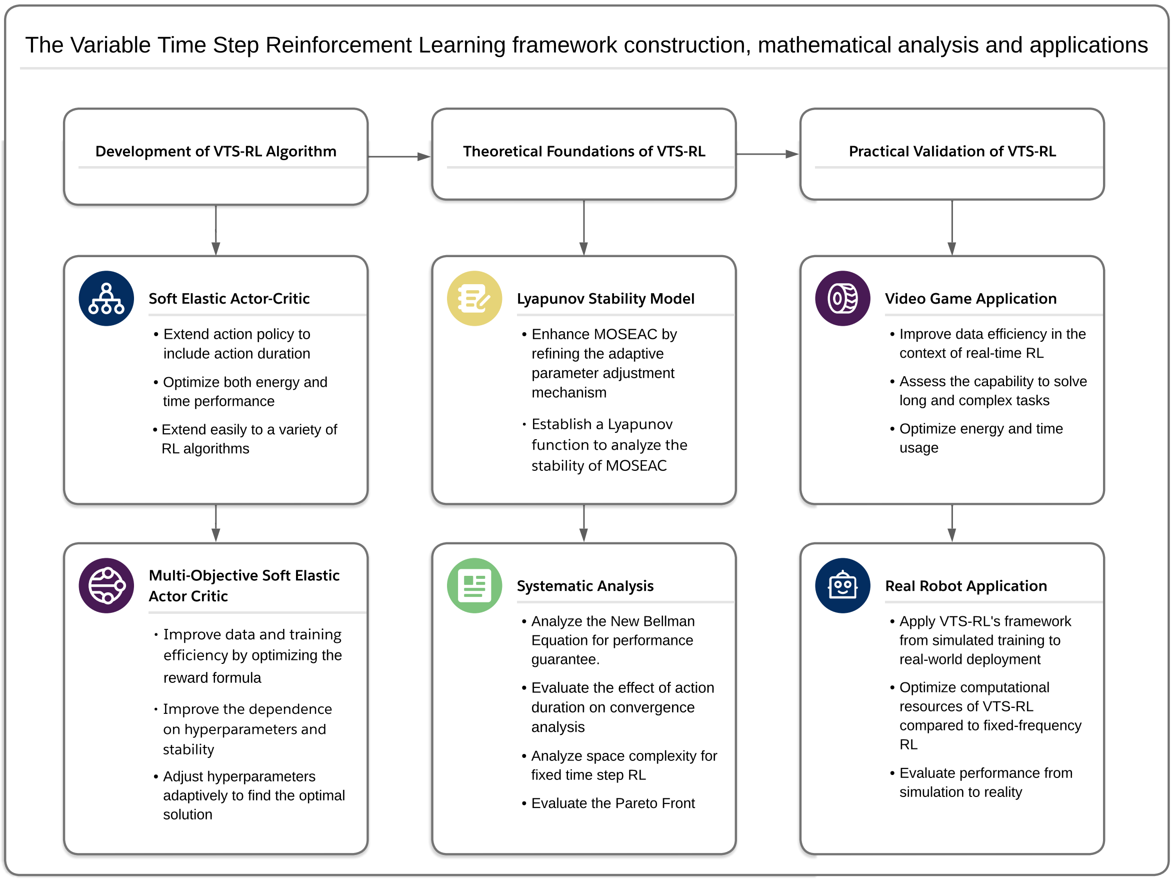
> Wang, Dong, and Giovanni Beltrame. *"Reinforcement Learning with Elastic Time Steps."* Submitted to IEEE Robotics and Automation Letters, 2024.

RW 3: Practical Validation and Deployment

To ensure the MOSEAC maintains convergence and performance guarantees without significant computational overhead (Objective 2) and the challenge of deploying VTS-RL in resource-constrained robotic systems (Objective 3), we deployed the improved MOSEAC on an AgileX Limo robot, validating its real-world applicability. This work includes performance guarantees, convergence proof, and complexity analysis, demonstrating energy efficiency on both CPU and GPU compared to fixed-frequency RL models. It also highlights the practical benefits of MOSEAC in handling real-world dynamic tasks efficiently.

This work was submitted to:

> Wang, Dong, and Giovanni Beltrame. *"Variable Time Step Reinforcement Learning for Robotic Applications."* Submitted to IEEE Transactions on Robotics, 2024.



The Variable Time Step Reinforcement Learning framework construction, mathematical analysis and applications
Development of VTS-RL Algorithm
Theoretical Foundations of VTS-RL
Practical Validation of VTS-RL
Soft Elastic Actor-Critic
• Extend action policy to include action duration
• Optimize both energy and time performance
• Extend easily to a variety of RL algorithms
Lyapunov Stability Model
• Enhance MOSEAC by refining the adaptive parameter adjustment mechanism
• Establish a Lyapunov function to analyze the stability of MOSEAC
Video Game Application
• Improve data efficiency in the context of real-time RL
• Assess the capability to solve long and complex tasks
• Optimize energy and time usage
Multi-Objective Soft Elastic Actor Critic
• Improve data and training efficiency by optimizing the reward formula
• Improve the dependence on hyperparameters and stability
• Adjust hyperparameters adaptively to find the optimal solution
Systematic Analysis
• Analyze the New Bellman Equation for performance guarantee.
• Evaluate the effect of action duration on convergence analysis
• Analyze space complexity for fixed time step RL
• Evaluate the Pareto Front
Real Robot Application
• Apply VTS-RL's framework from simulated training to real-world deployment
• Optimize computational resources of VTS-RL compared to fixed-frequency RL
• Evaluate performance from simulation to reality

Figure 1.1 Technical Route for the Development, Theoretical Foundations, and Practical Validation of the Variable Time Step Reinforcement Learning (VTS-RL) Framework

Figure 1.1 illustrates the technical route of our main research objective. Achieving these objectives aims to advance the theoretical foundations and practical applications of VTS-RL, leading to more efficient, adaptable, and reliable RL-based robotic systems.

1.5 Research Impact

We envision that our proposed methods will significantly impact the RL field for robotic systems, particularly in dynamic and resource-constrained environments. The development of VTS-RL frameworks, such as MOSEAC and SEAC, addresses critical challenges in optimizing task performance, energy consumption, and control frequencies.

A pertinent example of potential application is the development of a wearable robotic limb system for astronaut assistance during extravehicular activities (EVA) [15]. Maintaining an astronaut's position in space without the help of robotic arms is labor-intensive and challenging. Our VTS-RL framework, particularly MOSEAC, could be applied to such systems to dynamically adjust control strategies in response to varying impact conditions. This adaptability could ensure that the robotic limbs provide optimal support, reducing deviation and recovery time while preventing oscillation.

Furthermore, MOSEAC's application is particularly advantageous for space activities with limited onboard computational resources. By reducing computational demands and adhering to reactive programming principles, MOSEAC ensures controls are executed only when necessary. This efficiency makes it suitable for space missions, where performance must be maintained despite limited onboard computer capabilities. Additionally, by lowering computational requirements, VTS-RL allows more resources to be dedicated to critical tasks such as environmental perception and communication, enhancing overall mission efficiency.

During the development of these methods, extensive testing and validation were conducted using the rover robot platform. This demonstrated the practical benefits of MOSEAC in real-world applications. Validation included performance guarantees, convergence proof, and complexity analysis, highlighting MOSEAC's energy efficiency on CPU and GPU compared to traditional fixed-frequency RL models. These contributions are expected to advance the state of RL in robotics, promoting more efficient, adaptive, and sustainable robotic systems.

By addressing the key challenges in VTS-RL development and deployment, this research not only enhances the theoretical foundations of RL but also provides practical solutions for real-world applications. This work paves the way for more advanced and adaptive RL algorithms that can efficiently operate in various dynamic and resource-constrained environments, making significant strides in the fields of robotics and artificial intelligence.

1.6 Thesis Organization

This chapter outlines the thesis deliverables and main ideas, linking them to the research objectives. Chapter 1 provides background and motivation, problem statement, and research objectives.

Chapter 2 offers a literature review on RL fundamentals, RL applications in robotics, limitations of existing methods, and theoretical foundations related to VTS-RL.

Chapter 3 details the development of the Soft Elastic Actor Critic (SEAC), its reward formula, and validation in a Newtonian kinematics environment. Chapter 4 introduces the Multi-Objective Soft Elastic Actor Critic (MOSEAC), presenting the VTS-RL framework, pseudocode, and convergence analysis. Both are validated in Newtonian kinematics environments, addressing Research Objective 1. Chapter 5 improves the MOSEAC, providing pseudocode and proposing a convergence proof using a Lyapunov stability model, validated in a video game environment with sparse rewards, addressing Research Objectives 1 and 2. Chapter 6 applies MOSEAC to real-world robotic navigation tasks, evaluating its performance and energy-saving capabilities compared to fixed-frequency RL, and conducts a theoretical analysis of MOSEAC, covering Pareto optimality, performance guarantees, convergence analysis, and complexity analysis, addressing Research Objectives 2 and 3.

Chapter 7 summarizes the contributions, highlighting the advantages of VTS-RL, with commentary on related points and insights into current limitations and future research directions. Finally, Chapter 8 provides an overview of the development, validation, and practical implications of the VTS-RL framework and MOSEAC, emphasizing their significance in advancing RL research and applications.

CHAPTER 2 LITERATURE REVIEW

This chapter provides a comprehensive literature review covering the main themes relevant to this thesis. The key topics we discuss include:

1. **Review of Reinforcement Learning Algorithms and Their Mathematical Analysis:** Reviewing the convergence and performance guarantees of traditional RL.
2. **Traditional Reinforcement Learning in Robotics:** Reviewing RL applications in robotics, including the development history and critical contributions.
3. **Adaptive Frequency Control in Reinforcement Learning:** Elaborating the need for adaptive control frequencies in RL, highlighting its significance in improving efficiency and performance.
4. **Other Related Theoretical Works:** Reviewing other theoretical work related to ours, including Reactive Programming, Hierarchical Reinforcement Learning, Pareto Frontiers, and Lyapunov stability.

This review provides a detailed context for the advancements and challenges in the field, setting the foundation for the development and implementation of VTS-RL in real robotic applications.

2.1 Review of Reinforcement Learning Algorithms and Their Mathematical Analysis

Reinforcement learning (RL) as an important branch of machine learning [16, 17] has its development traced back to the 1950s. The foundational mathematical concept was introduced by Bellman in 1957 with the Markov Decision Process (MDP), which provided a solid theoretical basis for later research [18]. An MDP consists of a state space S, an action space A, state transition probabilities $P(s'|s,a)$, a reward function $R(s,a)$, and a discount factor γ. The objective is to find a policy $\pi(a|s)$ that maximizes the cumulative reward:

$$V^{\pi}(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| \pi\right]$$

Bellman also proposed the method of dynamic programming, which is crucial for solving RL problems [19]. The Bellman equation is central to these methods:

$$V(s) = \max_a \sum_{s'} P(s'|s,a)\left[R(s,a) + \gamma V(s')\right]$$

In the 1980s, the theory and algorithms of reinforcement learning were further developed. Watkins introduced the Q-learning algorithm in his 1989 doctoral thesis and proved its convergence under certain conditions [20]. The Q-learning update rule is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

Sutton's 1988 work on temporal-difference (TD) learning also became a significant theoretical foundation in RL [21].

2.1.1 Model-Based and Model-Free Reinforcement Learning

Reinforcement learning algorithms can be categorized into model-based [22, 23] and model-free [24, 25, 26] methods. Model-based methods rely on constructing an environment model to make decisions and plans, which are effective when the environment model is known or easy to construct. However, in complex and highly uncertain environments, building an accurate model can be very challenging.

Model-free methods, on the other hand, do not depend on an environment model but directly learn the optimal policy and value function from experience. These methods are applicable in a wider range of scenarios, especially when the environment model is difficult to construct. Both Q-learning and policy gradient [27] methods are examples of model-free RL.

2.1.2 Unbiased and Biase Value Estimation

A classic example of a model-free method is the Monte Carlo method [28, 29], which estimates the state value function $V(s)$ by averaging the returns obtained from multiple samples. This approach is essential for model-free RL as it evaluates the value function without needing a model of the environment's dynamics. By relying on actual returns from sampled trajectories, Monte Carlo methods bypass the need to explicitly estimate transition probabilities and reward functions, which can be highly complex or unknown in many real-world scenarios.

Monte Carlo methods provide unbiased value function estimates, ensuring that the learned policy converges to the optimal policy over time [30, 31]. Without such unbiased estimates, the learning process could be skewed, resulting in suboptimal or divergent policies [32, 33]. However, Monte Carlo methods require complete episodes for their estimates, which can be inefficient and impractical for environments with long or infinite horizons. The Monte Carlo update rule for state value functuion $V(s)$ is given by:

$$V(s) \approx \frac{1}{N} \sum_{i=1}^{N} G_i$$

where G_i is the total return from the i-th sample starting from state s, and N is the number of samples. As N increases, the estimate $V(s)$ converges to the true value.

In contrast, temporal-difference (TD) methods provide biased estimates but update value functions more frequently, typically at each time step. This allows TD methods to learn from incomplete episodes and be more sample-efficient, making them suitable for environments where it is impractical to wait for episode termination [21, 20]. The TD update rule for state value function $V(s)$ is given by:

$$V(s) \leftarrow V(s) + \alpha \left[R(s,a) + \gamma V(s') - V(s)\right]$$

While TD methods introduce bias, they reduce variance and often achieve faster convergence than Monte Carlo methods. This trade-off between bias and variance is a critical consideration in the design and selection of RL algorithms [34].

In summary, Monte Carlo estimation is a cornerstone of model-free RL [35, 29]. It enables algorithms to learn optimal policies directly from experience without needing an explicit environment model. This simplifies the learning process and enhances the robustness and applicability of RL algorithms in diverse and complex domains. However, integrating biased estimations like TD methods offers an alternative approach that balances efficiency and convergence, broadening the scope and effectiveness of model-free RL.

2.1.3 On-Policy and Off-Policy Reinforcement Learning

Reinforcement learning algorithms can also be classified into on-policy [36, 37] and off-policy [38, 33] methods. On-policy methods directly evaluate and improve the policy being executed, while off-policy methods can learn from data generated by a different policy.

On-policy methods such as SARSA (State-Action-Reward-State-Action) [39] evaluate and improve the same policy. The update rule for SARSA is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R(s,a) + \gamma Q(s',a') - Q(s,a)\right]$$

where (s', a') is the next state-action pair chosen according to the current policy π.

Off-policy methods like Q-learning use data generated by a behavior policy to learn the target policy. The Q-learning update rule is:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

On-policy methods generally offer better stability and convergence but may be less efficient in exploring new policies. Off-policy methods have higher sample efficiency [40].

2.1.4 Development of Policy Gradient Methods

In the 2000s, policy gradient methods became a research focus. Sutton et al. proposed policy gradient methods that directly optimize policy parameters to improve learning efficiency [27]. The policy gradient theorem provides the form of the gradient:

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) Q^\pi(s,a) \right]$$

In the same year, Konda and Tsitsiklis described the Actor-Critic algorithm, which is an important method in modern RL, combining policy gradient and value function approximation [41]. The Actor-Critic algorithm includes two main components: the Actor, which updates the policy $\pi_\theta(a|s)$, and the Critic, which evaluates the policy $V_w(s)$ or $Q_w(s,a)$. The key update rules are:

$$\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a|s) \delta$$

$$w \leftarrow w + \beta \delta \nabla_w V_w(s)$$

where the temporal difference error δ is:

$$\delta = R(s,a) + \gamma V_w(s') - V_w(s)$$

2.1.5 Advancements in Model-Free Deep Reinforcement Learning

With the development of deep learning, reinforcement learning has seen significant advancements in complex tasks. In 2015, Mnih et al. introduced the Deep Q-Network (DQN), which combined deep learning and reinforcement learning, significantly improving performance in complex tasks [42]. DQN is a classic off-policy algorithm that combines the Q-learning update rule with convolutional neural networks (CNN) [43] to handle high-dimensional state spaces.

Subsequently, Schulman et al. proposed the Proximal Policy Optimization (PPO) algorithm in 2017, which achieved significant results in policy optimization [44]. PPO is an on-policy algorithm that stabilizes the learning process by limiting the extent of policy updates. Its core idea is to constrain the policy changes in each update to ensure the new policy does not deviate too much from the old one, thereby improving training stability and efficiency.

Recently, Haarnoja et al. proposed the Soft Actor-Critic (SAC) algorithm, which combines the idea of maximum entropy reinforcement learning, aiming to maximize both the expected return and the policy entropy, demonstrating excellent performance in continuous control tasks [45]. The objective function of SAC is:

$$J(\pi) = \sum_{t=0}^{T} \mathbb{E}_{(s_t,a_t)\sim\rho_\pi} \left[R(s_t, a_t) + \alpha\mathcal{H}(\pi(\cdot|s_t))\right]$$

The policy update formula is:

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{(s_t,a_t)\sim\rho_\pi} \left[\nabla_\theta \log \pi_\theta(a_t|s_t) \left(\alpha \log \pi_\theta(a_t|s_t) - Q_\phi(s_t, a_t)\right)\right]$$

The Q-value update formula is:

$$J(Q_\phi) = \mathbb{E}_{(s_t,a_t)\sim\mathcal{D}} \left[\frac{1}{2} \left(Q_\phi(s_t, a_t) - (R(s_t, a_t) + \gamma\mathbb{E}_{s_{t+1}\sim P}[V_{\bar{\phi}}(s_{t+1})])\right)^2\right]$$

The state value function update formula is:

$$V_{\bar{\phi}}(s_t) = \mathbb{E}_{a_t\sim\pi} \left[Q_\phi(s_t, a_t) - \alpha \log \pi_\theta(a_t|s_t)\right]$$

In summary, the development of reinforcement learning has progressed from the introduction of fundamental theories to the evolution of complex algorithms.

2.1.6 MuZero and Recent Advances

In recent years, significant advances have been made in reinforcement learning, notably the introduction of the MuZero algorithm by Schrittwieser et al. in 2019 [46]. MuZero represents a breakthrough in model-based reinforcement learning by effectively learning both the model and the planning algorithm from data. Unlike previous model-based approaches, MuZero does not require an explicit model of the environment's dynamics but instead learns a latent state representation and performs lookahead search in this learned representation.

MuZero's key innovation is its ability to learn a compact representation of the environment's dynamics and value function directly from raw observations, which allows it to perform effective planning and decision-making. This approach has demonstrated remarkable performance in various challenging domains, including board games like Go, Chess, and Shogi, as well as complex video games like Atari.

The MuZero algorithm incorporates three main components:

- **Representation Network:** Converts observations into a latent state representation.
- **Dynamics Network:** Predicts the next latent state and reward given the current latent state and action.
- **Prediction Network:** Estimates the policy and value function from the latent state.

The update rules for MuZero involve backpropagating through these networks to minimize the prediction error and maximize the expected return, leveraging powerful neural network architectures to handle high-dimensional input spaces.

MuZero has set a new standard in reinforcement learning by combining the strengths of model-based and model-free approaches, enabling it to achieve state-of-the-art performance in a variety of challenging tasks.

2.1.7 Limitations

Despite these advancements, reinforcement learning still faces several challenges and limitations. One major issue is the exploration-exploitation trade-off, which remains an open problem in many RL applications. Balancing the need to explore new actions and states with the desire to exploit known rewarding actions is critical for efficient learning.

Another significant challenge is the sample inefficiency of many RL algorithms. Model-free methods, in particular, often require a large number of interactions with the environment to learn effective policies. This is particularly problematic in real-world applications where interactions can be costly or limited.

Moreover, the robustness and generalization of RL algorithms are active areas of research. Ensuring that learned policies can generalize well to unseen states and environments is crucial for deploying RL in practical scenarios.

In conclusion, the field of reinforcement learning has made remarkable progress over the past decades, from foundational theories to state-of-the-art algorithms like MuZero. Continued research and innovation will be essential to overcome current limitations and unlock the full potential of RL in various applications, particularly in robotics and other dynamic, real-world domains.

2.2 Traditional Reinforcement Learning in Robotics

2.2.1 Model-Based Reinforcement Learning in Robotics

Model-based reinforcement learning (MBRL) has been leveraged for various robotic applications, particularly in the context of low-level control of robotic components [22, 47]. MBRL methods build explicit models of the environment's dynamics, allowing for planning and decision-making to significantly enhance learning efficiency and reduce the number of physical interactions required.

One prominent application of MBRL in robotics is in low-level control tasks [48, 49]. These tasks involve controlling individual components of a robot, such as actuators, sensors, and other mechanical parts. By accurately modeling the dynamics of these components, MBRL can optimize control strategies that improve performance and efficiency. For instance, in robotic manipulation, MBRL can help control the movements of robotic arms precisely to achieve desired positions and orientations with minimal error [50].

MBRL has shown promise in optimizing energy efficiency and operational longevity in small-scale robotic systems, such as micro-robots or modular robots [22]. By predicting the outcomes of different control actions, these systems can make informed decisions that prolong battery life and enhance overall system stability [51].

Despite these applications, MBRL faces significant challenges that limit its broader application in robotics. The primary issue is the difficulty in constructing accurate models of complex, high-dimensional environments [52, 53]. In real-world scenarios, especially in robotics, the state space can be continuous and vast, making it hard to accurately capture all the necessary dynamics. Even if an accurate model can be constructed, the computational overhead associated with model learning and planning is often prohibitively high. Furthermore, the areas where MBRL has shown outstanding performance, such as low-level control, represent only a minimal subset of the vast potential applications in robotics. MBRL struggles to provide practical solutions for higher-level control strategies due to the abovementioned limitations. These challenges make MBRL less advantageous for many real-world robotic applications, where the need for real-time performance and adaptability is crucial, and thus, its potential impact is limited.

2.2.2 Advancements in Model-Free Reinforcement Learning for Robotics

Given the limitations of MBRL, significant advancements have been made in model-free reinforcement learning (RL) to address the challenges of continuous action spaces in robotics. RL was predominantly limited to discrete action spaces before the advent of algorithms like A2C and A3C [54, 41]. This limitation made RL algorithms unsuitable for real-world applications, including robotic control and complex video games, which often require continuous actions.

The Advantage Actor-Critic (A2C) and Asynchronous Advantage Actor-Critic (A3C) Algorithms marked a significant breakthrough in the field of RL by introducing policy gradient methods [27]. These methods enable RL to operate in continuous action spaces, enabling it to apply RL to complex control tasks [55].

A2C and A3C combine the benefits of value-based methods and policy-based methods. The key idea behind these algorithms is to use an actor-critic framework where the actor learns the policy (i.e., the mapping from states to actions), and the critic estimates the value function (i.e., the expected return of a state). The advantage function reduces the variance in policy updates, leading to more stable learning.

In A2C, the algorithm synchronously updates multiple environment instances, allowing more efficient use of computational resources. On the other hand, A3C operates asynchronously, running multiple copies of the agent in parallel and updating a global model. This parallelism speeds up learning and helps decorate the updates, leading to more robust learning.

The introduction of A2C and A3C had profound implications. These algorithms demonstrated that RL could be effectively used for robotic control by handling continuous actions, paving the way for RL to be applied to a broader range of tasks, including autonomous driving, robotic manipulation [56, 57], and sophisticated video games [58, 59].

2.2.3 Actor-Critic Based RL Applications in Robotics

Deep Deterministic Policy Gradient (DDPG) [60] was one of the early advancements in applying RL to continuous action spaces. DDPG combines the actor-critic framework with deterministic policy gradients, enabling efficient learning in high-dimensional spaces. However, DDPG is sensitive to hyperparameters and often struggles with stability and robustness issues. Twin Delayed Deep Deterministic Policy Gradient (TD3) [61] was introduced to address these challenges. TD3 incorporates improvements such as delayed policy updates and twin Q-networks to reduce overestimation bias and improve training stability.

Following DDPG, Proximal Policy Optimization (PPO) [44] emerged as a robust and efficient algorithm for robotic control. PPO optimizes the policy by ensuring that updates are minor and conservative, thus maintaining stability during training. It is particularly notable for its ability to perform well without extensive hyperparameter tuning, simplifying its application in various robotic tasks. PPO has been applied to robotic manipulation and navigation, demonstrating robust performance in functions such as object grasping [62, 63] and safe navigation [64, 65] in dynamic environments.

These RL algorithms have been applied to various robotic tasks, demonstrating their versatility and effectiveness. For example, DDPG and TD3 have been employed in tasks requiring precise control, such as robotic arm manipulation and autonomous vehicle control [66, 67]. These algorithms have enabled robots to perform tasks with high precision and adaptability, although they often require careful tuning of hyperparameters to achieve optimal performance. For instance, DDPG was effectively used in the Open Racing Car Simulator (TORCS) to train autonomous driving agents to learn complex driving behaviors and adapt to various driving conditions [68].

PPO has been utilized in robotic manipulation and navigation, providing robust performance in tasks such as object grasping and safe navigation in dynamic environments. For instance, PPO was applied in the Duckietown platform, where it was employed to train self-driving cars to navigate complex urban environments. This platform highlights PPO's robustness in handling real-world challenges in autonomous driving [69].

Despite their successes, these algorithms also faced significant challenges. One major limitation is their sensitivity to hyperparameters, leading to unstable training and suboptimal performance if not properly tuned [70, 71]. Additionally, the reliance on exploration parameters (like ϵ in PPO, DDPG, and TD3) leads to higher sensitivity to hyperparameters, often requiring extensive experimentation to find suitable settings.

In summary, while DDPG, TD3, and PPO significantly advanced the application of RL in robotics, they highlighted the need for more robust and less sensitive algorithms. The introduction of Soft Actor-Critic (SAC) [45, 72] addressed many of these issues by providing a more stable and efficient approach to learning in continuous action spaces, paving the way for more reliable and scalable applications in robotics.

The introduction of SAC brought a significant improvement in RL by incorporating the concept of entropy into the policy optimization process. This innovation allowed SAC to handle continuous action spaces more effectively, providing greater stability and robustness than previous algorithms such as PPO and DDPG [73]. SAC's unique approach involves maximizing both the expected reward and the entropy of the policy, which encourages exploration and results in more robust policies that are less sensitive to hyperparameters.

SAC optimizes a trade-off between the reward and the policy's entropy. This approach ensures that the learned policies are not only effective in achieving high rewards but also robust to variations and uncertainties in the environment. Additionally, SAC includes a temperature parameter that adjusts the importance of the entropy term. This parameter can be automatically tuned based on the learning environment, reducing the need for manual hyperparameter tuning and enhancing the algorithm's stability and performance.

SAC has been successfully applied to a variety of real-world robotic tasks, demonstrating its robustness and efficiency. One example is the SAC-PID control for mobile robots. Traditional PID control [74] struggles in environments that cannot be accurately modeled and where conditions change in real time. To address this, a hierarchical structure was developed, combining an upper controller based on SAC with a lower controller based on an incremental PID controller. SAC outputs optimal parameters for the PID controllers to compensate for real-time errors between the path and the mobile robot. This method was tested on Gazebo [75] and a real mecanum mobile robot, demonstrating strong robustness, generalization, and real-time performance compared to fuzzy PID control [76].

Another application of SAC is in robotic manipulation through Equivariant SAC. Recent advancements have shown that equivariant neural network models can improve sample efficiency in RL tasks. In this context, Equivariant SAC was applied to robotic manipulation, enabling policies to be learned entirely on physical robotic systems without the need for models, simulators, or offline datasets. This approach allowed for learning non-trivial manipulation tasks through on-robot experiences in less than an hour, highlighting the efficiency and effectiveness of Equivariant SAC in real-world applications [77].

SAC has also been adapted to control soft robots, which present unique control challenges due to their numerous passive degrees of freedom. In particular, SAC has been effectively used to control a continuum robot arm that operates with nine flexible rubber artificial muscles. The proposed Ensembled Light-weight model-Free reinforcement learning Network (ELFNet) demonstrated that SAC could learn control policies for this soft robot both in simulations and real-world settings. This method proved more suitable for compliant robots due to better sample efficiency and significant performance improvements, especially with a large number of passive DoFs [78].

In the context of mobile robot navigation, SAC has been compared with DDPG. Both techniques were used to control a robot to reach a target in an environment using inputs such as laser range findings, previous velocities, and the relative position and angle to the target. SAC demonstrated superior performance in navigating mobile robots compared to DDPG, showcasing its effectiveness in handling navigation tasks and providing better overall performance in avoiding collisions and reaching targets [79].

These examples illustrate SAC's versatility and effectiveness in various robotic tasks. The introduction of SAC has addressed many of the limitations of previous RL algorithms by providing a more stable and efficient approach to learning in continuous action spaces, paving the way for more reliable and scalable applications in robotics. Despite these advancements, there remain inherent challenges in RL when tackling complex, long-horizon decision-making tasks that involve large state and action spaces.

2.3 Adaptive Frequency Control in Reinforcement Learning

2.3.1 Limitations of These RL Algorithms with Fixed Control Rate

Despite the significant advancements and numerous applications of RL algorithms in robotics, these methods generally rely on a fixed frequency assumption for translating the discretized time into real continuous time. This fixed control frequency is rarely optimal, as the most appropriate control frequency should ideally be task-dependent. The Markov Decision Process (MDP) [11], a fundamental framework in RL, has inherent limitations when applied to real-world tasks due to this fixed frequency constraint.

The MDP framework assumes that the state transitions and actions are independent of the execution period. It does not concern itself with the duration between actions, meaning this period can be very short or very long. This oversight is a significant limitation when dealing with real-world robotic tasks, where the timing of actions is crucial for task performance. Specifically, the traditional RL framework has several drawbacks.

Tasks that require precise timing and coordination, such as robotic manipulation or autonomous driving, may experience degraded performance under a fixed frequency regime. For instance, tasks requiring rapid responses might suffer from delayed actions, while tasks that benefit from slower, deliberate actions might waste computational resources with unnecessary updates. This misalignment can lead to suboptimal performance and increased energy consumption. Moreover, maintaining a high control frequency, regardless of task requirements, leads to unnecessary energy expenditure. In battery-operated robotic systems, this can significantly reduce operational time and overall system efficiency. Fixed control frequencies do not adapt well to changing environmental conditions. For example, a mobile robot navigating through varying terrains or an industrial robot adjusting to different payloads requires adaptive control frequencies to maintain optimal performance and stability.

These examples underscore the necessity for developing VTS-RL algorithms that can dynamically adjust their control frequencies based on the task and environmental demands, leading to more efficient and effective robotic systems. Amin et al. investigate the impact of control frequency on RL performance, particularly in high-dimensional continuous control tasks with sparse rewards. The study conducts experiments using PolyRL in combination with DDPG and SAC algorithms across various MuJoCo environments such as Hopper-v2, HalfCheetah-v2, and Ant-v2 [80]. The results show optimizing control frequency is crucial for maximizing the effectiveness of reinforcement learning algorithms in complex environments, highlighting the importance of balancing exploration and exploitation through appropriate frequency adjustments [12].

Park et al. explore the effects of control frequency on RL performance by introducing a time discretization-invariant method called Safe Action Repetition (SAR) for policy gradient (PG) methods. The study reveals that reinforcement learning performance is highly sensitive to the discretization time scale (δ). When δ is too high, it limits the agent's ability to make fine-grained decisions, while too low a δ causes the variance of the PG estimator to explode and hinders exploration. As δ approaches zero, PG methods fail due to the increased variance of the gradient estimator, limited exploration ranges, and the infinite number of required decision steps [13].

Yann et al. investigate the effects of control frequency on RL performance, particularly in environments with random delays. The study highlights that traditional reinforcement learning algorithms struggle with delays because they complicate the credit assignment process. This problem is exacerbated at higher control frequencies, where frequent decision-making can lead to increased variance in the gradient estimator and hinder exploration. Conversely, lower control frequencies limit the agent's ability to make fine-grained decisions. They demonstrate that higher control frequencies can improve learning efficiency and task performance by effectively allowing the agent to receive and utilize informative reinforcement signals. However, they also note that the optimal control frequency must balance the need for fine-grained decision-making with the risk of increased variance and limited exploration [81].

In summary, although the MDP itself does not impose strict requirements on the control frequency for RL, its impact on RL performance is significant. Unlike discrete space RL with finite dimensions, continuous space RL is very sensitive to control frequency due to its high dimensionality and sparse rewards. To apply continuous space RL in real-world scenarios, such as robotic control, a suboptimal fixed control frequency that does not account for task requirements is unacceptable. Therefore, it is crucial to research and develop RL algorithms with dynamic control frequencies.

2.3.2 Research on RL Variable Control Frequency

The Continuous-Time Continuous-Options (CTCO) [82] framework introduces a novel approach to reinforcement learning by utilizing continuous-time decision-making and flexible option durations. Similar to ours, this methodology allows agents to make decisions with variable durations, providing a smooth and adaptable control mechanism that is particularly effective in continuous control tasks. By dynamically adjusting the decision frequency, CTCO enhances exploration and maintains robustness across different interaction frequencies, which traditional fixed-frequency RL algorithms often struggle with.

CTCO's flexibility in decision-making makes it highly suitable for real-world applications, as demonstrated in simulated continuous control tasks and real-world robotic experiments. The framework's ability to adapt interaction frequencies according to the task requirements leads to better exploration and more efficient learning, especially in environments with sparse rewards. By incorporating the soft actor-critic (SAC) algorithm and entropy regularization, CTCO promotes efficient exploration, ensuring that the learning process remains robust and stable.

However, despite its innovative approach, CTCO has some limitations. One significant challenge is tuning several hyperparameters, especially the dynamic adjustment of the discount factor γ using the hyperparameter τ. This dynamic discount factor makes CTCO's performance highly sensitive to environmental and task changes. CTCO tends to use a smaller γ throughout training, which can impair its ability for long-term planning and, consequently, its overall performance. The sensitivity to these hyperparameters means that extensive experimentation may be required to achieve optimal performance, which can be resource-intensive. Additionally, CTCO does not inherently optimize for task duration, which can result in longer task completion times. This aspect could be a drawback in time-sensitive applications where rapid decision-making and execution are crucial.

In summary, CTCO represents a significant advancement in reinforcement learning by enabling continuous-time decision-making and flexible option durations. Its strengths lie in improved exploration and robustness, making it well-suited for complex continuous control tasks. However, the framework's reliance on hyperparameter tuning, particularly the dynamic adjustment of the discount factor, and its potential for prolonged task durations highlight areas for further improvement.

Besides CTCO, to the best of our knowledge, there are no other RL algorithms within the community that directly adjust control frequencies. Most other works achieve changes in control frequency indirectly through various methods such as "learning to repeat [83]" or "sleep [84]" techniques. In the following paragraphs, we will provide a concise overview of these alternative approaches and how they achieve variable control frequencies.

Sahil et al. introduce the Fine Grained Action Repetition (FiGAR) framework [83]. This framework allows RL agents to decide both the action to execute and the duration for which it should be repeated, introducing temporal abstractions in the action space and enhancing the efficiency and performance of deep reinforcement learning algorithms. It allows the agent to learn temporal abstractions by predicting the number of time steps an action should be repeated. This leads to more efficient exploration and better policy learning, as the agent can effectively manage actions over varying time scales.

However, FiGAR's reliance on action repetition can sometimes lead to inefficiencies in scenarios where rapid adaptation to changing environments is crucial. While beneficial for stable environments, the fixed repetition duration might not be ideal for highly dynamic or unpredictable settings. In such cases, the agent might continue repeating an action when a quick change is necessary, potentially resulting in suboptimal performance.

Moreover, the approach of action repetition in FiGAR does not fundamentally improve data efficiency for computation-constrained platforms like robots. In simulations, executing a 10-second action once may be equivalent to executing a 1-second action ten times in terms of the outcome, but the latter demands ten times the computational resources. Therefore, altering control frequency in this manner does not reduce the computational load from a physical standpoint.

In summary, FiGAR represents a significant advancement in reinforcement learning by introducing the concept of fine-grained action repetition. This approach addresses the limitation of fixed action durations in traditional RL algorithms, offering a flexible and efficient solution for various control tasks. However, the framework's limited improvement in data efficiency on computation-constrained platforms makes the variable frequency approach lose its most important significance.

The concept of "learning to repeat" has been explored and expanded in two critical studies. Both studies focus on adapting control frequencies to optimize the execution of actions in reinforcement learning (RL) tasks.

Metelli et al. introduce the notion of action persistence in their work [85], where actions are repeated for a fixed number of steps to modify control frequency. Their proposed algorithm, Persistent Fitted Q-Iteration (PFQI), extends the traditional Fitted Q-Iteration (FQI) to learn the optimal value function for a given persistence level. This method balances the trade-off between expanding the policy space and managing sample complexity. Their theoretical analysis shows that increasing the control frequency (reducing action persistence) can make the learning problem more challenging due to higher sample complexity. Empirical results from benchmark domains demonstrate that appropriate persistence levels can significantly improve learning efficiency and policy performance.

In contrast, the paper by Lee et al. addresses the challenge of controlling multiple decision variables with different frequencies [86], formalized as factored-action Markov Decision Processes (FA-MDPs). They propose the Action-Persistent Policy Iteration (AP-PI) algorithm for optimizing c-persistent policies, ensuring convergence with minimal complexity increase. They introduce the Action-Persistent Actor-Critic (AP-AC) algorithm for high-dimensional tasks, which leverages neural networks to optimize policies for multiple control frequencies. Their experimental results show that AP-AC outperforms several baseline algorithms in continuous control tasks and traffic control simulations, demonstrating its efficiency and effectiveness in handling tasks with varying control frequencies.

However, both frameworks face notable challenges. PFQI's reliance on balancing policy space and sample complexity can complicate the learning process, particularly in dynamic settings. AP-AC's increased sample complexity and the implementation challenges associated with sophisticated neural network architectures present additional difficulties.

Chen et al. introduce a novel approach to address the challenges of cooperative charging in Wireless Rechargeable Sensor Networks (WRSNs) [84]. They identify that traditional Multi-Agent Reinforcement Learning (MARL) algorithms struggle in cooperative charging because they operate on fixed-length time steps. This rigidity does not accommodate the varying durations of different charging actions, leading to inefficient and suboptimal performance. To overcome this, the authors propose the VarLenMARL framework, which allows each MC to complete actions within variable-length time steps using the "sleep" method. The VarLenMARL framework incorporates a synchronized delay-tolerant trajectory collection mechanism to maintain temporal correctness and ensure the accurate calculation of global rewards.

One critical issue is the computational overhead of maintaining synchronized time-steps and delay-tolerant mechanisms, which could be challenging in large-scale deployments [87, 88]. Additionally, the framework still relies on periodic system checks to manage the synchronization, which may not eliminate computational loads [89]. Although the variable-length time-step mechanism mitigates some inefficiencies, it does not fundamentally alter the necessity for frequent synchronization and system checks, which remains a computational burden.

Facing these challenges, as described in section 4.1, we aimed to develop a VTS-RL framework to optimize both energy and time consumption. Despite our advances in optimizing both time and energy metrics, our approach diverges significantly from Hierarchical Reinforcement Learning (HRL) [90, 91]. Unlike HRL, which decomposes tasks into hierarchical levels with separate reward structures, we integrate multiple reward metrics into a single reward function to ensure user-friendliness and ease of implementation. This fundamental difference shapes the design and application of our VTS-RL framework compared to HRL.

2.4 Other Related Theoretical Works

2.4.1 Reactive Programming

Reactive programming is a paradigm focused on asynchronous data streams and change propagation [92]. It facilitates the development of systems that react to a sequence of inputs or events over time, making it particularly suitable for robotics control, where real-time responses to sensor inputs and environmental changes are crucial [93, 94].

Reactive programming centers around the concept of data streams and the propagation of changes. In a reactive system, the basic building blocks are observables and observers. Observables represent data streams that can emit items over time, while observers subscribe to these observables to react to emitted items.

Key principles include declarative code, asynchronous data flow, and event-driven architectures. Declarative code allows developers to specify what should happen rather than how it should happen, leading to more concise and readable code [95, 96]. Asynchronous data flow inherently supports asynchronous data handling, which is vital for robotics where operations often need to occur in parallel without blocking. Event-driven architectures are naturally suited for systems that need to respond to a wide range of stimuli from the robot's environment.

In the context of robotics, reactive programming provides a robust framework for handling the complex, asynchronous interactions between a robot and its environment. The primary advantages include real-time processing, scalability and flexibility, and fault tolerance [97, 98, 99]. Reactive programming ensures that robots can process sensor data and make decisions in real time. This is critical for applications like autonomous navigation, obstacle avoidance, and manipulation tasks. Reactive systems can scale efficiently with the increasing complexity and number of sensors and actuators in modern robots. Additionally, the reactive approach can improve the robustness of robotic systems by isolating and managing errors more effectively.

Reactive programming frameworks have been instrumental in developing sophisticated navigation systems for robots, particularly in environments that are dynamic and unpredictable [100]. These systems can process real-time data from various sensors, such as LIDAR [101], cameras [102], and ultrasonic sensors [103], to create a detailed map of the surroundings. By continuously updating this map, robots can effectively navigate through complex terrains, avoid obstacles, and find the most efficient paths to their destinations. One of the significant advantages of using reactive programming in this context is its ability to handle asynchronous events seamlessly [104]. For example, if a new obstacle suddenly appears in the robot's path, the reactive system can quickly process this information and adjust the robot's trajectory without any noticeable delay. This ensures that the robot can operate smoothly even in rapidly changing environments, enhancing its reliability and safety.

Robots are equipped with an array of sensors that provide critical information about their environment [105, 106]. Integrating data from these multiple sources in real-time is a complex task that can significantly benefit from reactive programming. By treating sensor outputs as continuous data streams, reactive programming allows for efficient and synchronized data processing. This approach enables the robot to maintain comprehensive situational awareness [107]. For instance, in a warehouse automation scenario, a robot might need to simultaneously monitor its position, the location of other robots, inventory levels, and environmental conditions such as temperature and humidity. Reactive programming frameworks can merge these diverse data streams into a coherent picture, enabling the robot to make informed decisions quickly. This level of integration is crucial for tasks that require high precision and coordination, such as picking and placing items, navigating tight spaces, and avoiding collisions with other robots or humans.

One of the hallmarks of reactive programming in robotics is its ability to facilitate real-time decision making [108, 109]. Robots often operate in environments where rapid responses to changes are essential. For example, in an agricultural setting, a robotic harvester needs to make split-second decisions based on the ripeness of fruits, the presence of obstacles, and the optimal path to minimize travel time and maximize efficiency. Reactive programming supports this by allowing the robot to process incoming data and update its actions almost instantaneously. This is achieved through the propagation of changes in data streams, where any change in input data triggers a cascade of updates throughout the system. As a result, the robot can continuously adapt to new information, making its operation more fluid and responsive [110]. This capability is particularly beneficial in applications requiring high levels of autonomy and adaptability, such as search and rescue missions, where the robot needs to navigate through debris and identify survivors under challenging conditions.

In summary, reactive programming significantly enhances the capabilities of robotic control systems by providing robust solutions for autonomous navigation, sensor data integration, and real-time decision making. By leveraging the principles of reactive programming, developers can create more adaptable, responsive, and efficient robots capable of performing complex tasks in diverse and dynamic environments. The continued evolution and adoption of these frameworks hold great promise for the future of robotics, paving the way for more advanced and intelligent robotic systems.

2.4.2 Hierarchical Reinforcement Learning (HRL)

Hierarchical Reinforcement Learning (HRL) has emerged as a pivotal approach in addressing the complexities of long-horizon decision-making tasks by breaking them down into simpler, more manageable subtasks. This survey provides an extensive overview of the advancements in HRL, focusing on the diverse methodologies developed to tackle challenges such as hierarchical policy learning, subtask discovery, transfer learning, and multi-agent learning.

HRL fundamentally enhances traditional RL by incorporating a hierarchical structure where high-level policies determine the optimal sequence of subtasks, and each subtask may itself be governed by lower-level policies. This decomposition is crucial in managing large state and action spaces, making HRL particularly effective in long-horizon tasks where standard RL algorithms struggle. The concept of temporal abstraction within HRL, where actions are represented over varying timescales, facilitates more efficient credit assignment and structured exploration, thereby improving learning efficiency and performance.

Pateria et al. [111] categorize the HRL approaches. It delineates the methods based on whether they involve subtask discovery, their application in single or multi-agent environments, and their focus on single or multiple tasks. This categorization helps researchers understand the progression and intersections of different HRL techniques, providing a clear framework.

HRL's practical applications, emphasizing its utility in complex domains such as continuous control [112], game playing [113], and robotic manipulation [114], where HRL has demonstrated significant performance improvements over traditional RL methods.

Our work also focuses on optimizing performance in multiple directions. However, our approach integrates various reward metrics into a *single reward function*, making it more adaptable to different RL algorithms and enhancing user-friendliness. This design philosophy fundamentally differs from the HRL principles.

In our second 4, third 5, and fourth 4 papers, we have explored dynamic changes in reward structures. Despite our commitment to user-friendliness, these dynamic adjustments have inevitably positioned us along the Pareto frontier [115, 116]. The following is a brief review of Pareto optimality and its applications in reinforcement learning.

2.4.3 Pareto Efficiency

Pareto efficiency is a fundamental concept in economics and optimization theory [117]. It represents a state where no individual can be better off without making someone worse. In other words, a Pareto efficient allocation is one where resources are allocated in the most efficient manner, given the system's constraints [118]. This concept is crucial for understanding resource allocation and the efficiency of different economic systems.

The significance of Pareto efficiency extends beyond economics into various fields such as engineering, public policy, and multi-objective optimization. In these contexts, it assesses the trade-offs between different objectives. For instance, in multi-objective optimization, solutions are evaluated based on their ability to improve one objective without degrading another. This allows for a more holistic approach to decision-making, where multiple criteria are considered simultaneously.

In practical applications, achieving Pareto efficiency can be challenging due to the complexity of balancing multiple objectives and competing interests. Various methods have been developed to identify Pareto-efficient solutions, including mathematical programming, evolutionary algorithms, and game theory [119]. These methods help find solutions that are not only efficient but also equitable, ensuring a fair distribution of resources or benefits among stakeholders.

Despite its widespread use and theoretical importance, Pareto efficiency has limitations. It does not account for the distribution of resources among individuals, meaning that a Pareto efficient outcome may still be inequitable. Additionally, it assumes that all preferences and utilities can be compared and measured, which is not always feasible in real-world scenarios. Therefore, while Pareto efficiency provides a valuable framework for understanding efficiency and trade-offs, it should be complemented with other considerations, such as equity and feasibility in practical applications [120].

2.4.4 Lyapunov Stability

Lyapunov stability is a cornerstone concept in studying dynamic systems and control theory [14]. It provides a mathematical framework to analyze the stability of equilibrium points in differential equations and dynamical systems. Specifically, Lyapunov stability examines whether small perturbations or deviations from an equilibrium state will decay over time, ensuring the system returns to equilibrium.

The core of Lyapunov stability theory lies in the construction of Lyapunov functions, scalar functions that measure the energy or potential of a system relative to its equilibrium point. A system is considered Lyapunov stable if a suitable Lyapunov function can be found that decreases over time, indicating that the system's trajectories will remain close to the equilibrium for all time. This approach is compelling because it does not require solving the differential equations explicitly; instead, it provides a way to infer stability from the properties of the Lyapunov function.

Lyapunov's methods are extensively used in various fields, including engineering (for example, applications of RL in power dispatch, etc.) [121, 122, 123], physics [124], and economics [125], to ensure the robustness and reliability of systems. In control theory, for instance, Lyapunov stability is employed to design controllers that stabilize systems in the presence of uncertainties and external disturbances [126]. Lyapunov's direct method is also widely used to prove the global stability of nonlinear systems by constructing a global Lyapunov function.

Despite its broad applicability, constructing an appropriate Lyapunov function can be challenging, especially for complex and high-dimensional systems. Researchers continue to develop new techniques and computational tools to simplify this process and extend the applicability of Lyapunov stability analysis. Nonetheless, Lyapunov stability remains a fundamental tool for understanding and ensuring the stability of dynamic systems, providing a robust theoretical foundation for practical applications in various scientific and engineering domains [126].

Our work demonstrates the rationality of integrating dynamic time control with Lyapunov models in our VTS-RL approach, particularly in robotic systems. Dynamic time control enhances flexibility by adjusting control actions based on the system's current state and environmental conditions, addressing the nonlinearity and time-variant dynamics typical in robotics. This adaptability ensures optimal performance through efficient resource utilization and effective response to varying situations.

Furthermore, Lyapunov models provide robust stability guarantees by employing a scalar function that decreases over time, ensuring system convergence to a desired equilibrium. By incorporating Lyapunov stability into the reinforcement learning framework, we ensure that the learning process remains stable, thereby preventing erratic behavior and potential system failure. This integration improves efficiency and performance and enhances robustness against uncertainties and disturbances, making it a highly effective control strategy for applications requiring high reliability and performance.

These studies collectively highlight the ongoing efforts to integrate variable action durations and adaptive control mechanisms into RL. While some good work has been done, it is clear that we are still in the early stages of effectively applying variable control frequencies and repetitive behaviors in practical applications. The potential of these innovations lies in their ability to enhance RL's applicability in real-world scenarios by improving learning efficiency and generalization to new tasks.

However, further research is essential to address the computational challenges and optimize these methods for dynamic and resource-constrained environments. Our work focuses on developing more efficient frameworks and reducing computational burdens. This is crucial for successfully deploying these advanced RL techniques in diverse and complex real-world applications, ranging from video games to robotic control. The VTS-RL method lacks comprehensive theoretical support, precise definitions, and practical implementations.

CHAPTER 3 ARTICLE 1: DEPLOYABLE REINFORCEMNENT LEARNING WITH VARIABLE CONTROL RATE

Preface: Deploying reinforcement learning (RL) controllers in real-world robots poses significant challenges, mainly due to the reliance on fixed-rate control strategies. This can lead to inefficiencies in computational and energy resources. This chapter addresses this issue by proposing a variable control rate approach integrated into the Soft Actor-Critic (SAC) algorithm, termed Soft Elastic Actor-Critic (SEAC). Our method allows the RL policy to determine both the action and the duration of the time step, enhancing adaptability and efficiency. We demonstrate the efficacy of SEAC through simulations, showing improved task completion times and reduced computational load compared to traditional fixed-rate policies. This approach holds promise for broader RL applications in resource-constrained robotic systems, providing a pathway towards more deployable and energy-efficient AI solutions.

Full Citation: Wang, Dong, and Giovanni Beltrame. "Deployable reinforcement learning with variable control rate." In Processing of the Advancement of Artificial Intelligence Conference, Deployable AI Workshop, 2024. Manuscript accepted for publication on December 12nd, 2023.

Deploying controllers trained with Reinforcement Learning (RL) on real robots can be challenging: RL relies on agents' policies being modeled as Markov Decision Processes (MDPs), which assume an inherently discrete passage of time. The use of MDPs results in that nearly all RL-based control systems employ a fixed-rate control strategy with a period (or time step) typically chosen based on the developer's experience or specific characteristics of the application environment. Unfortunately, the system should be controlled at the highest, worst-case frequency to ensure stability, which can demand significant computational and energy resources and hinder the deployability of the controller on onboard hardware. Adhering to the principles of reactive programming, we surmise that applying control actions *only when necessary* enables the use of simpler hardware and helps reduce energy consumption. We challenge the fixed frequency assumption by proposing a variant of RL with *variable control rate*. In this approach, the policy decides the action the agent should take as well as the duration of the time step associated with that action. In our new setting, we expand Soft Actor-Critic (SAC) to compute the optimal policy with a variable control rate, introducing the Soft Elastic Actor-Critic (SEAC) algorithm. We show the efficacy of SEAC through a proof-of-concept simulation driving an agent with Newtonian kinematics. Our experiments show higher average returns, shorter task completion times, and reduced computational resources when compared to fixed rate policies.

3.1 Introduction

Temporal aspects of reinforcement learning (RL), such as the duration of the execution of each action or the time needed for observations, are frequently overlooked. This oversight arises from the foundational hypothesis of the Markov Decision Process (MDP), which assumes the independence of each action undertaken by the agent [11]. As depicted in the top section of Figure 3.1, conventional RL primarily focuses on training an action policy, generally neglecting the intricacies of policy implementation. Some prior research approached the problem by splitting their control algorithm into two distinct components [127]: a learning part responsible for proposing an action policy, and a control part responsible for implementing the policy [128, 129, 130]. [1]

Translating action policies composed of discrete time steps into real-world applications generally means using a fixed control rate (e.g., 10 Hz). Practitioners typically choose the control rate based on their experience and the specific needs of each application, often without considering adaptability or responsiveness to changing environmental conditions. Imagine driving a car in a straight line in an extensive environment without obstacles: very few control actions are needed. On the contrary, driving in tight spaces with low tolerances and following complex paths can require a higher control frequency. Setting the control frequency is set to a fixed value requires it to be the worst-case (or an average if safety is not a concern) scenario that allows controllability of the system. In practical applications of reinforcement learning, especially in scenarios with constrained onboard computer resources, maintaining a consistently high fixed control rate can limit the availability of computing resources for other tasks and significantly increase energy consumption.

Furthermore, the inherent inertia of physical systems cannot be ignored, impacting the range of feasible actions. In such cases, an agent's control actions are closely tied to factors like velocity and mass, leading to considerably different outcomes when agents execute the same actions at different control rates.

Hence, applying RL directly to real-world scenarios can be challenging when the temporal dimension is not considered. The typical approach is to employ *a fast enough but fixed control rate that accommodates the worst-case scenario* for an application [130], resulting in suboptimal performance in most instances.

In this paper, we challenge RL's conventional fixed time step assumption, aiming to formulate faster and more energy-efficient policies. Our approach seamlessly integrates the temporal aspect

[1]Our paper is presented at Deployable AI Workshop at AAAI-2024

into the learning process. This modification can benefit onboard computers with limited resources so that deployed AI no longer consumes precious computing resources to compute unnecessary actions, effectively improving general system deployability. In our approach, the policy determines the following action and the duration of the next time step, making the entire learning process and applying policies *adaptive* to the specific demands of a given task. This paradigm shift follows the core principles of reactive programming [93]: as illustrated in the lower portion of Figure 3.1, in stark contrast to a strategy reliant on fixed execution times, adopting a dynamic execution time-based approach empowers the agent to achieve significant savings in terms of computational resources, energy consumption, and time expended. Moreover, our adaptive approach enables the integration of learning and control strategies, resulting in a unified system that enhances data efficiency and simplifies the pursuit of an optimal control strategy.

An immediate benefit of our approach is that the freed computational resources can be allocated to additional tasks, such as perception and communication, broadening the scope of RL applicability in resource-constrained robots. We view the variable control rate as promising for widely adopting RL in robotics.

3.2 Reinforcement Learning with A Fixed Control Rate

Before delving into the variable control rate RL, we provide a concise overview of fixed time step-based RL. A notable example of successful real-world reinforcement learning applications is Sony's autonomous racing car: Sony has effectively harnessed the synergy between reinforcement learning algorithms and a foundation of dynamic model knowledge to train AI racers that surpass human capabilities, resulting in remarkably impressive performance outcomes [131]. From a theoretical perspective, Li et al. [132] aimed to bolster the robustness of RL within non-linear systems, substantiating their advancements through simulations in a vehicular context. A shared characteristic among these studies is their dependence on a consistent control rate, typically 10 or 60 Hz. However, it is important to note that a successful strategy does not necessarily equate to an optimal one. As previously mentioned, time is critical in determining the system's performance, whether viewed from an application or theory perspective. The energy and time costs of completing a specific task determine an agent's level of general efficiency. A superior control strategy should minimize the presence of invalid instructions and ensure control actions are executed only when necessary. Hence, the duration of an individual action step should not be rigidly fixed; instead, it should vary based on the dynamic demands of the task.

We have designed a variable control rate reinforcement learning method to address the time issue. This approach trains the policy to predict the time for executing the action and the action itself, adapting to dynamic changes in demand. The goal is to overcome dynamic environment uncer-

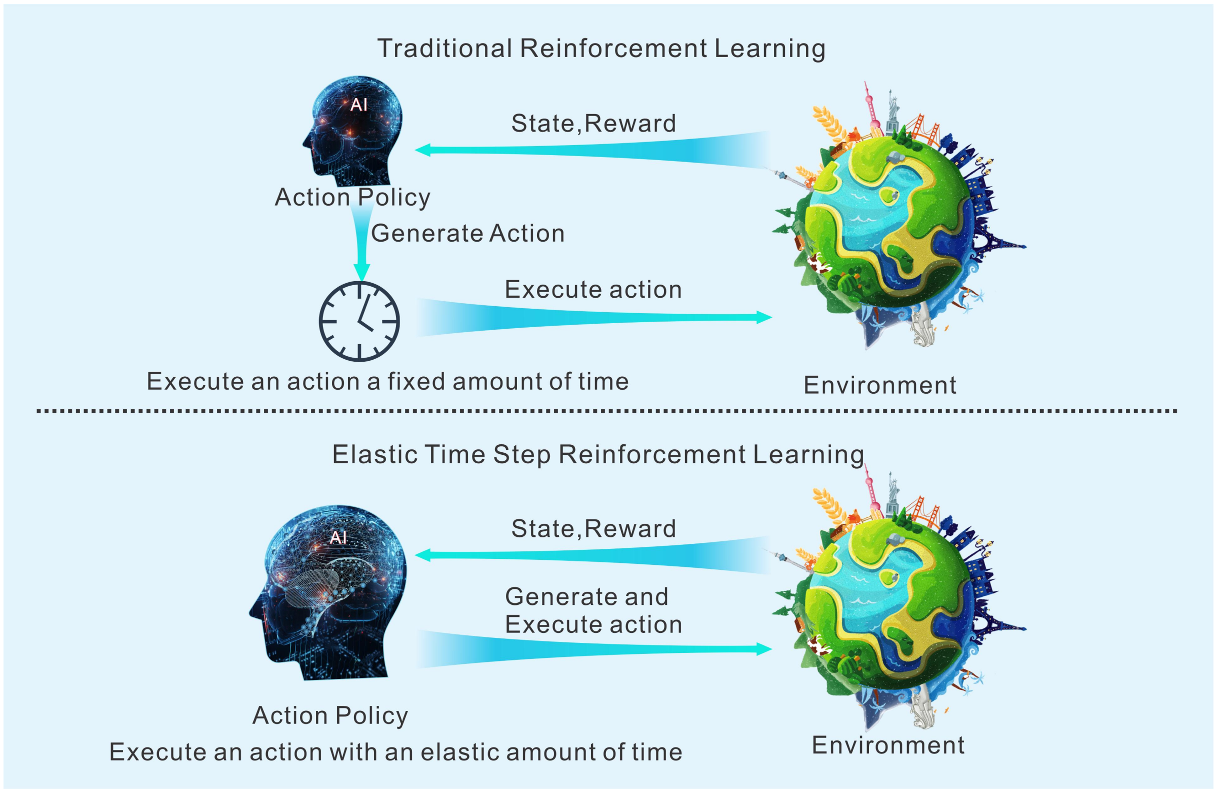

Traditional Reinforcement Learning
AI
State,Reward
Action Policy
Generate Action
Execute action
Execute an action a fixed amount of time
Environment
Elastic Time Step Reinforcement Learning
AI
State,Reward
Generate and
Execute action
Action Policy
Execute an action with an elastic amount of time
Environment

Figure 3.1 Comparing Elastic Time Step Reinforcement Learning and Traditional Reinforcement Learning

tainty and save the agent's computing resources. In a related study on changes in action execution time using reinforcement learning. Sharma et al. [83] introduced the concept of a learning-to-repeat strategy. Essentially, they found that when an agent performs the same action with the same execution time repeatedly, it can effectively alter the action execution time. However, their approach has limitations, particularly in simulating real-world scenarios. In simulations, executing a 10-second action and executing a 1-second action ten times may appear similar. Yet, their method doesn't account for potential changes in the agent's physical properties in natural environments. For instance, if the mass of the agent changes while it's performing a task, executing the same action simultaneously may not yield the same results. Moreover, the repeated execution of the same actions doesn't optimize computing resources. Allocating these resources to other tasks, like environment awareness, could significantly benefit onboard computers with limited computing capacity.

In addition, we also noticed that Chen et al. [84] changes the "control rate" in disguise by taking actions such as sleeping. However, it still needs a fixed control frequency to scan the agent's status to decide whether to take sleep action as the next step. High-frequency system status scanning and calculation also cannot effectively reduce the computing load. In addition to the previously mentioned scenarios, there exists a diverse range of time-sensitive reinforcement learning tasks spanning various domains. These tasks cover multiple fields, including robotics, electricity markets, and many others [133, 134, 135, 128]. However, using a fixed control rate is a common thread among these works and systems like robots, which often lack ample computing resources, can struggle to maintain a high and fixed control rate. The algorithm we designed can solve the problem of insufficient computing resources caused by maintaining too high a frequency, and it can be applied widely. It can train almost all RL strategies based on continuous control. For example, the problem of vehicles moving in a Newtonian dynamic environment; the classic robot arm problem: one assumes that the output of the strategy set is a force set. Suppose you want to quickly grab a fragile object, such as an egg, without considering other interferences, such as the material of the robot arm. In that case, the grip strength of the robot arm and direction are essential, and the duration of the force is even more critical. Beside the robotic related application, it can also be applied on Real-Time Strategy Applications, such as AI in gaming applications (Trackmania, Genshin Impact, and etc.). Especially those application scenarios deployed on smartphones with limited computing resources.

3.3 Reinforcement Learning with Variable Control Rate

A straightforward approach to variable time step duration is to monitor the completion of each execution action and dispatching the subsequent command. Indeed, in low-frequency control scenarios, there are typically no extensive demands for information delay or the overall time required to accomplish the task [games like Go or Chess, 136, 137]. However, in applications like robotics or autonomous driving, the required control frequency can vary from very high [7, 138, 6] to low depending on the state of the system. Following reactive programming principles[93], to control the system only when necessary we propose that the policy also outputs the duration of the current time step. Reducing the overall number of time steps conserves computational resources, reduces the agent's energy consumption, and enhances data efficiency.

Unfortunately, in most RL algorithms, such as Q learning [139] and the policy gradient algorithm [27], there is no concept of the action execution time, which is considered only in few works [140, 81]. When control frequency is taken into consideration, it is mostly related to specific control problems [141, 142], and assuming actions executed at a fixed rate.

We propose a reward policy incorporating the agent's energy consumption and the time taken to complete a task and extend Soft Actor-Critic [73] into the Soft Elastic Actor-Critic (SEAC) algorithm, detailed in the following.

It is worth noting that for using SEAC on a robot control, an appropriate control system would be necessary for a real-world implementation, e.g. a proportional-integral-derivative controller (PID) controller [143], an Extended Kalman filter (EKF)[144], and other essential elements. These components would need to be implemented to use SEAC into a real system environment. Nevertheless, we show that our system can indeed learn the duration of control steps and outperform established methods in a proof-of-concept implementation.

3.3.1 Energy and Time Concerned Reward Policy

As shown in Figure 3.2, our approach tackles a multi-objective optimization challenge, in contrast to conventional single-objective reinforcement learning reward strategies. We aim to achieve a predefined objective (metric 1) while minimizing energy consumption (metric 2) and time to complete the task (metric 3). To reduce the reward to a scalar, we introduce 3 weighting factors: α_t, α_ε, and α_τ, respectively assigned to our three metrics. It is important to note that we consider only the energy consumption associated with the computation of a time step (i.e. energy is linearly proportional to the number of steps) and not the energy consumption of the action itself (e.g moving a heavy object, taking a picture, etc.). Thus, our assessment of the agent's energy usage is solely based on the *computational load*.

At the same time, it is also necessary to minimize time consumption. Reducing the number of time steps does not necessarily mean reducing the overall time consumption under variable control rates. The MDP does not constrain the time the agent takes to complete a time step. In some extreme cases, for example, the agent took *an entire year* to meet *one step* only. Although it meets with the MDP, it is unacceptable for most application scenarios. Based on the above considerations, while we pursue the number of time-step minimization, we also pursue time-consuming minimization.

We assume that each action incurs a uniform energy consumption, denoted as ε. The time taken to execute an action is τ. In this context, the reward for one single step is represented by R, and the relationship can be expressed as follows:

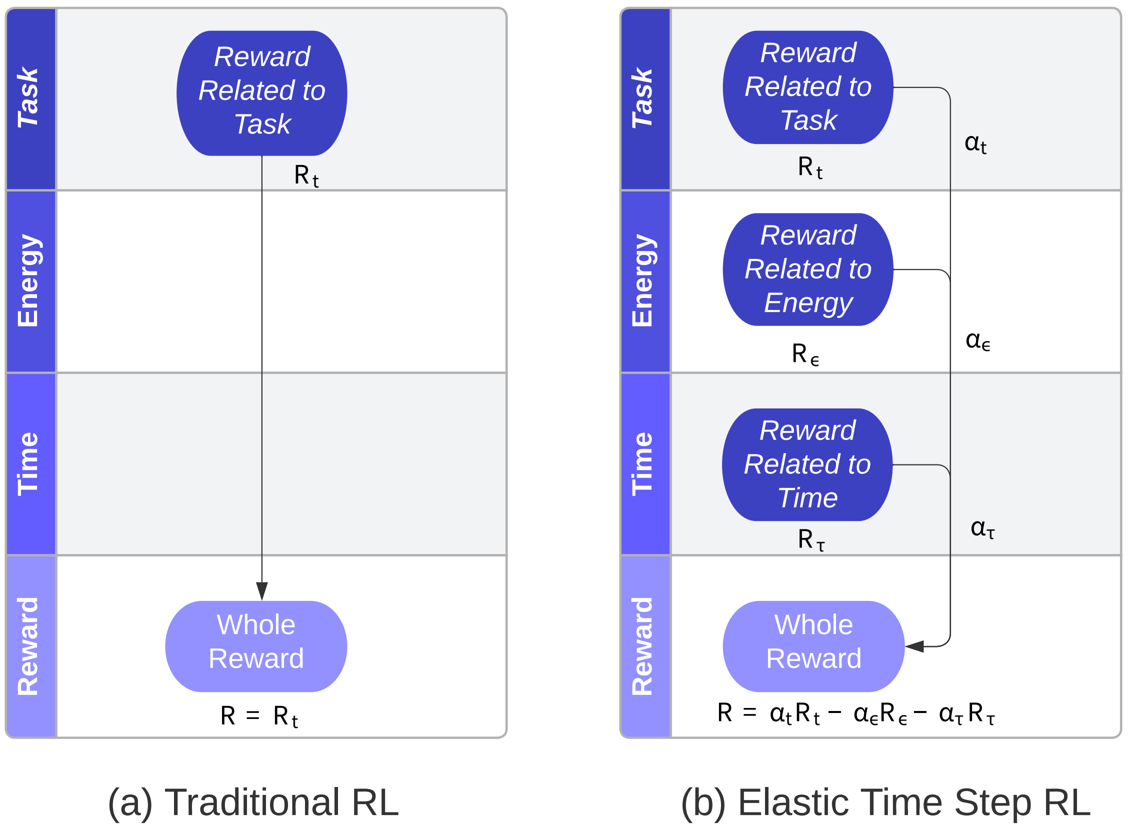


Figure 3.2 (a) the reward policy for traditional RL; (b) the reward policy for elastic time step RL

Reward Function

Definition 1. *The reward function is defined as:*

$$R = \alpha_t \cdot R_t - \alpha_\varepsilon \cdot R_\varepsilon - \alpha_\tau \cdot R_\tau$$

where

$R_t = r$, *r is the value determined by task setting;*

$R_\varepsilon = \varepsilon$, *$\varepsilon$ is the energy cost of one time step;*

$R_\tau = \tau$, *τ is the time taken to execute a time step.*

$\alpha_t, \alpha_\varepsilon, \alpha_\tau$ *are parametric weighting factors.*

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

While our algorithm aims to optimize multiple objectives, we take a simplified approach using a weighted strategy. Unlike the Hierarchical Reinforcement Learning (HRL) algorithm [90, 91], our method doesn't pursue Pareto optimality [115, 116] or involve multiple reward policies at different levels. Although our approach and HRL are designed to achieve various goals, they differ fundamentally in strategy settings. We've intentionally kept our strategy straightforward, making it user-friendly, computationally friendly, and adaptable for other algorithms. We've incorporated our reward policy into the SAC [73, 72] algorithm, extending it to SEAC. Importantly, this policy isn't exclusive to SAC – it can also be applied to other RL algorithms, such as PPO [44], TD3 [61], etc. This compatibility makes our algorithm easily referenceable and deployable to different applications.

3.3.2 The Verification Environment Design

Our SEAC verification work requires an environment that is non-discrete time and can reflect the full impact of time on action execution. This environment is not available within existing RL environments, we establish a test environment based on Gymnasium featuring variable action execution times, shown in Figure 3.3:

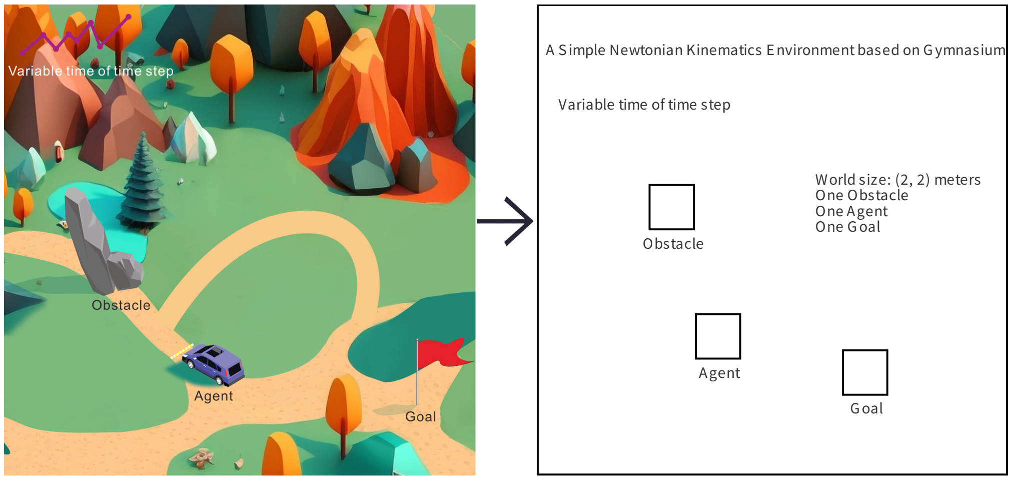

Variable time of time step
Obstacle
Agent
Goal
A Simple Newtonian Kinematics Environment based on Gymnasium
Variable time of time step
World size: (2, 2) meters
One Obstacle
One Agent
One Goal
Obstacle
Agent
Goal

Figure 3.3 A simple Newtonian Kinematics environment designed for verifying SEAC based on gymnasium.

To help readers grasp our algorithm better, we will delve into the design of the verification environment and provide a detailed explanation of how we implemented the algorithm in the upcoming section. This will give a clearer picture of the practical aspects of our work.

This environment is a continuous two-dimensional (2D) and consists of a starting point, a goal, and an obstacle. The task involves guiding an agent from the starting point to the goal while avoiding the obstacle. Upon resetting the environment, a new goal and obstacle are randomly generated. The conclusion of an epode is reached when the agent reaches the goal or encounters an obstacle. The agent is governed by Newton's laws of motion, including friction, details can be found in Defination 2 and Appendix A A.

The starting point of the agent is also randomly determined. If the goal or obstacle happen to be too close to the starting point, they are reset. Similarly, if the goal is too close to the location where the obstacle was generated, the obstacle's position is reset. This process continues until all three points are situated at least 0.05 meters apart from each other on a (2 x 2) meters map. Meanwhile, the maximum force in a single step is 100.0 Newton.

There are eleven dimensions of the state in the environment: the agent's position, the position of the obstacle, the position of the goal, the speed of the agent, the duration for last time step, and the force being applied for the last time step. It is worth noting that we are indeed using historical data (i.e. the movement and duration value of the preceding step), but refrain from using recurrent neural networks (RNN) [145, 146]. This decision stems from our concern that adopting recurrent architectures might deviate the overall reinforcement learning process from the Markov assumption [11, 147]: different decisions could arise from the same state due to the dynamic environment.

We consider 3 dimensions to the actions within the environment:

1. The time taken by the agent to execute the action;
2. the force being applied for the agent along the x axis;
3. the force being applied for the agent along the y axis.

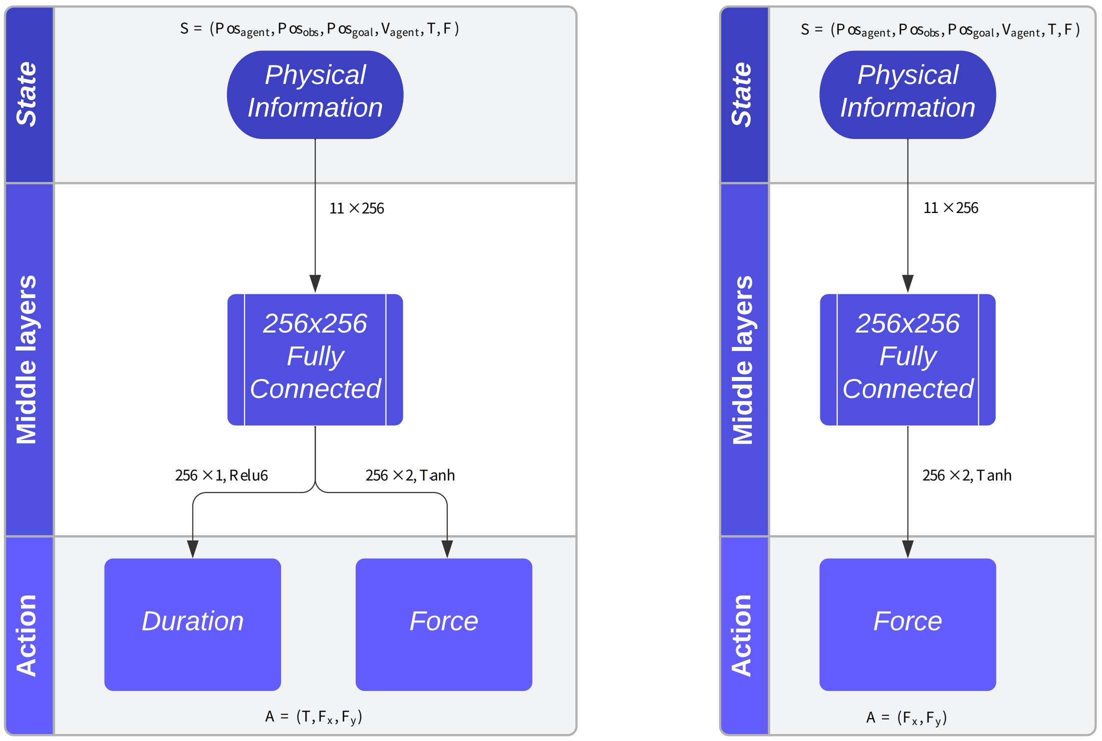
For instance, an action $a_t = (0.2, 50.0, -70.0)$ denotes that the agent is expected to apply 50.0 Newton force along the x axis and -70.0 Newton force along the y axis within 0.2 seconds. For more detailed environment settings, see Appendix A A.

3.4 Implementation of The SEAC

We expend the SAC algorithm to verify the validity of our Definition 1. Because we have not modified its loss functions, please refer to SAC [72] for its policy optimization.

We validate our reward strategy by taking the SAC algorithm and implementing fully connected neural networks [148] as both the actor and critic strategy. We assume the agent can explore the unknown environment as much as possible based on information entropy, giving a high probability that the agent can discover the optimal solution to complete the task.

As shown in Figure 3.4, we marked the specific values of S_t and A_t based on the environment we explained in Section 3-2 3.3.2, and the network structure of the Actor Policy. This will intuitively help readers realize the difference between SEAC and SAC. In contrast to the conventional RL, in addition to the environment's state data, we take the agent's action history values and bring them together to the form of states as the network's input, including the time spent performing the previous action (T), and the force in the last step (F_x and F_y). We regrad them as part of the state, because we believe position and velocity is insufficient to express the system's inertia. This setting can ensure that the Markov process can completely describe our environment, thereby ensuring the convergence of the RL algorithm. Additionally, our approach involves an extra component at the output: the duration time T for each action.



(a) SEAC (SAC with Elastic Time Steps) (b) SAC with Fixed Time Steps

Figure 3.4 (a) The SEAC ActorNetwork Architecture; (b) The SAC ActorNetwork Architecture.

When the Actor Network generates the action value A_t for the next step, the controller (Figure 3.4) will computes a range of control-related parameters (e.g., speed, acceleration, etc.) based on the action value and time. Under the context of our test environment, The related formulas are Newton Kinematics:

Newton Kinematics Formulas

Definition 2. *The Newton Kinematics formulas are:*

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

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

$$F_{aim} = mA;$$

$$F_{true} = F_{aim} \breve{} f_{friction};$$

$$\begin{aligned} f_{friction} &= \mu mg, If F_{aim} > f_{friction} \\ &\& V_{agent} \neq 0; \\ &= F_{aim}, If F_{aim} \leq f_{friction} \\ &\& V_{agent} = 0. \end{aligned}$$

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, m is the mass of the agent, μ is the friction coefficient, and g is the acceleration of gravity.

Through the duration and force generated by the policy, we can know the acceleration required to complete the policy and then calculate the speed. Then, we can compute the movement. However, due to friction, the actual acceleration will be inconsistent with the aimed acceleration, and the movement of the agent will be affected by Newtonian kinematics. Ultimately, the agent incorporates these actionable parameters into the environment, generating a new state and reward. This process is iterated until the completion of the task.

Our objective is for the agent to learn the optimal execution time for each step independently. We need to ensure that time is not considered as a negative value. Consequently, diverging from the single $Tanh$ [149] output layer typical in traditional RL Actor Networks, we separate the Actor Network's output layer into two segments: we use $Tanh$ as the output activation for the action value, and $ReLu6$ [150] for the output activation related to the time value.

As Definition 1, the precise reward configuration for our environment are outlined in Table 3.1. The hyperparameters settings can be found in Appendix BF.

It is noteworthy that our design does not substantially increase the model size. The recorded size of the network model in Appendix A is a mere 282.5KB after training. In comparison, the model size of the SAC network, of equivalent dimensions, is 280.5KB after training. The SEAC model is only 2KB larger, representing approximately a *7 ‰* increase in storage demand. This meticulous consideration ensures the model's deployability.

Table 3.1 Reward Policy for The Simple Newtonian Kinematics Environment

Reward Policy		
Name	Value	Annotation
	100.0	Reach the goal
r	-100.0	Crash on an obstacle
	$-1.0 \cdot D_{goal}$	D_{goal}: distance to goal
ϵ	1.0	Computational energy (Joule)
α_t	1.0	Task gain factor
α_ϵ	1.0	Energy gain factor
α_τ	1.0	Time gain factor

3.5 Experimental Results

We conducted six experiments for each of the three RL algorithms, employing various parameters within the environment described in Section III3.3.2 [2]. These experiments were conducted on a machine equipped with an Intel Core i5-13600K CPU and an NVIDIA RTX 4070 GPU, running Ubuntu 20.04. Subsequently, we selected the best-performing policy for each of these three algorithms to draw the graphs in Figures 3.5–3.8.

The frequency range for action execution spans from 1 to 100 Hz, and the agent's force value ranges from -100 to 100 Newton. We compared our results with the original SAC [72] and PPO [44] algorithms, both employing a fixed action execution frequency of 5.0 Hz.

We use the conventional average return graph and record the average time cost per task to clearly and intuitively represent our approach's performance. Furthermore, we generate a chart illustrating how the variable control rate policy works in action execution frequency for four tasks with the SEAC model. Finally, we draw a raincloud graph to visualize the disparities in energy costs among these three RL algorithms for one hundred different tasks.

Appendix BF provides all hyperparameter settings and implementation details. The average return results of all algorithms are shown in Figure 3.5, and their time-consuming results are shown in Figure 3.6:

Figure 3.5 and Figure 3.6 show that SEAC surpasses the baselines in terms of average return and time efficiency. The amount of data that SEAC needs to be trained has one more dimension than SAC and PPO (duration). So its training speed is not as fast as the above two at the beginning. However, when SEAC gradually reduces the amount of data by reducing the number of steps, its data efficiency improves significantly, starting from around 300,000 steps and finally converging before

[2]Our code is publicly available at Github: https://github.com/alpaficia/SEAC_Pytorch_release

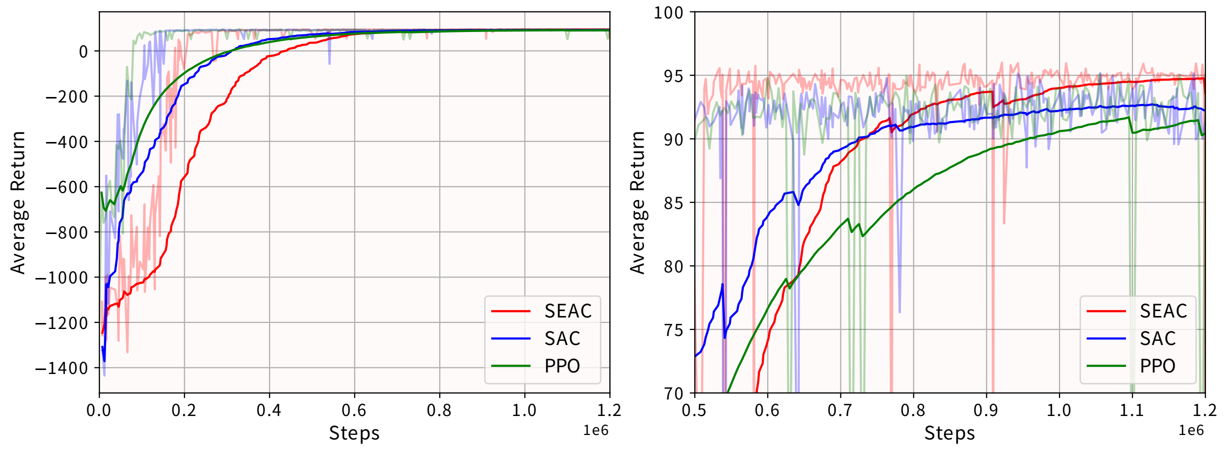


Figure 3.5 Average returns for three algorithms trained in 1.2 million steps. The figure on the right is a partially enlarged version of the figure on the left.

SAC and PPO around 900,000 to 1,200,000 steps. Fast training is also an exciting advantage for deployable AI.Meanwhile, SEAC, using the same policy optimization algorithm but incorporating an elastic time step, demonstrates higher and more stable final performance than SAC.

When considering the adaptation of action execution frequency within the SEAC model, we generate the variable control rate policy explanation charts for four distinct trials, as depicted in Figure 3.7, each utilizing different random seeds. As shown within the four charts, the first step of these SEAC policies is to use one or two *long period* steps to give the agent a large force to accelerate and to approach the goal, then use *a small time* to adjust the speed direction within a step, and then use high-frequency control rates to reach the target (subfigure 4) stably. Subfigures 3 and 4 show that it has learned to avoid obstacles.

Additionally, Figure 3.8 illustrates the energy cost (i.e. the number of time steps) across one hundred independent trial: as expected, SEAC minimizes energy with respect to PPO and SAC without affecting the overall average reward. It is worth noting that SAC and PPO are not optimising for energy consumption, so they are expected to have a large result spread. More interestingly, SEAC *both* reduces energy consumption *and* achieved a high reward. We maintain a uniform seed for all algorithms during this analysis to ensure fair and consistent results.

As shown in Figure 3.7, the agent's task execution strategy primarily focuses on minimizing the number of steps and the time required to complete the task. Notably, the agent often invests a substantial but justifiable amount of time in the initial movement phase, followed by smaller times for

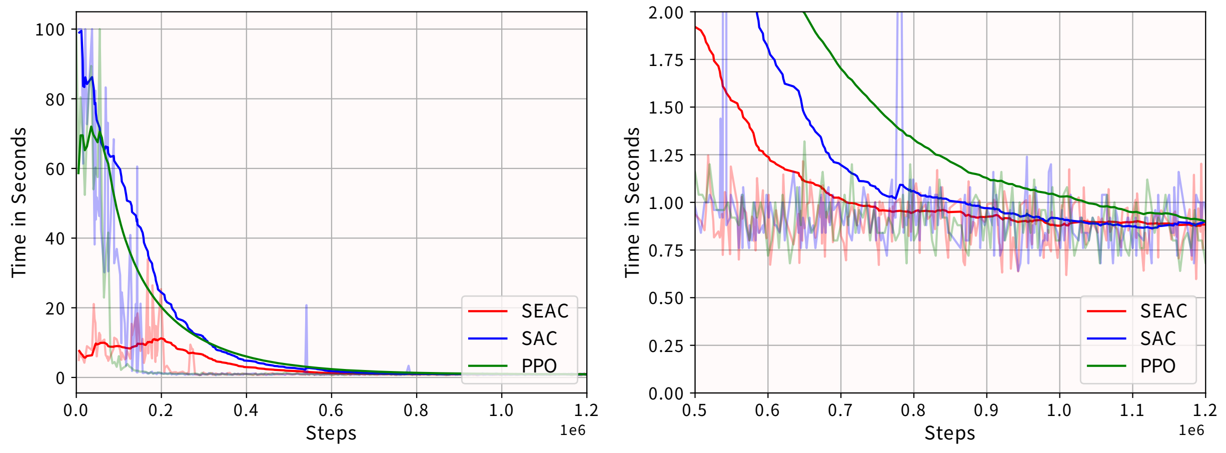


Figure 3.6 Average time cost per episode for three algorithms trained in 1.2 millions steps. The figure on the right is a partially enlarged version of the figure on the left.

subsequent steps to arrive at the goal. This pattern aligns with our core philosophy of minimizing energy and time consumption.

Figure 3.8 shows that the average computing energy consumption and its standard deviation of SEAC for completing the same task are far lower than the two fixed control rate-based reinforcement learning policies as comparison objects. Compared with the SAC baseline, the average computing energy consumption is reduced by arround 25% while taking less time to accomplish tasks. As we mentioned in Section IV3.4, when the storage load increased by about 7 ‰, the computing energy consumption dropped by around 25%, and the performance was not lost or even better. Compared with the fixed-time step RL policy, the variable-rate RL policy is undoubtedly *a better choice* for the deployable AI.

Furthermore, the variance in data distribution is notably reduced within the SEAC results. These findings underscore the algorithm's heightened stability in dynamic environments, further substantiating the practicality of our variable control rate-based RL reward policy.

3.6 Conclusions and Future Work

We propose a variable control rate-based reward policy that allows an agent to decide the duration of a time step in reinforcement learning, reducing energy consumption and increasing sample efficiency (since fewer time steps are needed to reach a goal). Reducing the number of time steps can be very beneficial when using robots with limited capabilities, as the newly freed computational

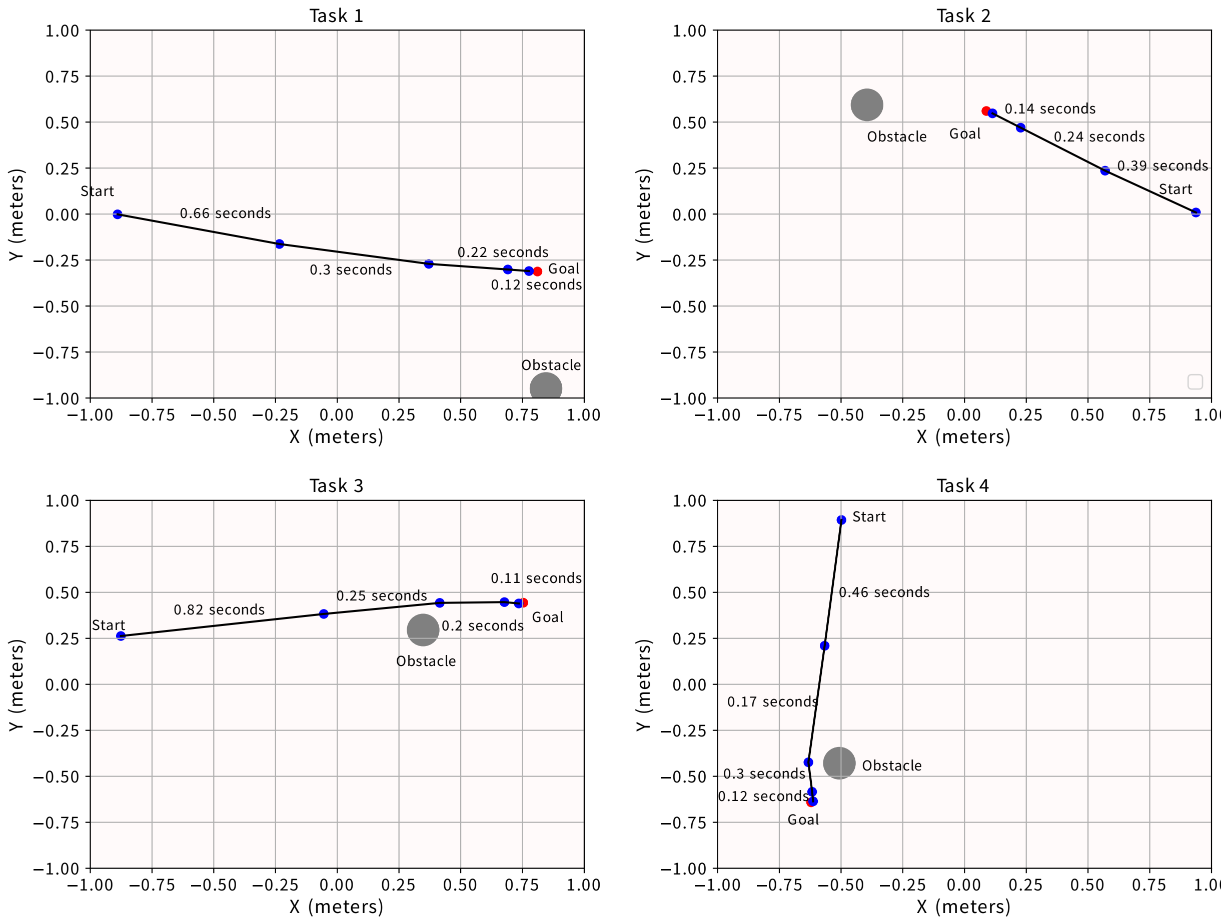


Figure 3.7 Four example tasks show how SEAC changes the control rate dynamically to adapt to the Newtonian mechanics environment and ultimately reasonably complete the goal.

resources can be used for other tasks such as perception, communication, or mapping. The overall energy reduction also increases the applicability of this policy to deployable AI on robotics.

We introduce the Soft Elastic Actor Critic (SEAC) algorithm and verify its applicability with a proof-of-concept implementation in an environment with Newtonian kinematics. The algorithm could be easily extended to real-world applications, and we invite the reader to refer to Section V3.5 and Appendix BF for the implementation details.

To the best of our knowledge, SEAC is the first reinforcement learning algorithm that simultaneously outputs actions and the duration of the following time step. Although the method would benefit from testing in more realistic and dynamic settings, such as Mujoco [80] or TMRL [151], we believe this method presents a promising approach to enhance RL's efficiency and energy conservation for deployment on real-world robotic systems.

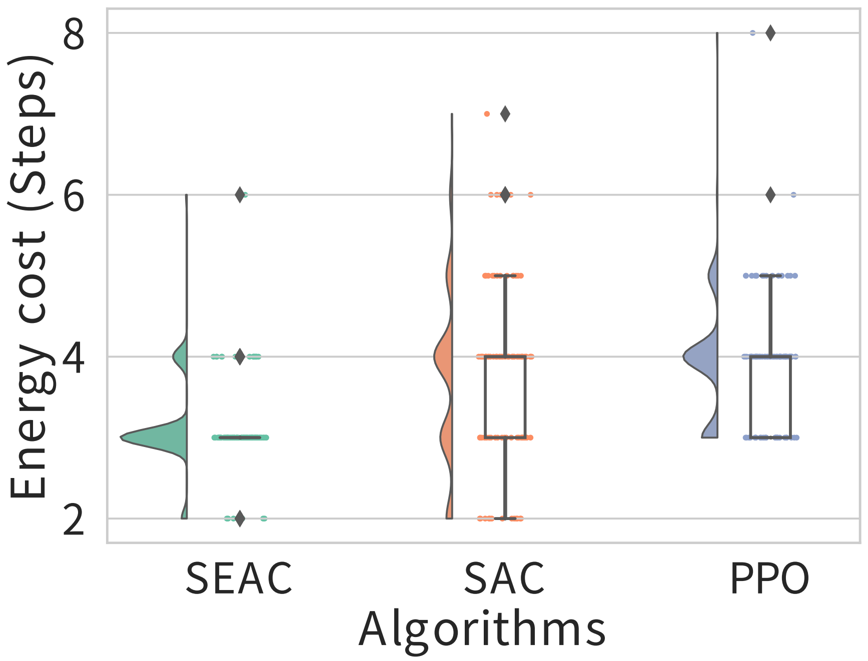


Figure 3.8 The energy cost for 100 trials. SEAC consistently reduces the number of time steps compared with PPO and SAC without affecting the overall average reward. Therefore, SAC and PPO are not optimizing for energy consumption and have a much larger spread.

CHAPTER 4 ARTICLE 2: MOSEAC: STREAMLINED VARIABLE TIME STEP REINFORCEMENT LEARNING

Preface: Traditional reinforcement learning (RL) methods often struggle with the limitations of fixed c ontrol l oops, l eading t o i nefficiencies in co mputational an d en ergy re sources. In this chapter, we introduce the Multi-Objective Soft Elastic Actor-Critic (MOSEAC) algorithm, which enhances RL by dynamically adjusting control frequencies based on the system's state and environmental conditions. This method simplifies h yperparameter t uning a nd i mproves task performance and energy efficiency. We v alidate M OSEAC t hrough s imulations i n a Newtonian kinematics environment, demonstrating significant i mprovements o ver e xisting m ethods like SEAC and CTCO. Our findings suggest that MOSEAC offers a versatile and efficient solution for various RL applications, particularly in dynamic and resource-constrained settings.

Full Citation: Wang, Dong, and Giovanni Beltrame. "MOSEAC: Streamlined Variable Time Step Reinforcement Learning." In Processing of the Reinforcement Learning Conference, Finding The Frame Workshop, 2024. Manuscript accepted for publication on June 4th, 2024.

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.

4.1 Introduction

Model-free deep reinforcement learning (RL) algorithms have achieved significant success in various domains, including gaming [152, 57] and robotic control [153, 5]. 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 [154, 93]. 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 [155].

Two notable VTS-RL algorithms are Soft Elastic Actor-Critic (SEAC) [155] and Continuous-Time Continuous-Options (CTCO) [82], 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 [156]. 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 [155]. 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) [61] or Proximal Policy Optimization (PPO) [44]. This adaptability facilitates the transition from fixed-time step to variable-time step reinforcement learning, significantly expanding the scope of its practical application.

4.2 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 [12, 13]. High frequencies may reduce exploration efficiency in some cases, but are needed in others such as environments with delays [81]. Wang and Karimi [156, 82] 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. [83] 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 [85, 86], but fails to account for physical properties or reduce computational demands, making its practical application challenging.

Additionally, Chen et al. [84] 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.

4.3 Algorithm Framework

To reduce hyperparameter dimensions, we combine SEAC's [155] 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 3.

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

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 4. *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 [156]'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 6), 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 5. *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 [155]'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 6. *The slope of the average reward (k_R) is:*

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

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 [29]. 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 7.

Definition 7. *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 5.

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

Algorithm 1: Multi-Objective Soft Elastic Actor and Critic

```
Require: a policy $\pi$ with a set of parameters $\theta$, $\theta'$, critic parameters $\phi$, $\phi'$, variable time
         step environment model $\Omega$, learning-rate $\lambda_p$, $\lambda_q$, reward buffer $\beta_r$, replay buffer
         $\beta$.
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 3 and 4
        $i \leftarrow i + 1$
    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 $t_i \geq k_{init} \quad \& \quad t_i \mid k_{update}$ 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 6 for $k_R$
            $\alpha_\varepsilon \leftarrow F_{update}(\alpha_m)$           → update $\alpha_m, \alpha_\varepsilon$ followed Definition 5
        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 [29]; k_{length} is maximum exploration steps per episode [29]; k_{init} is steps in the initial random exploration phase [29]. 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) [90, 91], which seeks Pareto optimality [115, 116] 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.

4.4 Empirical Analysis

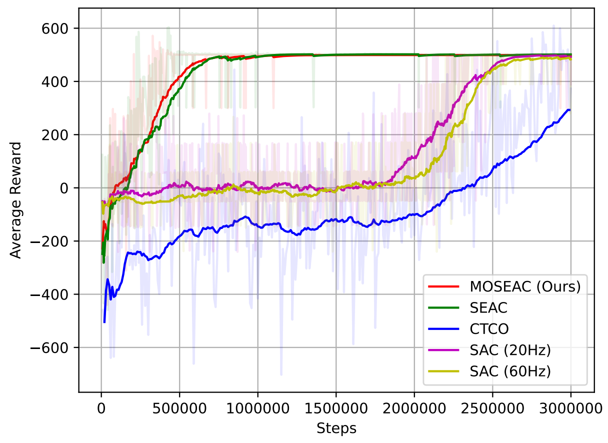
We conduct six experiments involving MOSEAC, SEAC, CTCO, and SAC algorithms (at 20 Hz and 60 Hz) within a Newtonian kinematics environment. A detailed description of the environment appears in Appendix B. These tests are performed on a computer equipped with an Intel Core i5-13600K and a Nvidia RTX 4070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

Figure 4.1 illustrates the result of training process. In Figure 4.1a, 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 4.1b.

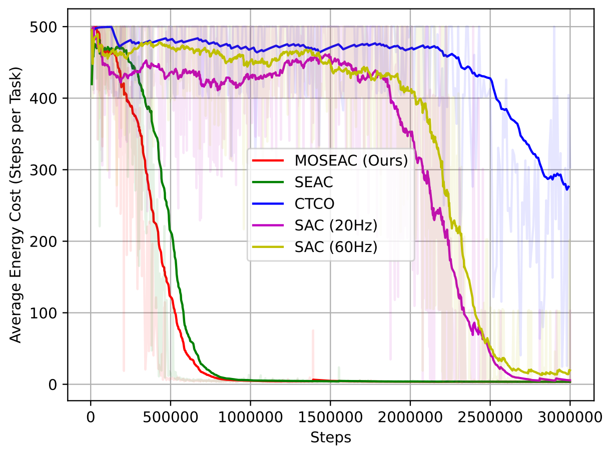
We compared MOSEAC's task performance with the best-performing SEAC after training. Due to the poor performance of CTCO and SAC, we excluded them from the analysis. Figure 4.2 presents performance metrics related to energy consumption and task duration. Specifically, Figure 4.2a shows the average energy consumption (measured in steps) for 300 tasks, and Figure 4.2b 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 J. Similarly, MOSEAC's time cost is significantly

[1]Our code is public available.



(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 4.1 Result graphs of five reinforcement learning algorithms.

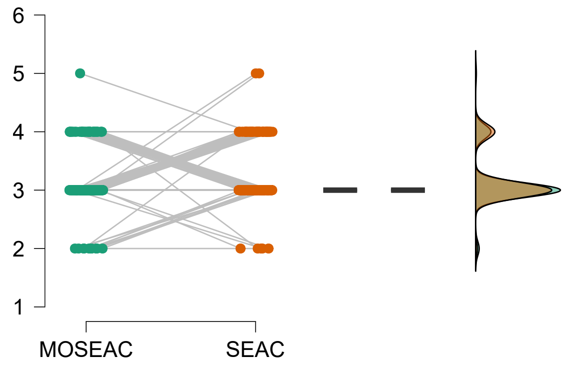
lower than SEAC's (z = -2.669, p = 0.004), with MOSEAC having a mean time cost of 0.905 (SD = 0.125), compared to SEAC's mean of 0.935 (SD = 0.127).

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.

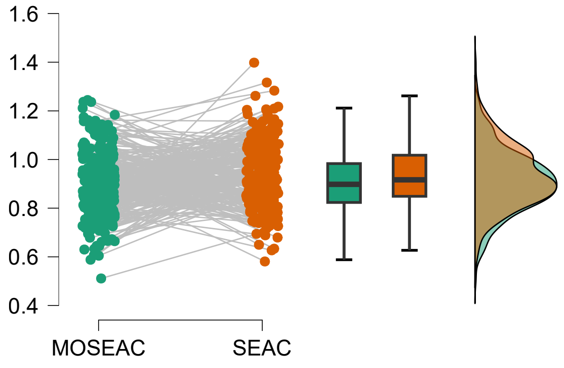
4.5 Conclusion and Future Work

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.



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



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

Figure 4.2 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.

CHAPTER 5 ARTICLE 3: REINFORCMENT LEARNING WITH ELASTIC TIME STEP

Preface: Traditional reinforcement learning (RL) methods are typically implemented with fixed control rates, leading to inefficiencies as optimal control rates vary with task requirements. In this paper, we introduce the Multi-Objective Soft Elastic Actor-Critic (MOSEAC) algorithm, which dynamically adjusts control frequency to minimize computational resources by selecting the lowest viable frequency. We show that MOSEAC converges and produces stable policies at the theoretical level, validated in a real-time 3D racing game. MOSEAC significantly outperformed other variable time step approaches in energy efficiency, training speed, and task effectiveness, demonstrating its potential for real-world RL applications in robotics.

Full Citation: Wang, Dong, and Giovanni Beltrame. "Reiforcement Learning with Elastic Time Step". Submitted to IEEE Robotics and Automation Letters on July 2nd, 2024.

Traditional Reinforcement Learning (RL) policies are typically implemented with fixed control rates, often disregarding the impact of control rate selection. This can lead to inefficiencies as the optimal control rate varies with task requirements. We propose the Multi-Objective Soft Elastic Actor-Critic (MOSEAC), an off-policy actor-critic algorithm that uses elastic time steps to dynamically adjust the control frequency. This approach minimizes computational resources by selecting the lowest viable frequency. We show that MOSEAC converges and produces stable policies at the theoretical level, and validate our findings in a real-time 3D racing game. MOSEAC significantly outperformed other variable time step approaches in terms of energy efficiency and task effectiveness. Additionally, MOSEAC demonstrated faster and more stable training, showcasing its potential for real-world RL applications in robotics.

keywords Reinforcement Learning, AI-Based Methods, Optimization and Optimal Control

5.1 Introduction

Model-free deep reinforcement learning (RL) algorithms have demonstrated significant value in diverse domains, from video games [136, 151] to robotic control [72, 131]. For instance, Sony's autonomous racing cars achieved remarkable results with fixed training frequencies of 10 Hz and 60 Hz [131].

However, using suboptimal, fixed control rates presents limitations, leading to excessive caution, wasted computational resources, risky behavior, and compromised control.

Variable Time Step Reinforcement Learning (VTS-RL) has been introduced to address issues related to fixed control rates in traditional RL. By leveraging reactive programming principles [154, 93], VTS-RL performs control actions only when necessary, reducing computational load and enabling variable action durations. For example, in robotic manipulation, VTS-RL allows a robot arm to adapt its control frequency dynamically, using lower frequencies for simple tasks and higher frequencies for complex maneuvers or delicate handling [82].

Two prominent VTS-RL algorithms are Soft Elastic Actor-Critic (SEAC [155]) and Continuous-Time Continuous-Options (CTCO) [82]. CTCO supports continuous-time decision-making with flexible option durations, enhancing exploration and robustness. However, it involves tuning multiple hyperparameters, such as radial basis functions (RBFs) and time-related parameters τ for its adaptive discount factor γ, making tuning complex in certain environments.

SEAC incorporates reward components related to task energy (number of actions) and task time, making it effective in time-constrained environments. Despite its advantages, SEAC demands careful tuning of hyperparameters to balance task, energy, and time costs to ensure optimal performance. The sensitivity of both SEAC and CTCO to hyperparameter settings presents a challenge for users aiming to fully exploit their capabilities.

We recently proposed Multi-Objective Soft Elastic Actor-Critic (MOSEAC [157]), which reduced the dimension of hyperparameters and the algorithm's dependence on hyperparameters by dynamically adjusting the hyperparameters corresponding to the reward structure.

We identify shortcomings in our previous work, where we proposed adapting α_m by monitoring reward trends. However, in some tasks, reward trends are not always stable, posing a risk of reward explosion. To address this, we introduce an upper limit α_{max} for α_m. We provid pseudocode for this improvement and established a corresponding Lyapunov stability function [14] to demonstrate the stability (convergence) of the new algorithm.

We evaluate MOSEAC in a racing game (Ubisoft TrackMania [158]), comparing it against CTCO [82], SEAC [155] and Soft Actor-Critic (SAC [73]). Our results demonstrate MOSEAC's improved training speed, stability, and efficiency. Our key contributions are:

1. **Algorithm Enhancement and Convergence Proof:** We introduce an upper limit α_{max} on the parameter α_m to prevent reward explosion, ensuring the stability and convergence of the MOSEAC algorithm. This enhancement is validated through a Lyapunov model, providing a proof of convergence and demonstrating the efficacy of the new adjustment mechanism.

2. **Enhanced Training Efficiency:** We show that MOSEAC achieves faster and more stable training than CTCO, highlighting its practical benefits and applicability in real-world scenarios.

The paper is organized as follows. Section 5.2 describes the current research status of variable time step RL. Section 5.3 introduces MOSEAC with its pseudocode and Lyapunov model. Section 5.4 describes the test environments. Section 5.5 presents the simulation parameters and results. Finally, Section 5.6 concludes the paper.

5.2 Related Work

Fixed control rates in RL often lead to inefficiencies. Research by [155] shows that suboptimal fixed rates can cause excessive caution or risky behavior, wasting resources and compromising control. Control rates significantly impact continuous control systems beyond computational demands. Some studies [12, 13] indicate that high control rates can degrade RL performance, while low rates hinder adaptability in complex scenarios.

Sharma et al. [83] proposed learning to repeat actions to mimic dynamic control rates, but this approach does not change the control frequency or reduce computational demands. Few studies have explored repetitive behaviors in real-world scenarios, such as those by Metelli et al. [85] and Lee et al. [86]. Chen et al. [84] introduced variable "control rates" using actions like "sleep," but still involved fixed-frequency checks.

Cui et al. [121] applied the Lyapunov model to verify the stability of RL algorithms and addressed handling the dynamics of power systems over time, although it still uses a fixed frequency to scan system states.

Introducing time steps of variable duration allows a robotic system to better adapt to its task and environment by adjusting control actions based on the system's current conditions, addressing the nonlinearity and time-variant dynamics typical in robotics [159]. This adaptability ensures optimal performance through efficient resource utilization and effective response to varying conditions [95, 93].

Lyapunov models provide robust stability guarantees by employing a scalar function that decreases over time, ensuring system convergence to a desired equilibrium. By incorporating Lyapunov stability into our reinforcement learning framework, we ensure that the learning process remains stable, thereby preventing erratic behavior and potential system failure [121, 122].

5.3 Multi-Objective Soft Elastic Actor and Critic

Our algorithm builds upon previous work [157], combining SEAC's hyperparameters for balancing task, energy, and time rewards through a simple multiplication method, and applying adaptive adjustments to the remaining hyperparameters. A key improvement is the introduction of an upper limit for the hyperparameter α_m. Below is an overview of the MOSEAC algorithm. This overview emphasizes the critical aspects of MOSEAC without delving into the detailed definitions. The reader can refer to [157] for details.

The reward in MOSEAC is:

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

where R_t is the task reward, R_τ is a time-dependent term, α_m is a weighting factor to modulate reward magnitude, and $\alpha_\varepsilon = 0.2 \cdot \left(1 - \frac{1}{1+e^{-\alpha_m+1}}\right)$ is a penalty parameter applied at each time step to reduce unnecessary actions.

To automatically set α_m to an optimal value, we dynamically adjust it during training. This adjustment mitigates convergence issues, specifically the problem of sparse rewards caused by suboptimal settings of α_m. MOSEAC increases α_m (and decrease α_ε) if the average reward is declining.

We introduce a hyperparameter ψ to dynamically adjust α_m based on observed trends in task rewards:

$$\alpha_m = \begin{cases} \alpha_m + \psi & \text{if } \alpha_m < \alpha_{max} \\ \alpha_m = \alpha_{max} & \text{otherwise} \end{cases}$$

In this work, we add α_{max} to ensure convergence and prevent reward explosion.

Algorithm 2 shows the pseudocode of MOSEAC. In short, MOSEAC extends the SAC algorithm by incorporating action duration D into the action policy set. This expansion allows the algorithm to predict the action and its duration simultaneously. The reward is calculated using section 5.3, and its changes are continuously monitored. If the reward trend declines, α_m increases linearly at a rate of ψ, without exceeding $alpha_{max}$. The action and critic networks are periodically updated like the SAC algorithm based on these preprocessed rewards.

The maximum number of training steps is denoted as t_{max} [29], while k_{length} indicates the maximum number of exploration steps per episode [29]. The initial random exploration phase comprises k_{init} steps [29]. k_{update} [29] is the update interval determining the frequency of updates for these neural networks used in the actor and critic policies. The reward R_i is computed as $R(S_i, A_i, D_i)$, where D_i falls within the interval $[D_{min}, D_{max}]$, representing the duration of the action.

Algorithm 2: Multi-Objective Soft Elastic Actor and Critic

```
Require: a policy $\pi$ with a set of parameters $\theta$, $\theta^{'}$, critic parameters $\phi$, $\phi^{'}$, variable time
         step environment model $\Omega$, learning-rate $\lambda_p$, $\lambda_q$, reward buffer $\beta_r$, replay buffer
         $\beta$.
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
        $A_i, D_i = \pi_\theta(S_i)$
        $S_{i+1}, R_i = \Omega(A_i, D_i)$
        $i \leftarrow i + 1$
    end
    $\beta_r \leftarrow 1/i \times \sum_0^i R_i$
    $\beta \leftarrow S_{0\sim i},\ A_{0\sim i},\ D_{0\sim i},\ R_{0\sim i},\ S_{1\sim i+1}$
    $i = 0$
    $t_i \leftarrow t_i + 1$
    if $t_i \geq k_{init} \quad \& \quad t_i \mid k_{update}$ then
        $Sample\, S,\ A,\ D,\ R,\ S^{'} from(\beta)$
        $\phi \leftarrow \phi - \lambda_q \nabla_\delta \mathcal{L}_Q(\phi,\ S,\ A,\ D,\ R,\ S^{'})$
                                                              $\rightarrow$ critic update
        $\theta \leftarrow \theta - \lambda_p \nabla_\theta \mathcal{L}_\pi(\theta,\ S,\ A,\ D,\ \phi)$
                                                              $\rightarrow$ actor update
        if $k_R(\beta_r)$ then
            $\alpha_m = \alpha_m + \psi$                              if$\alpha_m < \alpha_{max}$
            Or $\alpha_m = \alpha_{max}$                             otherwise
            $\alpha_\varepsilon \leftarrow F_{update}(\alpha_m)$
        end
        $\beta_r = 0$
                  $\rightarrow$ Re-record average reward values under new hyperparameters
    end
    Perform soft-update of $\phi^{'}$ and $\theta^{'}$
end
```

Our algorithm employs a scalarizing strategy to optimize multiple objectives, sweeping the design space by adapting α_m during training. Unlike Hierarchical Reinforcement Learning (HRL) [90, 91] which aims for Pareto optimality [116] through layered reward policies, our approach simplifies the process. We focus on ease of use and computational efficiency, ensuring our method can easily adapt to various algorithms.

It is worth noting that the choice of ψ is crucial for optimal performance, as ψ represents the sweeping step of our optimization (similar to the learning parameter in gradient descent). We suggest using the pre-set ψ value from our implementation to minimize the need for further adjustments. If training performance is inadequate, a high ψ value might cause the reward signal's gradient to change too quickly, leading to instability; in this case, a lower ψ is recommended. Conversely, if training progresses too slowly, a low ψ value might weaken the reward signal, hindering convergence; thus, increasing ψ could be beneficial.

5.3.1 Convergence Proof

With our reward function, the policy gradient is:

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

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

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

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

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.

The new policy parameter θ update rule is:

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

where β_k is the learning rate at step k.

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, and its upper limit is α_{max}, which guarantees algorithmic convergence and prevents reward explosion. α_ε decreases as defined.

2. **Learning Rate Conditions:** α_k and β_k must satisfy [41]:

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

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

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.

1. **Positive Scaling:** As $0 \leq R_\tau \leq 1$ and $\alpha_m \geq 0$, α_m only scales the reward without 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.

Under these conditions, MOSEAC will converge to a local optimum [29]:

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

5.3.2 Lyapunov Stability

To analyze the stability of MOSEAC, we define a Lyapunov function $V(t)$ and calculate its time derivative to ensure asymptotic stability:

$$V(t) = \frac{1}{2}\alpha_m(t)^2 + \sum_s \left[Q(s, a, D, \alpha_m) - Q^*(s, a, D)\right]^2$$

where $\alpha_m(t)$ varies with time, $Q(s, a, D, \alpha_m)$ is the current Q-value (D is the duration of action a), and $Q^*(s, a, D)$ is the ideal Q-value.

The time derivative of $V(t)$ is:

$$\begin{aligned}\dot{V}(t) = {} & \alpha_m(t)\dot{\alpha}_m(t) \\ & + \sum_s 2\left[Q(s, a, D, \alpha_m) - Q^*(s, a, D)\right] \\ & \dot{Q}(s, a, D, \alpha_m)\end{aligned}$$

Based on the linear growth model of α_m:

$$\dot{\alpha}_m(t) = \begin{cases} k & \text{if } \alpha_m(t) < \alpha_{\text{max}} \\ 0 & \text{if } \alpha_m(t) = \alpha_{\text{max}} \end{cases}$$

Replacing in $\dot{V}(t)$:

$$\dot{V}(t) = \begin{cases} \alpha_m(t)k + \sum_s 2\left[Q(s, a, D, \alpha_m) - Q^*(s, a, D)\right] \\ \quad \dot{Q}(s, a, D, \alpha_m), \quad \text{if } \alpha_m(t) < \alpha_{\text{max}} \\ \\ \sum_s 2\left[Q(s, a, D, \alpha_m) - Q^*(s, a, D)\right] \\ \quad \dot{Q}(s, a, D, \alpha_m), \quad \text{if } \alpha_m(t) = \alpha_{\text{max}} \end{cases}$$

Since $\alpha_m(t) > 0$ and $k > 0$, $\dot{V}(t) > 0$ when $\alpha_m(t) < \alpha_{\text{max}}$. However, when $\alpha_m(t)$ reaches α_{max}, $\dot{V}(t) = 0$, indicating that the system reaches a stable state.

5.4 Experimental Setup

We validate our MOSEAC in a real-time racing game, Trackmania [158]. Figure 5.1 illustrates the testing environment. In Trackmania, players race to complete the track as quickly as possible. We employed the map developed by the TMRL [151] team for a direct comparison of MOSEAC's performance against their SAC model. It is worth noting that training and deploying a policy for this game can only happen in real time [140], ensuring the realism of the overall experiments.

For controlling the car in the game, Trackmania offers: • gas control; • brake control; and • steering. The action value includes the rate at which these controls are applied. The input to the policy is the pixels as well as the numerical information shown on the heads-up display.

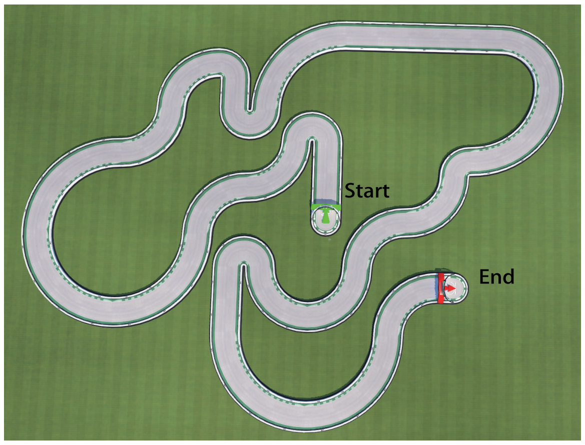


Figure 5.1 Top preview of the Trackmania track.

The reward system in the game is designed to encourage efficient path following: more rewards are given for covering more path points in a single move.This path is a series of evenly distributed points that collectively form the shortest route from the starting point to the end point of the racing game. This approach is consistent with the TMRL team's methodology [151]. The game uses realistic car physics, but there is no car damage or crash detection. Further details are available in Table 5.1.

Table 5.1 Trackmania Environments Details

State and action space of Trackmania		
	Data Space	Annotate
State Dimension	143	Details in Section 5.2
Car Speed	$(-1, 1)$	
Car Gear	$(-1, 1)$	
Wheel RPM	$(-1, 1)$	
RGB Image	$(64, 64, 3)$	RGB arrays
Action Dimension	4	
Gas Control	$(-1, 1)$	
Brake	$(-1, 1)$	
Yaw Control	$(-1, 1)$	
Control Rate	$(5, 30)\ Hz$	

To facilitate the agent's comprehension of how the controls influence speed and acceleration, we feed 4 sequential frames (and the intervals between them) to a convolutional neural network (CNN), whose output is an embedding contributing to the state representation (see Figure 5.2). The CNN distills features, transforming the image data from a matrix of dimensions (4, 64, 64, 3) to a compact (128, 1, 1) matrix.

To evaluate the impact of our variable time step approach, we used the same visual-based navigation and task reward policy across the tested approaches, varying only the control rate. It is worth noting that the task reward policy only includes time and disregards collisions.

Overall, our model inputs are the car's speed, gear, and wheel RPM, the current step number in one episode, along with the two most recent actions taken, for a final 143-dimensional state.

5.5 Experimental Results

We conducted experiments with MOSEAC, CTCO [82] and SEAC [155] on Trackmania for over 1320 hours[1]. These experiments were conducted on a I5-13600K computer with an NVIDIA RTX 4070 GPU. The final result video for MOSEAC is publicly available[2], with 43.202 seconds to complete the test track.

Before discussing the performance of MOSEAC and CTCO, it is essential to provide some background context. In our previous work, the SEAC model policies were trained using different hyperparameters [156]. These hyperparameters were explicitly tuned for SEAC, optimizing its performance in the given environment. Besides, the SAC model was trained by the TMRL team with a different set of hyperparameters tailored to their training approach [151]. Given these differences, directly comparing their training curves would be unfair; hence, we focus solely on the MOSEAC and CTCO training curves.

Reward signals are notably sparse in the environment set by the TMRL team. The reward is calculated based on the difference between the path coordinates after movement and the initial path coordinates, divided by 100 [151]. This sparse reward environment necessitates a very low-temperature coefficient alpha SAC [73], otherwise, the entropy component would dominate the reward. This insensitivity to reward signals can lead to poor training performance or even training failure. MOSEAC and CTCO are derived from SAC [157, 82], and they face similar constraints.

We have maintained this reward system to ensure a fair comparison with the TMRL team. Modifying the environment is not straightforward, as simply amplifying the reward signal may cause the agent to accumulate rewards rapidly, leading to overly optimistic estimates and unstable training. Such "reward explosion" can cause the agent to settle into local optima, neglecting long-term returns and reducing overall performance.

Using an adaptive temperature coefficient during training, we observed that the coefficient value becomes extremely low towards the end of the training. This results in a narrow distribution of actions, significantly slowing down the policy training and optimization process. This small tem-

[1]Our code is publicly available on GitHub: `https://github.com/alpaficia/TMRL_SEAC`
[2]`https://youtu.be/1aQ0xSK55nk`

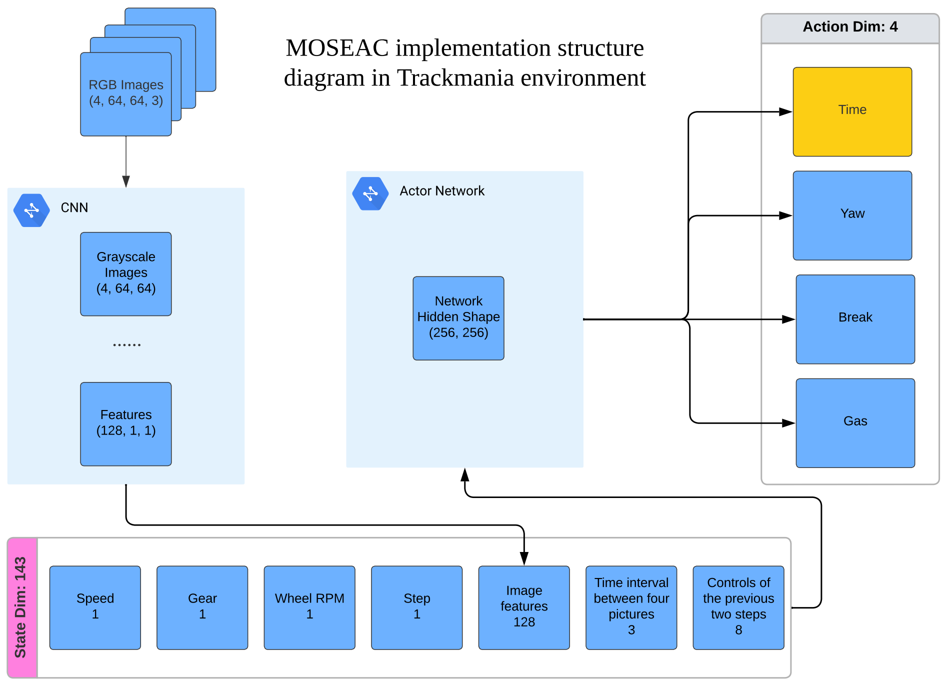

MOSEAC implementation structure diagram in Trackmania environment
RGB Images (4, 64, 64, 3)
CNN
Grayscale Images (4, 64, 64)
......
Features (128, 1, 1)
Actor Network
Network Hidden Shape (256, 256)
Action Dim: 4
Time
Yaw
Break
Gas
State Dim: 143
Speed 1
Gear 1
Wheel RPM 1
Step 1
Image features 128
Time interval between four pictures 3
Controls of the previous two steps 8

Figure 5.2 This is the implementation structure diagram of MOSEAC in the TrackMania environment. We use CNN to extract potential information in the environment and learn the extracted feature values based on rewards. The 143-dimensional state value and the 4-dimensional action value are shown in the figure. The time in the action value is not used for the current time step but for the next step.

perature coefficient causes both MOSEAC and CTCO to have a slow training and optimization process.

Figure 5.4 demonstrates that MOSEAC reaches a high point at around 6 million steps, indicating initial training success. However, Figure 5.5 shows that after 6 million steps, MOSEAC required over 14 million additional steps to approach the optimal value. This indicates that MOSEAC and CTCO require substantial time for initial training and further optimization in a sparse reward environment.

Despite these challenges, MOSEAC exhibits better training efficiency than CTCO. MOSEAC adapts to the environment more quickly, maintaining a more stable learning curve. In contrast, CTCO, influenced by its adaptive γ mechanism, tends to favor smaller γ values, impairing long-term planning and significantly slowing down training. Furthermore, MOSEAC achieves a higher final reward than CTCO, showcasing its superior adaptability and efficiency in sparse reward settings.

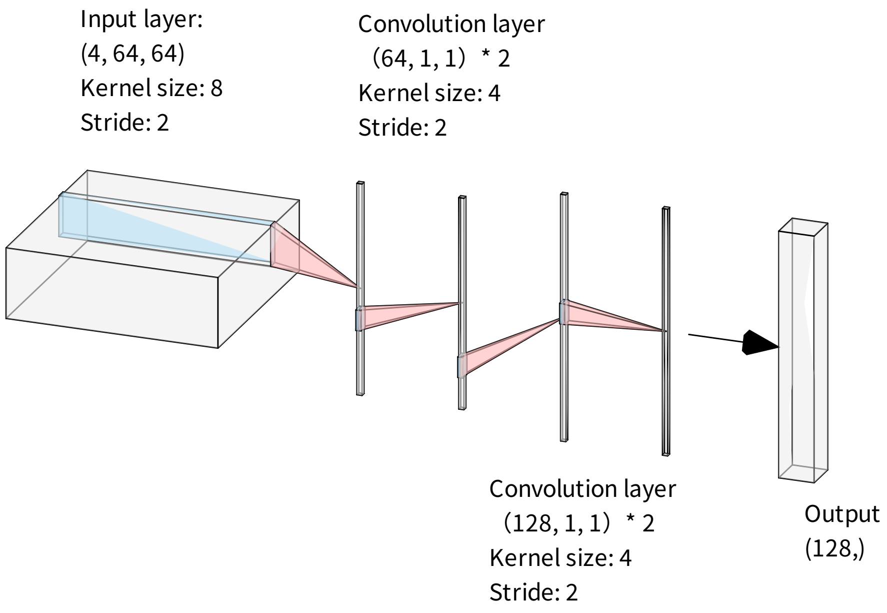


Figure 5.3 Our CNN structure diagram that we used to extract image features from the Trackmania video game. We convert RGB images into grayscale images and then input them into the CNN.

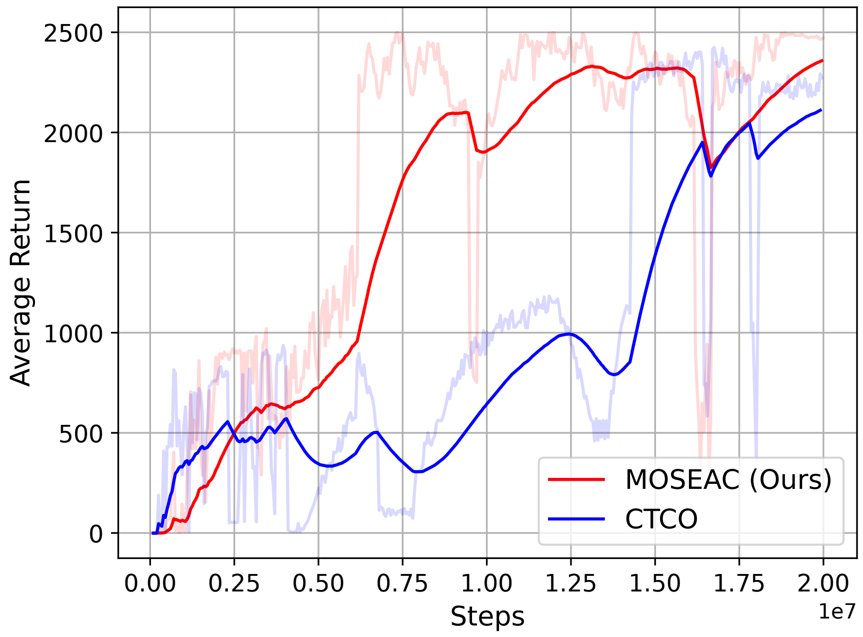


Figure 5.4 Training Progress of MOSEAC and CTCO: Average Return Over Time

Training in the Trackmania environment is real-time, and our 1320+ hours of training have demonstrated that MOSEAC can handle complex, sparse reward environments, albeit requiring significant time investment. Therefore, if a direct comparison with our results, specifically regarding training

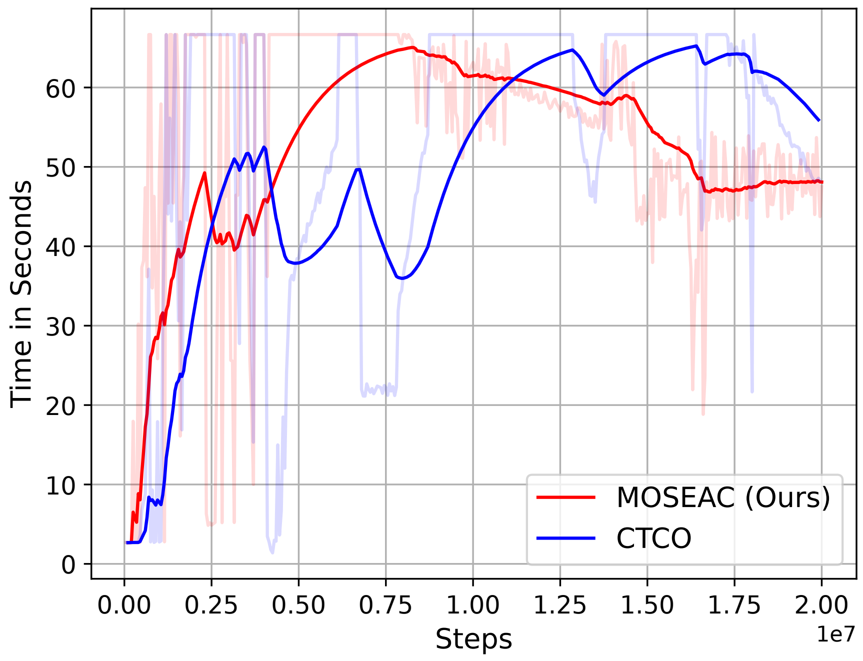


Figure 5.5 Training Progress of MOSEAC and CTCO: Time Consumption Over Steps

speed within the same reward environment, is unnecessary, we recommend optimizing the reward signals in the TMRL Trackmania environment. One possible optimization strategy could be to amplify the per-step reward signal appropriately. This adjustment theoretically allows for faster training of effective control policies, significantly enhancing training efficiency and saving substantial real-world time.

We compared the racetrack time (referred to as *time cost*) as well as the computational energy cost (in terms of the number of control steps) of MOSEAC with SEAC, CTCO, and SAC (20Hz) after training. Figure 5.6 illustrates the energy cost distribution and Figure 5.7 the time cost (lower is better in both cases). For both energy and time there is no overlap among the different methods, and the data follows a normal distribution (Shapiro-Wilk test of normality, see Table 5.2 and Table 5.4).

Using a paired sample T-test, MOSEAC showed lower energy in all of the trials compared with CTCO ($t = -64.85$, $df = 29$, $p \ll 0.001$), SEAC ($t = -35.90$, $df = 29$, $p \ll 0.001$), and SAC ($t = -196.35$, $df = 29$, $p \ll 0.001$). The statistics for the energy cost measures are presented in Table 5.3. Similarly, MOSEAC showed lower race time in all of the trials compared with CTCO ($t = -55, 67$, $df = 29$, $p \ll 0.001$), SEAC ($t = -32.92$, $df = 29$, $p \ll 0.001$), and SAC ($t = -41.06$, $df = 29$, $p \ll 0.001$). The statistics for the time cost measures are presented in Table 5.5.

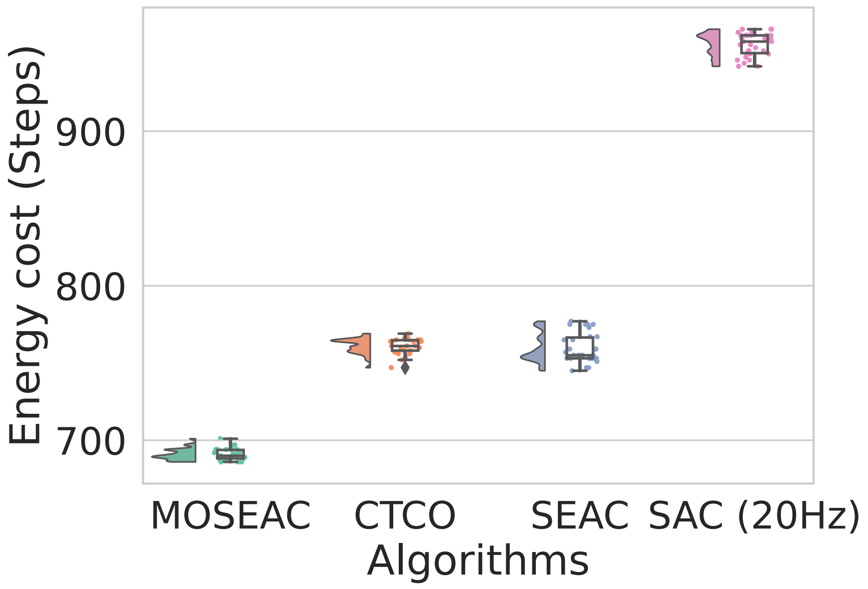


Figure 5.6 This figure compares the energy costs of four different algorithms: MOSEAC, CTCO, SEAC, and SAC (20Hz). The y-axis represents the energy cost in steps, while the x-axis lists the algorithms. The box plots show that MOSEAC has the lowest median energy cost, followed by SEAC, CTCO, and SAC (20Hz). This indicates that MOSEAC is the most energy-efficient among the four algorithms.

Table 5.2 Test of Normality (Shapiro-Wilk) - Energy cost difference

			W	p
MOSEAC	-	CTCO	0.947	0.142
MOSEAC	-	SEAC	0.948	0.150
MOSEAC	-	SAC (20Hz)	0.956	0.242

The improved performance of MOSEAC over SEAC, CTCO, and SAC (20Hz) can be attributed to its reward function and the integration of a state variable. 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, mainly 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 guide the agent's decisions more effectively, resulting in better energy efficiency and task completion speed in practical applications.

Table 5.3 Energy cost descriptives

	N	Mean	SD	SE	COV
MOSEAC	30	690.800	3.652	0.667	0.005
CTCO	30	760.933	4.989	0.911	0.007
SEAC	30	759.467	9.508	1.736	0.013
SAC (20Hz)	30	956.133	7.482	1.366	0.008

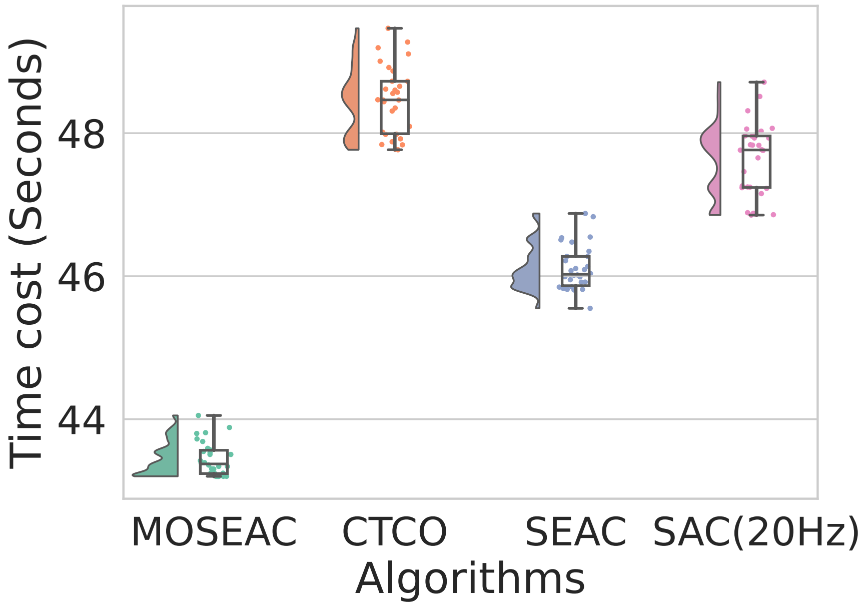


Figure 5.7 This figure compares the time costs of four different algorithms: MOSEAC, CTCO, SEAC, and SAC (20Hz). The y-axis represents the time cost in seconds, while the x-axis lists the algorithms. The box plots reveal that MOSEAC has the lowest median time cost, indicating faster task completion compared to the other algorithms. CTCO, SEAC, and SAC (20Hz) show higher median time costs, with MOSEAC outperforming them in terms of time efficiency.

5.6 Conclusions

In this paper, we introduced and evaluated the Multi-Objective Soft Elastic Actor-Critic (MOSEAC) algorithm, demonstrating its superior performance compared to CTCO, SEAC, and SAC (20Hz) in the sparse reward environment created by the TMRL team in the Trackmania game. MOSEAC features an innovative reward function that combines task rewards and time-related rewards, enhancing the reward signal when both task performance and time efficiency are high. This non-linear interaction balances these factors, leading to improved energy efficiency and task completion speed.

To prevent reward explosion, we introduced an upper limit (α_{max}) for the hyperparameter α_m, ensuring convergence and stability, validated using a Lyapunov model. The boundedness analysis

Table 5.4 Test of Normality (Shapiro-Wilk) - Time cost difference

			W	p
MOSEAC	-	CTCO	0.954	0.222
MOSEAC	-	SEAC	0.963	0.367
MOSEAC	-	SAC (20Hz)	0.968	0.478

Table 5.5 Time cost descriptives

	N	Mean	SD	SE	COV
MOSEAC	30	43.441	0.237	0.043	0.005
CTCO	30	48.463	0.481	0.088	0.010
SEAC	30	46.117	0.317	0.058	0.007
SAC (20Hz)	30	47.636	0.510	0.093	0.011

confirmed that α_m remains within the defined limits, maintaining the strength of the reward function and ensuring the robustness and reliability of the learning process.

Our experiments in the Trackmania environment, conducted over 1300 hours, validated these enhancements, showcasing MOSEAC's robustness and efficiency in real-world applications. The recorded best time for MOSEAC is 43.202 seconds, significantly faster than the other algorithms.

Building on MOSEAC's success, our future work will focus on extending the principles of variable time step algorithms to other reinforcement learning frameworks, such as Hierarchical Reinforcement Learning (HRL). This extension aims to address the complexities of long-term planning tasks more efficiently, further improving the adaptability and performance of RL algorithms in diverse and challenging environments, which brings benefits for solving complex problems by RL using real robot systems.

CHAPTER 6 ARTICLE 4: VARIABLE TIME STEP REINFORCEMENT LEARNING FOR ROBOTIC APPLICATIONS

Preface: Traditional reinforcement learning (RL) methods often struggle with the limitations of fixed control loops, leading to inefficiencies in computational and energy resources. In this paper, we introduce the Multi-Objective Soft Elastic Actor-Critic (MOSEAC) algorithm, which enhances RL by dynamically adjusting control frequencies based on the system's state and environmental conditions. This method simplifies hyperparameter tuning and improves task performance, training speed, and energy efficiency. We validate MOSEAC through simulations and real-world robotic experiments, demonstrating significant improvements over existing methods like SEAC and CTCO. Notably, MOSEAC significantly reduces CPU and GPU usage, conserving computational resources and extending battery life. Our findings suggest that MOSEAC offers a versatile and efficient solution for various RL applications, particularly in dynamic and resource-constrained settings.

Full Citation: Wang, Dong, and Giovanni Beltrame. "Variable Time Step Reinforcement Learning for Robotic Applications". Manuscript submitted to IEEE Transaction on Robotics on June 28th, 2024.

Traditional reinforcement learning (RL) generates discrete control policies, assigning one action per cycle. These policies are usually implemented as in a fixed-frequency control loop. This rigidity presents challenges as optimal control frequency is task-dependent; suboptimal frequencies increase computational demands and reduce exploration efficiency. Variable Time Step Reinforcement Learning (VTS-RL) addresses these issues with adaptive control frequencies, executing actions only when necessary, thus reducing computational load and extending the action space to include action durations. In this paper we introduce the Multi-Objective Soft Elastic Actor-Critic (MOSEAC) method to perform VTS-RL, validating it through theoretical analysis and experimentation in simulation and on real robots. Results show faster convergence, better training results, and reduced energy consumption with respect to other variable- or fixed-frequency approaches.

Keywords Variable Time Step Reinforcement Learning (VTS-RL), Deep Learning in Robotics and Automation, Learning and Adaptive Systems, Optimization and Optimal Control

6.1 Introduction

Deep reinforcement learning (DRL) algorithms have achieved significant success in gaming [152, 57] and robotic control [153, 5]. Traditional DRL employs a fixed control loop at set intervals (e.g., every 0.1 seconds). This fixed-rate control can cause stability issues and high computational demands, particularly in dynamic environments where the optimal action frequency varies.

Variable Time Step Reinforcement Learning (VTS-RL) has been recently proposed to address these issues. Based on reactive programming principles [154, 93], VTS-RL executes control actions only when necessary, reducing computational load and expanding the action space to include variable action durations. For example, in robotic manipulation, VTS-RL allows a robot arm to dynamically adjust its control frequency, using lower frequencies during simple, repetitive tasks and higher frequencies during complex maneuvers or when handling delicate objects [82].

Two notable VTS-RL algorithms are Soft Elastic Actor-Critic (SEAC) [155] and Continuous-Time Continuous-Options (CTCO) [82]. CTCO employs continuous-time decision-making with flexible option durations. CTCO requires tuning several hyperparameters, such as its radial basis functions (RBFs) and the time-related hyperparameters τ for its adaptive discount factor γ, complicating tuning in some environments.

SEAC incorporates reward terms for task energy (number of actions) and task time, making it effective in time-restricted environments like racing video games [156]. However, it also requires careful hyperparameter tuning (balancing task, energy, and time costs) to maintain performance. The sensitivity of SEAC and CTCO to hyperparameter settings challenges users to fully leverage their potential.

We introduce the Multi-Objective Soft Elastic Actor-Critic (MOSEAC) algorithm [157]. MOSEAC integrates action durations into the action space and adjusts hyperparameters based on observed trends in task rewards during training, reducing the need to set multiple hyperparameters. Additionally, our hyperparameter setting approach can be broadly applied to any continuous action reinforcement learning algorithm.

This adaptability facilitates the transition from fixed-time step to variable-time step reinforcement learning, significantly expanding its practical application.

In this paper, we provide an in-depth analysis of MOSEAC's theoretical performance, covering its framework, implementation, performance guarantees, convergence and complexity analysis. We also deploy the MOSEAC model as the navigation policy in simulation and on a physical AgileX Limo [1]. Our evaluation includes statistical performance analysis, trajectory similarity analysis between simulated and real environments, and a comparison of average computational resource consumption on both CPU and GPU.

MOSEAC allows the optimization of the control frequency for RL policies, as well as a reduction in computational load on the robots' onboard computer. The resources saved can be redirected to other critical tasks such as environmental sensing [160] and communication [161]. In addition, the MOSEAC principles can be applied to a variety of RL algorithms, making it a useful tool for the deployment of RL policies on physical robotics systems, reducing the real-to-sim gap.

The paper is organized as follows. Section 6.2 reviews the current research status of VTS-RL. Section 6.3 details the MOSEAC framework with its pseudocode. Section 6.4 presents the theoretical analysis of MOSEAC. Section 6.5 describes the implementation of our validation environment in the real world and the method to build the simulation environment. Section 6.6 presents the experiment results on both the simulation and the real Agilex Limo. Finally, Section 6.7 concludes the paper.

6.2 Related Work

The importance of action duration in reinforcement learning (RL) has been significantly underestimated, yet it is crucial for applying RL algorithms in real-world scenarios, impacting an agent's exploration capabilities. For instance, high frequencies may reduce exploration efficiency but are essential in delayed environments [81]. Recent studies by Amin et al. [12], and Park et al. [13] have highlighted this issue, showing that variable action durations can significantly affect learning performance. Building on these insights, Wang & Beltrame [156] and Karimi et al. [82] further explored how different frequencies impact learning, noting that excessively high frequencies can impede convergence. Their findings suggest that dynamic control frequencies, which adjust in real-time based on reactive principles, could enhance performance and adaptability.

Expanding on the idea of adaptive control, Sharma et al. [83] introduced a related concept of repetitive action reinforcement learning, where an agent performs identical actions over successive states, combining them into a larger action. This concept was further explored by Metelli et al. [85] and Lee et al. [86] in gaming contexts. However, despite its potential, this method does not fully address physical properties or computational demands, limiting its practical application.

Additionally, Chen et al. [84] proposed modifying the traditional control rate by integrating actions such as "sleep" to reduce activity periods. However, this approach still required fixed-frequency system checks, ultimately failing to reduce computational load as intended.

In the domain of traffic signal control, research focused on hybrid action reinforcement learning. The study highlighted the importance of synchronously optimizing stage specifications and green interval durations, underscoring the potential of variable action durations to improve decision-making processes in complex, real-time environments [162].

Furthermore, research on variable damping control for wearable robotic limbs has showcased the application of reinforcement learning to adjust control parameters adaptively in response to changing states. This study illustrated the effectiveness of RL in maintaining system stability and performance under varying conditions, reinforcing the value of dynamic control frequencies in practical applications [15].

These studies collectively highlight the ongoing efforts to integrate variable action durations and adaptive control mechanisms into RL. They also underscore the nascent stage of effectively integrating variable control frequencies and repetitive behaviors into practical applications, emphasizing their critical importance for enhancing algorithm performance and applicability in real-world scenarios.

6.3 Algorithm Framework

Soft Elastic Actor-Critic (SEAC) is an extension of the Soft Actor-Critic (SAC) algorithm that addresses the limitations of fixed control rates in reinforcement learning (RL) [155]. Traditional Markov Decision Processes (MDPs) in RL do not account for the duration of actions, assuming that all actions are executed over uniform time steps. This can lead to inefficiencies, as the time between two actions can vary widely, requiring a fixed-frequency control rate in practical deployments. SEAC breaks this assumption by dynamically adjusting the duration of each action based on the state and environmental conditions, following the principles of reactive programming. By incorporating action duration D into the action set, SEAC can decide on both the action and its duration to optimize energy consumption and computational efficiency.

MOSEAC extends SEAC [155] and combines its hyperparameters for balancing task, energy, and time rewards, and provides a method for automatically adjust its other hyperparamenters during training. Similar to SEAC, MOSEAC reward 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 8. *The reward associated with the state space is:*

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

where R_t is the task reward and 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 [163], 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 is designed to minimize 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 9. *The remap relationship between action duration and reward is:*

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

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

We automatically set α_m and α_ε during training. Based on previous results [156], we bind the increase of α_m to a decrease α_ε using a sigmoid function to mitigate convergence issues, specifically the problem of sparse rewards caused a large α_ε and a small α_m. This adjustment ensures that rewards are appropriately balanced, facilitating learning and convergence.

Definition 10. *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 [155]'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.

Overall, we update α_m based on the current trend in average reward during training. To guarantee stability, we ensure the change is monotonic, forcing a uniform sweep of the parameter space, and a maximum value of α_m, namely α_{max}. To determine the trend in 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 11. *The slope of the average reward (k_R) is:*

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

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 [29]. 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.

Definition 12. *We adaptively adjust the reward every k_{update} episodes. When $k_R < 0$:*

$$\begin{cases} \alpha_m = \alpha_m + \psi & \textit{if } \alpha_m < \alpha_{max} \\ \alpha_m = \alpha_{max} & \textit{otherwise} \end{cases}$$

where ψ serves as the sole additional hyperparameter necessary for adjusting the reward equation in our MOSEAC method during training. The parameter α_{max} represents the upper limit of α_m, ensuring algorithmic convergence and preventing reward explosion. Furthermore, α_ε is adjusted in accordance with Definition 10.

Algorithm 3 shows the pseudocode of MOSEAC. In short, MOSEAC extends the SAC algorithm by incorporating action duration D into the action policy set. This expansion allows the algorithm to predict the action and its duration simultaneously. The reward is calculated using 8, and its changes are continuously monitored. If the reward trend declines, α_m increases linearly at a rate of ψ, without exceeding $alpha_{max}$. The action and critic networks are periodically updated like the SAC algorithm based on these preprocessed rewards.

Here, t_{max} represents the maximum number of training steps [29]; k_{length} is the maximum number of exploration steps per episode [29]; k_{init} is the number of steps in the initial random exploration phase [29]. The reward $R(S_i, A_i, D_i)$ depends on state (S_i), action (A_i) and duration $D_i \in [D_{min}, D_{max}]$.

Overall, our reward scalarizes a multi-objective optimization problem including rewards for task, time, and energy. Unlike Hierarchical Reinforcement Learning (HRL) [90, 91], which seeks Pareto optimality [115, 116] with layered reward policies, our method simplifies the approach, making our strategy easily adaptable to various algorithms. While our goal is to avoid the complexity of Pareto optimization, dynamically adjusting the reward structure inevitably places us on the Pareto optimization curve. For a theoretical analysis of the Pareto curve and how α_{max} ensures algorithmic stability, please refer to Appendix I.

Algorithm 3: Multi-Objective Soft Elastic Actor and Critic (MOSEAC)

```
Require: a policy $\pi$ with a set of parameters $\theta$, $\theta^{'}$, critic parameters $\phi$, $\phi^{'}$, variable time
         step environment model $\Omega$, learning-rate $\lambda_p$, $\lambda_q$, reward buffer $\beta_r$, replay buffer
         $\beta$.
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
        $A_i, D_i = \pi_\theta(S_i)$
                                                  $\rightarrow$ simple action and its duration
        $S_{i+1}, R_i = \Omega(A_i, D_i)$
                                                  $\rightarrow$ compute reward with Definition 3 and 4
        $i \leftarrow i + 1$
    end
    $\beta_r \leftarrow 1/i \times \sum_0^i R_i$
                                                  $\rightarrow$ 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 $t_i \geq k_{init} \quad \& \quad t_i \mid k_{update}$ then
        $Sample\, S,\, A,\, D,\, R,\, S^{'} from(\beta)$
        $\phi \leftarrow \phi - \lambda_q \nabla_\delta \mathcal{L}_Q(\phi,\, S,\, A,\, D,\, R,\, S^{'})$
                                                  $\rightarrow$ critic update
        $\theta \leftarrow \theta - \lambda_p \nabla_\theta \mathcal{L}_\pi(\theta,\, S,\, A,\, D,\, \phi)$
                                                  $\rightarrow$ actor update
        if $k_R(\beta_r)$ then
            $\alpha_m = \alpha_m + \psi$                  if$\alpha_m < \alpha_{max}$
            Or $\alpha_m = \alpha_{max}$                  otherwise
                                                  $\rightarrow$ see Definition 6 for $k_R$
            $\alpha_\varepsilon \leftarrow F_{update}(\alpha_m)$
                                                  $\rightarrow$ update $\alpha_m, \alpha_\varepsilon$ followed Definition 5
        end
        $\beta_r = 0$
                          $\rightarrow$ Re-record average reward values under new hyperparameters
    end
    Perform soft-update of $\phi^{'}$ and $\theta^{'}$
end
```

Apart from a series of hyperparameters inherent to RL that need adjustment, such as learning rate, t_{max}, k_{update}, etc., ψ is the primary hyperparameter that requires tuning in MOSEAC. Determining the appropriate ψ value remains a critical optimization point. We recommend using the pre-set ψ value provided in our implementation to reduce the need for additional adjustments. If training performance is poor, a large ψ may cause the reward signal's gradient to change too rapidly, leading to instability, suggesting a reduction in ψ. Conversely, if training is slow, a small ψ may result in a weak reward signal, affecting convergence, suggesting an increase in ψ.

6.4 Theoretical Analysis

We have analyzed the theoretical performance of MOSEAC through various aspects, including performance guarantees, convergence and complexity analysis. Table 6.1 provides the notation for the following.

6.4.1 Performance Guarantees

In the standard SAC algorithm [45], 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.

When the action space is extended to include the action duration 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:

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

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 3.

Replacing Q^π the objective function for MOSEAC, incorporating the extended action space and the adaptive reward function, becomes:

$$J(\pi) = \mathbb{E}_{(s_t,a_t,D_t)\sim\rho_\pi}\Bigg[\sum_{t=0}^{\infty}\gamma^t\Big(\alpha_m(t)R_t\frac{D_{\min}}{D_t} \\ - \alpha_\varepsilon(t) + \alpha\mathcal{H}(\pi(\cdot|s_t))\Big)\Bigg]$$

where:

- $\alpha_m(t)$ is the adaptive parameter that increases monotonically when the average reward decreases, with an upper limit of α_{max}.
- $\alpha_\varepsilon(t)$ is the adaptive parameter that decreases monotonically when the average reward decreases.
- $\mathcal{H}(\pi(\cdot|s_t)) = -\mathbb{E}_{(a_t,D_t)\sim\pi(\cdot|s_t)}[\log\pi(a_t, D_t|s_t)]$ is the entropy of the policy.

The SAC policy improvement theorem [45] ensures that

$$Q^{\pi_k}(s,a,D) \geq Q^{\pi_{k-1}}(s,a,D)$$

for any iteration $k \geq 0$.

The soft Bellman equation in MOSEAC is:

$$Q^\pi(s,a,D) = \alpha_m(t)R_t\frac{D_{\min}}{D} - \alpha_\varepsilon(t) \\ + \gamma\mathbb{E}_{s'\sim P(\cdot|s,a,D)}\left[V^\pi(s')\right]$$

where the value function $V^\pi(s)$ is defined as:

$$V^\pi(s) = \mathbb{E}_{(a,D)\sim\pi(\cdot|s)}\Big[Q^\pi(s,a,D) \\ - \alpha\log\pi(a,D|s)\Big]$$

To prove convergence, we need to show that the soft Bellman operator is a contraction mapping. Let $\mathcal{T}$ be the soft Bellman operator. For any two Q-functions Q_1 and Q_2:

$$\begin{aligned}
\|\mathcal{T}Q_1 - \mathcal{T}Q_2\|_\infty = \sup_{s,a,D} \Big| & \alpha_m(t) R_t \frac{D_{\min}}{D} - \alpha_\varepsilon(t) \\
& + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a,D)} [V_1(s')] \\
& - \Big(\alpha_m(t) R_t \frac{D_{\min}}{D} - \alpha_\varepsilon(t) + \gamma \mathbb{E}_{s' \sim P(\cdot|s,a,D)} [V_2(s')] \Big) \Big| \\
\leq \sup_{s,a,D} \gamma \Big| & \mathbb{E}_{s' \sim P(\cdot|s,a,D)} [V_1(s') - V_2(s')] \Big| \\
\leq \gamma & \|V_1 - V_2\|_\infty \\
\leq \gamma & \|Q_1 - Q_2\|_\infty
\end{aligned}$$

Since $\gamma < 1$, the soft Bellman operator $\mathcal{T}$ is a contraction mapping. By Banach's fixed-point theorem, there exists a unique fixed point Q^* such that $Q^* = \mathcal{T}Q^*$ [164].

Let π^* be the optimal policy and π_k be the policy at iteration k. The error bound between the value functions of the optimal and learned policies is

$$\|V^{\pi^*} - V^{\pi_k}\| \leq \frac{2\alpha\gamma \log |\mathcal{A} \times [D_{\min}, D_{\max}]|}{(1-\gamma)^2}$$

With $|\mathcal{A} \times [D_{\min}, D_{\max}]|$ the size of the extended action space, the contraction property of the soft Bellman operator ensures that

$$\|V^{\pi^*} - V^{\pi_k}\|_\infty \leq \frac{2\alpha\gamma \log |\mathcal{A} \times [D_{\min}, D_{\max}]|}{(1-\gamma)^2}$$

where α is the coefficient of the entropy term and γ is the discount factor.

6.4.2 Convergence Analysis

With our reward function, the policy gradient is:

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

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

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

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

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.

The new policy parameter θ update rule is:

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

where β_k is the learning rate at step k.

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, and its upper limit is α_{max}, which guarantees algorithmic convergence and prevents reward explosion. α_ε decreases as defined.

2. **Learning Rate Conditions:** α_k and β_k must satisfy [41]:

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

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

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.

1. **Positive Scaling:** As $0 \le R_\tau \le 1$ and $\alpha_m \ge 0$, α_m only scales the reward without 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.

Under these conditions, MOSEAC will converge to a local optimum [29]:

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

6.4.3 Complexity Analysis

As MOSEAC closely follows SAC [45], but with an expanded action set that includes durations, its time complexity is similar to SAC, replacing the action set A with the expanded one $A \times D$. SAC and MOSEAC have 3 computational components:

- **Policy Evaluation:** consists of computing the value function $V^\pi(s)$ and the Q-value function $Q^\pi(s, a)$ for each state-action pair, i.e. $O(|\mathcal{S}| \cdot |\mathcal{A} \times D|)$.
- **Policy Improvement:** consists updating the policy parameters θ based on the policy gradient, involving the Q-value function and the entropy term. The time complexity is $O(|\mathcal{S}| \cdot |\mathcal{A} \times D|)$.
- **Value Function Update:** consists of computing the target value using the Bellman backup equation, with a time complexity of $O(|\mathcal{S}| \cdot |\mathcal{A} \times D|)$.

The overall time complexity for one iteration of the SAC algorithm is therefore

$$O(|\mathcal{S}| \cdot |\mathcal{A} \times D|)$$

The space complexity of MOSEAC is determined by the storage requirements for the policy, value functions, and other necessary data structures. As for the time complexity, MOSEAC follows SAC with a different action set:

- **Policy Storage**: Requires $O(|\theta|)$ space for the policy parameters.
- **Value Function Storage**: Requires $O(|\mathcal{S}| \cdot |\mathcal{A}| \cdot |D|)$ space for the value functions.

The overall space complexity is:

$$O(|\theta| + |\mathcal{S}| \cdot |\mathcal{A}| \cdot |D|)$$

The dynamic adjustment of α_m and α_ε adds a small computational overhead of $O(1)$ per update due to simple arithmetic operations and comparisons.

In summary, MOSEAC has higher computational complexity than SAC, but its dynamic adjustment mechanism and expanded action space provide greater adaptability and potential for improved performance in multi-objective optimization scenarios.

Table 6.1 Complexity Analysis Notation

Symbol	Description
$\mathcal{S}$	Set of states
$\mathcal{A}$	Set of actions
D	Action durations set
π_θ	Policy parameters θ
$\lvert\theta\rvert$	Policy parameter count
$V^\pi(s)$	Value function
$Q^\pi(s, a, D)$	Q-value function with D
α	Entropy temperature param.
α_m	Reward scaling param.
α_ϵ	Reward offset param.
R_t	Reward at time t
R_τ	Time-based reward
γ	Discount factor
ϕ	Value function params.
β_k	Learning rate at step k
$\mathcal{T}$	Soft Bellman operator
$\mathcal{H}(\pi(\cdot\lvert s))$	Policy entropy
ρ_π	State-action distribution
O	Complexity upper bound

6.5 Experiment

We conducted a systematic evaluation and validation of MOSEAC's performance on task completion ability (e.g., navigating to target locations, avoiding yellow lines), resource consumption (measured by the number of steps required to complete the task), and time cost. Additionally, we assessed the trajectory and control output similarity to report the differences between simulation and reality. Finally, we compared the computing resource usage between variable and fixed RL models.

Figure 6.1 illustrates our workflow. Our simulation to reality approach involves a process to ensure the reliability of MOSEAC when applied to a real-world Limo. The workflow includes the following steps:

1. **Manual Control and Data Collection**: Initially, we manually control the Limo to collect a dataset of its movements.
2. **Supervised Learning**: This dataset is then used to train an environment model in a supervised learning manner.
3. **Reinforcement Learning**: The trained environment model is applied in the reinforcement learning environment to train the MOSEAC model.
4. **Validation and Fine-Tuning**: The MOSEAC model is applied to the real Limo. Fine-tuning is performed based on the recorded movement data to ensure accurate translation from simulation to real-world application.
5. **Application**: Finally, the refined MOSEAC model is deployed for practical use and validated through real-world tasks.

6.5.1 Real Environment Set Up

Figure 6.2 shows Limo's real-world validation environment. We used an OptiTrack [2] system for real-time positioning. We use a 3x3 meter global coordinate frame with its center aligned with the center of the 2D map, recording the positions of yellow lines within the map. The Limo's navigable area is confined within the map boundaries, excluding four enclosed regions. Specific values of these positions, the specifications of the Agilex Limo, and other metrics can be found in Appendix C.

6.5.2 Simulation Environment

For our kinematic modeling, we use an Ackerman model:

Definition 13. *The Ackerman kinematic model.*

Table 6.2 gives the symbols and descriptions.

The forces and accelerations are calculated as:

$$\begin{aligned}
F_{friction} &= -\mu_k M g \\
a_{friction} &= \frac{F_{friction}}{M} = -\mu_k g \\
a_{power} &= \frac{P}{M} \\
a_{net} &= a_{power} + \frac{V_{target} - V}{\Delta t} + a_{friction}
\end{aligned}$$

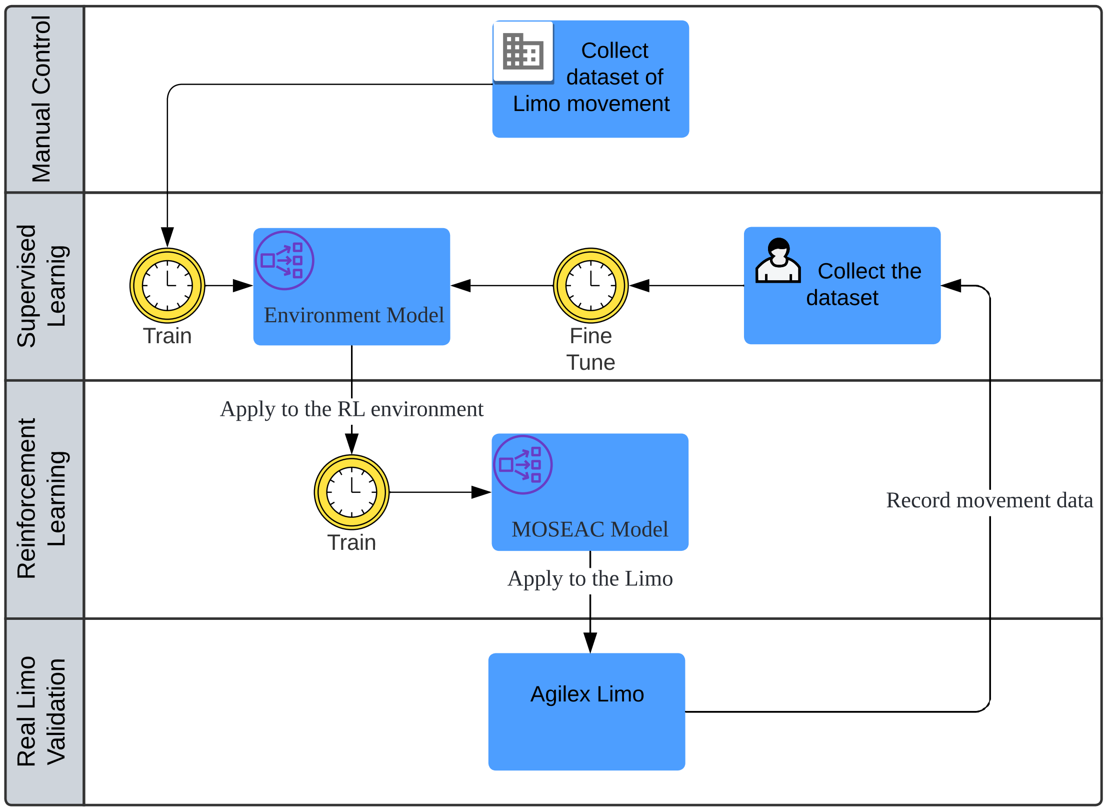


Figure 6.1 The workflow for our MOSEAC implementaion to the Agilex Limo, we use a joystick to control the Limo movement for the initial environmental data collection (top).

The updates for velocity and position are:

$$V_{new} = V + a_{net}\Delta t$$
$$X_{new} = X + V_{new}\cos(\theta_{new})\Delta t$$
$$Y_{new} = Y + V_{new}\sin(\theta_{new})\Delta t$$

This formulation accounts for the dynamics of the Limo vehicle, considering friction, power, and vehicle mass, ultimately predicting the new position. This kinematic model is referred tp as $M_{Ackerman}$.

We employ supervised learning [165, 166] to model the motion dynamics from the real-world environment to the simulation environment. This model predicts kinematic and physical information,

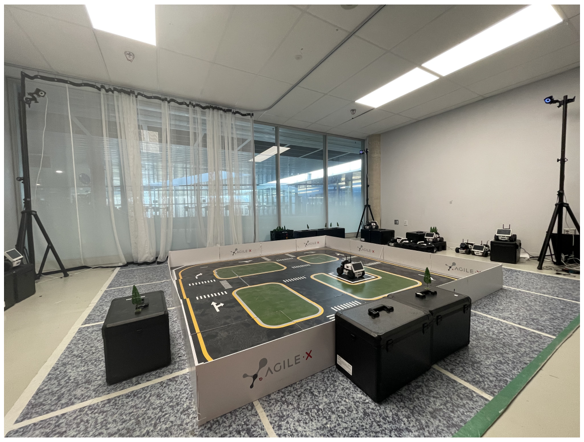


Figure 6.2 This photo depicts the real-world environment used to validate the performance of MOSEAC on the Agilex Limo. The cameras on the left, right, and middle stands are three of the four cameras comprising the OptiTrack positioning system.

Table 6.2 List of Symbols and Descriptions

Symbol	**Description**
X and Y	Current positions of Limo
V	Current velocity of Limo
θ	Current angular velocity of Limo
Δt	Control duration
V_{target}	Control linear velocity
δ	Control angular velocity
μ_k	Coefficient of kinetic friction
P	Power factor (which can be negative) of Limo
$g = 9.81\,\text{m/s}^2$	Gravitational acceleration
$M = 4.2\,\text{kg}$	Mass of the Limo (measured)
$L = 0.204\,\text{m}$	Distance between the centers of the front and rear wheels (measured)

μ_k and P, across different regions within the environment. We select a Transformer Model [1] [167] to predict these data. The input, output, and its loss function are defined in Definition 14.

[1]Our code is publicly available on GitHub with the shape and related hyperparameters of this Transformer Model inside.

Definition 14. *The input* $\mathbf{X}$ *is:*

$$\mathbf{X} = \left(X, Y, V, \theta, \Delta t, V_{target}, \delta\right)$$

The output $\mathbf{O}$ *is:*

$$\mathbf{Y} = \left(\mu_k, P\right)$$

Using $M_{Ackerman}$ *from Definition 13, the loss function of the Transformer model* T *is computed as:*

$$\begin{aligned} \textit{predict} &= M_{Ackerman}(\mathbf{X}, \mathbf{Y}, g) \\ \textit{loss} &= \frac{1}{N}\sum_{i=1}^{N}\left(\textit{predict}_i - \textit{target}_i\right)^2 \end{aligned}$$

The loss function is defined as the Mean Squared Error (MSE) between the predicted positions and the target positions recorded in the dataset:

where:

- $\textit{predict}_i$ *are the predicted positions for the* i*-th data point*
- $\textit{target}_i$ *are the actual positions for the* i*-th data point*

After learning the kinematic model, we need to address the state definition of MOSEAC. It should include the necessary environmental information to enable the model to understand the positions of obstacles in the environment. Although we have recorded all the positions of these yellow lines, it is not reasonable to input them directly as state values. This would compromise the generality of our approach, and a large amount of invariant fixed position information might cause the neural network processing units to struggle with capturing the critical information steps or lead to overfitting [168, 169].

We developed a simulated lidar system centered on Limo's current position. This system generates 20 rays, similar to lidar operation, to detect intersections with enclosed regions in the environment. These intersections provide information about restricted areas, as shown in Figure 6.3.

Since the enclosed regions data in our simulation are based on real-world measurements, the gap between the virtual and real world is negligible. We designated eight turning points on this map as navigation endpoints, with a specific starting point for Limo.

The state dimensions for our MOSEAC model include: • robot current position, • goal position (chosen among 8 random locations), • linear velocity, • steering angle, • 20 lidar points, and • pre-

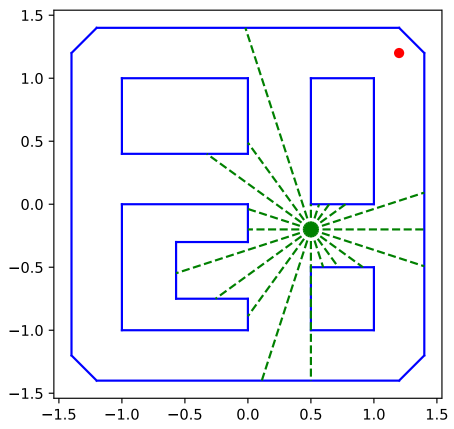


Figure 6.3 The simulated lidar system generates 20 rays from Limo's position, calculates their intersections with environment enclosed regions, and returns the nearest intersection points for each ray.

vious control duration, linear and angular velocity. The action dimensions include control duration, linear velocity, and angular velocity.

Let R_t represent the task reward and T denote the termination flag (where 1 indicates termination and 0 indicates continuation). The function $d(\mathbf{p}_1, \mathbf{p}_2)$ measures the distance between positions $\mathbf{p}_1$ and $\mathbf{p}_2$. The variables $\mathbf{a}_{\text{new}}$ and $\mathbf{t}$ denote the new position of the agent and the target position, respectively. The initial distance from the starting position to the target position is d_0, and δ specifies the threshold distance determining whether Limo is sufficiently close to a point. Additionally, C_{inner} indicates a collision with enclosed regions, while C_{outer} indicates a collision with the map boundary.

Implementing a penalty mechanism instead of a termination mechanism in our reward definition offers significant advantages: the penalty mechanism provides incremental feedback, allowing the model to continue learning even after errors, thereby avoiding frequent resets that could disrupt the training process. This approach reduces overall training time and encourages the agent to explore a broader range of strategies, leading to a more comprehensive understanding of the environment's dynamics. However, we still terminate an experiment if the robot wonders outside the map boundary to stop meaningless exploration. The reward R_t is:

$$R_t = \begin{cases} R_{\text{boundary\_penalty}}, & \text{if } C_{\text{inner}}(\mathbf{a}_{\text{new}}) \\ R_{\text{success\_reward}}, & \text{if } d(\mathbf{a}_{\text{new}}, \mathbf{t}) \leq \delta \\ R_{\text{failure\_penalty}}, & \text{if } C_{\text{outer}}(\mathbf{a}_{\text{new}}) \\ R_{\text{distance\_reward}} = d_0 - d(\mathbf{a}_{\text{new}}, \mathbf{t}), & \text{otherwise} \end{cases}$$

The termination flag T is determined as:

$$T = \begin{cases} 0, & \text{if } C_{\text{inner}}(\mathbf{a}_{\text{new}}) \\ 1, & \text{if } d(\mathbf{a}_{\text{new}}, \mathbf{t}) \leq \delta \\ 1, & \text{if } C_{\text{outer}}(\mathbf{a}_{\text{new}}) \\ 0, & \text{otherwise} \end{cases}$$

The initial data collected using remote control was skewed for short action durations, causing some discrepancies between the physical robot behaviour and the behaviour in simulation. To address these discrepancies we fine-tuned a Transformer model that approximates Limo's kinematic model by collecting new real-world movement data.

We begin by freezing all the layers of the Transformer model except the final fully connected layer. This ensures that the initial generalized features are preserved while adapting the model's outputs to the specific characteristics of the Limo dataset. The training process is:

1. **Freeze Initial Layers**: Initially, all layers except the final fully connected layer are frozen. θ_i represents the parameters of the i-th layer:

$$\text{if } i < N_{\text{unfrozen}}, \quad \nabla_{\theta_i} L = 0$$

where L is the loss function and N_{unfrozen} is the number of layers that are not frozen.

2. **Gradual Unfreezing**: After training the final layer, we incrementally unfreeze the preceding layers and fine-tune the model further. This is done iteratively, where at each stage, one additional layer is unfrozen and the model is re-trained. The number of unfrozen layers increases until all layers are eventually fine-tuned:

$$\text{if } i \geq N_{\text{unfrozen}}, \quad \nabla_{\theta_i} L \neq 0$$

3. **Early Stopping**: To prevent overfitting, we employ early stopping. If the validation loss does not improve for a specified number of epochs, training is halted:

$$\text{if } \Delta L_{\text{valid}} > 0 \text{ for } n \text{ epochs}, \quad \text{stop training}$$

where ΔL_{valid} is the change in validation loss and n is the patience parameter.

4. **Evaluation**: The model's performance is evaluated on a test dataset after each training phase. The final model is selected based on the lowest validation loss.

The total training process can be described by the following optimization problem:

$$\min_{\theta} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} \mathcal{L}(f_\theta(x), y)$$

where $\mathcal{D}_{\text{train}}$ is the training dataset, f_θ is the model with parameters θ, and $\mathcal{L}$ is the loss function 14. The fine-tuning process specifically involves updating the parameters layer by layer, ensuring that the generalized pre-trained knowledge is gradually adapted to the specific nuances of the new Limo dataset.

This fine-tuning strategy allows the use of pre-trained models, adapting them to new tasks with potentially smaller datasets while maintaining robust performance and preventing overfitting.

For more detailed information please refer to Appendix D.

6.6 Results

We conducted six experiments involving MOSEAC, MOSEAC without the α_{max} limitation, SEAC, CTCO, and SAC (at 20 Hz and 60 Hz, SAC20 and SAC60, respectively) within the simulation environment and applied these trained models to the real Limo. These training tests were performed on a computer equipped with an Intel Core i5-13600K CPU and an Nvidia RTX 4070 GPU running Ubuntu 22.04 LTS. The deployment tests were conducted on an Agilex Limo equipped with a Jetson Nano [170] (with a Cortex-A57 CPU and Maxwell 128 cores GPU) running Ubuntu 18.04 LTS.

We trained our RL model using PyTorch [171] within a Gymnasium environment [172]. Subsequently, we converted the model parameters to ONNX format [171]. Finally, we utilized TensorRT [173] to build an inference engine from the ONNX model in the AgileX Limo local environment and deployed it for RL navigation tasks [2]. More system information details can be found in Appendix E.

Figure 6.4 and Figure 6.5 illustrate the results of the training process. Several key insights can be drawn from these figures:

Figure 6.4 indicates that MOSEAC consistently improves over time compared to the other algorithms, suggesting that MOSEAC adapts well to the environment and maintains a stable learning curve. While SEAC also shows stable performance, it converges slightly slower than MOSEAC. Additionally, Figure 6.4 illustrates that MOSEAC exhibits higher action duration robustness than CTCO. MOSEAC utilizes a fixed discount factor (γ), ensuring stable long-term planning capabilities without the negative impact of varying γ. In contrast, CTCO's performance is susceptible to the choice of action duration range, significantly affecting its γ. As training progresses, CTCO tends

[2]Our code is publicly available on GitHub with the hyperparameters of the MOSEAC model inside.

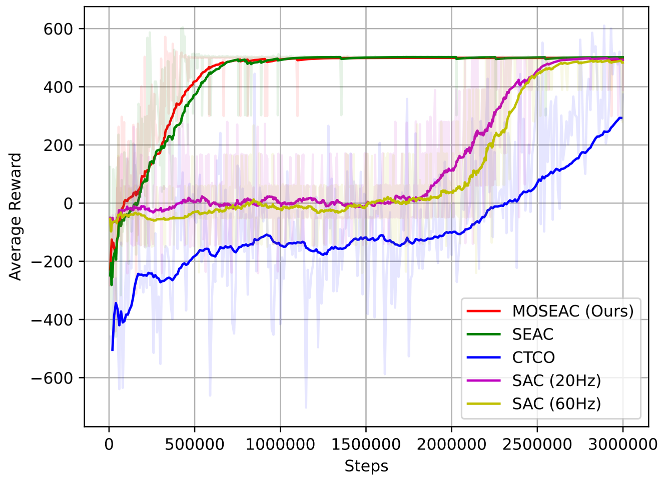


Figure 6.4 Average returns of 5 reinforcement learning algorithms over 2.5M steps during training.

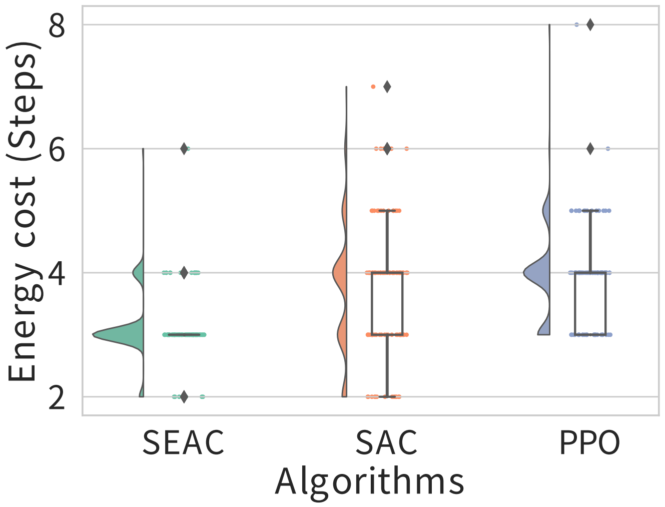


Figure 6.5 Average energy costs of 5 reinforcement learning algorithms over 2.5M steps during training.

to favor smaller γ values, placing greater demands on its τ parameter that controls the range of γ. This sensitivity makes CTCO less adaptable to diverse environments, whereas MOSEAC maintains consistent performance. Furthermore, Figure 6.5 provides a comparative analysis of energy costs among the different algorithms. MOSEAC demonstrates lower energy costs per task, indicating higher energy efficiency during training. This efficiency is critical for practical applications where compute resource consumption is a concern.

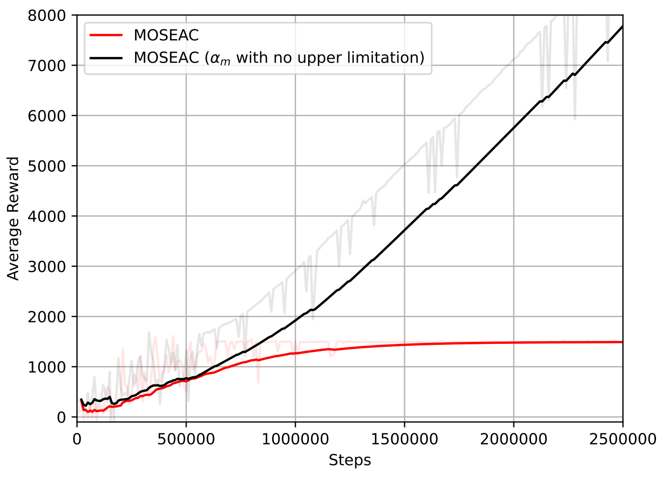


Figure 6.6 Average returns of MOSEAC and MOSEAC (without α_m limitation) in 2.5M steps during the training.

We compared the effects of imposing an upper limit on the parameter α_m versus allowing it to increase without restriction in the context of our MOSEAC algorithm. Our findings indicate significant differences in performance stability and energy efficiency between the two approaches.

Introducing an upper limit on α_m (denoted as α_{max}) is required to prevent reward explosion. Figure 6.6 shows that MOSEAC has consistent and stable improvement in average reward. In contrast, MOSEAC without the upper limit initially followed a similar trend but eventually diverged, leading to instability and potential reward explosion. This divergence suggests that without the upper limit, α_m may increase uncontrollably, destabilizing the reward structure.

However, as shown in Figure 6.7, MOSEAC without an upper limit on α_m generally exhibited lower average energy costs compared to MOSEAC with the upper limit. This suggests that while the absence of an upper limit on α_m may lead to reward explosion, it does not significantly impair task performance in practice.

We update α_m based on the declining reward trend. However, in random multi-objective tasks with varying target locations, the average reward can fluctuate significantly, causing α_m to be updated continuously. This can lead to an unbounded increase in reward during training when α_m is unrestricted, amplifying the reward signal. The increased gradient variability results in rapid strategy adjustments, enhancing short-term energy efficiency. Despite the potential for reward explosion, the impact on task performance is minimal, with notable improvements in energy efficiency.

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.

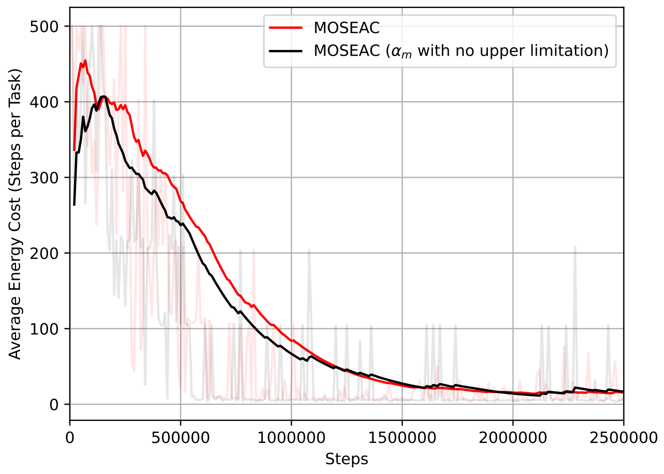


Figure 6.7 Average Energy cost of MOSEAC and MOSEAC (without α_m limitation) in 2.5M steps during the training.

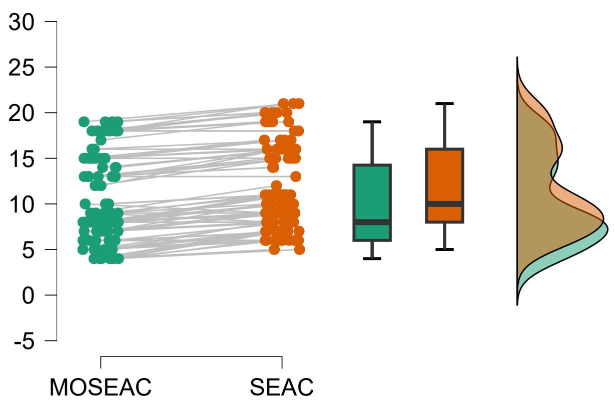


Figure 6.8 Energy cost (as number of time steps) for 100 random tasks. We use the same seed for MOSEAC and SEAC to ensure that the tasks are the same for the two algorithms.

Figure 6.8 and Figure 6.9 illustrate the energy and time distributions for MOSEAC and SEAC methods. Given the lack of normality in the data distribution, we use the Wilcoxon signed-rank test to compare the paired samples. For energy consumption (Figure 6.8), MOSEAC demonstrates significantly lower median and overall energy usage than SEAC ($W = 35.0, z = -7.904, p < .001$). For time efficiency (Figure 6.9), MOSEAC also shows a significantly lower median time, indicating quicker task completion than SEAC ($W = 502.0, z = -6.956, p < .001$). The descriptive statistics and test results are summarized in Appendix J.

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 inde-

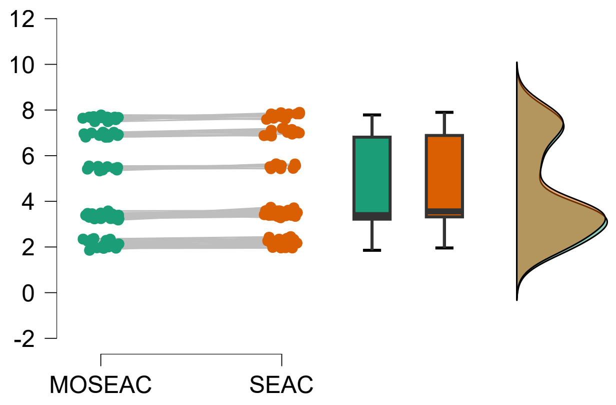


Figure 6.9 Time cost for 100 random tasks. We use the same seed for MOSEAC and SEAC to ensure that the tasks are the same for the two algorithms.

pendently, 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.

We conducted real-world tests on the Agilex Limo using ROS [174][3]. The Limo robot navigated to random and distinct endpoints using the MOSEAC model as its control policy for Ackerman steering [4]. We collected these data and calculated trajectory and control output similarities with respect to simulation.

For trajectory similarity, we comine all trajectory data in to one trajetory, the ATE (Average Trajectory Error)[177] is 0.036 meters, indicating minimal deviations between the actual and simulated paths. Additionally, the Dynamic Time Warping (DTW) [178] value of 0.531 supports the high degree of similarity between the temporal sequences of the trajectories. These metrics suggest that our method enables the Limo to follow the planned paths with high precision, closely mirroring the simulation.

Regarding control output similarity, the Mean Absolute Error (MAE) is 0.002, and the Mean Squared Error (MSE) is $4.49E-05$. These low error values indicate that the control outputs in the real world closely match those in the simulation. Our method effectively minimizes the

[3]Our code for ROS workspace is publicly available on GitHub, which also provides support for ROS 2 [175] and Docker [176]

[4]Video avaliable here: `https://youtu.be/VhTa66WqxoU`

discrepancies in control signals, ensuring that the robot's actions are consistent across different environments.

Overall, the empirical data supports the theoretical claims regarding MOSEAC's performance in trajectory fidelity and control output consistency.

We also recorded the computing resource usage, highlighting the efficiency of the MOSEAC algorithm in terms of energy consumption, particularly computational energy. Figure 6.10 provides a comparison of the average usage per second of CPU and GPU resources between MOSEAC and SAC (Soft Actor-Critic) algorithms running at different frequencies (10 Hz and 60 Hz).

The CPU usage data reveals that MOSEAC significantly reduces computational load compared to SAC at both 10 Hz and 60 Hz. Specifically, MOSEAC utilizes only $11.40\% \pm 0.12$ of the CPU resources, whereas SAC requires $16.80\% \pm 0.14$ at 10 Hz and $31.41\% \pm 1.47$ at 60 Hz. Similarly, the GPU usage data indicates that MOSEAC is more efficient, using only $2.80\% \pm 0.07$ compared to SAC's $13.79\% \pm 0.05$ at 10 Hz and $27.86\% \pm 0.19$ at 60 Hz. This reduction conserves energy (increasing battery life) and frees processing power for tasks like perception and communication. The descriptive statistics and test results are summarized in Appendix K.

6.7 Conclusions

In this paper, we presented the Multi-Objective Soft Elastic Actor-Critic (MOSEAC) algorithm, a Variable Time Step Reinforcement Learning (VTS-RL) method designed with adaptive hyperparameters that respond to observed reward trends during training. Our analysis included theoretical performance guarantees, convergence analysis, and practical validation through simulations and real-world navigation tasks using a small rover.

We compared MOSEAC with other VTS-RL algorithms, such as SEAC [155] and CTCO [82]. While SEAC and CTCO improved over traditional fixed-time step methods, they still required extensive hyperparameter tuning and did not achieve the same level of efficiency and robustness as MOSEAC. SEAC's reward structure, though effective, was less adaptable, and CTCO's sensitivity to action duration further limited its practical application.

Additionally, the empirical data demonstrated that MOSEAC significantly outperforms traditional SAC algorithms, particularly in terms of computational resource efficiency. MOSEAC's reduced CPU and GPU usage frees up resources for other critical tasks, such as environment perception and map reconstruction, enhancing the robot's operational efficiency and extending battery life. This makes MOSEAC highly suitable for long-term and complex missions.

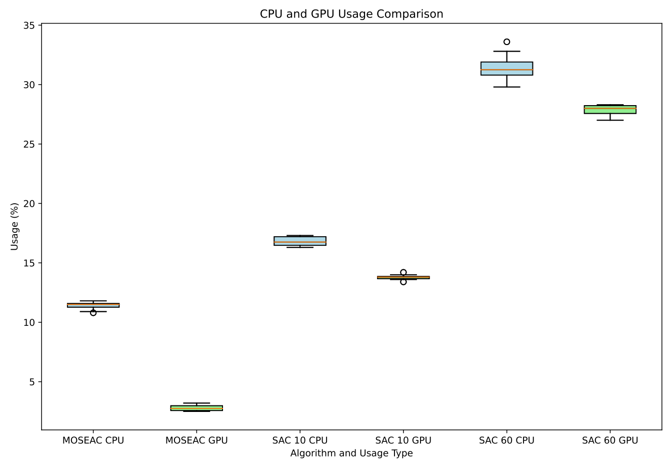


Figure 6.10 Comparison of Average Compute Resource Usage Across Different Methods

Our findings validate the robustness and applicability of MOSEAC in real-world scenarios, providing strong evidence of its potential to lower hardware requirements and improve data efficiency in reinforcement learning deployments.

Our future work will further refine the algorithm, particularly in the adaptive tuning of hyperparameters, to enhance its performance and applicability. We aim to apply MOSEAC to a broader range of robotic projects, including smart cars and robotic arms, to fully leverage its benefits in diverse practical settings.

CHAPTER 7 GENERAL DISCUSSION

7.1 Summary of Works

7.1.1 Findings and Contributions

The Soft Elastic Actor-Critic (SEAC) algorithm represents a significant advancement in reinforcement learning by introducing the concept of Variable Time Step Reinforcement Learning (VTS-RL). As the first VTS-RL algorithm, the significance of SEAC is manifold. SEAC breaks the limitations of fixed frequency control by allowing the control frequency to adapt dynamically based on task requirements. By incorporating task energy and time rewards into the reward function, SEAC ensures that the algorithm not only focuses on task completion but also on minimizing the time and energy required. This multi-objective optimization improves task efficiency and energy utilization without adding computational complexity. As the first VTS-RL algorithm, SEAC sets the groundwork for subsequent research in this area. It establishes a theoretical basis and reference framework for future algorithms like MOSEAC. SEAC's approach of balancing task, time, and energy rewards highlights the potential of multi-objective optimization in reinforcement learning, which can be extended to other domains, driving further advancements in RL.

Building on the foundation of SEAC, the Multi-Objective Soft Elastic Actor-Critic (MOSEAC) algorithm further refines the VTS-RL concept. MOSEAC reduces the complexity of hyperparameter tuning by requiring only a single hyperparameter to guide exploration. This is achieved through an adaptive reward mechanism that dynamically adjusts based on observed task rewards. Validation in a Newtonian kinematics environment demonstrated that MOSEAC achieves higher task performance and training efficiency with fewer time steps and lower energy consumption. This indicates that MOSEAC accelerates the training process while maintaining high task performance.

The application of MOSEAC in the Agilex Limo robot represents a culmination of the research, showcasing its real-world efficacy and providing a demonstration of its capabilities. The MOSEAC was validated through both physical simulations and real-world experiments using the Agilex Limo robot. These tests involved navigating various target positions and complex environments. The experiments were meticulously documented, including hardware and software configurations, ensuring that other researchers could replicate the study. The results confirmed that MOSEAC effectively completes navigation tasks with energy efficiency and task completion times compared to fixed time step algorithms. This highlights the algorithm's robustness and adaptability in real-world scenarios. MOSEAC demonstrated significant improvements in reducing energy consumption and optimizing task performance, making it highly suitable for practical applications. The mathematical analysis detailed the adaptive α_m adjustment mechanism, which linearly increases with an upper limit, ensuring bounded and convergent reward functions. The inclusion of ψ in the reward function facilitates multi-objective optimization without increasing computational complexity, balancing various objectives effectively. Theoretical analysis confirmed MOSEAC's ability to achieve stable convergence in diverse environments, ensuring consistent performance across different scenarios.

These findings collectively demonstrate the robustness, efficiency, and practical applicability of MOSEAC, providing a solid foundation for future research and deployment in diverse real-world scenarios. The approach taken in validating MOSEAC on the Agilex Limo robot exemplifies its potential for broader adoption and adaptation in various robotic applications.

7.1.2 Features and Advantages

In many practical applications, task completion time and energy consumption are critical performance metrics. Traditional fixed time-step algorithms struggle to optimize both simultaneously. MOSEAC addresses this by incorporating task energy and time reward components into the reward function, achieving multi-objective optimization. Experimental results demonstrate that MOSEAC completes tasks with fewer time steps and significantly reduces energy consumption, thereby improving system efficiency and sustainability. Training efficiency and convergence speed are essential indicators of reinforcement learning algorithm performance. MOSEAC enhances these by using an adaptive adjustment of control frequency and reward mechanisms. By introducing a non-linear reward framework where the action duration reward and task reward are multiplicatively related, MOSEAC enables the agent to more quickly understand the impact of action duration on overall task reward. This accelerates the training process, as evidenced by experiments in a Newtonian kinematics environment, showing higher task performance in shorter training times and significantly improved data efficiency.

During the training phase, MOSEAC's adaptive adjustment mechanism for control frequency and reward settings improves efficiency and convergence speed. This dynamic adjustment allows the algorithm to quickly adapt to different environments and task requirements, ensuring stable convergence. In the deployment phase, experimental validation with the Agilex Limo robot demonstrates MOSEAC's robustness and adaptability across various target positions and complex environments. The design of expanding the action dimension and applying new reward formulas ensures that MOSEAC is flexible and adaptable to different tasks and environments.

Ensuring theoretical stability and convergence is critical for reliable algorithm deployment. MOSEAC has undergone detailed mathematical analysis to verify its convergence and complexity, ensuring stability across different environments. The mathematical foundation confirms that MOSEAC can balance multiple objectives, optimizing task completion time and energy consumption, thereby providing robust theoretical support for its practical application. Different tasks and environments may have specific requirements for reinforcement learning algorithms. SAC, an off-policy algorithm, performs well in many scenarios but may not be as suitable in environments that require frequent policy updates, where on-policy algorithms like PPO are more effective. MOSEAC broadens the action dimension by introducing action duration and applying new reward formulas, allowing it to extend beyond SAC to other reinforcement learning algorithms, such as PPO and TD3. The VTS-RL framework, backed by mathematical theory, ensures stability and effectiveness across various algorithms, offering consistent performance improvements and adaptability.

In traditional reinforcement learning algorithms, hyperparameter tuning is a complex and time-consuming process, with the setup of hyperparameters like learning rate and temperature coefficient (alpha) being crucial for performance and convergence speed. MOSEAC retains these traditional RL hyperparameters but introduces an adaptive reward mechanism that dynamically adjusts the key hyperparameter α_m. This reduces the user's burden in tuning multiple hyperparameters, making MOSEAC more user-friendly. While MOSEAC does not eliminate the need for all RL hyperparameters, it simplifies the tuning process by adding only one additional hyperparameter, thus lowering the entry barrier.

In summary, MOSEAC as a VTS-RL algorithm offers significant advantages in optimizing task completion time and energy consumption, improving training efficiency and convergence speed, enhancing robustness and adaptability, ensuring theoretical stability and convergence, providing broad algorithm applicability, and simplifying hyperparameter tuning. These features not only make MOSEAC important in theoretical research but also support its extensive deployment in practical applications.

7.2 Limitations

While MOSEAC demonstrates significant advantages, it also presents certain limitations that need to be addressed for broader applicability and effectiveness in real-world scenarios.

Firstly, the algorithm's adaptive adjustment mechanism and complex reward framework may increase computational resource requirements. While this is not a significant issue when training on high-performance computing resources, it can be more challenging for on-policy algorithms, mainly when training directly on resource-limited robots without requiring simulation-to-reality transfer [179]. This increased demand may affect training efficiency and outcomes in such constrained environments.

Secondly, despite reducing some of the hyperparameter tuning complexity by introducing an adaptive reward mechanism, MOSEAC still requires careful tuning of the critical hyperparameter ψ. Although specific adjustment recommendations are provided in these articles above, the optimal value of ψ may vary depending on the task and environment, indicating potential for further optimization. Currently, α_m is adjusted linearly, but more efficient methods may exist, and we have not found these methods yet. This presents a challenge for users who lack extensive hyperparameter experience.

Thirdly, like all reinforcement learning algorithms, MOSEAC's performance in simulation environments may not fully reflect its real-world applicability. Experimental settings are typically controlled and idealized, while real-world environments are noisy, uncertain, and dynamically changing. If the algorithm is not trained directly in real environments but rather in simulated ones, the quality of simulation data versus real-world data can significantly impact performance. Additionally, some existing reinforcement learning frameworks and tools (e.g., Mujoco [80]) may not fully support the extension to include time dimensions, limiting direct application.

Fourthly, MOSEAC relies on a complex reward signal design for multi-objective optimization. This requires a deep understanding of the task and environment and may necessitate redesigning and adjusting reward signals for different application scenarios [180]. The complexity of reward signal design can increase the development and debugging time, especially when facing new tasks or unforeseen environmental changes [181]. While these designs can enhance algorithm performance, they also impose higher demands on developers, increasing the application's difficulty and cost.

Fifth, when applied to multi-agent reinforcement learning, the non-uniform control rate method will lead to inconsistent state updates. Multiple agents may need to reach an agreement confirmation before the state update can be performed, which places additional hardware and communication quality requirements. MOSEAC has not yet been verified on a multi-agent reinforcement learning platform, one of our future work directions.

Finally, introducing the time dimension in MOSEAC requires developers to set appropriate time ranges (minimum and maximum durations). Inappropriate time settings can degrade or even nullify the algorithm's performance, particularly in real-time reinforcement learning training (as we mentioned in Chapter 6). This adds another layer of complexity to the algorithm's design and debugging, potentially making it less effective than traditional methods in some tasks. Therefore, further research and validation are needed to ensure MOSEAC's suitability and stability in these specific tasks and environments.

These limitations highlight the areas where MOSEAC requires further development and validation to ensure its robustness and effectiveness across diverse real-world applications.

7.3 Future Research

Based on the successful application of the MOSEAC in single-agent reinforcement learning, we propose the following three potential future research directions to expand further and optimize the algorithm's application:

7.3.1 Expanding to Multi-Agent Reinforcement Learning (MARL)

Expanding MOSEAC to multi-agent reinforcement learning [182, 183] environments is a promising research direction. By leveraging MOSEAC's adaptive adjustment mechanism and time dimension design, it is possible to coordinate the behaviors of multiple agents, achieving global optimization. However, the reward design in multi-agent environments is more complex, requiring the development of reward mechanisms suitable for agent cooperation. Additionally, computational resource requirements will significantly increase, necessitating optimization of resource usage and improved training efficiency. Training stability in multi-agent environments is also a critical challenge, mainly when applied to real-world robotics, where communication delays between multiple robots prevent real-time state updates. If each robot has different action durations, asynchronous state updates based on communication are a potential challenge [184]. Experimental validation of MOSEAC's performance in these environments is required.

7.3.2 Extension of Action Time Dimensions in Traditional Pre-trained RL Models

The extension of action time dimensions to make traditional pre-trained RL models more usable in different tasks and environments is another important research direction. MOSEAC's adaptive adjustment mechanism and time dimension design can further enhance the adaptability of pre-trained models, improving their performance in multi-task environments. Specifically, dynamically adjusting action time dimensions can optimize computational resource usage and improve training efficiency. This direction can enhance the generality of pre-trained models and reduce the time required for tuning and optimization in new tasks. There is existing research [185] on eliminating models' dependence on predefined action space sizes and structures, allowing models to generalize to new tasks and environments, which provides technical and theoretical support for our ideas.

7.3.3 Application to Safe Reinforcement Learning

The application of MOSEAC to safe reinforcement learning [186, 187] can provide new approaches to developing algorithms that avoid unsafe behaviors during training and deployment. MOSEAC's time dimension design allows agents to respond to potential hazards quickly, avoiding unsafe behaviors. The adaptive adjustment mechanism allows agents' actions to be dynamically evaluated and adjusted to meet safety constraints [188]. Additionally, incorporating safety-related reward terms in the reward design can enhance safety while ensuring task efficiency. This direction requires extensive validation in practical applications, especially in high-safety-requirement scenarios such as outdoor autonomous driving and industrial robotics.

Through these three research directions, we aim to further expand and optimize the MOSEAC's application scope. These directions can enhance the algorithm's adaptability and effectiveness and demonstrate its potential and advantages in practical applications. With continuous research and experimentation, we believe MOSEAC will achieve significant breakthroughs in multi-agent environments, pre-trained models, and safe reinforcement learning.

CHAPTER 8 CONCLUSION

The research presented in this thesis introduces and validates the Variable Time Step Reinforcement Learning (VTS-RL) framework, with a particular focus on the development and application of the Soft Elastic Actor-Critic (SEAC) and Multi-Objective Soft Elastic Actor-Critic (MOSEAC) algorithms. This work addresses critical inefficiencies in traditional reinforcement learning (RL) approaches by allowing dynamic adjustment of control frequencies based on task requirements, thus optimizing computational resource usage and improving task performance and energy efficiency.

8.1 Key Contributions

The SEAC algorithm pioneers the VTS-RL approach by integrating variable control frequencies into the RL framework. SEAC's ability to dynamically adjust control rates based on immediate task demands sets a new standard in RL, significantly enhancing task efficiency and energy utilization. Building on SEAC, the MOSEAC algorithm further refines this approach by simplifying hyperparameter tuning through an adaptive reward mechanism. This innovation reduces the complexity of algorithm deployment and improves training efficiency and task performance.

The practical application of MOSEAC on the Agilex Limo robot demonstrates the real-world efficacy of the VTS-RL framework. Through rigorous physical simulations and real-world experiments, MOSEAC has shown substantial improvements in energy efficiency and task completion times compared to fixed time step algorithms. These findings underscore the robustness and adaptability of MOSEAC in various robotic applications.

8.2 Theoretical and Practical Impact

The theoretical analysis presented in this thesis confirms the stability and convergence of the MOSEAC. By establishing a robust mathematical foundation for the VTS-RL framework, this research ensures that MOSEAC can maintain consistent performance across different environments and tasks. The adaptive α_m adjustment mechanism and the inclusion of ψ in the reward function facilitate multi-objective optimization without increasing computational complexity, enabling MOSEAC to effectively balance multiple objectives.

The impact of our research extends beyond immediate improvements in reinforcement learning performance. The introduction of VTS-RL and MOSEAC establishes a new standard for future RL algorithms by incorporating variable time steps, thus enhancing their applicability across diverse fields.

In autonomous driving, the ability to adjust control frequencies based on real-time road conditions can lead to safer and more efficient navigation systems. Similarly, variable time steps in industrial robotics can optimize production processes by reducing energy consumption and improving response times.

The VTS-RL framework also significantly advances intelligent systems in resource-constrained environments. By optimizing computational resource usage, VTS-RL enables the deployment of RL algorithms on platforms with limited processing power, such as embedded systems and mobile robots. This facilitates broader adoption of RL technologies in areas like healthcare, agriculture, and environmental monitoring, where efficient and adaptable control strategies are crucial.

The implementation and validation of these methods on the AgileX LIMO robot platform demonstrate the practical benefits of MOSEAC in real-world applications. This research not only enhances the theoretical foundations of RL but also provides tangible solutions for real-world challenges, paving the way for advanced RL algorithms that can operate efficiently in dynamic and resource-constrained environments.

8.3 Limitations and Future Work

While MOSEAC offers significant advancements, it also presents certain limitations, such as increased computational resource requirements and the complexity of reward signal design. Future research should focus on expanding MOSEAC to multi-agent reinforcement learning environments, extending action time dimensions in traditional pre-trained RL models, and applying MOSEAC to safe reinforcement learning. These directions will further enhance the adaptability, efficiency, and applicability of the VTS-RL framework in diverse real-world scenarios.

In conclusion, the VTS-RL framework and the MOSEAC represent significant steps forward in the field of reinforcement learning. By addressing the limitations of fixed control frequencies and introducing dynamic adjustment mechanisms, this research paves the way for more efficient, adaptable, and reliable RL-based robotic systems. The findings and contributions of this thesis provide a solid foundation for future advancements in reinforcement learning, with broad implications for both theoretical research and practical applications.

REFERENCES

[1] AgileX Robotics, “Agilex limo - multi-modal mobile robot with ai modules,” https://www.globenewswire.com/, accessed: [date].

[2] OptiTrack, “Optitrack - motion capture systems,” https://www.optitrack.com, accessed: [date].

[3] M. Lapan, *Deep Reinforcement Learning Hands-On: Apply modern RL methods to practical problems of chatbots, robotics, discrete optimization, web automation, and more*. Packt Publishing Ltd, 2020.

[4] Y. Li, “Deep reinforcement learning: An overview,” *arXiv preprint arXiv:1701.07274*, 2017.

[5] B. Singh, R. Kumar, and V. P. Singh, “Reinforcement learning in robotic applications: a comprehensive survey,” *Artificial Intelligence Review*, vol. 55, no. 2, pp. 945–990, 2022.

[6] T. Hester, M. Quinlan, and P. Stone, “Rtmba: A real-time model-based reinforcement learning architecture for robot control,” in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 85–90.

[7] J. Hwangbo, I. Sa, R. Siegwart, and M. Hutter, “Control of a quadrotor with reinforcement learning,” *IEEE Robotics and Automation Letters*, vol. 2, no. 4, pp. 2096–2103, 2017.

[8] K. Shao, Z. Tang, Y. Zhu, N. Li, and D. Zhao, “A survey of deep reinforcement learning in video games,” *arXiv preprint arXiv:1912.10944*, 2019.

[9] K. Souchleris, G. K. Sidiropoulos, and G. A. Papakostas, “Reinforcement learning in game industry—review, prospects and challenges,” *Applied Sciences*, vol. 13, no. 4, p. 2443, 2023.

[10] O. Delalleau, M. Peter, E. Alonso, and A. Logut, “Discrete and continuous action representation for practical rl in video games,” *arXiv preprint arXiv:1912.11077*, 2019.

[11] J. R. Norris, *Markov chains*. Cambridge university press, 1998, no. 2.

[12] S. Amin, M. Gomrokchi, H. Aboutalebi, H. Satija, and D. Precup, “Locally persistent exploration in continuous control tasks with sparse rewards,” *arXiv preprint arXiv:2012.13658*, 2020.

[13] S. Park, J. Kim, and G. Kim, "Time discretization-invariant safe action repetition for policy gradient methods," *Advances in Neural Information Processing Systems*, vol. 34, pp. 267–279, 2021.

[14] A. M. Lyapunov, "The general problem of the stability of motion," *International journal of control*, vol. 55, no. 3, pp. 531–534, 1992.

[15] S. Zhao, T. Zheng, D. Sui, J. Zhao, and Y. Zhu, "Reinforcement learning based variable damping control of wearable robotic limbs for maintaining astronaut pose during extravehicular activity," *Frontiers in Neurorobotics*, vol. 17, p. 1093718, 2023.

[16] M. Mohri, A. Rostamizadeh, and A. Talwalkar, *Foundations of machine learning*. MIT press, 2018.

[17] M. I. Jordan and T. M. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, 2015.

[18] R. Bellman, "A markovian decision process," *Journal of mathematics and mechanics*, pp. 679–684, 1957.

[19] ——, "Dynamic programming," *science*, vol. 153, no. 3731, pp. 34–37, 1966.

[20] C. J. C. H. Watkins, "Learning from delayed rewards," 1989.

[21] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, pp. 9–44, 1988.

[22] A. S. Polydoros and L. Nalpantidis, "Survey of model-based reinforcement learning: Applications on robotics," *Journal of Intelligent & Robotic Systems*, vol. 86, no. 2, pp. 153–173, 2017.

[23] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine *et al.*, "Model-based reinforcement learning for atari," *arXiv preprint arXiv:1903.00374*, 2019.

[24] J. Gläscher, N. Daw, P. Dayan, and J. P. O'Doherty, "States versus rewards: dissociable neural prediction error signals underlying model-based and model-free reinforcement learning," *Neuron*, vol. 66, no. 4, pp. 585–595, 2010.

[25] T. Degris, P. M. Pilarski, and R. S. Sutton, "Model-free reinforcement learning with continuous action in practice," in *2012 American control conference (ACC)*. IEEE, 2012, pp. 2177–2182.

[26] A. L. Strehl, L. Li, E. Wiewiora, J. Langford, and M. L. Littman, "Pac model-free reinforcement learning," in *Proceedings of the 23rd international conference on Machine learning*, 2006, pp. 881–888.

[27] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," *Advances in neural information processing systems*, vol. 12, 1999.

[28] J. Hammersley, *Monte carlo methods*. Springer Science & Business Media, 2013.

[29] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[30] A. Doucet, M. K. Pitt, G. Deligiannidis, and R. Kohn, "Efficient implementation of markov chain monte carlo when using an unbiased likelihood estimator," *Biometrika*, vol. 102, no. 2, pp. 295–313, 2015.

[31] M. Vihola, "Unbiased estimators and multilevel monte carlo," *Operations Research*, vol. 66, no. 2, pp. 448–462, 2018.

[32] J. Asmuth and M. L. Littman, "Learning is planning: near bayes-optimal reinforcement learning via monte-carlo tree search," *arXiv preprint arXiv:1202.3699*, 2012.

[33] S. Fujimoto, D. Meger, and D. Precup, "Off-policy deep reinforcement learning without exploration," in *International conference on machine learning*. PMLR, 2019, pp. 2052–2062.

[34] M. Kearns and S. Singh, "Near-optimal reinforcement learning in polynomial time," *Machine learning*, vol. 49, pp. 209–232, 2002.

[35] A. Lazaric, M. Restelli, and A. Bonarini, "Reinforcement learning in continuous action spaces through sequential monte carlo methods," *Advances in neural information processing systems*, vol. 20, 2007.

[36] S. Singh, T. Jaakkola, M. L. Littman, and C. Szepesvári, "Convergence results for single-step on-policy reinforcement-learning algorithms," *Machine learning*, vol. 38, pp. 287–308, 2000.

[37] M. A. Wiering and H. Van Hasselt, "Two novel on-policy reinforcement learning algorithms based on td (λ)-methods," in *2007 IEEE International Symposium on Approximate Dynamic Programming and Reinforcement Learning*. IEEE, 2007, pp. 280–287.

[38] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3389–3396.

[39] G. A. Rummery and M. Niranjan, *On-line Q-learning using connectionist systems*. University of Cambridge, Department of Engineering Cambridge, UK, 1994, vol. 37.

[40] M. Hausknecht, P. Stone, and O.-p. Mc, "On-policy vs. off-policy updates for deep reinforcement learning," in *Deep reinforcement learning: frontiers and challenges, IJCAI 2016 Workshop*. AAAI Press New York, NY, USA, 2016.

[41] V. Konda and J. Tsitsiklis, "Actor-critic algorithms," *Advances in neural information processing systems*, vol. 12, 1999.

[42] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[43] K. O'shea and R. Nash, "An introduction to convolutional neural networks," *arXiv preprint arXiv:1511.08458*, 2015.

[44] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[45] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[46] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel *et al.*, "Mastering atari, go, chess and shogi by planning with a learned model," *Nature*, vol. 588, no. 7839, pp. 604–609, 2020.

[47] F. Ebert, C. Finn, S. Dasari, A. Xie, A. Lee, and S. Levine, "Visual foresight: Model-based deep reinforcement learning for vision-based robotic control," *arXiv preprint arXiv:1812.00568*, 2018.

[48] N. O. Lambert, D. S. Drew, J. Yaconelli, S. Levine, R. Calandra, and K. S. Pister, "Low-level control of a quadrotor with deep model-based reinforcement learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 4, pp. 4224–4230, 2019.

[49] A. Kupcsik, M. P. Deisenroth, J. Peters, A. P. Loh, P. Vadakkepat, and G. Neumann, "Model-based contextual policy search for data-efficient generalization of robot skills," *Artificial Intelligence*, vol. 247, pp. 415–439, 2017.

[50] N. Kovincic, H. Gattringer, A. Müller, M. Weyrer, A. Schlotzhauer, L. Kaiser, and M. Brandstötter, "A model-based strategy for safety assessment of a robot arm interacting with humans," *PAMM*, vol. 19, no. 1, p. e201900247, 2019.

[51] V. Pong, S. Gu, M. Dalal, and S. Levine, "Temporal difference models: Model-free deep rl for model-based control," *arXiv preprint arXiv:1802.09081*, 2018.

[52] I. Osband and B. Van Roy, "Model-based reinforcement learning and the eluder dimension," *Advances in Neural Information Processing Systems*, vol. 27, 2014.

[53] A. Plaat, W. Kosters, and M. Preuss, "Deep model-based reinforcement learning for high-dimensional problems, a survey," *arXiv preprint arXiv:2008.05598*, 2020.

[54] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1928–1937.

[55] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.

[56] K. Zhu and T. Zhang, "Deep reinforcement learning based mobile robot navigation: A review," *Tsinghua Science and Technology*, vol. 26, no. 5, pp. 674–691, 2021.

[57] J. Ibarz, J. Tan, C. Finn, M. Kalakrishnan, P. Pastor, and S. Levine, "How to train your robot with deep reinforcement learning: lessons we have learned," *The International Journal of Robotics Research*, vol. 40, no. 4-5, pp. 698–721, 2021.

[58] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev *et al.*, "Grandmaster level in starcraft ii using multi-agent reinforcement learning," *Nature*, vol. 575, no. 7782, pp. 350–354, 2019.

[59] S. E. Li, "Deep reinforcement learning," in *Reinforcement learning for sequential decision and optimal control*. Springer, 2023, pp. 365–402.

[60] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[61] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International conference on machine learning*. PMLR, 2018, pp. 1587–1596.

[62] Y. Huang, D. Liu, Z. Liu, K. Wang, Q. Wang, and J. Tan, "A novel robotic grasping method for moving objects based on multi-agent deep reinforcement learning," *Robotics and Computer-Integrated Manufacturing*, vol. 86, p. 102644, 2024.

[63] Q. Zheng, Z. Peng, P. Zhu, Y. Zhao, R. Zhai, and W. Ma, "An object recognition grasping approach using proximal policy optimization with yolov5," *IEEE Access*, 2023.

[64] L. Zhang, J. Peng, W. Yi, H. Lin, L. Lei, and X. Song, "A state-decomposition ddpg algorithm for uav autonomous navigation in 3d complex environments," *IEEE Internet of Things Journal*, 2023.

[65] A. M. Abdulghani, M. M. Abdulghani, W. L. Walters, and K. H. Abed, "Ai safety approach for minimizing collisions in autonomous navigation." *Journal of Artificial Intelligence (2579-0021)*, vol. 5, 2023.

[66] F. Zhang, J. Xiong, S. Yuan, and K. Wen, "Research of improved td3 robotic arm path planning using evolutionary algorithm," in *2023 WRC Symposium on Advanced Robotics and Automation (WRC SARA)*. IEEE, 2023, pp. 269–275.

[67] Y. Fan, H. Dong, X. Zhao, and P. Denissenko, "Path-following control of unmanned underwater vehicle based on an improved td3 deep reinforcement learning," *IEEE Transactions on Control Systems Technology*, 2024.

[68] S. Wang, D. Jia, and X. Weng, "Deep reinforcement learning for autonomous driving," *arXiv preprint arXiv:1811.11329*, 2018.

[69] M. Srouji, J. Zhang, and R. Salakhutdinov, "Structured control nets for deep reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2018, pp. 4742–4751.

[70] M. Franz, L. Wolf, M. Periyasamy, C. Ufrecht, D. D. Scherer, A. Plinge, C. Mutschler, and W. Mauerer, "Uncovering instabilities in variational-quantum deep q-networks," *Journal of The Franklin Institute*, vol. 360, no. 17, pp. 13 822–13 844, 2023.

[71] J. Seo, S. Kim, A. Jalalvand, R. Conlin, A. Rothstein, J. Abbate, K. Erickson, J. Wai, R. Shousha, and E. Kolemen, "Avoiding fusion plasma tearing instability with deep reinforcement learning," *Nature*, vol. 626, no. 8000, pp. 746–751, 2024.

[72] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel *et al.*, "Soft actor-critic algorithms and applications," *arXiv preprint arXiv:1812.05905*, 2018.

[73] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," in *International conference on machine learning*. PMLR, 2018, pp. 1861–1870.

[74] O. Engineering, "Pid controller: Types, what it is & how it works," *Omega Engineering*, 2021. [Online]. Available: https://www.omega.com/en-us/resources/pid-controllers

[75] Open Source Robotics Foundation, "Gazebo," https://gazebosim.org, 2021.

[76] X. Yu, Y. Fan, S. Xu, and L. Ou, "A self-adaptive sac-pid control approach based on reinforcement learning for mobile robots," *International journal of robust and nonlinear control*, vol. 32, no. 18, pp. 9625–9643, 2022.

[77] D. Wang, M. Jia, X. Zhu, R. Walters, and R. Platt, "On-robot learning with equivariant models," *arXiv preprint arXiv:2203.04923*, 2022.

[78] R. Morimoto, S. Nishikawa, R. Niiyama, and Y. Kuniyoshi, "Model-free reinforcement learning with ensemble for a soft continuum robot arm," in *2021 IEEE 4th International Conference on Soft Robotics (RoboSoft)*. IEEE, 2021, pp. 141–148.

[79] J. C. de Jesus, V. A. Kich, A. H. Kolling, R. B. Grando, M. A. d. S. L. Cuadros, and D. F. T. Gamarra, "Soft actor-critic for navigation of mobile robots," *Journal of Intelligent & Robotic Systems*, vol. 102, no. 2, p. 31, 2021.

[80] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *2012 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, 2012, pp. 5026–5033.

[81] Y. Bouteiller, S. Ramstedt, G. Beltrame, C. Pal, and J. Binas, "Reinforcement learning with random delays," in *International conference on learning representations*, 2021.

[82] A. Karimi, J. Jin, J. Luo, A. R. Mahmood, M. Jagersand, and S. Tosatto, "Dynamic decision frequency with continuous options," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2023, pp. 7545–7552.

[83] S. Sharma, A. Srinivas, and B. Ravindran, "Learning to repeat: Fine grained action repetition for deep reinforcement learning," *arXiv preprint arXiv:1702.06054*, 2017.

[84] Y. Chen, H. Wu, Y. Liang, and G. 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)*. IEEE, 2021, pp. 1–9.

[85] A. M. Metelli, F. Mazzolini, L. Bisi, L. Sabbioni, and M. Restelli, "Control frequency adaptation via action persistence in batch reinforcement learning," in *International Conference on Machine Learning*. PMLR, 2020, pp. 6862–6873.

[86] J. Lee, B.-J. Lee, and K.-E. Kim, "Reinforcement learning for control with multiple frequencies," *Advances in Neural Information Processing Systems*, vol. 33, pp. 3254–3264, 2020.

[87] L. Lin, W. Zhou, Z. Yang, and J. Liu, "Deep reinforcement learning-based task scheduling and resource allocation for noma-mec in industrial internet of things," *Peer-to-Peer Networking and Applications*, vol. 16, no. 1, pp. 170–188, 2023.

[88] X. Li and S. Dong, "Research on efficient reinforcement learning for adaptive frequency-agility radar," *Sensors*, vol. 21, no. 23, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/23/7931

[89] M. Qin, S. Sun, W. Zhang, H. Xia, X. Wang, and B. An, "Earnhft: Efficient hierarchical reinforcement learning for high frequency trading," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 38, no. 13, 2024, pp. 14 669–14 676.

[90] T. G. Dietterich, "Hierarchical reinforcement learning with the maxq value function decomposition," *Journal of artificial intelligence research*, vol. 13, pp. 227–303, 2000.

[91] S. Li, R. Wang, M. Tang, and C. Zhang, "Hierarchical reinforcement learning with advantage-based auxiliary rewards," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[92] Z. Wan and P. Hudak, "Functional reactive programming from first principles," in *Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation*, 2000, pp. 242–252.

[93] E. Bregu, N. Casamassima, D. Cantoni, L. Mottola, and K. Whitehouse, "Reactive control of autonomous drones," in *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, 2016, pp. 207–219.

[94] L. P. Kaelbling *et al.*, "An architecture for intelligent reactive systems," *Reasoning about actions and plans*, pp. 395–410, 1987.

[95] E. Bainomugisha, A. L. Carreton, T. v. Cutsem, S. Mostinckx, and W. d. Meuter, "A survey on reactive programming," *ACM Computing Surveys (CSUR)*, vol. 45, no. 4, pp. 1–34, 2013.

[96] E. Czaplicki and S. Chong, "Asynchronous functional reactive programming for guis," *ACM SIGPLAN Notices*, vol. 48, no. 6, pp. 411–422, 2013.

[97] G. Biggs and B. MacDonald, "A survey of robot programming systems," in *Proceedings of the Australasian conference on robotics and automation*, vol. 1, 2003, pp. 1–3.

[98] I. Pembeci, H. Nilsson, and G. Hager, "Functional reactive robotics: An exercise in principled integration of domain-specific languages," in *Proceedings of the 4th ACM SIGPLAN international conference on Principles and practice of declarative programming*, 2002, pp. 168–179.

[99] A. Desai, S. Qadeer, and S. A. Seshia, "Programming safe robotics systems: Challenges and advances," in *Leveraging Applications of Formal Methods, Verification and Validation. Verification: 8th International Symposium, ISoLA 2018, Limassol, Cyprus, November 5-9, 2018, Proceedings, Part II 8*. Springer, 2018, pp. 103–119.

[100] C. Pezzato, C. H. Corbato, S. Bonhof, and M. Wisse, "Active inference and behavior trees for reactive action planning and execution in robotics," *IEEE Transactions on Robotics*, vol. 39, no. 2, pp. 1050–1069, 2023.

[101] Y. Jiang, P. Peng, L. Wang, J. Wang, J. Wu, and Y. Liu, "Lidar-based local path planning method for reactive navigation in underground mines," *Remote Sensing*, vol. 15, no. 2, p. 309, 2023.

[102] C. Peng, Z. Fei, and S. G. Vougioukas, "Gnss-free end-of-row detection and headland maneuvering for orchard navigation using a depth camera," *Machines*, vol. 11, no. 1, p. 84, 2023.

[103] Y. Zhuang, X. Sun, Y. Li, J. Huai, L. Hua, X. Yang, X. Cao, P. Zhang, Y. Cao, L. Qi *et al.*, "Multi-sensor integrated navigation/positioning systems using data fusion: From analytics-based to learning-based approaches," *Information Fusion*, vol. 95, pp. 62–90, 2023.

[104] G. Salvaneschi and P. Weisenburger, "Bridging between active objects: Multitier programming for distributed, concurrent systems," in *Active Object Languages: Current Research Trends*. Springer, 2024, pp. 92–122.

[105] N. Senel, K. Kefferpütz, K. Doycheva, and G. Elger, "Multi-sensor data fusion for real-time multi-object tracking," *Processes*, vol. 11, no. 2, p. 501, 2023.

[106] T. Wan, B. Shao, S. Ma, Y. Zhou, Q. Li, and Y. Chai, “In-sensor computing: materials, devices, and integration technologies,” *Advanced materials*, vol. 35, no. 37, p. 2203830, 2023.

[107] D. Schreckenghost, K. Holden, M. Greene, T. Milam, and C. Hamblin, “Effect of automating procedural work on situation awareness and workload,” *Human Factors*, vol. 65, no. 6, pp. 1161–1172, 2023.

[108] M. F. I. Khan and A. K. M. Masum, “Predictive analytics and machine learning for real-time detection of software defects and agile test management,” *Educational Administration: Theory and Practice*, vol. 30, no. 4, pp. 1051–1057, 2024.

[109] J. Haase, P. B. Walker, O. Berardi, and W. Karwowski, “Get real get better: A framework for developing agile program management in the us navy supported by the application of advanced data analytics and ai,” *Technologies*, vol. 11, no. 6, p. 165, 2023.

[110] J. Kaur and K. S. Mann, “Ai based healthcare platform for real time, predictive and prescriptive analytics using reactive programming,” in *Journal of Physics: Conference Series*, vol. 933, no. 1. IOP Publishing, 2017, p. 012010.

[111] S. Pateria, B. Subagdja, A.-h. Tan, and C. Quek, “Hierarchical reinforcement learning: A comprehensive survey,” *ACM Computing Surveys (CSUR)*, vol. 54, no. 5, pp. 1–35, 2021.

[112] K. Hausman, J. T. Springenberg, Z. Wang, N. Heess, and M. Riedmiller, “Learning an embedding space for transferable robot skills,” in *International Conference on Learning Representations*, 2018.

[113] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, “Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation,” *Advances in neural information processing systems*, vol. 29, 2016.

[114] A. Gupta, V. Kumar, C. Lynch, S. Levine, and K. Hausman, “Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning,” *arXiv preprint arXiv:1910.11956*, 2019.

[115] I. Kacem, S. Hammadi, and P. Borne, “Pareto-optimality approach for flexible job-shop scheduling problems: hybridization of evolutionary algorithms and fuzzy logic,” *Mathematics and computers in simulation*, vol. 60, no. 3-5, pp. 245–276, 2002.

[116] M. S. Monfared, S. E. Monabbati, and A. R. Kafshgar, “Pareto-optimal equilibrium points in non-cooperative multi-objective optimization problems,” *Expert Systems with Applications*, vol. 178, p. 114995, 2021.

[117] V. Pareto, *Manual of political economy: a critical and variorum edition.* OUP Oxford, 2014.

[118] K. Deb, “Multi-objective optimisation using evolutionary algorithms: an introduction,” in *Multi-objective evolutionary optimisation for product design and manufacturing*. Springer, 2011, pp. 3–34.

[119] A. Rapoport, *Game theory as a theory of conflict resolution.* Springer Science & Business Media, 2012, vol. 2.

[120] S. Vajda, *Mathematical programming*. Courier Corporation, 2009.

[121] W. Cui, Y. Jiang, and B. Zhang, “Reinforcement learning for optimal primary frequency control: A lyapunov approach,” *IEEE Transactions on Power Systems*, vol. 38, no. 2, pp. 1676–1688, 2022.

[122] Y. Chow, O. Nachum, E. Duenez-Guzman, and M. Ghavamzadeh, “A lyapunov-based approach to safe reinforcement learning,” *Advances in neural information processing systems*, vol. 31, 2018.

[123] T. J. Perkins and A. G. Barto, “Lyapunov design for safe reinforcement learning,” *Journal of Machine Learning Research*, vol. 3, no. Dec, pp. 803–832, 2002.

[124] J. Tailleur and J. Kurchan, “Probing rare physical trajectories with lyapunov weighted dynamics,” *Nature Physics*, vol. 3, no. 3, pp. 203–207, 2007.

[125] M. Diehl, R. Amrit, and J. B. Rawlings, “A lyapunov function for economic optimizing model predictive control,” *IEEE Transactions on Automatic Control*, vol. 56, no. 3, pp. 703–707, 2010.

[126] R. Freeman and P. V. Kokotovic, *Robust nonlinear control design: state-space and Lyapunov techniques*. Springer Science & Business Media, 2008.

[127] G. Williams, N. Wagener, B. Goldfain, P. Drews, J. M. Rehg, B. Boots, and E. A. Theodorou, “Information theoretic mpc for model-based reinforcement learning,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 1714–1721.

[128] Z. Yang, K. Merrick, L. Jin, and H. A. Abbass, “Hierarchical deep reinforcement learning for continuous action control,” *IEEE transactions on neural networks and learning systems*, vol. 29, no. 11, pp. 5174–5184, 2018.

[129] M. Zanon and S. Gros, “Safe reinforcement learning using robust mpc,” *IEEE Transactions on Automatic Control*, vol. 66, no. 8, pp. 3638–3652, 2020.

[130] A. R. Mahmood, D. Korenkevych, B. J. Komer, and J. Bergstra, “Setting up a reinforcement learning task with a real-world robot,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4635–4640.

[131] P. R. Wurman, S. Barrett, K. Kawamoto, J. MacGlashan, K. Subramanian, T. J. Walsh, R. Capobianco, A. Devlic, F. Eckert, F. Fuchs *et al.*, “Outracing champion gran turismo drivers with deep reinforcement learning,” *Nature*, vol. 602, no. 7896, pp. 223–228, 2022.

[132] J. Li, J. Ding, T. Chai, F. L. Lewis, and S. Jagannathan, “Adaptive interleaved reinforcement learning: Robust stability of affine nonlinear systems with unknown uncertainty,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 33, no. 1, pp. 270–280, 2020.

[133] S. Nasiriany, H. Liu, and Y. Zhu, “Augmenting reinforcement learning with behavior primitives for diverse manipulation tasks,” in *2022 International Conference on Robotics and Automation (ICRA)*. IEEE, 2022, pp. 7477–7484.

[134] F. Pardo, A. Tavakoli, V. Levdik, and P. Kormushev, “Time limits in reinforcement learning,” in *International Conference on Machine Learning*. PMLR, 2018, pp. 4045–4054.

[135] Z. Zhang, D. Zhang, and R. C. Qiu, “Deep reinforcement learning for power system applications: An overview,” *CSEE Journal of Power and Energy Systems*, vol. 6, no. 1, pp. 213–225, 2019.

[136] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, “Mastering the game of go with deep neural networks and tree search,” *nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[137] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, “A general reinforcement learning algorithm that masters chess, shogi, and go through self-play,” *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[138] T. Hester and P. Stone, “Texplore: real-time sample-efficient reinforcement learning for robots,” *Machine learning*, vol. 90, pp. 385–429, 2013.

[139] C. J. Watkins and P. Dayan, “Q-learning,” *Machine learning*, vol. 8, pp. 279–292, 1992.

[140] S. Ramstedt and C. Pal, “Real-time reinforcement learning,” *Advances in neural information processing systems*, vol. 32, 2019.

[141] S. Adam, L. Busoniu, and R. Babuska, “Experience replay for real-time reinforcement learning control,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 2, pp. 201–212, 2011.

[142] P. Almási, R. Moni, and B. Gyires-Tóth, “Robust reinforcement learning-based autonomous driving agent for simulation and real world,” in *2020 International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2020, pp. 1–8.

[143] R. Singh, M. Ierapetritou, and R. Ramachandran, “System-wide hybrid mpc–pid control of a continuous pharmaceutical tablet manufacturing process via direct compaction,” *European Journal of Pharmaceutics and Biopharmaceutics*, vol. 85, no. 3, pp. 1164–1182, 2013.

[144] Y. Dai, S. Yu, Y. Yan, and X. Yu, “An ekf-based fast tube mpc scheme for moving target tracking of a redundant underwater vehicle-manipulator system,” *IEEE/ASME Transactions on Mechatronics*, vol. 24, no. 6, pp. 2803–2814, 2019.

[145] W. Zaremba, I. Sutskever, and O. Vinyals, “Recurrent neural network regularization,” *arXiv preprint arXiv:1409.2329*, 2014.

[146] Z. C. Lipton, J. Berkowitz, and C. Elkan, “A critical review of recurrent neural networks for sequence learning,” *arXiv preprint arXiv:1506.00019*, 2015.

[147] F. A. Gers, J. Schmidhuber, and F. Cummins, “Learning to forget: Continual prediction with lstm,” *Neural computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[148] B. Müller, J. Reinhardt, and M. T. Strickland, *Neural networks: an introduction*. Springer Science & Business Media, 1995.

[149] B. L. Kalman and S. C. Kwasny, “Why tanh: choosing a sigmoidal function,” in *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, vol. 4. IEEE, 1992, pp. 578–581.

[150] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, “Mobilenets: Efficient convolutional neural networks for mobile vision applications,” *arXiv preprint arXiv:1704.04861*, 2017.

[151] tmrl, “tmrl main page,” https://github.com/trackmania-rl/tmrl, 2023.

[152] R. Liu, F. Nageotte, P. Zanne, M. de Mathelin, and B. Dresp-Langley, “Deep reinforcement learning for the control of robotic manipulation: a focussed mini-review,” *Robotics*, vol. 10, no. 1, p. 22, 2021.

[153] N. Akalin and A. Loutfi, “Reinforcement learning approaches in social robotics,” *Sensors*, vol. 21, no. 4, p. 1292, 2021.

[154] R. Majumdar, A. Mathur, M. Pirron, L. Stegner, and D. Zufferey, “Paracosm: A test framework for autonomous driving simulations,” in *International Conference on Fundamental Approaches to Software Engineering*. Springer International Publishing Cham, 2021, pp. 172–195.

[155] D. Wang and G. Beltrame, “Deployable reinforcement learning with variable control rate,” *arXiv preprint arXiv:2401.09286*, 2024.

[156] ——, “Reinforcement learning with elastic time steps,” *arXiv preprint arXiv:2402.14961*, 2024.

[157] ——, “Moseac: Streamlined variable time step reinforcement learning,” *arXiv preprint arXiv:2406.01521*, 2024.

[158] U. T. 2023, “Trackmania main page,” https://www.ubisoft.com/en-us/game/trackmania/trackmania, 2023.

[159] K. Shin and N. McKay, “Minimum-time control of robotic manipulators with geometric path constraints,” *IEEE Transactions on Automatic Control*, vol. 30, no. 6, pp. 531–541, 1985.

[160] P.-Y. Lajoie, B. Ramtoula, Y. Chang, L. Carlone, and G. Beltrame, “Door-slam: Distributed, online, and outlier resilient slam for robotic teams,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 1656–1663, 2020.

[161] S. Wan, Z. Gu, and Q. Ni, “Cognitive computing and wireless communications on the edge for healthcare service robots,” *Computer Communications*, vol. 149, pp. 99–106, 2020.

[162] E. Even-Dar, S. Mannor, and Y. Mansour, “Action elimination and stopping conditions for reinforcement learning,” in *Proceedings of the 20th International Conference on Machine Learning (ICML-03)*, 2003, pp. 162–169.

[163] N. Gottipati *et al.*, “To the max: Reinventing reward in reinforcement learning,” *arXiv preprint arXiv:2402.01361*, 2020. [Online]. Available: https://ar5iv.labs.arxiv.org/html/2402.01361

[164] S. Banach, “Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales,” *Fundamenta mathematicae*, vol. 3, no. 1, pp. 133–181, 1922.

[165] M. Rybczak, N. Popowniak, and A. Lazarowska, “A survey of machine learning approaches for mobile robot control,” *Robotics*, vol. 13, no. 1, p. 12, 2024.

[166] Authors, “Supervised and unsupervised deep learning applications for visual slam in robotics,” in *IMECE*. ASME, 2022, p. V003T04A010.

[167] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” 2023.

[168] Authors, “Assessing the impact of distribution shift on reinforcement learning performance,” *arXiv preprint arXiv:2402.03590*, 2024. [Online]. Available: https://arxiv.org/abs/2402.03590

[169] M. Katakura *et al.*, “Reinforcement learning model with dynamic state space tested on target search tasks for monkeys: Extension to learning task events,” *Frontiers in Robotics and AI*, 2023. [Online]. Available: https://www.frontiersin.org/articles/10.3389/frobt.2023.00856/full

[170] “Jetson nano module,” https://developer.nvidia.com/embedded/jetson-nano, 2019.

[171] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, “Pytorch: An imperative style, high-performance deep learning library,” in *Advances in Neural Information Processing Systems*, vol. 32, 2019, pp. 8024–8035. [Online]. Available: https://proceedings.neurips.cc/paper/2019/hash/bdbca288fee7f92f2bfa9f7012727740-Abstract.html

[172] M. Towers, J. K. Terry, A. Kwiatkowski, J. U. Balis, G. d. Cola, T. Deleu, M. Goulão, A. Kallinteris, A. KG, M. Krimmel, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, A. T. J. Shen, and O. G. Younis, “Gymnasium,” Mar. 2023. [Online]. Available: https://zenodo.org/record/8127025

[173] “Tensorrt,” https://developer.nvidia.com/tensorrt, 2021.

[174] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “Ros: an open-source robot operating system,” in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[175] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, “Ros 2: Towards a performance-centric and real-time robotics framework,” in *Robotics: Science and Systems (RSS) Workshop on Real-time and Performance in Robotic Systems*, Online, 2020.

[176] D. Inc., *Docker: Open Platform for Developing, Shipping, and Running Applications*, 2013, available at https://docs.docker.com/.

[177] H. Ryan, M. Paglione, and S. Green, "Review of trajectory accuracy methodology and comparison of error measurement metrics," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, 2004, p. 4787.

[178] M. Müller, "Dynamic time warping," *Information retrieval for music and motion*, pp. 69–84, 2007.

[179] W. Zhu, X. Guo, D. Owaki, K. Kutsuzawa, and M. Hayashibe, "A survey of sim-to-real transfer techniques applied to reinforcement learning for bioinspired robots," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 34, no. 7, pp. 3444–3459, 2021.

[180] J. Eschmann, "Reward function design in reinforcement learning," *Reinforcement learning algorithms: Analysis and Applications*, pp. 25–33, 2021.

[181] R. Devidze, G. Radanovic, P. Kamalaruban, and A. Singla, "Explicable reward design for reinforcement learning agents," *Advances in Neural Information Processing Systems*, vol. 34, pp. 20 118–20 131, 2021.

[182] K. Zhang, Z. Yang, and T. Başar, "Multi-agent reinforcement learning: A selective overview of theories and algorithms," *Handbook of reinforcement learning and control*, pp. 321–384, 2021.

[183] D. Huh and P. Mohapatra, "Multi-agent reinforcement learning: A comprehensive survey," *arXiv preprint arXiv:2312.10256*, 2023.

[184] A. Wong, T. Bäck, A. V. Kononova, and A. Plaat, "Deep multiagent reinforcement learning: Challenges and directions," *Artificial Intelligence Review*, vol. 56, no. 6, pp. 5023–5056, 2023.

[185] V. Sinii, A. Nikulin, V. Kurenkov, I. Zisman, and S. Kolesnikov, "In-context reinforcement learning for variable action spaces," *arXiv preprint arXiv:2312.13327*, 2023.

[186] S. Gu, L. Yang, Y. Du, G. Chen, F. Walter, J. Wang, Y. Yang, and A. Knoll, "A review of safe reinforcement learning: Methods, theory and applications," *arXiv preprint arXiv:2205.10330*, 2022.

[187] J. Garcıa and F. Fernández, "A comprehensive survey on safe reinforcement learning," *Journal of Machine Learning Research*, vol. 16, no. 1, pp. 1437–1480, 2015.

[188] T. Yamagata and R. Santos-Rodriguez, “Safe and robust reinforcement-learning: Principles and practice,” *arXiv preprint arXiv:2403.18539*, 2024.

[189] M. Riedmiller, R. Hafner, T. Lampe, M. Neunert, J. Degrave, T. van de Wiele, V. Mnih, and N. Heess, “Learning by playing - solving sparse reward tasks from scratch,” in *Proceedings of the 35th International Conference on Machine Learning*, 2018. [Online]. Available: https://arxiv.org/abs/1802.09464

[190] M. G. Bellemare, S. Srinivasan, G. Ostrovski, T. Schaul, D. Saxton, and R. Munos, “Unifying count-based exploration and intrinsic motivation,” *arXiv preprint arXiv:1704.08302*, 2017. [Online]. Available: https://arxiv.org/abs/1704.08302

[191] S. Levine, C. Finn, T. Darrell, and P. Abbeel, “End-to-end training of deep visuomotor policies,” *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016. [Online]. Available: https://arxiv.org/abs/1604.07316

[192] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[193] V. Pareto, *Manual of Political Economy*. New York: Augustus M. Kelley, 1971.

[194] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, “A fast and elitist multiobjective genetic algorithm: Nsga-ii,” *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

APPENDIX A   VALIDATION ENVIRONMENT DETAILS IN (CH. 3)

The Spatial Information of our environment are:

Table A.1 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	6	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	

The state dimensions are:

State

Definition 15. *State*
We have only one starting point, endpoint, and obstacle in the set environment. 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 action dimensions are:

Action

Definition 16. *Action*
Each action should have a corresponding execution duration.

$$a_t = (a_t, a_{fx}, a_{fy})$$

Where:
a_t = *The time 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.*

APPENDIX B DETAILS OF NEWTONIAN KINEMATICS ENVIRONMENT IN OF (CH. 4)

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 18 for its state value information, Definition 19 for its action value information, and Definition 17 for the physical formula it follows.

As Definition 8, the precise reward configuration for our environment are outlined in Table B.1.

Definition 17. *The Newton Kinematics formulas are:*

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

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

$$F_{aim} = mA;$$

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

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

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 B.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 18:

Definition 18. *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 B.2:

Table B.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	

The action dimensions are shown in Definition 19:

Definition 19. *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.*

APPENDIX C REAL ENVIRONMENT IN (CH. 6)

Table C.1 displays the coordinate information of the enclosed regions in the real environment; Table C.2 shows the specifications of the Agilex Limo; Table C.3 shows the key performance metrics of the OptiTrack system.

Table C.1 Enclosed regions information (yellow lines arrays) in the real environment (in meters)

Zone	Coordinates
zone_left_down	[[0.0, 0.0], [-1.0, 0.0]], [[-1.0, 0.0], [-1.0, -1.0]], [[-1.0, -1.0], [0.0, -1.0]], [[0.0, -1.0], [0.0, -0.75]], [[0.0, -0.75], [-0.57, -0.75]], [[-0.57, -0.75], [-0.57, -0.3]], [[-0.57, -0.3], [0.0, -0.3]], [[0.0, -0.3], [0.0, 0.0]]
zone_left_up	[[0.0, 0.4], [0.0, 1.0]], [[0.0, 1.0], [-1.0, 1.0]], [[-1.0, 1.0], [-1.0, 0.4]], [[-1.0, 0.4], [0.0, 0.4]]
zone_right_up	[[0.5, 0.0], [1.0, 0.0]], [[1.0, 0.0], [1.0, 1.0]], [[1.0, 1.0], [0.5, 1.0]], [[0.5, 1.0], [0.5, 0.0]]
zone_right_down	[[0.5, -1.0], [0.5, -0.5]], [[0.5, -0.5], [1.0, -0.5]], [[1.0, -0.5], [1.0, -1.0]], [[1.0, -1.0], [0.5, -1.0]]

Table C.2 AgileX Limo Specifications [1]

Category	Specifications
Dimensions	322mm x 220mm x 251mm
Weight	4.2 kg
Maximum Speed	1 m/s
Maximum Climbing Capacity	25° (omni-wheel, differential, Ackermann steering), 40° (tracked mode)
Ground Clearance	24 mm
Battery	12V Li-ion 5600mAh
Run Time	40 minutes of continuous operation
Standby Time	2 hours
Charging Time	2 hours
Operating Temperature	-10°C to 40°C
IP Rating	IP22 (splash-proof, dust-proof)
Sensors	IMU: MPU6050 LiDAR: EAI X2L Depth Camera: ORBBEC DaBai
CPU	ARM64 Quad-Core 1.43GHz (Cortex-A57)
GPU	128-core NVIDIA Maxwell™ @ 921MHz
Communication	Wi-Fi, Bluetooth 5.0
Operating System	Ubuntu 18.04, ROS1 Melodic

Table C.3 Key Performance Metrics of the OptiTrack System [2]

Performance Metric	**Value**	**Description**
Camera Resolution	Up to 4096x2160	High resolution for detailed tracking
Frame Rate	Up to 360 FPS	Ensures smooth motion capture
Latency	As low as 3ms	Provides real-time feedback
Field of View (FOV)	56° to 100°	Suitable for various capture needs
Synchronization Precision	<1μs	Multi-camera synchronized capture
Marker Tracking Accuracy	<0.5mm	Precise 3D spatial positioning
Operating Range	Up to 30m	Suitable for large-scale capture environments
Optical Resolution	12 MP	High-quality image data
Ambient Light Suppression	Strong	Reduces interference from ambient light
Data Interface	Gigabit Ethernet	Fast data transmission

APPENDIX D SIMULATION ENVIRONMENT IN (CH. 6)

The position of the goal is randomized from these eight points in each episode. Setting multiple endpoints offers several advantages. It enhances the navigation policy's generalization capability by training the agent to adapt to various goals rather than a single target. This promotes extensive exploration, prevents the agent from getting stuck in local optima, and increases robustness by preparing the agent to handle dynamic or uncertain target positions in real-world deployments [189].

Additionally, it improves data efficiency by allowing the agent to learn from diverse experiences in a single training session. Multiple endpoints provide more prosperous reward signals, accelerating learning through varied success and failure contexts. Moreover, it simulates realistic scenarios where multiple destinations are shared, thus increasing the practical value of the trained model. Lastly, this approach raises task complexity, challenging the agent to develop more sophisticated strategies and ultimately enhancing overall performance [190].

Table D.1 List of goal positions with their coordinates

Goal Position	**Coordinates**
Goal Position 1	[1.2, 1.2]
Goal Position 2	[1.2, -1.2]
Goal Position 3	[-1.2, 1.2]
Goal Position 4	[-1.2, -1.2]
Goal Position 5	[1.2, 0]
Goal Position 6	[0, 1.2]
Goal Position 7	[-1.2, 0]
Goal Position 8	[0, -1.2]

The start point is a fixed point as: [-0.2, -0.5]. To enhance the stability of the agent's initial position and reduce the risk of misalignment in real-world deployment, we introduce uniform noise to the starting coordinates. This approach aims to minimize data uncertainty caused by position discrepancies. By adding noise uniformly in the range $[-0.05, 0.05]$, we can achieve the desired effect. Let:

- $\mathrm{a}_{\mathrm{location}}$ be the agent's position
- $U(a, b)$ denote a uniform distribution in the range $[a, b]$

The updated position formula is:

$$\mathbf{a}_{\text{location}} = \begin{bmatrix} -0.2 + U(-0.05, 0.05) \\ -0.5 + U(-0.05, 0.05) \end{bmatrix}$$

This method ensures that the uniform noise maintains the desired variability without introducing bias [191].

All in all, the state dimension is 49, and their shape and space are shown in Table B.2.

Table D.2 Noteable, Limo cannot make the control duration correctly if the duration is not in this time range.

State details			
Name	Shape	Space	Annotation
Limo Position	$[2,]$	$[-1.5, 1.5]$	in (X, Y), (meters)
Goal Position	$[2,]$	$[-1.5, 1.5]$	in (X, Y), (meters)
Linear Velocity	$[1,]$	$[-1.0, 1.0]$	
Steering Angle	$[1,]$	$[-1.0, 1.0]$	
Control duration	$[1,]$	$[0.02, 0.5]$	in Seconds
Previous linear velocity	$[1,]$	$[-1.0, 1.0]$	
Angular velocity	$[1,]$	$[-1.0, 1.0]$	
20 radar point positions	$[40,]$	$[-1.5, 1.5]$	reshape from $[20, 2]$, in (X, Y)

Table D.3 Noteable, Limo cannot make the control duration correctly if the duration is not in this time range.

Action details			
Name	Shape	Space	Annotation
Control duration	$[1,]$	$[0.02, 0.5]$	in Seconds
Linear velocity	$[1,]$	$[-1.0, 1.0]$	
Angular velocity	$[1,]$	$[-1.0, 1.0]$	

Table D.4 Reward Value Settings for Limo Environment

Reward Settings		
Name	Value	Annotation
cross_punish	-30.0	cross with the enclosed regions
success_reward	500.0	success to reach the goal
dead_punish	-100.0	go out of the map
δ	0.2	if limo close enough to a point, in meters

APPENDIX E SYSTEM DETAILS IN (CH. 6)

Table E.1 Training PC Details

PC Key Softwares' Ecosystem	
Name	Value
Nvidia Driver Version	450.67
Cuda Version	11.8
cuDNN Version	8.9.7
Python Version	3.8
Torch Version	2.1.0
ONNX Version	1.13.1
Gymnasium Version	0.29.1

Table E.2 Agilex Limo Details

Agilex Limo Key Softwares' Ecosystem	
Name	Value
Nvidia JetPack Version	4.6.4
Cuda Version	10.2_r440
cuDNN Version	8.2.1
TensorRT Version	8.2.1.3
Python Version	3.6
Pycuda Version	2020.1
ONNX Version	1.13.1
Limo Controller Version	2.4

Our experiments used the Agilex Limo equipped with a Jetson Nano running JetPack version 4.6.4. While newer versions such as 6.0+ are available, compatibility and support limitations necessitated our use of JetPack 4.6.4. Specifically, the older versions of cuDNN and TensorRT required for our setup are no longer available for direct download, although this may affect reproducibility. To assist others in replicating our results, we have provided custom-built support packages on our GitHub.

APPENDIX F HYPERPARAMETER SETTING OF SEAC IN (CH. 3)

Table F.1 Hyperparameters Setting of SEAC

Hyperparameter sheet		
Name	Value	Annotation
Total steps	$3e6$	
γ	0.99	Discount factor
Net shape	$(256, 256)$	
batch_size	256	
a_lr	$2e4$	Learning rate of Actor Network
c_lr	$2e4$	Learning rate of Critic Network
max_steps	500	Maximum steps for one episode
α	0.12	
η	-3	Refer to SAC [72]
T_i	5.0	Time duration of rest compared RL algorithms, in HZ
min_frequency	1.0	Minimum control frequency, in HZ
max_frequency	100.0	Maximum control frequency, in HZ
Optimizer	Adam	Refer to Adam [192]
environment steps	1	
Replaybuffer size	$1e6$	
Number of samples before training start	$5 \cdot max\_steps$	
Number of critics	2	

APPENDIX G HYPERPARAMETERS OF MOSEAC IN (CH. 4)

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 [72]
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 8)
ψ	$1e-4$	Monotonically increasing H-parameter (Algorithm 3)
Optimizer	Adam	Refer to Adam [192]
environment steps	1	
Replaybuffer size	$1e6$	
Number of samples before training start	$5 \cdot max\_steps$	
Number of critics	2	

APPENDIX H CONVERGENCE ANALYSIS OF MOSEAC IN (CH. 4)

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 4.3.

SAC Algorithm Overview In the standard SAC algorithm [45], 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.

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 3.f

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.

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.

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.

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 11).
   - α_ε decreases with the Definition 10.

2. **Learning Rate Conditions:**
   The learning rates α_k and β_k must satisfy the following conditions [41]:

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

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

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.

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.

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. [29],

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

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

APPENDIX I ANALYSIS OF THE ADAPTIVE α_M ADJUSTMENT IN MOSEAC IN (CH. 6)

In MOSEAC we use an adaptive adjustment scheme for α_m, specifically a simple linear increment with an upper limit $\alpha_{\max}$. The inclusion of ψ and the reward adjustment strategy inherently involves Pareto optimization. Below, we provide a mathematical analysis of the stability of our scheme and its relation to the Pareto front.

Adaptive Adjustment Scheme for α_m

Let α_m be adjusted linearly over time with an upper limit $\alpha_{\max}$:

$$\alpha_m(t) = \min(\alpha_{m,0} + kt, \alpha_{\max})$$

where $\alpha_{m,0}$ is the initial value, k is the increment rate, and t represents the time or iteration index.

The stability of this linear adjustment can be analyzed by examining the impact on the reward function R and the policy updates.

Reward Function and ψ

The reward function R in the MOSEAC algorithm can be expressed as:

$$R = r + \psi$$

where r is the immediate reward and ψ is an adjustment term that influences the reward based on the adaptive α_m.

Given the linear increment of α_m, the adjustment term ψ can be represented as:

$$\psi(t) = f(\alpha_m(t)) = f(\alpha_{m,0} + kt) \text{ for } \alpha_m(t) < \alpha_{\max}$$

$$\psi(t) = f(\alpha_{\max}) \text{ for } \alpha_m(t) \geq \alpha_{\max}$$

where f is a function that defines how ψ depends on α_m.

Stability Analysis

To analyze the stability of the reward function under this adaptive scheme, we examine the boundedness and convergence of R. The stability is ensured if the cumulative reward remains bounded and the policy converges to an optimal policy over time.

1. **Boundedness**: The linear increment of α_m with an upper limit should ensure that R does not grow unbounded. This requires that $f(\alpha_m)$ grows at a controlled rate.

   $$|R| = |r + f(\alpha_{m,0} + kt)| \leq M \text{ for } \alpha_m(t) < \alpha_{\max}$$

   $$|R| = |r + f(\alpha_{\max})| \leq M \text{ for } \alpha_m(t) \geq \alpha_{\max}$$

   where M is a constant, indicating that R remains bounded.

2. **Convergence**: The policy π should converge to an optimal policy π^*. The convergence is influenced by the adjustment term ψ and the learning rate α_m.

   $$\lim_{t \to \infty} \pi(t) = \pi^*$$

   If α_m is adjusted linearly with an upper limit, the learning rate should decrease over time to ensure convergence.

Multi-Objective Optimization

The multi-objective optimization aspect of the MOSEAC algorithm arises from the trade-off between multiple objectives in the reward function. The inclusion of ψ introduces a multi-objective optimization problem, where the algorithm aims to optimize the cumulative reward while balancing different aspects influenced by ψ.

The Pareto front represents the set of optimal policies where no single objective can be improved without degrading another [193]. The linear increment of α_m with an upper limit and the presence of ψ ensure that the algorithm explores the policy space effectively, converging to solutions on the Pareto front [194].

APPENDIX J STATISTICS FOR ENERGY AND TIME CONSUMPTIONS IN (CH. 6)

Statistic results for the MOSEAC and SEAC on energy and time consumptions are shown in Table J.1.

Table J.1 Descriptives

	N	Mean	SD	SE	COV
MOSEAC_Energy	100	10.130	4.733	0.473	0.467
SEAC_Energy	100	11.660	4.740	0.474	0.407
MOSEAC_Time	100	4.341	2.038	0.204	0.469
SEAC_Time	100	4.456	2.044	0.204	0.459

Table J.2 Test of Normality (Shapiro-Wilk)

			W	p
MOSEAC_energy	-	SEAC_energy	0.916	< 0.001
MOSEAC_time	-	SEAC_time	0.993	0.894

Table J.3 Paired Samples T-Test

Measure 1		Measure 2	W	z	p
MOSEAC_Energy	-	SEAC_Energy	35.000	-7.904	<0.001
MOSEAC_Time	-	SEAC_Time	502.000	-6.956	<0.001

APPENDIX K STATISTICS FOR COMPUTE RESOURCES IN (CH. 6)

Table K.1 Descriptives

	N	Mean	SD	SE	COV
MOSEAC CPU usage (%)	88	11.400	0.370	0.131	0.032
SAC 10 Hz CPU usage (%)	88	16.800	0.400	0.141	0.024
SAC 60 Hz CPU usage (%)	88	31.413	1.298	0.459	0.041
MOSEAC GPU usage (%)	88	2.800	0.283	0.100	0.101
SAC 10 Hz GPU usage (%)	88	13.787	0.242	0.085	0.018
SAC 60 Hz GPU usage (%)	88	27.863	0.463	0.164	0.017

Table K.2 Test of Normality (Shapiro-Wilk)

		W	p
Measure 1	Measure 2	W	p
MOSEAC CPU usage (%)	SAC 10 Hz CPU usage (%)	0.947	0.685
	SAC 60 Hz CPU usage (%)	0.960	0.806
MOSEAC GPU usage (%)	SAC 10 Hz GPU usage (%)	0.835	0.067
	SAC 60 Hz GPU usage (%)	0.901	0.296

Table K.3 Paired Samples T-Test

Measure 1		Measure 2	W	z	p
MOSEAC CPU usage (%)	-	SAC 10 Hz CPU usage (%)	-	-2.521	0.004
	-	SAC 60 Hz CPU usage (%)	-	-2.521	0.004
MOSEAC GPU usage (%)	-	SAC 10 Hz GPU usage (%)	-	-2.521	0.007
	-	SAC 60 Hz GPU usage (%)	-	-2.521	0.007