Decision Frequency Adaptation in Reinforcement Learning Using Continuous Options with Open-Loop Policies

by

Amirmohammad Karimi

A thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science

Department of Computing Science

University of Alberta

Abstract

In classic reinforcement learning(RL) for continuous control, agents make decisions at discrete and fixed time intervals. The duration between decisions becomes a crucial hyperparameter. Setting it too short may increase the problem's difficulty by requiring the agent to make numerous decisions to achieve its goal, while setting it too long can result in the agent losing control over the system. However, physical systems do not necessarily require a constant control frequency. For learning agents, it is often preferable to make decisions with a low frequency when possible and a high frequency when necessary. Previously, control frequency adaptation methods in temporal-abstraction RL have been proposed. However, like classic RL, these methods often do not consider physical time and treat task time steps discretely. This can make the learning experience sensitive to the underlying task interaction frequency. We propose a framework called Continuous-Time Continuous-Options (CTCO), where the agent chooses options as open-loop sub-policies of variable durations. These options are defined in continuous time and can interact with the system at any desired frequency providing smooth extended continuous actions. We demonstrate the effectiveness of CTCO by comparing its performance to classical RL and temporal-abstraction RL methods on simulated and real-world continuous control tasks with various action-cycle times. We show that our algorithm's performance is not affected by the choice of task interaction frequency. Moreover, we show the benefit of having open-loop options over simple action repetition. Furthermore, we demonstrate the efficacy of CTCO in facilitating exploration in a real-world visual reaching task with sparse rewards for a 7 DOF robotic arm.

Preface

The main idea of continuous-time continuous options was formed in my collaboration with Samuele Tosatto; we equally contributed to the policy structure design and policy gradient theorem derivation. The idea of using normalized RBFs was given by Samuele. I designed and implemented the CTCO actor-critic framework and the experiments in this thesis. Jun Jin instructed me on how to set up and control the Franka Emika robot arm. Design and implementation of the robotic visual-reacher task is done by myself.

The Results of this work were submitted as A. Karimi, J. Jin, J. Luo, R. Mahmood, M. Jagersand and S. Tosatto, "Variable Decision Frequency Option Critic," to 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2023).

Acknowledgements

I want to express my sincere gratitude to my supervisors, Martin Jagersand and Rupam Mahmood, for their support, guidance, and encouragement throughout my master's program. Their expertise has been instrumental in shaping my research and shaping me as a researcher. I am truly fortunate to have had such great mentors.

I would also like to extend my special thanks to Samuele Tosatto, whose supervision and counselling played a crucial role in completing this thesis. I am grateful for his guidance every step of the way.

I wish to acknowledge Martha White, my committee member, for her valuable time and effort in reviewing this thesis.

I am thankful to all the Reinforcement Learning and Artificial Intelligence (RLAI) group and the Robot LAIR lab members at the University of Alberta.

I thank the University of Alberta, Alberta Machine Intelligence Institute (Amii), and the CCAI Chairs Program for funding this program and the Huawei Noah's Ark Lab for providing the Franka-Emika robot arm. In addition, I acknowledge the access to Compute Canada, whose resources enabled me to run experiments efficiently.

Finally, I am immensely grateful to my family and friends for their love and encouragement throughout my journey. Their support kept me motivated during challenging times, and I am genuinely thankful for their presence in my life.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Robotics has the potential to improve human life by automating a wide range of tasks. However, classical methods for controlling robots are often not equipped to handle the complexity of these tasks. The desired behaviours in such tasks can be highly complex to express, and those tasks are often highly dynamic in nature. Reinforcement learning (RL) is a promising approach to learning how to perform such tasks. RL enables the agent to learn desired behaviours through experience and interaction with the task [39]. In RL, the agent takes actions at specific time-steps, which are often determined by the task itself. In continuous control and robotics tasks, these time-steps are typically equally spaced in real-time, i.e. every Δt second. However, current state-of-the-art RL methods ignore the physical time between these time-steps and interpret them as discrete events. This can lead to varying learning performances across different interaction frequencies [42]. A favourable RL agent should consider the physical time and be able to perform robustly across different interaction time scales.

RL agents select actions from their policy based on the current state of the environment, and these actions are applied at each time-step. When the actions are primitive or low-level, the decision process occurs at every time-step, which can provide a wide range of control behaviours. However, this approach has certain drawbacks from the perspective of the learning agent. In continuous control tasks, the effect of actions on the environment state takes time to develop. This results in high sample complexity and challenging exploration. For instance, a random uniform policy applied at high frequency often only explores a limited area around the initial state. In particular, learning in goal-based tasks or settings with sparse rewards may drastically suffer from ineffective exploration [1], [30], [47]. In terms of policy learning, it becomes increasingly challenging for an agent to determine the optimal action as the time interval Δt approaches zero in action-value-based methods. This is because the advantage of each action becomes insignificant [4], [42]. Moreover, policy gradient methods are also ineffective in this scenario due to the divergent variance of the policy gradient [30].

We argue that the decision frequency does not necessarily need to be fixed to the interaction frequency. Typically, the decision frequency should be higher when the system is hard to control, while it can be lower otherwise [15]. A good example is the swing-up pendulum task. During the "swinging phase", the system is simple to control, and one can decide to swing left or right with low frequency until the pendulum is upright. In the stabilization phase, however, the system is unstable, and one needs to apply frequent small torques in different directions to compensate for perturbations. In other words, one can make the learning algorithm more efficient by allowing the agent to make decisions less frequently when possible.

We suppose a continuous control task is represented as a base Markov Decision Process (MDP) with fixed time-steps of duration Δt seconds. Our goal is to enable our agent to adapt its decision frequency by forming extended continuous courses of action that can last for multiple numbers of time-steps in the base MDP. Our approach involves utilizing the options framework introduced by Sutton *et al.* [41]. We represent the extended actions as options with open-loop policies and termination conditions that are defined by a time-out function. Specifically, the agent selects a trajectory of continuous actions as well as the duration of executing this trajectory in continuous time. This approach transforms the base MDP to a new augmented MDP where time-steps are not necessarily equally spaced in time. The action space of this MDP would be the option policy and its duration in the base MDP.

The concept of explicitly choosing the duration of extended actions has been studied in a body of works for adapting the control frequency which are known as action repetition RL (ARRL) methods [19], [24], [26], [36]. In ARRL methods, the agent repeats an action for multiple time-steps. In contrast to ARRL methods, where the agent chooses the number of repetitions from a finite set of integers, we choose to sample the duration in seconds as a real number. In this way, the choice of durations is not sensitive to the frequency of interaction. There have been works to compensate for this limitation that scale the maximum possible number of repetitions with frequency [30]. However, the increased size of possible repetitions intensifies the problem's difficulty in both value-based and policy-gradient methods.

From the view of action abstraction, policies based on repeating an action are limited in representational power and exploration, especially in continuous control tasks where a smoothly changing control signal may be desirable. Our method provides smooth courses of action as options. Options are introduced in Hierarchical Reinforcement Learning (HRL), where the agent increases the level of abstraction while maintaining flexibility in control [38]. Better exploration and achieving higher learning performance are shown when using options both in discrete and continuous action spaces [3], [50]. However, the need to learn each sub-policy brings the same issues as in classical RL when the physical time between time-steps is ignored. To address this problem, we set options to have open-loop policies that output primitive actions as a function of the physical time elapsed from the start of option execution. Instead of discrete values, options in our method are chosen as continuous values that parameterize the option policy.

1.1 Contributions

In this thesis, we propose the Continous-Time Continuous-Options (CTCO) framework as a new approach to reinforcement learning for continuous control. The contributions of this work can be summarized as follows:

- Our framework can be seen as a hybrid between full HRL and Action Repetition RL. We develop a system that utilizes temporal abstraction that selects options similar to HRL but from a continuous multidimensional space $\boldsymbol{\Omega}$. We form the sub-policy associated with an option as an open-loop controller parameterized with $\boldsymbol{\omega} \in \boldsymbol{\Omega}$. Normalized Radial-basis functions are used to represent these sub-policies that change in time, providing more expressive options than a constant action repetition.

- CTCO augments the space of option policies with continuous values defining option durations. This duration determines the termination condition of the selected option. In this way, our approach can adapt its decision frequency regardless of the interaction frequency. Building on the state-of-the-art classical RL method, soft-actor-critic(SAC) introduced by Haarnoja *et al.* [14], we have implemented a SAC-based algorithm where the actions are pairs of an option and its duration.

- We perform a series of experiments to show the robustness of our framework to different interaction frequencies as well as the possible advantages of open-loop options over repeated actions. We set the performance measure as in continuous-time RL defined by Yildiz *et al.* [46] and compare the performance of CTCO against one method from each of the classical, hierarchical, and action repetition RL methods. Our experiments are conducted in simulated classic RL benchmarks from DeepMind Control Suite and on a 7-DoF real robotic manipulator consisting of sparse and dense reward settings.

This thesis is structured as follows. In Chapter 2, we provide an introduction to the concepts of robot learning, focusing on reinforcement learning in the context of robotics and continuous control. We also discuss related works in decision frequency adaptation. Chapter 3 presents the fundamental concepts of reinforcement learning, which serve as the building blocks for our proposed methods. In Chapter 4, we introduce our framework CTCO, including its policy formulation and the actor-critic framework. Chapter 5 covers the empirical analysis of our method. Finally, in Chapter 6, we summarize the work done, evaluation results, and discuss limitations and possible future directions.

Chapter 2

Background

This chapter presents a brief literature review of robot learning techniques, with an emphasis on reinforcement learning. It commences by framing learning tasks as the acquisition of policies and skills, along with introducing various policy structures. Subsequently, the method of reinforcement learning in the context of robotics is discussed. Finally, state-of-the-art reinforcement learning approaches for addressing the challenge of adapting control frequency are reviewed. Reinforcement learning, the technique employed in this thesis, is given particular attention and elaborated on further in the technical background section.

2.1 Robot Learning

There is a vast range of potential applications for robots, such as in services, homes, factories, healthcare and others. In each of these fields, there could be a variety of utilizations of a robot in different environments requiring the robot to make changes in the environment through observing states and applying actions. These environments are typically unstructured and unpredictable, where the robot may have to deal with unforeseen and new situations that the robot's designers do not expect. These challenges make hand design and programming by humans tedious and mostly impossible. Therefore, a part of the research in robotics has been trying to answer how a robotic agent can learn to affect its environment to achieve some goal given the environmental challenges. That research expands on methods in classical control, learning from human demonstrations, high-level planning and reinforcement learning [16], [18].

2.2 Learning a Skill Policy

Performing a task means achieving some objective. A desired behaviour or skill is what an agent follows to achieve the task objective. Therefore, learning to do a task is learning such skills. In the context of learning a skill, a policy is a set of rules or guidelines that determine how an agent should behave in a given state to achieve the desired objective. Typically skills are formed as stochastic policies where the policy maps state to action probabilities [18]. Two important components of policies are the action space, which defines the set of actions that an agent can take in response to its environment, and the policy structure, which determines how the agent chooses which action to take in a given situation. In the following, we will explore different types of action spaces, followed by various policy structures used in robot learning.

2.2.1 Action Spaces

In order to make changes in the environment, a robot needs to send control signals to its actuators, such as the torque of an electric motor, pressure in hydraulic actuators and force in cable-driven actuators. The action space chosen for the policy is closely related to the control signal. Selecting the actions as the same as the control signals can benefit robots with complex models and dynamics. However, in most applications, an additional controller exists between the policy and the actuator. Using an extra controller enables designers to benefit from prior works in control for robotics, such as PID controllers. Desired values to be controlled can be the actions of the policy; for instance, the policy may choose the positions, velocities and accelerations of robot arm joints or torque and forces of an electric motor as the input for a controller. The policy may also determine other values associated with the configurations of the controller, such as gains of a PID controller in the action space. Another benefit of having an additional controller is that the policy may work at a lower frequency than the frequency of interaction with a robot actuator. For example, a controller can work at 1KHz, but policy actions are chosen at a much less rate [18].

2.2.2 Policy Structures

The choice of the policy structure directly influences the policy's representational power in terms of how restricted it can be. Very general representations can bring the finest level of control in a task and achieve optimal behaviour; however, more limited representations may result in better data efficiency and generalization [16], [18]. Therefore the choice of the policy structure is critical as it determines the class of behaviour. The following proposes a range of highly general to highly restricted policy structures.

Non-parametric Policies

Non-parametric policies are the most expressive representations. The size of the policy can grow as needed by the complexity and data size in training. We can mention Nearest-neighbor approaches, gaussian processes, Riemannian motion policies and locally weighted regression methods in this area [2], [32]. In these methods achieving high-quality generalization typically needs a large amount of data [18].

Generic Fixed-size Parametric Policies

Policies with these structures have a fixed-complexity parametric representation. The structure and representation power of the policy is assumed to be more restricted than non-parametric policies. Look-up tables in discrete settings and linear combinations of basis functions such as tile coding [39], the Fourier basis [17], neural networks [21], decision tree classifiers, and support vector machines in continuous settings are among examples of fixed-size parametric policies. In designing these kinds of policies, the choice of number and definitions of the parameters matters, given that the representational power is fixed. For instance, tabular representations can describe any discrete function but cannot generalize to unseen states and actions; on the other hand, policies consisting of linear combinations of basis functions can naturally generalize to novel conditions both in discrete and continuous settings as a change in one parameter may globally change the policy behaviour. It should be noted that the success in generalization depends on the assumptions made about the shape and smoothness of the policy.

Restricted Parametric Policies

As policy representation grows in complexity, issues such as overfitting, sample complexity, and generalization become more noticeable, particularly in robotics tasks. Consequently, researchers have proposed policies with structures specifically customized for robotics to mitigate these challenges. Some of the constrained approaches are outlined below:

Linear Quadratic Regulators: These methods try to find optimal trajectories around a given or learned trajectory, assuming that the system's dynamics are linear, the objective is quadratic in the state, and the state is fully observable. Achieving the optimal behaviour is guaranteed when the linear-quadratic conditions are met, and knowledge of dynamics and cost function are fully known along with a known trajectory to stabilize around [51].

Dynamic Movement Primitives: DMPs allow for learning simple and generalizable policies from a few demonstrations of skills. These policies can also be improved by reinforcement learning [34]. Many robotic movements are formed by a goal configuration and the shape of the movement. DMPs benefit from this fact by deploying differential equations for converging to a goal configuration in the movement while shaping the movement using non-linear functions. A little data is needed for learning DMPs, and generalization could be achieved in cases where the class of motions comprehensively describes the desired skills. While policies learned in these methods are deterministic, there are other approaches, such as probabilistic variants of DMP frameworks called ProMPs which can generate distributions of trajectories [29] as well as Gaussian Mixture Regressions [9] which models the likelihood of states over time and can produce multimodal distributions over trajectories describing different behaviours with some variance.

Goal-based Policies

The most limited types of policy structures are those that are goal-based, where the policy parameters are primarily used to achieve a specific goal configuration. As a result, the strategies produced by these policies are usually fixed, such as navigating to a specific goal point [37]. In some cases, only a small number of parameters may be adjusted, such as in the case of PID controllers or motion planning.

2.3 Reinforcement Learning in the Context of Robotics

In reinforcement learning (RL), robots (agents) can autonomously learn skills through trial-and-error interactions with an environment. Each interaction is typically defined as observing the environment, taking action, and getting scalar feedback called reward over time-steps determined by the task. Reinforcement learning aims to learn a policy that maximizes the sum of rewards over time. A wide variety of problems in robotics can be framed in this scheme. Also, RL can be used to learn the parameters of any given policy representation as long as a reward function is present to promote the desired behaviour. Researchers have used RL to train robots to perform tasks such as grasping objects, flying helicopters, and walking on two or four legs [16], [18].

There are several different methods in RL that can be broadly grouped into the categories of model-free or model-based and value-based or policy-search methods. In the following, each category is introduced with examples of applications in robotics.

Model-based Methods

In these methods, transition models are learned from data which are often hard to learn and inaccurate. Researchers have used approximated transition models to aid exploration and policy search and improve data efficiency [11], [12], [20]. These methods are advantageous when learning a model of transition is more straightforward than learning the optimal policy. Also, if there are non-stationarities or changes in the task, the learned model can be transferred for learning a new policy without needing additional data acquisition. Moreover, especially in robotics, having a transition model can help sampling data where it is hard or impossible due to practical issues such as putting the robot in a particular state and sampling different actions. On the other hand, as mentioned earlier, learning accurate transition models is complicated, and the inaccuracy in models can inject bias in learning the policy.

Model-free Methods

In these methods, policies are learned directly via interactions without access to a transition model. Model-free methods are favourable in tasks with complex dynamics where learning a transition model is more challenging than learning a policy. However, model-free methods may need much more data than model-based methods; also, if there are changes in the task, additional data collecting is needed for learning a new policy. According to Lillicrap *et al.* [22], these techniques have been employed to control a robotic arm to reach a target. Similarly, Mnih *et al.* have implemented these methods for playing Atari games. In addition, real-world applications of robotic manipulation and locomotion have also been explored by Levine *et al.* [21], Mahmood *et al.* [23], and Bloesch *et al.* [5].

Value-based Methods

A part of RL methods tries to learn the value of states or state-action pairs to select actions with the highest value given a state. Low variance properties, sample efficiency and guaranteed convergence to optimal policies in discrete settings make value-based methods favourable. These methods may not be suitable for robotics tasks when there is noise, state features are poor, the state space is high-dimensional, and the action space is continuous. Mnih *et al.* [27] and Zhang *et al.* [49] are among the works that use Deep-Q networks as value-based methods in robotics.

Policy Search Methods

In contrast to value-based methods, policy search methods directly parameterize the policy and search for parameters. These methods are a good match for robotic tasks as they can handle continuous action spaces and scale well with high dimensional states. Policy search methods are either actor-only or actor-critic. Actor-only methods tend to directly optimize the policy without learning any value function. Most of these methods that estimate the gradient of the policy are based on the REINFORCE algorithm [45]. These methods often suffer from high variance in gradient estimations. On the other hand, actor-critic methods learn both parameterized policy as the actor, and value function as the critic that can have shared parameters [14], [20], [25], [35], [40]. These methods have the advantage of both value-based methods and policy search methods. Because of using a bootstrapped approximation of value-function, the variance in gradient estimations is decreased, and sample efficiency is achieved. These advantages make them state-of-the-art methods in reinforcement learning and robotics and the larger area of deep RL [13], [33].

2.4 Related Works in Decision Frequency Adaptation

2.4.1 Action Repetition

Static Action Repetition

In both real-life and simulated environments, the concept of repeating actions is a common occurrence used by agents to control their behaviour during task execution. Traditional methods typically employ a fixed action repetition scheme, where the agent repeats the same action for a specified number of time steps. For example, Braylan *et al.* [7] studied the effect of different frame skip values on performance in Atari games and found that some games exhibited significantly better performance with higher frame skip values.

In their paper, Metelli *et al.* [24] proposed a new hyper-parameter for improving performance in reinforcement learning called “action persistence.” This parameter is represented by a fixed value, k, which specifies how many times an action should be repeated before it is changed to modify the control frequency. The authors developed a new k-persistent policy based on a classical Markovian stationary policy. They also introduced a Persistent Bellman operator derived from the original Bellman operator. They implemented the Persistent Fitted Q-Iteration algorithm, which can estimate the value function at a given persistence. The experimental results presented in the paper demonstrated that introducing persistence can improve sample efficiency, as reducing the control frequency can lead to better performance.

The paper by Dabney *et al.* [10] introduced “temporally-extended ϵz-greedy exploration”. This method involves selecting an exploration probability ϵ, a set of options Ω, and a sampling distribution p that has support on Ω. At each step, the agent either follows the current policy π for one step with probability $1 - \epsilon$, or with probability ϵ, it samples a new option and executes it until it reaches a termination condition. The authors demonstrated the effectiveness of this method in both tabular and deep reinforcement learning, particularly in environments with sparse rewards, where it can lead to significant improvements in exploration and performance.

Dynamic Action Repetition

This part explores various approaches for achieving dynamic action repetition, which refers to an agent’s ability to select different repetition rates.

Lakshminarayanan *et al.* [19] proposed enlarging the action space by pairing actions with a specific repetition value. The authors employed this framework with two popular Deep RL algorithms, DQN and A3C, and applied them mainly to Atari games. The Augmented DQN was implemented by duplicating the network’s last layer and outputting the Q-values for actions at two different repetition rates. However, a drawback of this approach is that repetition rates are hyperparameters, and there is no automatic adaptation of frequency.

Another approach, proposed by Sharma *et al.* [36], introduces the concept of a skip network that is used jointly with the original network to select the action repetition, deploying policy gradient methods. However, a drawback of this implementation is that the second network is not action-dependent, meaning that the action repetition chosen in a specific state is an average of all actions. This can lead to sub-optimal performance in states where one action is impulsive, and the other has a high action repetition value.

Another approach to achieving dynamic action repetition is presented by Yu *et al.* [47]. The authors propose a secondary binary policy to decide whether to repeat the previous action or select a new one. This method, called Temporally Abstract Actor-Critic (TAAC), extends a policy gradient method where the primary network selects a new action at each state. However, the action is only chosen once a second network decides whether it is more advantageous to repeat the previous action or choose a new one.

In previous related work of Ni and Jang [28], the task is framed in continuous-time MDP where the agent selects a primitive action and the continuous time to apply it on the environment. In their approach, the time scale is chosen from a predetermined interval between δ_{min} and δ_{max}. The reward function is approximated to be constant during the action execution. Additionally, a penalty is used for choosing small time-scales by constraining the average time scale. The policy is optimized using soft-actor-critic. The authors have only tested their approach on simulated tasks without real-world experiments.

Chapter 3

Technical Background

This chapter provides technical background for the proposed reinforcement learning method by introducing the key concepts and components that serve as the building blocks. These include the basic definitions in RL, function approximation in RL, actor-critic policy gradient methods, maximum entropy RL, and temporal abstraction and its relation to continuous-time RL. Understanding these building blocks is crucial to grasp the proposed method and its underlying principles fully.

3.1 Reinforcement Learning Framework

3.1.1 Definitions

In reinforcement learning, the goal is to learn an optimal behaviour to maximize a scalar accumulative return through trial and error [39]. RL frameworks have two components called agent and environment that interact with each other. The agent and environment interaction is often supposed to happen in a Markov Decision Process (MDP) framework. An MDP is defined as a tuple $\mathcal{M} := (\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \gamma, \mu_0)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ defines the reward function, $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}^+$ is the transition probability distribution, $\gamma \in [0, 1)$ is the discounting factor, and μ_0 is the initial state distribution [39].

In the above discrete MDP formulation, an agent interacts with the environment in time-steps. At each time-step $t \in \mathbb{Z}$, the agent receives the current environment state $\mathbf{s}_t$ and, using a policy π, takes action $\mathbf{a}_t$. In the next time-step, a reward $r_{t+1} = r(\mathbf{s}_t, \mathbf{a}_t)$ and state $\mathbf{s}_{t+1} \sim p(\cdot \mid \mathbf{s}_t, \mathbf{a}_t)$ will be observed by the agent. In this way, for each initial state $\mathbf{s}_0 \sim \mu_0$, an interaction trajectory of $\mathbf{s}_0, \mathbf{a}_0, r_1, \mathbf{s}_1, \mathbf{a}_1, r_2, \ldots$ is produced.

Generally, in an MDP setting, a stochastic policy π can be defined as a mapping from the trajectory of interactions past each time-step t, $\mathcal{H}^t = (\mathbf{s}_0, \mathbf{a}_0, r_0, \cdots, \mathbf{s}_t)$ to probabilities over the action-space as $\pi : \mathcal{H} \times \mathcal{A} \to \mathbb{R}^+$. A Markovian assumption for the policy results in reducing the past trajectory to the current state of the agent and leads to the policy definition as $\pi : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^+$. Thus, an agent takes actions by sampling $\mathbf{a}_t \sim \pi(\cdot \mid \mathbf{s}_t)$. A policy can also be deterministic when it maps a state to a particular action.

Tasks in RL are defined either as episodic or continuing. An episodic task has a termination state $\mathbf{s}_T$. When the agent arrives at it, the interaction trajectory is ended, and the environment state will be reset with respect to the initial state distribution μ_0. On the other hand, in continuing tasks, the agent will continue to interact with the environment for an infinite number of time-steps. In this thesis, all the tasks are considered episodic.

The performance of an RL agent is measured based on the discounted accumulated rewards, known as return when following its policy π as

$$J_\pi := \mathop{\mathbb{E}}_{\mu_\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)\right].$$

Here, μ_π shows the distribution of interaction trajectories associated with policy π.

A value function $v^\pi(\mathbf{s})$ is denoted as the expected return when the agent follows π form state $\mathbf{s}$ as

$$v^\pi(\mathbf{s}) := \mathop{\mathbb{E}}_{\mu_\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)\right] \quad \mathbf{s}_0 = \mathbf{s} \quad \forall \mathbf{s} \in \mathcal{S}.$$

An action-value function, or q-function, $q^\pi(\mathbf{s}, \mathbf{a})$ is defined similarly as the expected return following policy π if taking action $\mathbf{a}$ at state $\mathbf{s}$

$$q^\pi(\mathbf{s}, \mathbf{a}) := \mathop{\mathbb{E}}_{\mu_\pi}\left[\sum_{t=0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t)\right] \quad \mathbf{s}_0 = \mathbf{s}, \mathbf{a}_0 = \mathbf{a} \quad \forall \mathbf{s} \in \mathcal{S}, \mathbf{a} \in \mathcal{A}.$$

v^π and q^π can also be defined recursively in the form of Bellman equations

$$v^\pi(\mathbf{s}) = \sum_{\mathbf{a}\in\mathcal{A}} \pi(\mathbf{a} \mid \mathbf{s})(r(\mathbf{s},\mathbf{a}) + \gamma \sum_{\mathbf{s}'\in\mathcal{S}} p(\mathbf{s}' \mid \mathbf{s},\mathbf{a}) v^\pi(\mathbf{s}')),$$
$$q^\pi(\mathbf{s},\mathbf{a}) = r(\mathbf{s},\mathbf{a}) + \gamma \sum_{\mathbf{s}'\in\mathcal{S}} p(\mathbf{s}' \mid \mathbf{s},\mathbf{a})(\sum_{\mathbf{a}'\in\mathcal{A}} \pi(\mathbf{a}' \mid \mathbf{s}') q^\pi(\mathbf{s}',\mathbf{a}')).$$

In the case of continuous action or state space, the summation becomes integral.

By definition, an optimal policy $\pi^* \in \Pi$, where Π is a set of all possible policies, has optimal value function v^* as

$$v^*(\mathbf{s}) = \max_{\pi\in\Pi} v^\pi(\mathbf{s}) \quad \forall \mathbf{s} \in \mathcal{S},$$

and optimal q-function as

$$q^*(\mathbf{s},\mathbf{a}) = \max_{\pi\in\Pi} q^\pi(\mathbf{s},\mathbf{a}) \quad \forall \mathbf{s} \in \mathcal{S}, \mathbf{a} \in \mathcal{A}.$$

When the state and action spaces are tabular, and the true dynamics of the environment are known, we can compute the optimal value function v^* and optimal action-value function q^* using Dynamic Programming (DP) methods. DP methods involve applying the Bellman optimal operator iteratively, starting from an arbitrary initial value function until convergence. Additionally, DP methods can be used to find both the optimal value functions and optimal policy by applying the policy evaluation and policy improvement processes iteratively. However, if the transition model is unknown, we can relax this assumption using model-free Temporal Difference (TD) methods to approximate the value functions for a specific policy π.

3.1.2 Function Approximation

In the case of large or continuous state and action spaces, tabular TD methods become impractical for finding true value functions for a specific policy π. Instead, we can use function approximation techniques to approximate v^π and q^π as $\hat{v}_{\boldsymbol{\theta}}$ and $\hat{q}_{\boldsymbol{\theta}}$, respectively, where $\boldsymbol{\theta}$ is a vector parameter. The same TD errors in tabular methods can be used as regression errors to learn the parameters $\boldsymbol{\theta}$ by minimizing an MSE objective function

Algorithm 1 A Generic Actor Critic Algorithm

```
Initialize $\boldsymbol{\theta}, \boldsymbol{\phi}$
for each time-step $t$ do
    $\mathbf{a}_t \sim \pi_\phi(. \mid \mathbf{s}_t)$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_\theta \nabla J(\theta)$
    $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \alpha_\phi \nabla J_\pi(\boldsymbol{\phi})$
end for
```

$$J_Q(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\mu^\pi}\left[\Big(r(\mathbf{s},\mathbf{a}) + \gamma \hat{q}_\theta(\mathbf{s}',\mathbf{a}') - \hat{q}_\theta(\mathbf{s},\mathbf{a})\Big)^2\right],$$

where μ^π denotes the stationary distribution under policy π. Stochastic Gradient Descent (SGD) can be used to optimize $\boldsymbol{\theta}$ iteratively to minimize $J_Q(\boldsymbol{\theta})$ as

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \alpha_\theta \Big(\widetilde{q}(\mathbf{s},\mathbf{a}) - \hat{q}_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a})\Big) \nabla \hat{q}_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a}),$$

where α_θ is the step size, and $\widetilde{q}(\mathbf{s},\mathbf{a})$ is the target value.

In action-value methods, a policy is implicitly formed by finding the action with the maximum value in a state. Instead, policy gradient methods parameterize the policy as $\pi_{\boldsymbol{\phi}}(. \mid \mathbf{s})$ and explicitly define the distribution of actions given a state, which is parameterized by some vector parameter $\boldsymbol{\phi}$. The performance measure will also be a function of this parameter, defined as $J_\pi(\boldsymbol{\phi})$. Therefore, the policy improvement process can be constructed as updating $\boldsymbol{\phi}$ due to the gradient of performance with respect to the parameter:

$$\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \alpha_\phi \nabla J_\pi(\boldsymbol{\phi}).$$

Parameterizing both the value functions and the policy results in actor-critic methods. These methods try to estimate a value function based on the data collected from agent experience and update the policy by estimating the gradient of the performance w.r.t policy parameter. A general scheme of an actor-critic method is shown in algorithm 1. All the methods implemented in this thesis categorize as actor-critic methods where neural networks with some learnable weights are used as function approximators.

3.1.3 Maximum Entropy Reinforcement Learning

In maximum entropy reinforcement learning, the agent's objective function considers not only the rewards obtained but also the stochasticity of the policy. To measure this stochasticity, the entropy of the policy in visited states is augmented to the performance measure as follows:

$$J_\pi := \mathop{\mathbb{E}}_{\mu_\pi}\left[\sum_{t=0}^{\infty}\gamma^t\Big(r(\mathbf{s}_t,\mathbf{a}_t)+\alpha\mathcal{H}(\pi(.\mid\mathbf{s}_t))\Big)\right], \tag{3.1}$$

where α is a temperature parameter that controls the trade-off between the policy's stochasticity and the rewards obtained [52].

Soft Actor-Critic

Soft Actor-Critic(SAC) is a policy gradient actor-critic method introduced by Haarnoja *et al.* [14] that optimizes the maximum entropy objective in (3.1). A data buffer $\mathcal{D}$ is used to store transition information. In this algorithm, a soft-value and soft-q function are defined as

$$v(\mathbf{s}) := \mathop{\mathbb{E}}_{\mathbf{a}\sim\pi(.|\mathbf{s})}\Big[q(\mathbf{s},\mathbf{a})-\alpha\log\pi(\mathbf{a}\mid\mathbf{s})\Big],$$
$$q(\mathbf{s},\mathbf{a}) := \mathop{\mathbb{E}}_{\mathbf{s}'\sim p(.|\mathbf{s},\mathbf{a})}\Big[r(\mathbf{s},\mathbf{a})+\gamma v(\mathbf{s}')\Big].$$

The value functions and the policy are parameterized with $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$, respectively, and modelled by neural networks. A target network parameterized with $\widetilde{\boldsymbol{\theta}}$, with the same model as $\boldsymbol{\theta}$, is used to stabilize the training [27]. Parameter $\boldsymbol{\theta}$ is optimized due to the following objective

$$J_Q(\theta) = \mathop{\mathbb{E}}_{\mathbf{s},\mathbf{a},\mathbf{s}'\sim\mathcal{D}}\left[\Big(q_{\boldsymbol{\theta}}(\mathbf{s},\mathbf{a})-(r(\mathbf{s},\mathbf{a})+\gamma v_{\widetilde{\theta}}(\mathbf{s}'))\Big)^2\right],$$
$$\text{with } v_{\widetilde{\theta}}(\mathbf{s}) = \mathop{\mathbb{E}}_{\mathbf{a}\sim\pi(.|\mathbf{s})}\Big[q_{\widetilde{\theta}}(\mathbf{s},\mathbf{a})-\alpha\log\pi_\phi(\mathbf{a}\mid\mathbf{s})\Big].$$

Policy parameters, $\boldsymbol{\phi}$, are optimized due to an alternative objective:

$$J_\pi(\phi) = \mathop{\mathbb{E}}_{\mathbf{s}\sim\mathcal{D},\mathbf{a}\sim\pi_\phi(.|\mathbf{s})}\Big[q_\theta(\mathbf{s},\mathbf{a})-\alpha\log\pi_\phi(\mathbf{a}\mid\mathbf{s})\Big].$$

Algorithm 2 Soft Actor Critic Algorithm

```
**Initialize:** $\boldsymbol{\theta}, \boldsymbol{\phi}, \alpha$
**for** each time-step $t$ **do**
    $\mathbf{a}_t \sim \pi_\phi(. \mid \mathbf{s}_t)$
    observe $\mathbf{s}_{t+1}, r_{t+1}$
    $\mathcal{D} \leftarrow \mathcal{D} \cup \{\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}, r_{t+1}\}$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \lambda_\theta \nabla J(\theta)$
    $\boldsymbol{\phi} \leftarrow \boldsymbol{\phi} + \lambda_\phi \nabla J_\pi(\boldsymbol{\phi})$
    $\alpha \leftarrow \alpha + \lambda_\alpha \nabla J(\alpha)$
    $\widetilde{\boldsymbol{\theta}} \leftarrow \tau\boldsymbol{\theta} + (1-\tau)\widetilde{\boldsymbol{\theta}}$
**end for**
```

In practice, a multidimensional Gaussian distribution is often used to model the policy. Neural networks parameterized with $\boldsymbol{\phi}$ implement the mean and variance of such distributions. When computing the gradients of the policy objective with respect to the parameters, the gradients need to flow through the sampling operation of the policy distribution. However, the sampling operation is not differentiable, making it challenging to compute gradients with respect to the mean and standard deviation. Reparameterizing actions as

$$\mathbf{a}_t = f_{\boldsymbol{\phi}}(\mathbf{s}_t, \boldsymbol{\epsilon}_t) \quad \text{with } \boldsymbol{\epsilon}_t \sim \mathcal{N}(0, \mathcal{I})$$

enables SAC to efficiently compute gradients for updating the policy network.

The temperature parameter α can also be learned due to

$$J(\alpha) = \underset{\mathbf{s}\sim\mathcal{D}, \mathbf{a}\sim\pi_\phi(.|\mathbf{s})}{\mathbb{E}} \left[\alpha(-\log \pi_\phi(\mathbf{a} \mid \mathbf{s}) - H) \right],$$

where H is an adjustable target entropy. A generic pseudo code of SAC is shown in 2.

3.2 Continuous Time RL

In Reinforcement Learning, the dynamics of the environment are assumed to follow an MDP framework, while in classical control, the state dynamics are defined by differential equations in the continuous-time domain. If we denote the continuous-time state by $\boldsymbol{s}(t)$, then the change in the state can be expressed as follows:

$$d\boldsymbol{s}(t) = f(\boldsymbol{s}(t), \boldsymbol{a}(t))\ dt + \sigma(\boldsymbol{s}(t), \boldsymbol{a}(t))d\boldsymbol{W}(t), \quad t > 0,$$

where $\boldsymbol{a}(t) = \mu(\boldsymbol{s}(t))$ is the system input or action, f defines the dynamics in form of differential equations, and W and σ account for the stochasticity in the system [44]. By defining a reward function $r(\boldsymbol{s}(t), \boldsymbol{a}(t))$, the goal of control would be maximizing a discounted integral of rewards over time

$$J_\mu = \mathbb{E}\Bigg[\int_0^\infty e^{-\rho t} r(\mathbf{s}(t), \mathbf{a}(t))dt\Bigg], \tag{3.2}$$

where ρ is a time-constant. Similar to classical RL , value function of state $\mathbf{s}$ is defined as

$$v_\mu(\mathbf{s}) = \mathbb{E}\Bigg[\int_0^\infty e^{-\rho t} r(\mathbf{s}(t), \mathbf{a}(t))dt \mid \mathbf{s}(0) = \mathbf{s}\Bigg]. \tag{3.3}$$

The system can evolve in continuous time but control decisions are made discretely, meaning that an action $\mathbf{a}(t)$ defined in continuous time with duration d is chosen and applied at state $\mathbf{s}(t)$ and after execution the system is at some state $\mathbf{s}'$ [6], [31]. In this transition a reward can be defined as:

$$R(\mathbf{s}, \mathbf{a}, d) = \int_t^{t+d} e^{-\rho(\kappa - t)} r(\mathbf{s}(\kappa), \mathbf{a}(\kappa))d\kappa.$$

Thus if a policy π is choosing the action (and it's duration) then the corresponding value function in (3.3) can be redefined recursively as:

$$v_\pi(\mathbf{s}) = \mathop{\mathbb{E}}_{\mathbf{a}, d \sim \pi(.|\mathbf{s})}\Bigg[R(\mathbf{s}, \mathbf{a}, d) + e^{-\rho d} v_\pi(\mathbf{s}')\Bigg],$$

and similarly the action-value function in continuous time is defined as

$$q_\pi(\mathbf{s}, \mathbf{a}, d) = \mathop{\mathbb{E}}_{\mathbf{a}', d' \sim \pi(.|\mathbf{s}')}\Bigg[R(\mathbf{s}, \mathbf{a}, d) + e^{-\rho d} q_\pi(\mathbf{s}', \mathbf{a}', d')\Bigg].$$

3.3 Temporal Abstraction in RL

Semi-Markov Decision Processes

Semi-Markov Decision Processes(SMDPs) are a special kind of MDP appropriate for modeling continuous-time discrete-event systems. In SMDPs the actions take variable amounts of time and the goal is to model transitions given temporally-extended courses of action [41]. This model is closely related to the system described in 3.2.

Suppose the underlying base system is an MDP, with regular, single-step transitions, then we can have the concept of an extended action, as the same as in SMDPs, when the course of action is several discrete time-steps. We denote the term **option** for these courses of actions. The relation between the MDP, SMDP and options over the MDP are shown in figure 3.1.

Options

Options are known as sequences of simple actions that allow an agent to perform extended actions over time. An option consists of three parts: a policy, a termination function, and an initiation set. An option, represented as $o : \{\mathcal{I}_o, \beta_o, \pi_o\}$, can only be chosen if the current state $\mathbf{s}$ is within the initiation set $\mathcal{I}_o$; In this thesis we assume initiation set is equal to the statespace. When the option o is being executed, the agent will follow its policy π_o and continue doing so until the termination condition is met, at which point a new option will be selected [41].

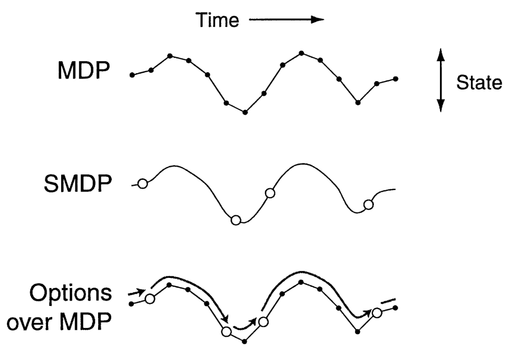


Figure 3.1: Illustration of a base system that is an MDP, with regular, single-step transitions, while the options define potentially larger transitions, like those of an SMDP, that may last for a number of discrete steps [41].

The reward associated with choosing an option o in state $\mathbf{s}$ is defined as:

$$r(\mathbf{s}, o) = \mathbb{E}\Big[r_t + \gamma r_{t+1} + ... + \gamma^{k-1} r_{t+k-1}\Big]. \tag{3.4}$$

The value function for a state in an SMDP with options can also be defined similarly as in MDPs, as the expected sum of discounted rewards and the value of the next state as:

$$v(\mathbf{s}) = \mathbb{E}\Bigg[r_t + \gamma r_{t+1} + ... + \gamma^{k-1} r_{t+k-1} + \gamma^k v(\mathbf{s}_{t+k})\Bigg].$$

Similarly, the Q-value for an option o in state $\mathbf{s}$, can be defined as:

$$q(\mathbf{s}, o) = \mathbb{E}\Bigg[r_t + \gamma r_{t+1} + ... + \gamma^{k-1} r_{t+k-1} + \gamma^k q(\mathbf{s}_{t+k}, o')\Bigg].$$

In these equations, r_t is the reward at time step t, γ is the discount factor, s_{t+k} is the state after k steps, and o' is the next option to be selected after k steps; It is important to note that the value of k is not fixed and is a random variable. This thesis focuses on scenarios where k is established as a timeout before option execution. To address this, options are considered semi-Markov options, enabling policies and termination conditions to rely on all preceding events since the option was initiated, according to Sutton *et al.* [41].

3.3.1 Semi-Markov Options

Building on the concepts introduced in 3.2 and 3.3, we now consider semi-Markov options with termination conditions as a time-out function. In this case, we can see the transitions from one option to the other as one time-step of an MDP $\mathcal{M}' := (\mathcal{S}' = \mathcal{S}, \mathcal{A}', p', \mathcal{R}', \gamma', \mu_0' = \mu_0)$, that is an augmentation of a simple base MDP $\mathcal{M} = (\mathcal{S}, \mathcal{A}, p, \mathcal{R}, \gamma, \mu_0)$. The action space of $\mathcal{M}'$ would be the space of possible option policies, $\mathbf{\Omega}$, paired with natural numbers $\mathbb{N}$ as the length of execution, $\mathcal{A}' = \mathbf{\Omega} \times \mathbb{N}$. $\mathcal{M}'$ will have the transition model $p'(\mathbf{s}' \mid \mathbf{s}, \pi_o, k)$ for $\pi_o \in \mathbf{\Omega}$, $k \in \mathbb{N}$ and a reward model similar to 3.4 but with deterministic length of execution defined as $r'(\mathbf{s}, o, k) = \underset{\pi_o}{\mathbb{E}}\Bigg[r_t + \gamma r_{t+1} + ... + \gamma^{k-1} r_{t+k-1}\Bigg]$. It should be noted that the discounting factor for $\mathcal{M}'$ is not constant and varies with option length k as $\gamma' = \gamma^k$.

If a policy π is choosing options and execution lengths, then the performance of such policy in $\mathcal{M}'$ would be defined as

$$J_\pi = \mathbb{E}\Big[\sum_{i=0}^{\infty} \gamma^{\sum_{j=0}^{j=i-1} k_j} r'(\mathbf{s}_i, o_i, k_i)\Big]. \tag{3.5}$$

Chapter 4

Continuous-Time Continuous Option Actor Critic

This chapter outlines the proposed approach and techniques used in this study to achieve the research objectives. We propose a framework in which an RL agent optimizes its behaviour while adapting its decision frequency. Our framework, Continuous-Time Continuous-Options (CTCO), includes a policy that chooses extended continuous actions with varying lengths. The policy only decides on the next course of action once the current one is finished, allowing the system to adapt its decision-making frequency.

We suppose a continuous control task is given as a base MDP $\mathcal{M}(\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \mu_0, \gamma)$ with some constant time-interval Δt seconds between time-steps. CTCO interacts with MDP $\mathcal{M}$ through applying the extended actions. CTCO forms a policy π which samples extended actions $\mathbf{a}_{t:t+\lceil \frac{d}{\Delta t} \rceil}$ and continuous durations $d \in \mathbb{R}^+$, with $\mathbf{a}_i \in \mathcal{A}$ for $i = t \cdots t + \lceil \frac{d}{\Delta t} \rceil - 1$. Durations are continuous values; therefore, CTCO can adapt its decision frequency regardless of the interaction frequency $\frac{1}{\Delta t}$. Meanwhile, by optimizing its policy, our framework can produce desired courses of action in the base MDP. We choose to optimize the policy in an actor-critic framework based on the soft-actor-critic(SAC) algorithm [13]. Our choice is based on two main advantages of SAC over other actor-critic algorithms. Firstly, using an entropy regularization term promotes exploration and helps maintain diversity in the policy. Additionally, SAC's use of a replay buffer can make it more sample efficient than other algorithms.

The following describes how CTCO uses temporal abstraction and formulates its policy to choose extended actions. Then, we show how our framework optimizes its policy in a policy gradient actor-critic framework.

4.1 The Policy Formulation

In this section, we describe the formulation of the policy of our framework and how it interacts with a continuous control task MDP.

4.1.1 Options and Durations as High-level Actions

Based on concepts in section 3.3, we denote an extended action $a_{t:t+k}$ as a sequence of primitive actions executed by a semi-Markov option $o \in \mathcal{O}$ with some intra-option policy $\pi_o \in \boldsymbol{\Omega}$ and termination β_o as a time-out of k time-steps in the base MDP $\mathcal{M}(\mathcal{S}, \mathcal{A}, \mathcal{R}, p, \mu_0, \gamma)$. In this setting, $\mathcal{O}$ is the space of options and $\boldsymbol{\Omega}$ is the space of intra-option policies. The interaction of CTCO with MDP $\mathcal{M}$ is equivalent to the interaction of CTCO with a high-level MDP $\mathcal{M}^H(\mathcal{S}^H, \mathcal{A}^H, \mathcal{R}^H, p^H, \mu_0^H, \gamma^H)$ with state-space $\mathcal{S}^H := \mathcal{S}$, action-space $\mathcal{A}^H := \boldsymbol{\Omega} \times \mathbb{N}$, reward function $\mathcal{R}^H := \mathcal{S} \times \boldsymbol{\Omega} \times \mathbb{N} \rightarrow \mathbb{R}$, transition distribution $p^H := \mathcal{S} \times \boldsymbol{\Omega} \times \mathbb{N} \times \mathcal{S} \rightarrow \mathbb{R}$, initial state distribution $\mu_0^H := \mu_0$, and discounting $\gamma^H := \gamma$. One interaction in the ith time-step of $\mathcal{M}^H$ consists of observing the state $\mathbf{s}_i = \mathbf{s}_t$ and sampling an intra-option policy π_{o_i} and its duration d_i from a policy $\pi(.,.|\mathbf{s}_i)$ then applying the intra-option policy π_{o_i} in the base MDP for $k = \lceil \frac{d}{\Delta t} \rceil$ time-steps and finally observing the $\mathbf{s}_{i+1} = \mathbf{s}_{t+k}$ and reward $r_i = r_t + \gamma r_{t+1} + ... + \gamma^{k-1} r_{t+k-1}$.

Now we need our RL agent to be able to choose an option o and, accordingly, π_o to generate a continuous course of action in one decision. However, the space of all possible intra-option policies, $\boldsymbol{\Omega}$, is infinitely large without any structure that would make sampling π_o intractable. To overcome this limitation, we associate an intra-option policy π_o with a vector parameter $\boldsymbol{\omega} \in \boldsymbol{\Omega}$, where we have reduced $\boldsymbol{\Omega}$ to a multidimensional space with real values. Therefore we formally define the stochastic policy as $\pi : \mathcal{S} \times \boldsymbol{\Omega} \times \mathbb{R} \rightarrow \mathbb{R}^+$. We denote $\boldsymbol{\omega}$ a continuous option that parameterizes a deterministic intra-option policy

$\pi_{\boldsymbol{\omega}}$ to generate actions according to the following

$$\boldsymbol{\omega}, d \sim \pi(.,. \mid \mathbf{s} = \mathbf{s}_t) \tag{4.1}$$

$$\mathbf{a}_{t+k} = \pi_{\boldsymbol{\omega}}(\mathbf{s}_{t+k}, k\Delta t, d) \text{ For } k = 0, \cdots, \lceil \frac{d}{\Delta t} \rceil - 1, \tag{4.2}$$

where policy $\pi_{\boldsymbol{\omega}}$ can be any function, neural network, or program that generally can depend on a set of parameters $\boldsymbol{\omega}$, and the state $\mathbf{s}_{t+k}$, elapsed time from the start of option execution $k\Delta t$, and option duration d.

According to the definition of $\mathcal{M}^H$, $(\boldsymbol{\omega}, d)$ corresponds to an option and is considered the high-level action in this MDP. Considering this, the reward definition in section 3.3.1 and the intra-option policy definition in (4.2), we obtain the high-level reward associated with an option $(\boldsymbol{\omega}, d)$ in $\mathcal{M}^H$ as:

$$r(\mathbf{s} = \mathbf{s}_t, \boldsymbol{\omega}, d) = \sum_{k=0}^{\lceil \frac{d}{\Delta t} \rceil} \gamma^k r(\mathbf{s}_{t+k}, \pi_{\boldsymbol{\omega}}(\mathbf{s}_{t+k}, k\Delta t, d)). \tag{4.3}$$

If a policy π is choosing $\boldsymbol{\omega}, d$ then according to (3.5) the performance measure of π in high-level MDP $\mathcal{M}^H$ becomes

$$J_\pi = \mathop{\mathbb{E}}_{\mu^\pi} \left[\sum_{i=0}^{\infty} \gamma^{\sum_{j=0}^{j=i-1} \lceil \frac{d_j}{\Delta t} \rceil} r(\mathbf{s}_i, \boldsymbol{\omega}_i, d_i) \right]. \tag{4.4}$$

Figure 4.1 illustrates the interaction between the CTCO agent and the environment. More visualizations of how CTCO works can be found in supplementary videos.

4.1.2 Open-loop Intra-option Policies

Until now, we have shown how our method can vary the decision frequency by choosing an intra-option policy $\pi_{\boldsymbol{\omega}}$ and its duration d to generate an extended action. In this section, we demonstrate the structure of such intra-option policies.

We encode deterministic option policies using a linear parametric model without state dependency to achieve a simple model with a few parameters. Option policies without state dependency result in open-loop controllers that are undesirable, particularly in the presence of stochasticity. However, the policy over options can compensate for the lack of feedback by selecting suitable

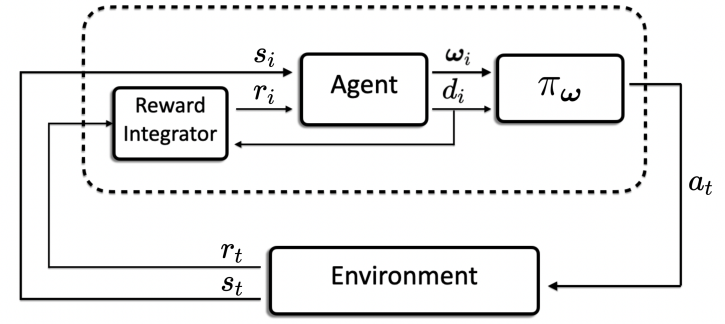


Figure 4.1: Diagram of a CTCO agent interacting with a task environment. At each decision point t in the based MDP, this agent observes a state $\mathbf{s}_i = \mathbf{s}_t$ and chooses option parameter $\boldsymbol{\omega}_i$ and duration d_i. Then executes intra-option policy $\pi_{\boldsymbol{\omega}_i}$ due to (4.2) for duration d_i while integrating rewards according to (4.3).

options as frequently as needed for the cost of increased decision frequency. Equation (4.5), describes a deterministic open-loop intra-option policy that outputs actions $\mathbf{a} \in \mathcal{A}$ as a linear combination of some time-dependent features $\boldsymbol{\phi}(t) : \mathbb{R} \rightarrow \mathbb{R}^{|\mathcal{A}| \times |\boldsymbol{\Omega}|}$ with coefficients $\boldsymbol{\omega} \in \boldsymbol{\Omega}$.

$$\mathbf{a}_{t+k} = \pi_{\boldsymbol{\omega}}(k\Delta t, d) = \boldsymbol{\phi}(k\Delta t/d) \times \boldsymbol{\omega} \quad \text{For } k = 0, \cdots, \lceil \frac{d}{\Delta t} \rceil - 1 \tag{4.5}$$

We take inspiration from movement primitives and use normalized radial basis functions (RBFs) to encode the features $\boldsymbol{\phi}$, resulting in smooth and low-jerk action trajectories that are suitable for robotic applications. The number of RBFs, N_{RBF}, determines the complexity of the action trajectory. With one RBF, we obtain an option policy with constant action, while with more RBFs, we obtain a more complex intra-option policy. Moreover, the dimensionality of option parameters $\boldsymbol{\omega}$ will be set as $|\boldsymbol{\Omega}| = N_{RBF}|\mathcal{A}|$, meaning that every N_{RBF} elements of $\boldsymbol{\omega}$ are used to generate one element of the low-level action vector. The general form of $\boldsymbol{\phi}$ with N_{RBF} features is shown in (4.6).

$$\boldsymbol{\phi}(t) = \left[\frac{e^{\frac{-(t-c_j)^2}{2h_j}}}{\sum_{k=1}^{N_{RBF}} e^{\frac{-(t-c_k)^2}{2h_k}}} \right]_{i,j} \text{For } i \in \{1, \cdots, |\mathcal{A}|\}, j \in \{1, \cdots, N_{RBF}\}, \tag{4.6}$$

here c_j and h_j are centers and widths of RBFs respectively. Since we are using normalized RBFs, generated actions in (4.5), will have the same range of values as of the corresponding elements in $\boldsymbol{\omega}$. Figure 4.2 shows normalized RBFs for $N_{RBF} = 3$ and 3 randomly sampled continuous options when $|\mathcal{A}| = 1$.

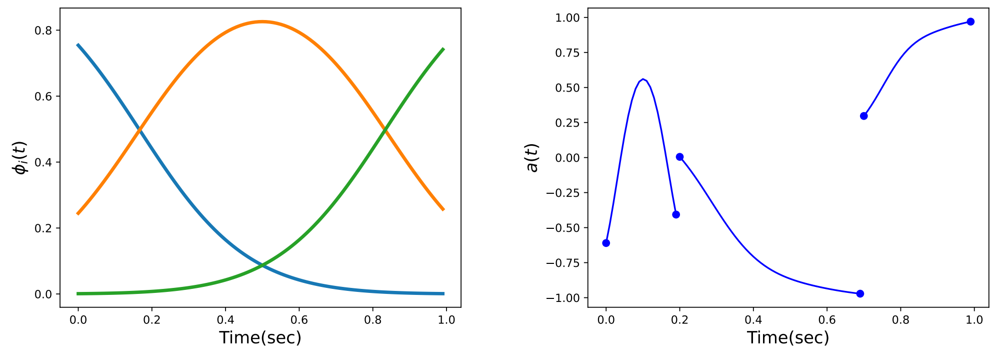


Figure 4.2: The left plot shows three equally spaced RBFs. The right plot shows three sample options vs time, generated from RBFs in the left.

4.2 A Soft Actor-Critic Framework

Up to this point, we have presented a mathematical framework that outlines how a policy π generates extended actions as options and interacts with an MDP. In the following section, we discuss the implementation and optimization of such a policy. Among many choices in policy gradient methods, we implement our algorithm following the soft actor-critic (SAC) architecture [14].

The policy π is responsible for selecting the parameters $\boldsymbol{\omega}$ and the duration d of the options. The stochastic policy takes the current state of the system $\mathbf{s}$ as input and determines the probability density of the parameter vector and the duration conditionally independent, as in

$$\boldsymbol{\omega}, d \sim \pi(.,.|\mathbf{s}).$$

We model the probability density function of $\boldsymbol{\omega}$ as a multidimensional Gaussian distribution. However, the probability density function of d can only be defined over positive values. To address this issue, we transform a Gaussian distribution to a pdf defined over positive numbers by applying an invertible function that maps samples from the Gaussian distribution to positive values. In our case, we choose the sigmoid function for this purpose. We use a neural network with weights $\boldsymbol{\theta}$ to implement the policy. The network outputs $\boldsymbol{\mu}_\theta^{\boldsymbol{\omega}}(\mathbf{s})$, $\boldsymbol{\sigma}_\theta^{\boldsymbol{\omega}}(\mathbf{s})$, $\mu_\theta^d(\mathbf{s})$, $\sigma_\theta^d(\mathbf{s})$ as the mean and variance of the Gaussian distributions to sample the continuous option $\boldsymbol{\omega}$ and the duration d given an observation of state $\mathbf{s}$ according to the following

$$\boldsymbol{\omega} \sim \mathcal{N}(\boldsymbol{\mu}_\theta^{\boldsymbol{\omega}}(\mathbf{s}), \boldsymbol{\sigma}_\theta^{\boldsymbol{\omega}}(\mathbf{s})) \tag{4.7}$$

$$d = d_{max}\text{Sigm}(d^-) \text{ with } d^- \sim \mathcal{N}(\mu_\theta^d(\mathbf{s}), \sigma_\theta^d(\mathbf{s})). \tag{4.8}$$

Here d is limited to continuous values in $(0, d_{max})$.

4.2.1 Policy Evaluation.

We include an entropic regularization term in the performance objective that encourages the exploration of different option parameters $\boldsymbol{\omega}$ and durations d as discussed in 3.1.3. Furthermore, since high-frequency decision-making is problematic for training performances, we include a new component called *high-frequency penalization* to discourage the policy from choosing small durations. The high-frequency penalization, β_h, is a constant positive scalar subtracted from the objective each time the policy makes a decision. This modifies the objective definition in (4.4) to

$$J_\pi(\theta) = \mathop{\mathbb{E}}_{\mu^\pi}\left[\sum_{i=0}^{\infty} \gamma^{\sum_{j=0}^{j=i-1}\lceil \frac{d_j}{\Delta t}\rceil} \left(r_i + \beta_E \mathcal{H}(\pi_\theta(\cdot,\cdot \mid \mathbf{s}_i)) - \beta_h\right)\right]. \tag{4.9}$$

where β_E and β_h are the entropic regularizer and the high-frequency penalty term. To see how β_h encourages for longer durations, we take out the term associated with the high-frequency penalty in the overall objective in (4.9) as $\mathop{\mathbb{E}}_{\mu^\pi}\left[-\sum_{i=0}^{\infty} \gamma^{\sum_{j=0}^{j=i-1}\lceil \frac{d_j}{\Delta t}\rceil}\beta_h\right]$. Since $\gamma < 1$, larger values of d will increase this term.

Following the SAC framework and the fact that actions are pair of option parameter and duration, $(\boldsymbol{\omega}, d)$, we define the Q-function by the following

Bellman equation:

$$Q^{\pi}(\mathbf{s},\boldsymbol{\omega},d) := \underset{\substack{\mathbf{s}'\sim p(.|\mathbf{s},\boldsymbol{\omega},d)\\ \boldsymbol{\omega}',d'\sim\pi_{\boldsymbol{\theta}}(.,.|\mathbf{s}')}}{\mathbb{E}}\left[r(\mathbf{s},\boldsymbol{\omega},d)-\beta_h+\gamma^{\lceil\frac{d}{\Delta t}\rceil}(Q^{\pi}(\mathbf{s}',\boldsymbol{\omega}',d')-\beta_E\log\pi_{\boldsymbol{\theta}}(\boldsymbol{\omega}',d'|\mathbf{s}'))\right], \tag{4.10}$$

where $\boldsymbol{\omega}', d' \sim \pi(\mathbf{s}')$ are next option parameter and duration sampled in next state $\mathbf{s}'$. In practice, such an equation cannot be solved in closed form. Similar to the implementation of SAC by Haarnoja *et al.* [13], we approximate the Q-function and the target Q-function, as $\hat{Q}_{\chi}$ and $\hat{Q}_{\chi'}$, using neural networks parameterized by χ and χ'. To learn these weights, we minimize the bellman error in (4.11). We stochastically estimate the gradient of this objective as $\hat{\nabla}J_Q(\chi)$ to update critic parameters χ and χ' accordingly.

$$J_Q(\chi) = \underset{(\mathbf{s},\boldsymbol{\omega},d,r,\mathbf{s}')\sim\mathcal{D}}{\mathbb{E}}\left[\frac{1}{2}\Big(r-\beta_h+\gamma^{\lceil\frac{d}{\Delta t}\rceil}(\hat{Q}_{\chi'}(\mathbf{s}',\boldsymbol{\omega}',d')-\beta_E\log\pi(\boldsymbol{\omega}',d'\mid\mathbf{s}')) - \hat{Q}_{\chi}(\mathbf{s},\boldsymbol{\omega},d)\Big)^2\right]. \tag{4.11}$$

4.2.2 Policy Improvement.

To optimize the parameterized policy π_θ, we need to compute the gradient of the objective in (4.9) w.r.t to the policy parameters $\boldsymbol{\theta}$ and update $\boldsymbol{\theta}$.

$$\nabla_\theta J_\pi(\theta) = \nabla_\theta \underset{\mu^\pi}{\mathbb{E}}\left[\sum_{i=0}^{\infty}\gamma^{\sum_{j=0}^{j=i-1}\lceil\frac{d_j}{\Delta t}\rceil}\left(r_i+\beta_E\mathcal{H}(\pi_\theta(\cdot,\cdot\mid\mathbf{s}_i))-\beta_h\right)\right]. \tag{4.12}$$

In the following we show that the policy gradient for continuous control in an MDP with variable discounting will have the same form as for an MDP with a constant discounting factor. Following the policy improvement in SAC, we first reparameterize the policy as in (4.14) *and* (4.13). Then we obtain the gradient of Q-function w.r.t policy paramter to derive the policy gradient.

$$\boldsymbol{\omega} = \boldsymbol{f}_\theta^{\boldsymbol{\omega}}(\mathbf{s};\boldsymbol{\epsilon}) := \boldsymbol{\mu}_\theta^{\boldsymbol{\omega}}(\mathbf{s}) + \boldsymbol{\epsilon}\boldsymbol{\sigma}_\theta^{\boldsymbol{\omega}}(\mathbf{s}) \quad \text{with } \boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I}), \tag{4.13}$$

$$d = f_\theta^d(\mathbf{s};\epsilon) := d_{max}\mathrm{Sigm}(\mu_\theta^d(\mathbf{s})+\epsilon\sigma_\theta^d(\mathbf{s})) \quad \text{with } \epsilon\sim\mathcal{N}(0,1). \tag{4.14}$$

According to the Bellman equation we have gradient of Q w.r.t to θ as:

$$\begin{aligned}
&\nabla_\theta Q^\pi(\mathbf{s},\boldsymbol{\omega},d)\\
&= \nabla_\theta \mathop{\mathbb{E}}_{\substack{\mathbf{s}'\sim p(.|\mathbf{s},\boldsymbol{\omega},d)\\ \boldsymbol{\omega}',d'\sim\pi(.,.|\mathbf{s}')}}\left[r(\mathbf{s},\boldsymbol{\omega},d)-\beta_h+\gamma^{\lceil\frac{d}{\Delta t}\rceil}(Q^\pi(\mathbf{s}',\boldsymbol{\omega}',d')+\beta_E\mathcal{H}(\pi_\theta(.,.|\mathbf{s}')))\right]\\
&= \nabla_\theta \mathop{\mathbb{E}}_{\substack{\mathbf{s}'\sim p(.|\mathbf{s},\boldsymbol{\omega},d)\\ \boldsymbol{\omega}',d'\sim\pi(.,.|\mathbf{s}')}}\left[r(\mathbf{s},\boldsymbol{\omega},d)-\beta_h+\gamma^{\lceil\frac{d}{\Delta t}\rceil}(Q^\pi(\mathbf{s}',\boldsymbol{\omega}',d')-\beta_E\log\pi_\theta(\boldsymbol{\omega}',d'|\mathbf{s}'))\right]
\end{aligned}$$

Using the reparameterized policy in (4.13) and (4.14) we get:

$$\begin{aligned}
\nabla_\theta Q^\pi(\mathbf{s},\boldsymbol{\omega},d) = \mathop{\mathbb{E}}_{\mathbf{s}',\epsilon'}\Bigg[\gamma^{\lceil\frac{d}{\Delta t}\rceil}\Bigg(&\nabla_{\boldsymbol{\omega}}Q^\pi(\mathbf{s}',\boldsymbol{\omega}',d')\nabla_\theta f_\theta^{\boldsymbol{\omega}}(\mathbf{s}',\epsilon')+\nabla_d Q^\pi(\mathbf{s}',\boldsymbol{\omega}',d')\nabla_\theta f_\theta^{d}(\mathbf{s}',\epsilon')\\
&+\nabla_\theta Q^\pi(s',\boldsymbol{\omega}',d')-\beta_E\nabla_\theta\log(\pi_\theta(\boldsymbol{\omega}',d'|\mathbf{s}'))\Bigg)\Bigg]. \quad (4.15)
\end{aligned}$$

Recursively replacing $\nabla_\theta Q^\pi(\mathbf{s}',\boldsymbol{\omega}',d')$ results in:

$$\begin{aligned}
\nabla_\theta Q^\pi(\mathbf{s},\boldsymbol{\omega},d) = \mathop{\mathbb{E}}_{\mu_\pi}\Bigg[\sum_{i=0}^{\infty}\gamma^{\sum_{j=0}^{i}\lceil\frac{d_j}{\Delta t}\rceil}\Bigg(&\nabla_{\boldsymbol{\omega}}Q^\pi(\mathbf{s}_{i+1},\boldsymbol{\omega}_{i+1},d_{i+1})\nabla_\theta f_\theta^{\boldsymbol{\omega}}(\mathbf{s}_{i+1},\epsilon_{i+1})\\
+\nabla_d Q^\pi(\mathbf{s}_{i+1},\boldsymbol{\omega}_{i+1},d_{i+1})\nabla_\theta f_\theta^{d}(\mathbf{s}_{i+1},\epsilon_{i+1})&-\beta_E\nabla_\theta\log(\pi_\theta(\boldsymbol{\omega}_{i+1},d_{i+1}|\mathbf{s}_{i+1}))\Bigg)\\
|_{\boldsymbol{\omega}_{i+1}=f_\theta^{\boldsymbol{\omega}}(\mathbf{s}_{i+1},\epsilon_{i+1}),d_{i+1}=f_\theta^{d}(\mathbf{s}_{i+1},\epsilon_{i+1})}\Bigg] &\quad \text{for} \quad \mathbf{s}_0=\mathbf{s},\boldsymbol{\omega}_0=\boldsymbol{\omega},d_0=d. \quad (4.16)
\end{aligned}$$

Now we derive the policy gradient:

$$\begin{aligned}
\nabla_\theta J_\pi &= \nabla_\theta\mathop{\mathbb{E}}_{\mu_\pi}\left[\sum_{i=0}^{\infty}\gamma^{\sum_{j=0}^{i-1}\lceil\frac{d_j}{\Delta t}\rceil}(R(\mathbf{s}_i,\boldsymbol{\omega}_i,d_i)-\beta_h+\beta_E\mathcal{H}(\pi_\theta(.,.|\mathbf{s}_i))\right]\\
&= \nabla_\theta\mathop{\mathbb{E}}_{\mathbf{s}_0,\boldsymbol{\omega}_0,d_0}\left[Q^\pi(\mathbf{s}_0,\boldsymbol{\omega}_0,d_0)-\beta_E\log\pi_\theta(\boldsymbol{\omega}_0,d_0|\mathbf{s}_0))\right]\\
&= \mathop{\mathbb{E}}_{\mathbf{s}_0,\epsilon_0,\boldsymbol{\epsilon}_0}\Bigg[\nabla_\theta Q^\pi(\mathbf{s}_0,\boldsymbol{\omega}_0,d_0)+\nabla_{\boldsymbol{\omega}}Q^\pi(\mathbf{s}_0,\boldsymbol{\omega}_0,d_0)\nabla_\theta f_\theta^{\boldsymbol{\omega}}(\mathbf{s}_0,\boldsymbol{\epsilon}_0)+\nabla_d Q^\pi(\mathbf{s}_0,\boldsymbol{\omega}_0,d_0)\\
&\qquad\nabla_\theta f_\theta^{d}(\mathbf{s}_0,\epsilon_0))-\beta_E\nabla_\theta\log(\pi_\theta(\boldsymbol{\omega}_0,d_0|\mathbf{s}_0))|_{\boldsymbol{\omega}_0=f_\theta^{\boldsymbol{\omega}}(\mathbf{s}_0,\boldsymbol{\epsilon}_0),d_0=f_\theta^{d}(\mathbf{s}_0,\epsilon_0)}\Bigg]\\
&\qquad\qquad\text{(By reparameterizaiton trick)}\\
&= \mathbb{E}_{\mu_\pi}\Bigg[\sum_{i=0}^{\infty}\gamma^{\sum_{j=0}^{i-1}\lceil\frac{d_j}{\Delta t}\rceil}\Bigg(\nabla_{\boldsymbol{\omega}}Q^\pi(\mathbf{s}_i,\boldsymbol{\omega}_i,d_i)\nabla_\theta f_\theta^{\boldsymbol{\omega}}(\mathbf{s}_i,\epsilon_i)+\nabla_d Q^\pi(\mathbf{s}_i,\boldsymbol{\omega}_i,d_i)\\
&\qquad\nabla_\theta f_\theta^{d}(\mathbf{s}_i,\epsilon_i))-\beta_E\nabla_\theta\log(\pi_\theta(\boldsymbol{\omega}_i,d_i|\mathbf{s}_i))\Bigg)|_{\boldsymbol{\omega}_i=f_\theta^{\boldsymbol{\omega}}(\mathbf{s}_i,\epsilon_i),d_i=f_\theta^{d}(\mathbf{s}_i,\epsilon_i)}\Bigg].\\
&\qquad\qquad\text{(By replacing } \nabla_\theta Q^\pi(\mathbf{s}_0,\boldsymbol{\omega}_0,d_0) \text{ using (4.16))}
\end{aligned}$$

Discounting can be dropped to stochastically sample states from the discounted distribution $d_{\pi,\gamma}$. Therefore we have the policy gradient as:

$$\nabla_\theta J_\pi = \mathbb{E}_{d_{\pi,\gamma}}\Bigg[\nabla_{\boldsymbol{\omega}} Q^\pi(\mathbf{s},\boldsymbol{\omega},d)\nabla_\theta f_\theta^{\boldsymbol{\omega}}(\mathbf{s},\epsilon) + \nabla_d Q^\pi(\mathbf{s},\boldsymbol{\omega},d)\nabla_\theta f_\theta^d(\mathbf{s},\epsilon)) \\ - \beta_E \nabla_\theta \log(\pi_\theta(\boldsymbol{\omega},d_i|\mathbf{s}))|_{\boldsymbol{\omega}=f_\theta^{\boldsymbol{\omega}}(\mathbf{s},\epsilon),d=f_\theta^d(\mathbf{s},\epsilon)}\Bigg]. \quad (4.17)$$

Finally we incorporate the approximated Q-function to estimate the policy gradient as in (4.18).

$$\hat{\nabla}_\theta J_\pi(\theta) = \underset{\mathbf{s}\sim\mathcal{D},\epsilon,\boldsymbol{\epsilon}\sim\mathcal{N}(\mathbf{0},\mathbf{I})}{\mathbb{E}}\Bigg[\nabla_{\boldsymbol{\omega}} Q_\chi^\pi(\mathbf{s},\boldsymbol{\omega},d)\nabla_\theta \boldsymbol{f}_\theta^{\boldsymbol{\omega}}(\mathbf{s},\boldsymbol{\epsilon}) + \nabla_d Q_\chi^\pi(\mathbf{s},\boldsymbol{\omega},d)\nabla_\theta f_\theta^d(\mathbf{s},\epsilon) \\ - \beta_E \nabla_\theta \log \pi_\theta(\boldsymbol{\omega},d|\mathbf{s})\Big|_{\boldsymbol{\omega}=\boldsymbol{f}_\theta^{\boldsymbol{\omega}}(\mathbf{s},\boldsymbol{\epsilon}),d=f_\theta^d(\mathbf{s},\epsilon)}\Bigg]. \quad (4.18)$$

The complete algorithm of CTCO is described in algorithm 3.

Algorithm 3 Continuous-Time Continuous-Option Actor-Critic

```
Input: A policy $\pi$ with a set of parameters $\boldsymbol{\theta}$, Intra-option policy model $\pi_{(\omega)}$, critic parameters $\chi, \chi'$, learning-rates $\lambda_q, \lambda_p$, replay buffer $\mathcal{D}$.
$i = 0$, observe $\mathbf{s}_0$
while True do
    $\boldsymbol{\omega}_i, d_i \sim \pi_\theta(.,.|\mathbf{s}_i)$                ▷ Sample option and duration
    Execute $\pi_{\boldsymbol{\omega}_i}(k\Delta t, d_i)$ For $k = 0, \cdots, \lceil \frac{d_i}{\Delta t} \rceil - 1$
    Observe $\mathbf{s}_{i+1}$ and compute $r_i$ with (4.3)
    Store $\mathbf{s}_i, \boldsymbol{\omega}_i, d_i, r_i, \mathbf{s}_{i+1}$ in $\mathcal{D}$
    $\chi \leftarrow \chi - \lambda_q \hat{\nabla}_\chi J_Q(\chi)$                ▷ Update critic
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \lambda_p \hat{\nabla}_\theta J_\pi(\theta)$                ▷ Update actor
    Perform soft-update of $\chi'$.
    $i \leftarrow i + 1$
end while
```

Chapter 5

Empirical Evaluation

In this chapter, we present the experimental results of our work. We start by describing the experimental setup, and then we focus on answering the following questions:

1. Can our algorithm adapt its decision frequency independently from the interaction frequency in continuous control tasks?

2. Does our algorithm benefit from having options with more complex behaviour than action repetition?

3. How does our algorithm perform on a real-robotic task with sparse rewards?

5.1 Experimental Setup.

This section describes the experimental setup used for our experiments, including evaluation criteria, task specifications, and algorithms implementations in simulation and real-world settings.

5.1.1 Evaluation Criteria

Continuous control tasks are defined in continuous time. However, they are often modelled as discretized MDPs with some time-interval Δt. When evaluating the performance of a reinforcement learning algorithm, we consider the performance measure in continuous time as in (3.2), regardless of the discretization frequency. We assume that the frequency is high enough to neglect the effect of time resolution. To compute the performance measure J_π in a base MDP $\mathcal{M}(\mathcal{S}, \mathcal{A}, \mathcal{R}, p, d_0, \gamma)$ with time-interval Δt, we set $\gamma = e^{-\rho \Delta t}$ due to an approximation of the integral as summation when $\Delta t \to 0$:

$$\sum_{0}^{\infty} \gamma^t r(\mathbf{s}_t, \mathbf{a}_t) \Delta t \approx \int_{0}^{\infty} e^{-\rho t} r(\mathbf{s}(t), \mathbf{a}(t)) dt. \tag{5.1}$$

In this thesis, we assume time-constant $\rho = 0.4$ for all tasks. For example, a time-discretization of $\Delta t = 0.05$ seconds would result in $\gamma = 0.98$.

5.1.2 Considerations in Algorithms Implementations

When learning simulated tasks, optimization happens synchronously while the agent interacts with the task in the same process. Unlike discrete simulation timesteps, the time of the real-world proceeds during the agent computations. Therefore, we ensure the action inference time is less than the action cycle time, and optimization steps are done asynchronously in a parallel process [48]. Moreover, the number of updates an agent does in the real-world is limited by the computation resources and is independent of the time discretization. Therefore, we modify learning algorithms to maintain a fixed number of updates per time unit in both simulated and real-world experiments. The implementations are publicly available [1].

5.1.3 Tasks

Here we present the details of the simulated and real-world continuous control environments used in this thesis. In all of the experiments, to ensure that all reinforcement learning algorithms are optimizing and being evaluated for the same objective, we modify the task rewards by multiplying them by Δt. Furthermore, we change the maximum episode lengths according to the interaction frequency to maintain the same episode length in task time.

[1] https://github.com/amir-karimi96/continuous-time-continuous-option-policy-gradient

Simulated Tasks

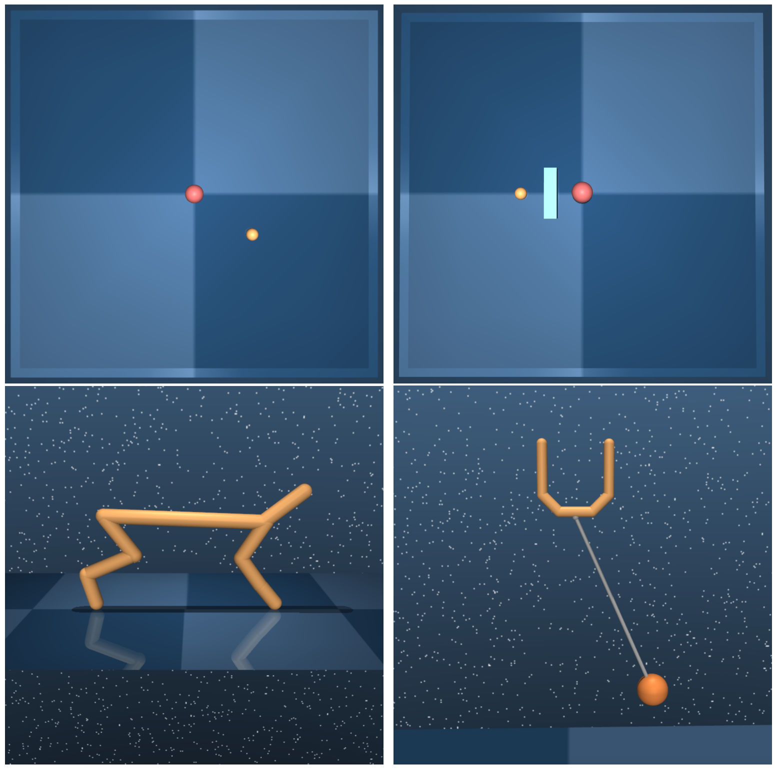


Figure 5.1: Tasks of Point-Mass, Point-Mass-with-Obstacle, Cheetah, Ball-in-cup

Figure 5.1 depicts images of simulated tasks used in this thesis. Three simulated tasks of Point-Mass, Cheetah and Ball-in-Cup are taken from DeepMind Control Suite [43].

The task of Point-Mass-with-Obstacle is a modification of Point-Mass, where the action is the desired position of the object. A PID controller internally controls the object's position. Moreover, to make the task more challenging, we discourage discontinuity in actions by not performing desired positions further than 4cm from the object position and adding a negative reward of -0.2. The reset coordinates of object is randomly set with $x_{reset} \in [-0.2, -0.15]cm$ and $y_{reset} \in [-0.05, 0.05]cm$. The obstacle and goal positions are constant.

Real-world Task of Visual-Reacher

In this part we describe the real-world task of visual reacher. First the physical setup is presented and then the task specifications are introduced.

In this task, the goal is to reach a static bean bag randomly placed on a table using a robotic arm equipped with a camera on its wrist. The physical setup of the task is shown in figure 5.2. The robot has the bean bag attached to its wrist by a string, which allows it to randomly set the bean bag's position on the table. To ensure the safety of the robot and avoid hitting objects, the position of the end-effector is bounded to a box with dimensions $30 \times 50 \times 30\ cm^3$.

We use a 7 DoF Franka-Emika Panda robotic arm. The actuation commands and sensory data are communicated using ROS protocol. There are three ROS nodes. The first node is the controller device that directly actuates the robot. The second node is an interface between an agent and the robot. This node needs to communicate with the controller device at a fixed rate of 1000 Hz. We have used a Jetson-Nano minicomputer with real-time OS for this node. The third node is a workstation where policy inference and learning happen. The task environment, which is defined in this node, sends actions to and receives sensory data from the robot at an arbitrary rate through communicating with the interface node. All communications happen in a wired network using ROS topics and messages. The camera sends images to the workstation through USB.

We implement the task environment with the standard structure of RL environments as in OpenAI Gym [8]. The agent controls the angular velocities of 7 joints resulting in a seven-dimensional action-space. The action values are limited to [-0.3,0.3]rad/s for each joint.

The observation consists of the most recent camera image and joint configurations. The observation-space is a tuple of $80 \times 60 \times 3$ RGB image, seven joint positions and seven joint velocities.

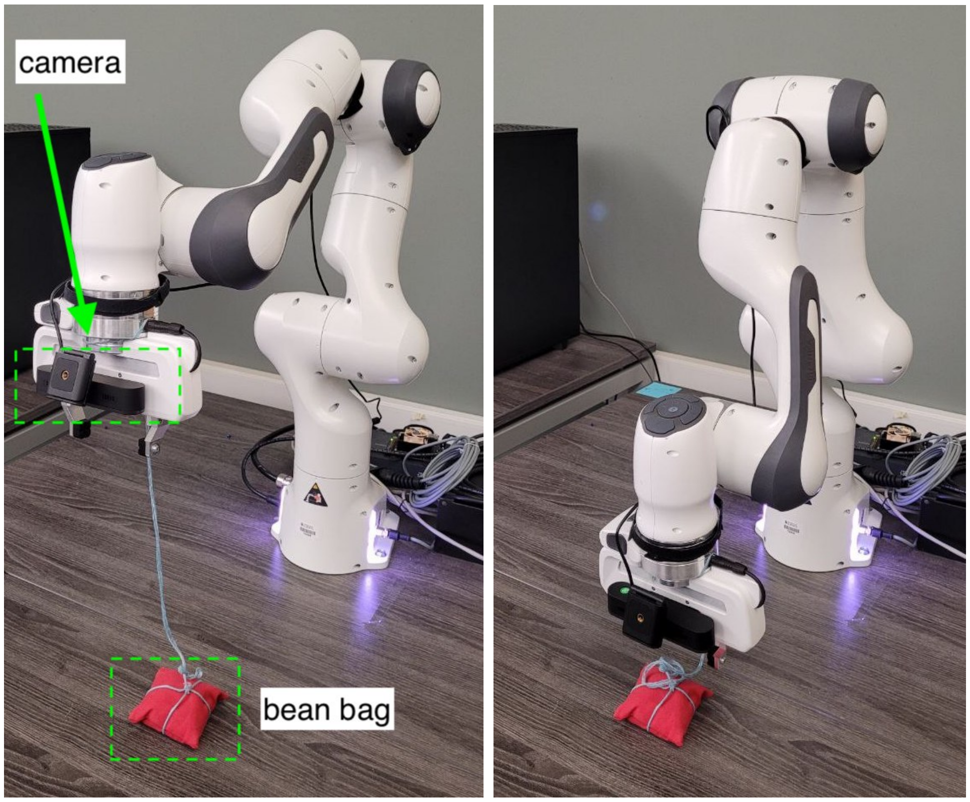


Figure 5.2: **Visual Reacher**. Before each episode, the robot randomly sets the position of the red bean bag on the table. The goal of the robot is to reach the bean bag by using visual inputs.

The sparse reward R is computed as

$$R(\mathbf{s}, \mathbf{a}) = \begin{cases} 1 & \text{if } \rho(\mathbf{s}) \geq 0.0125 \\ 0 & \text{otherwise} \end{cases}$$

with

$$\rho(\mathbf{s}) = \frac{1}{w \times h} \sum_{p \in \text{red pixels}} (0.5 - |p_x|)(0.5 - |p_y|)$$

where $\rho(\mathbf{s})$ is a metric that shows how big and close to the center the bean bag is in the image. $w = 80$, $h = 60$ are image dimensions and $p_x, p_y \in [-0.5, 0.5]$ are normalized coordinates of red pixels in the camera image.

Each episode takes 8s to complete, and the total number of time steps is determined according to the interaction frequency. Before the start of each episode, the target position of the bean bag is randomly sampled in an area of

$20 \times 30\ cm^2$. The robot places the bean bag in the target position using the string. Then all the joints are reset to a particular position.

5.2 Robustness w.r.t to the Interaction Frequency

In this part, we empirically analyze the learning performance of our method in different environment interaction frequencies. We hypothesize CTCO performs similarly across different interaction frequencies due to sampling the option durations in continuous time, while RL frameworks which choose an action for each task time-step have their learning performance influenced by the interaction frequency. Particularly they are not robust to high-frequency interactions. We first examine this robustness over different sets of $N_{RBF} \in \{1, 2, 3\}$ and $\beta_h \in \{0, 0.005, 0.01\}$ hyperparameters. Then we compare the frequency sensitivity of our method against classic RL by using soft actor-critic (SAC) [13], ARRL by using fine-grained action repetition (FiGAR-SAC) [36], and HRL by using double actor-critic (DAC-PPO) [50]. In this experiment, we use three simulated benchmarking tasks of Point-Mass, Cheetah and Ball-in-Cup from DeepMind Control Suite [43]. Details of these tasks are discussed in 5.1.3. We test algorithms in three tasks for four different interaction frequencies of $F \in \{50, 100, 250, 500\text{Hz}\}$. We evaluate the performance of each algorithm due to (5.1) after learning for 400 minutes of task time and compute the average performance over 30 runs with different random seeds.

5.2.1 Results

Tables 5.1, 5.2, and 5.3 show that the performance of CTCO in each task is robust across different interaction frequencies for a given combination of hyperparameters. The choice of β_h hyperparameter is critical for the algorithm's performance. For instance, in the task of *cheetah*, when β_h is 0.01, the policy favours longer options and cannot react fast enough to find a good policy. When β_h is set to zero, the algorithm's performance can be sensitive to the interaction frequency, as the policy may converge to the shortest possible durations. The performance of the algorithm is not significantly affected by the choice of N_{RBF} hyperparameter. In some cases, higher values of N_{RBF} perform better, especially when β_h is higher. This suggests that CTCO can benefit from options when choosing longer durations.

Table 5.1: Final performance of different hyperparameter settings for each interaction frequency in task of Point-Mass after 400 minutes of task time.

Hyperparameters		**Interaction Frequency (Hz)**			
$\boldsymbol{\beta_h}$	$\boldsymbol{N_{RBF}}$	$\boldsymbol{F = 50}$	$\boldsymbol{F = 100}$	$\boldsymbol{F = 250}$	$\boldsymbol{F = 500}$
0.0	1	1.08 ± 0.07	1.06 ± 0.07	1.06 ± 0.07	1.04 ± 0.08
0.0	2	1.04 ± 0.08	1.02 ± 0.07	1.00 ± 0.08	1.01 ± 0.08
0.0	3	1.01 ± 0.08	0.98 ± 0.08	0.95 ± 0.10	0.99 ± 0.09
0.005	1	1.12 ± 0.07	1.11 ± 0.07	1.11 ± 0.07	1.11 ± 0.07
0.005	2	1.08 ± 0.07	1.05 ± 0.08	1.06 ± 0.07	1.07 ± 0.07
0.005	3	1.04 ± 0.08	1.05 ± 0.08	1.02 ± 0.08	1.00 ± 0.10
0.01	1	1.13 ± 0.07	1.12 ± 0.07	1.12 ± 0.07	1.12 ± 0.07
0.01	2	1.11 ± 0.07	1.10 ± 0.07	1.08 ± 0.07	1.06 ± 0.08
0.01	3	1.09 ± 0.07	1.07 ± 0.07	1.06 ± 0.07	1.04 ± 0.08

Table 5.2: Final performance of different hyperparameter settings for each interaction frequency in task of Cheetah after 400 minutes of task time.

Hyperparameters		**Interaction Frequency (Hz)**			
$\boldsymbol{\beta_h}$	$\boldsymbol{N_{RBF}}$	$\boldsymbol{F = 50}$	$\boldsymbol{F = 100}$	$\boldsymbol{F = 250}$	$\boldsymbol{F = 500}$
0.0	1	0.85 ± 0.03	0.86 ± 0.03	0.87 ± 0.02	0.86 ± 0.03
0.0	2	0.89 ± 0.03	0.88 ± 0.02	0.89 ± 0.02	0.88 ± 0.02
0.0	3	0.92 ± 0.02	0.92 ± 0.02	0.89 ± 0.02	0.86 ± 0.03
0.005	1	0.67 ± 0.03	0.64 ± 0.05	0.67 ± 0.04	0.65 ± 0.03
0.005	2	0.65 ± 0.04	0.68 ± 0.04	0.68 ± 0.06	0.64 ± 0.06
0.005	3	0.81 ± 0.03	0.80 ± 0.03	0.79 ± 0.02	0.74 ± 0.07
0.01	1	0.58 ± 0.02	0.58 ± 0.02	0.58 ± 0.02	0.58 ± 0.02
0.01	2	0.59 ± 0.04	0.61 ± 0.03	0.64 ± 0.03	0.57 ± 0.03
0.01	3	0.70 ± 0.04	0.70 ± 0.04	0.68 ± 0.06	0.68 ± 0.05

Table 5.3: Final performance of different hyperparameter settings for each interaction frequency in task of Ball-in-Cup after 400 minutes of task time.

Hyperparameters		**Interaction Frequency (Hz)**			
$\boldsymbol{\beta_h}$	$\boldsymbol{N_{RBF}}$	$\boldsymbol{F = 50}$	$\boldsymbol{F = 100}$	$\boldsymbol{F = 250}$	$\boldsymbol{F = 500}$
0.0	1	1.70 ± 0.12	1.75 ± 0.11	1.82 ± 0.09	1.76 ± 0.10
0.0	2	1.83 ± 0.08	1.85 ± 0.08	1.84 ± 0.08	1.86 ± 0.08
0.0	3	1.84 ± 0.08	1.85 ± 0.08	1.86 ± 0.08	1.83 ± 0.09
0.005	1	1.73 ± 0.11	1.67 ± 0.13	1.74 ± 0.09	1.75 ± 0.10
0.005	2	1.81 ± 0.08	1.82 ± 0.08	1.82 ± 0.08	1.82 ± 0.09
0.005	3	1.80 ± 0.08	1.85 ± 0.08	1.84 ± 0.07	1.79 ± 0.11
0.01	1	1.69 ± 0.12	1.66 ± 0.11	1.70 ± 0.10	1.68 ± 0.13
0.01	2	1.75 ± 0.09	1.80 ± 0.09	1.78 ± 0.08	1.74 ± 0.09
0.01	3	1.76 ± 0.10	1.77 ± 0.08	1.78 ± 0.10	1.78 ± 0.09

In the following, we show the importance of decision frequency adaptation by comparing CTCO($\beta_h = 0.05, N_{RBF} = 3$) against SAC, DAC-PPO and FiGAR-SAC algorithms in different interaction frequencies. SAC and DAC-PPO agents have a constant decision frequency equal to the interaction frequency. FiGAR-SAC agent can alter its decision frequency by choosing the number of action repetitions from 1 to 10.

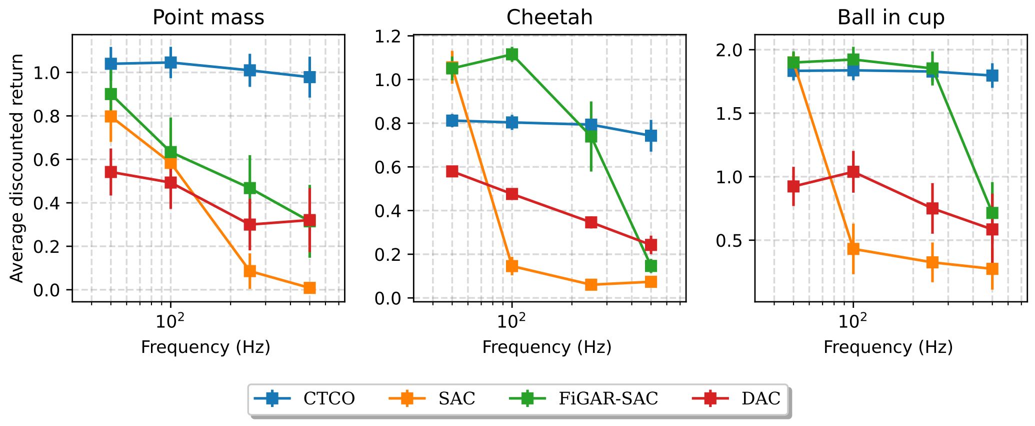


Figure 5.3: Performance Comparison of RL algorithms in three control tasks with varying interaction frequencies. Results from CTCO($\beta_h = 0.05, N_{RBF} = 3$), SAC, FiGAR-SAC, and DAC-PPO after 400 minutes of task time, averaged over 30 runs with 95% confidence interval.

Figure 5.3 shows that our algorithm maintains almost constant performance across different frequencies. While the performance of SAC, FiGAR-

SAC and DAC-PPO is influenced by interaction frequency. Specifically, at the highest interaction frequency, our method performs better than all other algorithms in all tasks. In the task of Cheetah, CTCO has sub-optimal performance, suggesting that with dense reward, simpler algorithms like SAC and FiGAR-SAC that can have the highest representational power may achieve higher performances.

5.3 Advantage of Continuous Options over Action Repetition

In this section, we investigate if our method can benefit from having option policies other than repeating actions for the cost of introducing a new hyperparameter N_{RBF} and increased action-space dimensionality. Policies based on action repetition have the most representational power when the repetition duration is one time-step. However, in this case, the exploration will be very limited, as shown in the previous section 5.2. On the other hand, CTCO can generate extended actions that are more complex than just repeating an action. We hypothesize that in cases where the exploration is crucial and, at the same time, smoothly changing actions are desired our method can benefit from parameterized options.

We aim to validate our hypothesis by creating a modified test case for the Point-Mass task, which involves placing an obstacle between the object and the goal position. We also change the action space to the desired position of the object. Actions will be effective only when the position commands are within a predefined distance from the object's position. In this way, and given that this environment has sparse rewards, learning a good policy requires effective exploration while taking smoothly changing actions. To show the effectiveness of parameterized options in our method over action repetition, we test CTCO with hyperparameters $\beta_h \in \{0, 0.005\}$ and $N_{RBF} \in \{1, 2, 3\}$, FiGAR-SAC and SAC to learn the task of Point-Mass-with-Obstacle.

5.3.1 Results

Figure 5.4 shows the learning performances of CTCO, SAC, and FiGAR-SAC in the task of Point-Mass-with-Obstacle for 400 minutes of task time. It can be seen that having more expressive options benefits exploration and learning performance in the task of Point-Mass-with-Obstacle. This result shows that action repetition policies, including CTCO with $N_{RBF} = 1$, are not suitable when the continuity in actions is desirable.

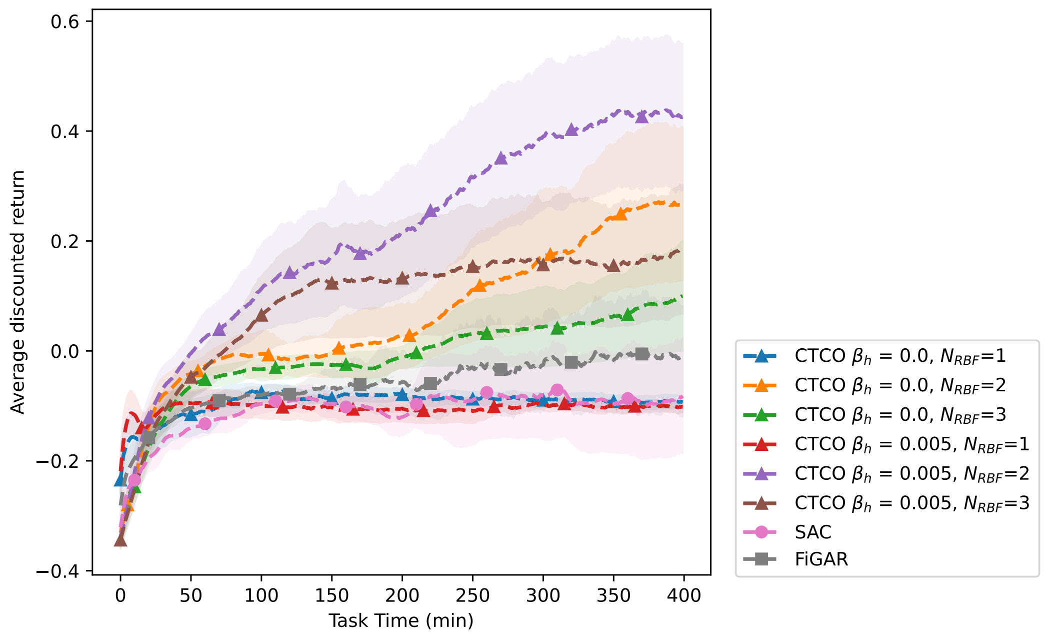


Figure 5.4: Comparison of action repetition and continuous options in the task of Point-Mass-with-Obstacle with $\Delta t = 0.02s$. Average performance of CTCO with different configurations, SAC and FiGAR-SAC over 30 runs with 95% confidence intervals are depicted.

Figure 5.5 depicts sample solutions of action repetition and CTCO agents with different option complexity in the task of Point-Mass-with-Obstacle. Note that in this task, primitive actions are desired 2D positions. It is evident that reaching the goal state with action repetition needs many right decisions. Whereas with more complex options, the continuously changing action increases the chance of reaching the goal state with better exploration.

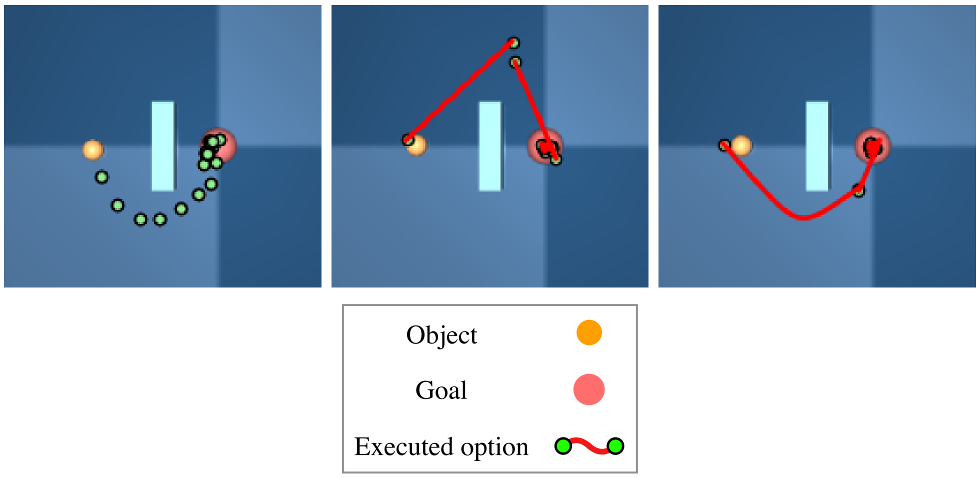


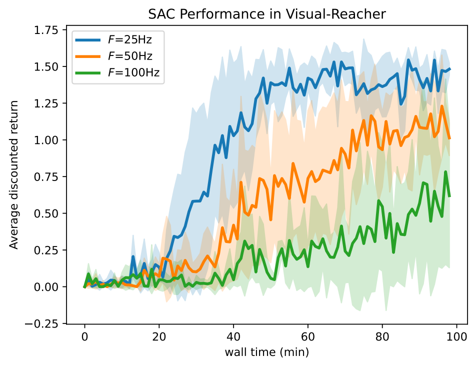
Figure 5.5: Visualization of actions in task of Point-Mass-with-Obstacle. From left to right, sample solutions of FiGAR-SAC, CTCO with $N_{RBF} = 2$, and CTCO with $N_{RBF} = 3$ are shown.

5.4 Real-world Evaluation

To test the ability of CTCO to work in a real-world scenario, we designed a sparse reward visual-reaching task using a robotic arm as described in 5.1.3. Although target reaching can be efficiently solved by using classic robotic techniques such as visual servoing, object detection, and planning, it is still valuable to evaluate the effectiveness of RL algorithms in the presence of real-world challenges in robot learning.

In this experiment, we assess the learning performance of CTCO with $\beta_h = 0.02$, $N_{RBF} = 2$ for different interaction frequencies of 25, 50 and 100 Hz. We also test the performance of SAC in these frequencies to emphasize the importance of temporally extended actions in the exploration for real-world goal-based tasks. For implementation details, refer to 5.1.

5.4.1 Results



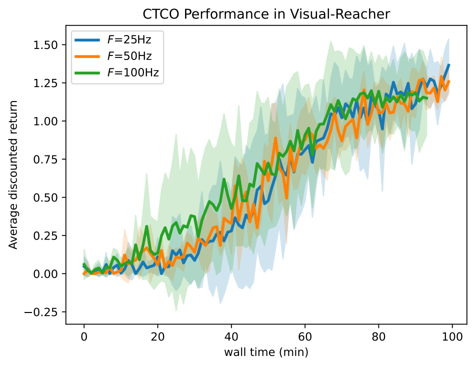


Figure 5.6: Learning performance of CTCO(right) and SAC(left) in real-world task of visual-reacher. Average of 5 runs with 95% confidence intervals for 100 minutes of task time.

Figure 5.6 shows that CTCO can robustly learn the challenging task of sparse reward visual-reaching in a practical time regardless of the interaction frequency. Although SAC performs better in the least interaction frequency, this method cannot learn properly with small time-interval interactions. This result justifies the possible benefits of infrequent decision-making in learning real-world tasks in several points. First of all, the slow dynamics of the environment makes exploration challenging with high-frequency decision-making. In addition to the exploration issues, limited resources in real-world scenarios, such as the memory for a data buffer with image data, are less of a problem with fewer interaction samples in time.

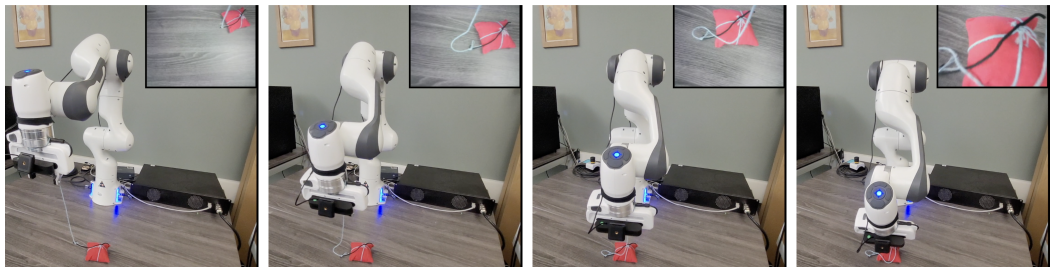


Figure 5.7: Series of figures showing robot motion deploying CTCO agent.

Figure 5.7 shows the learned behaviuor of CTCO in task of visual-reacher for one episode.

Chapter 6

Conclusion

In this thesis, we addressed the problem of learning to adapt the decision frequency in reinforcement learning for continuous control. Classic reinforcement learning algorithms that are defined in discrete time decide which action to apply at fixed time intervals. Choosing too short decision intervals causes exploration issues, whereas the system can become uncontrollable with too long decision intervals. As a new approach to adapting the decision frequency, we proposed a reinforcement learning framework where the agent selects continuous extended actions with variable continuous durations. This approach provides robustness with respect to the underlying interaction frequency and promotes high-level, smooth exploration. We empirically showed this robustness and the possible benefits of continuous options over simple action repetition in simulated and real-world robotic tasks.

6.1 Limitations and Future Work

While removing the decision frequency hyper-parameter, our algorithm suffers from two drawbacks: it introduces two new hyper-parameters and has open-loop option policies. The two new hyper-parameters define the number of RBFs composing the option policies and the high-frequency penalization. The agent's performance can be sensitive to the choice of these hyperparameters for different reward scales and task complexity levels. A future research direction could address learning the high-frequency penalization automatically. In our work, option policies are open-loop controllers. Therefore, they cannot respond to unexpected state changes in stochastic environments. For instance, we qualitatively show in the supplementary videos that in the task of visual reaching with a moving object, when running a policy pre-trained with static objects, CTCO may fail in tracking the object if the object location is changed too fast. Technically, the policy can counteract this deficit by choosing low-duration options (thus increasing the frequency of the feedback loop). However, lower-duration policies are undesirable since they complicate learning. This limitation can be compensated using closed-loop sub-policies or implementing termination policies [30].

References

[1] 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.

[2] C. G. Atkeson and S. Schaal, "Robot Learning From Demonstration," in *Proceedings of the Fourteenth International Conference on Machine Learning*, Morgan Kaufmann Publishers Inc., 1997, pp. 12–20.

[3] P.-L. Bacon, J. Harb, and D. Precup, "The Option-Critic Architecture," in *Proceedings of the AAAI Conference on Artificial Intelligence*, Issue: 1, vol. 31, 2017.

[4] L. C. Baird, "Reinforcement learning in continuous time: Advantage updating," in *Proceedings of 1994 IEEE International Conference on Neural Networks (ICNN'94)*, IEEE, vol. 4, 1994, pp. 2448–2453.

[5] M. Bloesch, J. Humplik, V. Patraucean, *et al.*, "Towards real robot learning in the wild: A case study in bipedal locomotion," in *Conference on Robot Learning*, PMLR, 2022, pp. 1502–1511.

[6] S. Bradtke and M. Duff, "Reinforcement learning methods for continuous-time markov decision problems," *Advances in neural information processing systems*, vol. 7, 1994.

[7] A. Braylan, M. Hollenbeck, E. Meyerson, and R. Miikkulainen, "Frame skip is a powerful parameter for learning to play atari," in *Workshops at the Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[8] G. Brockman, V. Cheung, L. Pettersson, *et al.*, "OpenAI Gym," *arXiv:1606.01540*, 2016, arXiv: 1606.01540. [Online]. Available: `http://arxiv.org/abs/1606.01540` (visited on 10/02/2019).

[9] S. Calinon, F. Guenter, and A. Billard, "On learning, representing, and generalizing a task in a humanoid robot," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 37, no. 2, pp. 286–298, 2007.

[10] W. Dabney, G. Ostrovski, and A. Barreto, "Temporally-extended {\epsilon}-greedy exploration," *arXiv preprint arXiv:2006.01782*, 2020.

[11] M. P. Deisenroth and C. E. Rasmussen, "PILCO: A Model-based and Data-efficient Approach to Policy Search," in *Proceedings of the 28th International Conference on International Conference on Machine Learning*, ser. ICML'11, event-place: Bellevue, Washington, USA, Omnipress, 2011, pp. 465–472, ISBN: 978-1-4503-0619-5. [Online]. Available: `http://dl.acm.org/citation.cfm?id=3104482.3104541` (visited on 10/02/2019).

[12] C. Finn and S. Levine, "Deep visual foresight for planning robot motion," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2017, pp. 2786–2793.

[13] T. Haarnoja, S. Ha, A. Zhou, J. Tan, G. Tucker, and S. Levine, "Learning to walk via deep reinforcement learning," *arXiv preprint arXiv:1812.11103*, 2018.

[14] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor," in *Proceeding of the 35th International Conference on Machine Learning*, 2018, pp. 1856–1865.

[15] S. Kleff, A. Meduri, R. Budhiraja, N. Mansard, and L. Righetti, "High-frequency nonlinear model predictive control of a manipulator," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, 2021, pp. 7330–7336.

[16] J. Kober, J. A. Bagnell, and J. Peters, "Reinforcement learning in robotics: A survey," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1238–1274, 2013.

[17] G. Konidaris, S. Osentoski, and P. Thomas, "Value function approximation in reinforcement learning using the fourier basis," in *Twenty-fifth AAAI conference on artificial intelligence*, 2011.

[18] O. Kroemer, S. Niekum, and G. Konidaris, "A Review of Robot Learning for Manipulation: Challenges, Representations, and Algorithms," *arXiv preprint arXiv:1907.03146*, 2019.

[19] A. Lakshminarayanan, S. Sharma, and B. Ravindran, "Dynamic Action Repetition for Deep Reinforcement Learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 31, 2017.

[20] S. Levine, N. Wagener, and P. Abbeel, "Learning contact-rich manipulation skills with guided policy search (2015)," *arXiv preprint arXiv:1501.05611*, 2015.

[21] 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.

[22] T. P. Lillicrap, J. J. Hunt, A. Pritzel, *et al.*, "Continuous Control with Deep Reinforcement Learning," in *International Conference on Learning Representations*, arXiv: 1509.02971, 2016. [Online]. Available: `http://arxiv.org/abs/1509.02971` (visited on 10/02/2019).

[23] A. R. Mahmood, D. Korenkevych, G. Vasan, W. Ma, and J. Bergstra, "Benchmarking Reinforcement Learning Algorithms on Real-World Robots," in *Conference on Robot Learning*, PMLR, 2018, pp. 561–591.

[24] 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.

[25] V. Mnih, A. P. Badia, M. Mirza, *et al.*, "Asynchronous Methods for Deep Reinforcement Learning," in *Proceedings of the 33rd International Conference on Machine Learning*, 2016, pp. 1928–1937.

[26] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[27] V. Mnih, K. Kavukcuoglu, D. Silver, *et al.*, "Human-Level Control Through Deep Reinforcement Learning," en, *Nature*, vol. 518, no. 7540, pp. 529–533, 2015, ISSN: 0028-0836, 1476-4687. DOI: `10.1038/nature14236`. [Online]. Available: `http://www.nature.com/articles/nature14236` (visited on 10/02/2019).

[28] T. Ni and E. Jang, "Continuous control on time," in *ICLR 2022 Workshop on Generalizable Policy Learning in Physical World*, 2022.

[29] A. Paraschos, C. Daniel, J. Peters, and G. Neumann, "Probabilistic Movement Primitives," in *Advances in Neural Information Processing Systems (NIPS)*, mit press, 2013.

[30] 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.

[31] R. E. Parr, *Hierarchical control and learning for Markov decision processes.* University of California, Berkeley, 1998.

[32] C. E. Rasmussen, *Gaussian processes in machine learning.* Springer, 2004.

[33] M. Riedmiller, R. Hafner, T. Lampe, *et al.*, "Learning by playing solving sparse reward tasks from scratch," in *International conference on machine learning*, PMLR, 2018, pp. 4344–4353.

[34] S. Schaal, "Dynamic movement primitives-a framework for motor control in humans and humanoid robotics," in *Adaptive motion of animals and machines*, Springer, 2006, pp. 261–280.

[35] J. Schulman, S. Levine, P. Moritz, M. Jordan, and P. Abbeel, "Trust Region Policy Optimization," in *Proceedings of the 32nd International Conference on Machine Learning*, 2015, pp. 1889–1897.

[36] S. Sharma, A. Srinivas, and B. Ravindran, "Learning to repeat: Fine Grained Action Repetition for Deep Reinforcement Learning," *arXiv preprint arXiv:1702.06054*, 2017.

[37] Z. Su, O. Kroemer, G. E. Loeb, G. S. Sukhatme, and S. Schaal, "Learning to switch between sensorimotor primitives using multimodal haptic signals," in *International Conference on Simulation of Adaptive Behavior*, Springer, 2016, pp. 170–182.

[38] D. Surovik, O. Melon, M. Geisert, M. Fallon, and I. Havoutis, "Learning an Expert Skill-Space for Replanning Dynamic Quadruped Locomotion over Obstacles," 2021, Publisher: Journal of Machine Learning Research.

[39] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* MIT press, 2018.

[40] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy Gradient Methods for Reinforcement Learning with Function Approximation," in *Advances in Neural Information Processing Systems*, 2000, pp. 1057–1063.

[41] R. S. Sutton, D. Precup, and S. Singh, "Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning," *Artificial intelligence*, vol. 112, no. 1-2, pp. 181–211, 1999, Publisher: Elsevier.

[42] C. Tallec, L. Blier, and Y. Ollivier, "Making deep q-learning methods robust to time discretization," in *International Conference on Machine Learning*, PMLR, 2019, pp. 6096–6104.

[43] Y. Tassa, Y. Doron, A. Muldal, *et al.*, "Deepmind control suite," *arXiv preprint arXiv:1801.00690*, 2018.

[44] H. Wang, T. Zariphopoulou, and X. Y. Zhou, "Reinforcement learning in continuous time and space: A stochastic control approach.," *J. Mach. Learn. Res.*, vol. 21, no. 198, pp. 1–34, 2020.

[45] R. J. Williams, "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning," *Machine learning*, vol. 8, no. 3-4, pp. 229–256, 1992.

[46] C. Yildiz, M. Heinonen, and H. Lähdesmäki, "Continuous-time model-based reinforcement learning," in *International Conference on Machine Learning*, PMLR, 2021, pp. 12 009–12 018.

[47] H. Yu, W. Xu, and H. Zhang, "Taac: Temporally abstract actor-critic for continuous control," *Advances in Neural Information Processing Systems*, vol. 34, pp. 29 021–29 033, 2021.

[48] Y. Yuan and A. R. Mahmood, “Asynchronous reinforcement learning for real-time control of physical robots,” in *2022 International Conference on Robotics and Automation (ICRA)*, IEEE, 2022, pp. 5546–5552.

[49] F. Zhang, J. Leitner, M. Milford, B. Upcroft, and P. Corke, “Towards vision-based deep reinforcement learning for robotic motion control,” *arXiv preprint arXiv:1511.03791*, 2015.

[50] S. Zhang and S. Whiteson, “Dac: The Double Actor-Critic Architecture for Learning Options,” *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[51] K. Zhou, J. C. Doyle, and K. Glover, *Robust and optimal control*, 1996.

[52] B. D. Ziebart, A. L. Maas, J. A. Bagnell, A. K. Dey, *et al.*, “Maximum entropy inverse reinforcement learning.,” in *Aaai*, Chicago, IL, USA, vol. 8, 2008, pp. 1433–1438.