Towards Stable Reinforcement Learning in Non-Episodic Tasks

by

Sathwik Karnik

B.S., Electrical Engineering and Computer Science, Mathematics
Massachusetts Institute of Technology (2022)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2023

Authored by: Sathwik Karnik
Department of Electrical Engineering and Computer Science
August 11, 2023

Certified by: Pulkit Agrawal
Assistant Professor
Thesis Supervisor

Accepted by: Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Towards Stable Reinforcement Learning in Non-Episodic Tasks

by

Sathwik Karnik

Submitted to the Department of Electrical Engineering and Computer Science
on August 11, 2023, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Despite recent advances in deep reinforcement learning (RL), deploying RL policies in robotics often leads to various challenges. The typical training paradigm in RL involves the rollouts of policies executed in a finite horizon or episodes. However, such policies may struggle to generalize well in various non-episodic tasks, including both object manipulation and locomotion. In this thesis, we study the challenges that arise from non-episodic tasks in two settings: (1) object manipulation in the Habitat Home Assistant Benchmark (HAB) [18] and (2) locomotion in the MuJoCo suite [20].

In the first of these two settings, we study the failure modes of the baseline methods and characterize much of the failures as being due in part to the instabilities in object placement and the lack of error recovery in the setting of open-loop task planning. We consider a possible approach to address this issue by modifying the steady-state termination condition in the RL objective to place the object at the goal position for a longer horizon. We next consider an error-corrective policy using inverse-kinematics (IK) following the execution of the RL policy. The integration of an IK policy leads to a significant improvement in the final task success rate from 41.8% to 65.3% in SetTable, one of the three tasks in the HAB.

In the second setting, we consider extrapolation in the non-episodic task of locomotion in the MuJoCo suite. Typical RL policies are trained for a finite horizon, but may need to be executed for a longer horizon during deployment in locomotion tasks. However, current RL approaches may fail to generalize beyond the training horizon. To address this issue, we consider the use of time-to-go embeddings as part of the observations. Specifically, we introduce the use of a constant time-to-go embedding in the setting where the horizon is much longer during evaluation or deployment. We find limited evidence of improvements in the average episode returns during evaluation in 6 tasks in the MuJoCo suite.

Thesis Supervisor: Pulkit Agrawal
Title: Assistant Professor

Acknowledgments

There are several people in my life that truly made all the difference over the past several years.

Professor Leslie Kaelbling deserves special thanks for introducing me to the world of robotics research. Though I did not end up doing research in task and motion planning, Leslie was my first professor in machine learning and later became my research mentor, guiding me through my evolution from classical robotics to robot learning.

One could say that my past year in the Improbable AI Lab completed this evolution (though with a healthy dose of skepticism!), after enjoying Professor Pulkit Agrawal's seminar in Computational Sensorimotor Learning (CSL). I have been truly privileged to have been advised by Pulkit over the past year. His mentorship and support have shaped my research outlook and my perspective on the future of robotics.

Additionally, I would like to thank my research collaborators over the years. Gustavo Goretkin deserves special recognition for his guidance and patience as my first mentor. I would also like to thank Chandler Squires and Rohan Chitnis for their guidance and support in two separate projects in my senior year. Over the past year, I have enjoyed the mentorship of Zhang-Wei Hong, whose guidance and depth of knowledge as an experienced RL researcher have been invaluable.

Lastly, I would not be here without the endless support of my parents, Vishwanath and Rathna. Finally, I am truly lucky to have an older brother Karthik who has always been there with me.

Contents

List of Figures

Chapter 1

Introduction

A key objective of the robotics community has been the development of robots that can serve as assistants in performing manual chores, especially in home-like environments. To this end, we focus on algorithms that can enable robots to perform object manipulation and locomotion tasks in the mobile manipulation setting. The purpose of this thesis is to address important questions that arise from solutions to mobile manipulation tasks.

To illustrate such problems, we first focus on the Habitat Home Assistant Benchmark (HAB) released by Meta in 2022 [18]. In this challenge, a mobile manipulator (i.e., a home robot) is tasked with achieving objectives in a home-like simulator environment. The challenge presented seeks to find a way to seamlessly execute various skills, including picking and placing, opening and closing, and navigation. The first objective of this thesis is the development of methods for improving the accuracy of completing a curated set of tasks, as provided by the Habitat 2.0 Challenge. In this setting, we assume that an open-loop high-level task plan is provided to the agent. The objective of the agent is to execute each skill and switch from one skill to the next skill. In this thesis, we first examine a significant failure mode in current open-loop solutions to the HAB – namely, instabilities arising from object placements. In Chapter 2, we consider various solutions and illustrate that a corrective inverse-kinematics policy helps improve performance.

While the first part of this thesis focuses on the challenges associated with ob-

ject manipulation, the second part focuses on the problem of non-episodic tasks using policies trained using reinforcement learning (RL). In particular, we consider issues that arise from the common assumption in RL that the policy for solving the infinite-horizon problem extends beyond the time-limited finite-horizon problem that is constrained in practical applications of RL. To this end, we examine the use of the time-to-go in observation inputs to RL policies. We introduce methods for using time-awareness as a means to extrapolate beyond the finite horizon used during the training of the RL policy. We illustrate our findings in on-policy RL methods in the MuJoCo [20] suite of locomotion tasks, illustrating both the benefits and limitations of our method.

This thesis spans two adjacent problems in the search of discovering control policies for stable behavior for non-episodic tasks in robotics settings.

Chapter 2

Stability of Object Placements in the Home Assistant Benchmark

In 2022, Meta released the Habitat 2.0 Rearrangement Challenge and the Home Assistant Benchmark (HAB), a benchmark suite for performing mobile manipulation tasks in a home-like environment [18]. The HAB consists of three mobile manipulation problems: TidyHouse, PrepareGroceries, and SetTable. In each of these tasks, a Fetch robot [1] is asked to rearrange objects from various parts of a virtual apartment. As part of this rearrangement, the robot may need to navigate to different parts of the apartment, open or close a drawer, and pick and place objects. The basic premise of each task is that the agent is provided sensory information about the goal and its progress towards the goal.

The challenge of mobile manipulation has been studied from various perspectives, including reinforcement learning (RL) and task and motion planning (TAMP). Despite the appeal of deploying a single end-to-end RL policy (referred to as *monolithic RL*) for commanding low-level controls, such methods have not had much success in mobile manipulation tasks [18]. On the other hand, TAMP provides a hierarchical approach with a high-level task planner and a low-level motion planner, thus enabling an agent to discover control policies for solving different tasks. TAMP solutions may either involve a precomputation of the task plan before deployment (open-loop) or an integration of the task planner and motion planner with feedback during deployment

(closed-loop).

Prior work evaluated on the HAB ([18], [8]) has studied solutions with open-loop task planning with skills trained with RL (denoted as TP+SRL). Gu et. al. [8] propose region-based rewards for training navigation skills and illustrate a marked improvement over the baseline performance. However, in SetTable, for instance, the success rate of the full task sequence is still under 30% when evaluated cross-layout on the held-out test set and 41.8% on the validation set, as provided by Gu et. al. [8].

In this work, we explore a common failure of open-loop task planning in mobile manipulation: incorrect object placements. We define an object placement as being *goal-stabilizing* if the policy successfully places the object and the object stays in the designated goal region in the subsequent timesteps where the object is not required for manipulation. In long-horizon mobile manipulation tasks where the task planner is an open-loop controller, the success of object placements depends on a goal-stabilizing placing policy, as the agent has no mode of error-recovery. As a case study, we focus on the HAB and first consider the task specification of the place skill in SRL. We first provide an alternate task specification that modifies the prior success criteria for the place skill. Lastly, we consider a solution that augments the SRL with inverse kinematics (IK) to control the position of the end effector for goal-stabilizing object placements.

2.1 Mobile Manipulation Failures from Object Placements

In this section, we consider the failure modes of mobile manipulation in the Habitat Home Assistant Benchmark and approaches to address these issues from the perspective of the task specification of RL placing policies in the framework of an open-loop task planner.

2.1.1 Failure Cases of Mobile Manipulation in the Home Assistant Benchmark (HAB)

To understand the failure modes of open-loop task planning with skills trained using RL (TP+SRL), we illustrate the various outcomes of evaluation on the HAB. In the context of Figure 2-1, we refer to a place execution as being *unstable* if the object placement is not goal-stabilizing. Figure 2-1 illustrates the breakdown of evaluation outcomes, including successes, failures due to unstable place skill executions, or other failures. As evaluated in the validation set, in SetTable, the most challenging of the three HAB mobile manipulation tasks, the baseline TP+SRL has a success rate of 41.8% (and a failure rate of 58.2%), and of the unsuccessful episodes, 23.4% are due to failures in place_0 (i.e., the first placing task in the task sequence) and 54.6% are due to failures in place_1 (i.e., the second placing task in the task sequence). Figure 2-1 highlights that the object placements are not goal-stabilizing and present a significant failure mode in open-loop mobile manipulation task planning.

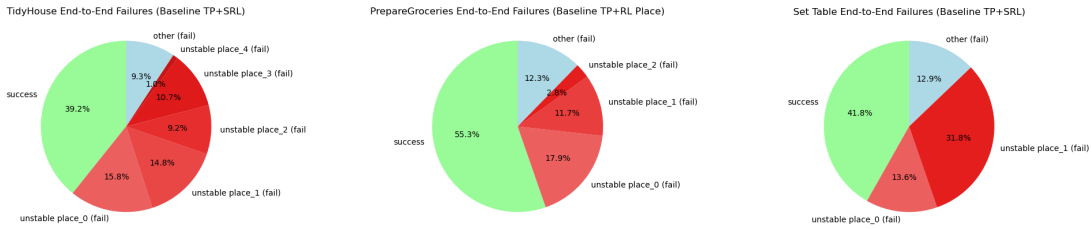


Figure 2-1: This figure shows the failure modes of the baseline TP+SRL [8] in the HAB when evaluated on the validation set. Best viewed zoomed in.

2.1.2 Task Specification of Object Placement in Reinforcement Learning

To better understand why the place skill in SRL is not goal-stabilizing, we now examine the prior task specification ([18], [8]) of the placing task in TP+SRL. Let $\mathbf{x}_o, \mathbf{x}_g$ be positions of the object and its goal, respectively, and let $d^g_{o,t}$ be the Euclidean distance between $\mathbf{x}_o$ and $\mathbf{x}_g$ at time t. Let $\mathbf{x}_{ee}, \mathbf{x}_r$ be positions of the end effector and

its resting goal, respectively, and let $d^r_{ee,t}$ be the Euclidean distance between $\mathbf{x}_o$ and $\mathbf{x}_g$ at time t. For context in the Habitat simulator, the robot is assumed to receive continuous sensor readings of d^g_o and $d^r_{ee,t}$ [18].

Let $\mathbb{1}_{holding}$ indicate whether the robot is currently holding an object (and similarly $\mathbb{1}_{!holding}$ for not holding an object), and let $\mathbb{1}_{success}$ denote the success measure for the object placement task. For thresholds $\epsilon_g, \epsilon_r \geq 0$, the object placement success measure is defined as:

$$\mathbb{1}_{success} = (d^g_o \leq \epsilon_g) \wedge \mathbb{1}_{!holding} \wedge (d^r_{ee} \leq \epsilon_r)$$

In other words, if the object position is within ϵ_g of the goal position, the robot is no longer grasping an object, and its end effector is within ϵ_r of the resting position, then the policy is deemed successful. However, this task specification does not enforce constraints on the future positions of the object. In other words, during the training of an RL policy, an episode may be terminated before the object is at rest, as long as the object is within the acceptable goal region. This can lead to unintended consequences, as an object may fall and escape the goal region.

To address this problem, we consider a slight modification of this task specification in which the object placement policy is deemed successful if the object is additionally not grasped and at the goal position for Δ consecutive timesteps:

$$\mathbb{1}_{success} = \left[\bigwedge_{i=0}^{\Delta-1} (d^g_{o,t-i} \leq \epsilon_g) \wedge \mathbb{1}_{!holding}\right] \wedge (d^r_{ee} \leq \epsilon_r)$$

We hypothesize that this specification will encourage the agent to place the object to be within the goal for more time. To understand the need for the "not holding" indicator $\mathbb{1}_{!holding}$, we note that grasping the object may be an easier way for ensuring the object placement to not fall outside of the goal region for Δ timesteps. We hypothesize that this indicator $\mathbb{1}_{!holding}$ is needed for the duration of the Δ timesteps to ensure that we do not encourage the agent to discover a policy of holding the object near the goal. Lastly, we note that Δ is a hyperparameter that can be chosen during training.

2.2 Goal-Stabilizing RL Place Skill Policies using Inverse Kinematics

While the mobility of a robot's base expands its reach in the workspace, the robot's placing actions can be unstable. For instance, the robot may displace its base while placing an object on a surface. This presents the challenge of making goal-stabilizing object placements. Another source of instability may arise from the robot dropping an object from a high vertical distance, resulting in higher variations in the object's contacts with the goal surface. To mitigate these issues, we make use of inverse kinematics (IK) upon the execution of the RL-based object placement policy.

In the context of the Habitat simulator, the grasping action is emitted as a scalar control; when the scalar is negative, the gripper is holding the object, and when the scalar is positive, the gripper is not holding an object. Based on the scalar emitted by the agent, the agent can release or grasp objects. When the RL policy outputs a grasping action that is positive, the object would typically be ejected from the gripper. Instead, in our IK-based approach, we make a switch from executing the RL policy to executing the IK policy.

The IK-based policy executes a control sequence generated from inverse kinematics to move from its initial position to the desired end position of the end effector. To find a desired end effector position, we consider the initial $(x, y, z) \in \mathbb{R}^3$ position of the end effector at the start of IK policy. This position must be directly above the goal region, given both the fact that the RL-based policy output would command the gripper to drop the grasped object and that the RL-based policy has a high success rate evaluated individually (see Section 2.1.2). Thus, the desired end effector position may be chosen to be $(x, y, z - \varepsilon)$ for some chosen $\varepsilon \geq 0$. Once the robot executes the IK-generated plan, the policy commands a positive grasping action, thus dropping the object.

2.3 Evaluation on the Habitat 2.0 Home Assistant Benchmark

We evaluate our methods in the three tasks given the provided validation set [8] of the Habitat Home Assistant Benchmark, as each of the three tasks deploy the placing skill. We evaluate on 100 episodes across 3 seeds used for training and 3 seeds for evaluation (for a total of 9 seeds and 900 episodes). Figure 2-2 illustrates the success curves (reported as the mean and standard error) across the task sequences. The success of each task depends on the continued successes of the previous tasks. For instance, in SetTable, close_1 fails if the object placed during place_1 falls outside of the goal region. Additionally, Figure 2-3 illustrates the failure modes of TP+SRL using inverse kinematics, similar to Figure 2-1.

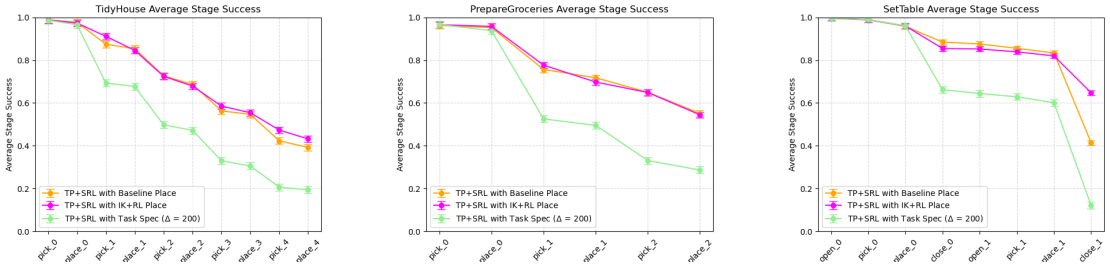


Figure 2-2: Average Success Rates in the Home Assistant Benchmark

Figure 2-2 shows that the use of inverse kinematics in substantially increases the final task success rate in SetTable from 41.8% to 65.3%. This improvement in success is directly attributed to the goal-stabilizing behavior, as evidenced by the reduction in failures due to object placement from a total of 45.4% (13.6% for place_0 and 31.8% for place_1) in the baseline TP+SRL to 18.0% in the TP+SRL and IK placing skill. Interestingly, training the RL policy with our task specification of $\Delta = 200$ timesteps and for 200M frames (instead of the defaults of 200 steps and 100M frames) led to worse performance in each of the three tasks. Additionally, for both the TidyHouse and PrepareGroceries tasks, the use of inverse kinematics does not have a substantial effect in performance. In the next section, we investigate an explanation for why inverse kinematics does not improve the performance significantly in these two HAB

tasks.

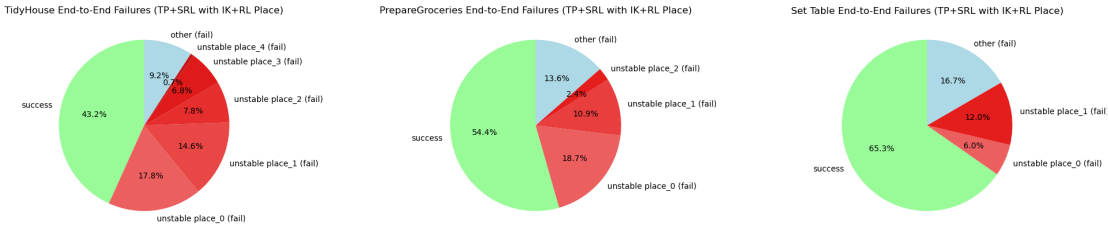


Figure 2-3: This figure shows the failure modes of the TP+SRL with inverse kinematics used in the placing skill [8] in the HAB when evaluated on the validation set. Best viewed zoomed in.

2.4 Does the Success Rate of Object Placements Depend on the Object Type?

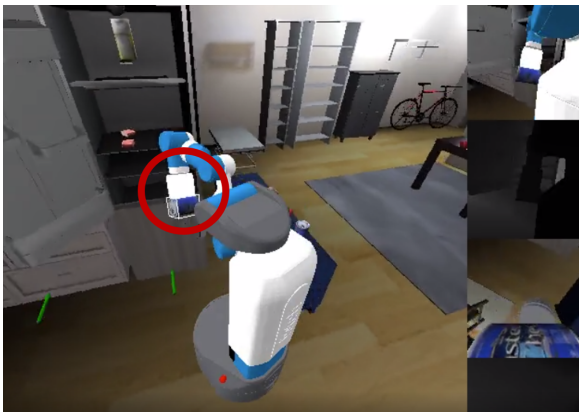
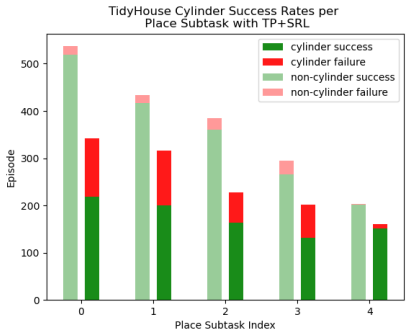
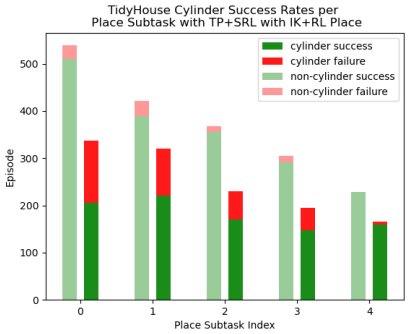


Figure 2-4: This figure depicts the Fetch robot grasping a can (in red) in the Habitat simulator.

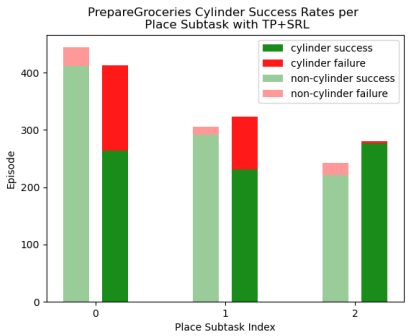
To study the limitations of modifying only the place skill, we analyze the relationship between the object placement success rate and the object types. In fact, we find that the majority of the failure cases in the TidyHouse and PrepareGroceries tasks result from difficulties associated with placing cylinders (i.e., the target object to be placed is a soup can). We illustrate in Figure 2-5 the number of goal stabilizing object placement failures that are associated with placing cylinders and placing non-cylinders.



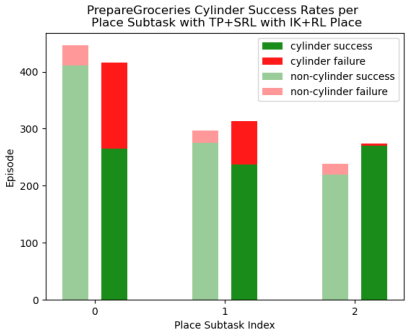
(a) TidyHouse (TP+SRL)



(b) TidyHouse (TP+SRL with IK+RL)



(c) PrepareGroceries (TP+SRL)



(d) PrepareGroceries (TP+SRL with IK+RL)

Figure 2-5: Success Rates of Placing Cylinders and Non-Cylinders in TidyHouse and PrepareGroceries

Figure 2-5 illustrates that the majority of the failures associated with placing objects in TidyHouse and PrepareGroceries is associated with placing cylinder objects. This is challenging even when switching from the RL placing policy to the inverse kinematics policy, as evidenced by a comparison between the two columns of Figure 2-5. This may be explained by the fact that our inverse kinematics solution still strictly follows the open-loop task plan and does not perform an adjustment to the grasp (see Figure 2-4). This is especially undesirable in placing cylinders, as the jaws of the gripper may obstruct the two flat surfaces of the cylinders. On the other hand, SetTable does not involve cylinder target objects, thus benefiting greatly from the use of inverse kinematics.

2.5 Discussion

In this work, we identified and analyzed a common failure mode of open-loop task planning in the context of Meta's Habitat 2.0 Home Assistant Benchmark. We illustrated that failures due to the instabilities in object placement lead to failures in mobile manipulation tasks. We considered multiple approaches to address these failures, including modifying the task specification to encourage the agent to place the object in the goal position for more timesteps. Finally, we illustrated the benefits and limitations of an error-corrective inverse-kinematics policy for more stable object placements (or goal-stabilizing execution, as we define here).

The use of inverse kinematics enables us to exploit the Cartesian action space for stable object placements. This switch from an RL policy with joint space actions to inverse kinematics leads us to consider the possibility of a single RL policy executing such actions with delicate manipulation in a seamless way.

Previous work has analyzed the comparison of various action spaces for learning manipulation tasks and has considered the use of impedance control for performing actions in RL settings and, thus, make use of a task space controller, in contrast with configuration space controllers. The use of an impedance controller has been shown to have benefits in peg manipulation tasks [22] and other manipulation tasks, such as door opening and surface wiping [10]. Additionally, prior work has considered unifying the joint and Cartesian action spaces and have illustrated its benefits through implicit policies [6], much like the policies trained from implicit behavior cloning [5].

A key limitation of the methods used in our work was the assumption of an open-loop task planner. This led our work to use inverse kinematics to search in the Cartesian space and solve for the joint angles for producing more goal-stabilizing action sequences. One way to address this would be to employ a closed-loop task and motion planner to enable for integrated planning and error recovery when the preconditions of a particular predicate are not satisfied [7]. Such solutions are advantageous in practice, though could be limited by the planning time needed for robot execution and the need for symbolic representations in satisfying preconditions and achieving

the desired effects of predicates. Recent work has also considered executing open-loop plans generated from a STRIPS planner [4], as we do here, but using Logical Geometric Programming to search over the limited set of relative object poses, thus leading to faster wall-clock time execution [11]. Such methods may provide promising alternative approaches that

Chapter 3

Learning to Extrapolate the Horizon in Locomotion Tasks

3.1 Introduction

The current paradigm of reinforcement learning (RL) trains the agent by rolling out trajectories in the environment, observing the returns (i.e., sum of rewards over time), and updating the policy to increase the returns. We can characterize the agent's task as being either episodic or non-episodic. Many practical applications are non-episodic, which are common in cyclical tasks (e.g. locomotion) or goal-reaching tasks (e.g. dextrous manipulation).

In the practical training procedure in RL, the trajectory rollouts are time-limited by the choice of the *training horizon*. However, in non-episodic tasks, such policies trained in the finite-horizon setting may be deployed for horizons longer than the training horizon. Despite the appeal of typical RL, the learned policies may not generalize well beyond the training horizon or in the infinite-horizon setting.

The discount factor γ plays an important role in affecting the agent's ability to make such a generalization. In particular, setting $\gamma < 1$ can guarantee the convergence of the value function, but it can also limit the effective planning horizon [3]. Solving the infinite-horizon task in RL may involve estimating the value function using the undiscounted rewards ($\gamma = 1$) at future states. However, setting the discount factor

$\gamma = 1$ can be impractical, as the value function can diverge when there is no absorbing state in the observation space. Additionally, in theory, setting the discount factor γ to be large (but still $\gamma < 1$) could still lead to high variances in the value estimates [19].

Given that the discount factor provides a way to specify the effective planning horizon, one may be able to approximate the optimal policy for a discounted infinite-horizon task by finding the optimal policy in the undiscounted finite-horizon task [9]. As part of our ongoing empirical work, we claim that this approximation could lead to lower variance in the value estimates by including the current number of timesteps remaining (or *time-to-go*) as part of the observation.

On its own, using the time-to-go embedding as part of the observation presents challenges in the infinite-horizon setting. Practically, the episode length may not be known for computing the time-to-go and the variance in the value estimates could increase at a prohibitive rate as the episode length grows. To address these issues, we propose methods that aim to approximate the policy for solving an undiscounted finite-horizon task at every timestep. In this work, we do so by inputting a constant time-to-go in the observation during the deployment of the policy in the infinite-horizon task. During the training procedure, the policy π and value function V receive as input $s \| \varphi(t_{go})$ – the concatenation of the observation s and the φ embedding of time-to-go t_{go}. However, during the evaluation, we query the policy with $\pi(s||\varphi(C))$ for constant C at every timestep.

Our work makes several contributions, distinct from prior work on time-to-go embeddings in observations. In particular, we introduce a geometric time-embedding, different from the linear time-embedding previously studied [13] Importantly, we consider the use of our time-embedding as part of the observation input to RL policies and training with episode length T_{train} while evaluating the performance with the episode length $T_{test} \gg T_{train}$. To this end, we consider using a constant time-to-go embedding input as part of the observation during deployment and illustrate its benefits across six tasks in the MuJoCo locomotion suite [20].

3.2 Reinforcement Learning Policies Beyond the Training Horizon

In this section, we contextualize the objective of typical RL and provide intuition to explain its limits in generalization across the time horizon in simulated locomotion environments.

3.2.1 Typical Objective in Reinforcement Learning

Formally, in RL, the objective is to learn a control policy π in a Markov Decision Process (MDP). An agent starts at an initial state sampled from an initial state distribution $\rho(s_o)$. At each timestep t, the agent observes the state s_t and chooses an action $a_t \sim \pi(\cdot \mid s_t)$, after which the agent receives reward $r_t = R(s_t, a_t)$ and transitions to the state s_{t+1}, as defined by the MDP. In the setting of online RL, the objective is to learn $\pi^* = \text{argmax}_\pi J(\pi)$, where $J(\pi)$ is the expected γ-discounted infinite-horizon return:

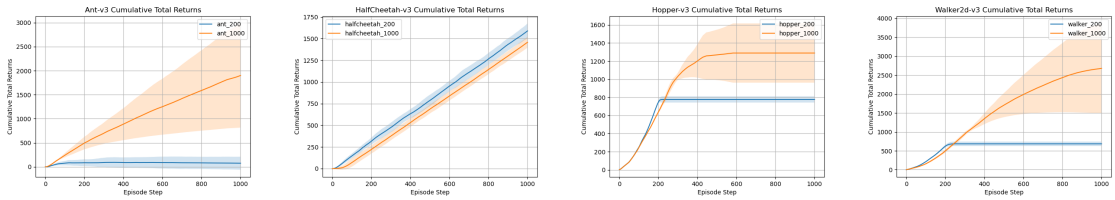
$$J(\pi) = \mathbb{E}_{s_0 \sim \rho_0, a_t \sim \pi(\cdot|s_t), s_{t+1} \sim \mathcal{T}(\cdot|s_t, a_t)} \Big[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \Big]$$

In practice, directly solving the infinite-horizon problem may not feasible and a finite time limit is often imposed.

3.2.2 Limited Capacity of RL to Generalize Beyond the Training Horizon

A question that arises from imposing such time limits is: how well does the policy perform beyond the time limit of an episode during training? This question leads us to our choice of evaluation metric. In particular, we measure how well a policy can make such a generalization by training using an episode time limit of T_{train} and testing using an episode time limit of T_{test}. In our focus on non-episodic tasks, we set $T_{test} > T_{train}$.

Figure 3-1 illustrates performance gaps in the cumulative episode returns corresponding to two sets of policies in the MuJoCo suite of locomotion tasks [20]. One set of policies is trained with a maximum episode length of 200 timesteps, and the other set of policies is trained with a maximum episode length of 1000 timesteps. We note that, despite the cyclical nature of locomotion, the policies trained for 200 timesteps do not match the performance of the policies trained for 1000 timesteps in three of the four tasks (Ant-v3, Hopper-v3, and Walker2d-v3).



(a) Ant-v3 (b) HalfCheetah-v3 (c) Hopper-v3 (d) Walker-v3

Figure 3-1: This figure illustrates the cumulative evaluation episode returns with two sets of policies: one trained with episodes of length 200 timesteps and another trained with episodes of length 1000 timeteps.

By imposing a time limit for the episode length in RL training, the objective is to train a policy to maximize the discounted sum of rewards within the episode. Oftentimes, there are robot tasks that may require infinite-horizon planning; one notable example is robot locomotion. A robot trained with a locomotion policy for a short horizon may not generalize well beyond the time limit. In fact, we observed how the length of the training episode affects the performance in Figure 3-1, where we compared two sets of policies with different training episode lengths.

3.2.3 Role of the Discount Factor in Generalization

The discount factor γ plays an important role in the ability of an RL policy to generalize well beyond the training horizon. Practically, the discount factor is used to weight the relative importances of future rewards. A policy trained with a low discount factor can lead to actions that are myopic, as the value estimates more heavily discount the rewards from future timesteps; a policy trained with a high discount factor can lead to actions that are made aware of rewards from future timesteps, as is

desirable in non-episodic tasks. In other words, the choice of γ can limit the effective training horizon.

Despite the apparent benefit of higher discount factors in non-episodic tasks, it is important to consider that $\gamma < 1$ can theoretically guarantee the convergence of the value function [3]. On the other hand, setting $\gamma = 1$ can theoretically lead to the divergence of the value function when there are no sink states (or absorbing states) in the observation space of the MDP. Even large values of γ (but still $\gamma < 1$) have been found to lead to difficulties in training and could lead to a higher variance in the value estimates [19].

We note that the optimal policy in the discounted return infinite-horizon setting can be approximated by finding the optimal policy in the finite-horizon setting [9]. What this means is that planning with an infinite horizon and discounted rewards can be analogous to planning with a finite horizon and undiscounted rewards at every timestep. This claim may be intuitively explained by the fact that the discount factor provides a way to limit the effective planning horizon [9]. Given the nature of the finite horizon setting, where a time limit (or horizon) is imposed as part of the task, we may incorporate the current number of timesteps remaining (or *time-to-go*) in the episode as part of the observation. The inclusion of the time-to-go as part of the observation theoretically guarantees the convergence of the value function even when $\gamma = 1$, as the recursive calculation of the value function follows the dynamic programming paradigm [2].

3.2.4 Time-to-Go Embedding

Let s_t denote the observation at time t, and for episode time-limit T, let $\varphi(t, T)$ denote the embedding of time-to-go t_{go} that is appended to s_t before executing the policy $\pi(a_t \mid (s_t \,\|\, \varphi(t, T)))$. The inclusion of the time-to-go as part of the observation space has not been thoroughly studied. The idea was proposed in the context of various approaches to address time limits in practical applications of RL [13], which studied time-to-go, as well as partial-episode bootstrapping. Importantly, the prior motivation for the time-to-go in the observations differs from ours, as the previous

work explores time-to-go in the context of solving the finite-horizon problem. In contrast, we motivate the use of time-to-go as a means to extrapolate the policy beyond the training episode during deployment.

The prior work [13] illustrates the benefits of using time-to-go as part of the observation in the MuJoCo suite in comparison with a baseline implementation of proximal policy optimization (PPO) [17]. In their results [13], the use of time-to-go has a marked improvement in the average training episode returns over the performance of the baseline. However, in an attempt to reproduce these results, we found that the results could not be reproduced to match the relative performances in comparison with the baseline. In particular, using an optimized set of hyperparameters for PPO tuned in the MuJoCo suite [15], we illustrate in Figure A-1 our unsuccessful attempt at reproducing the results of the prior work on time-awareness, matching prior unsuccessful attempts to reproduce the results [14].

At a glance, we observe that the deployment of a policy with the current time-to-go embedding may require inputting the episode length, thus making it undesirable in the infinite-horizon setting. Additionally, the behavior of the policy for the same time-unaware observation will be dependent on the time-to-go. Thus, the present use of time-to-go in the observation may not be suitable for infinite-horizon or non-episodic tasks.

3.2.5 Constant Time-Embedding During Evaluation

Central to our proposed method is a *constant* time-to-go input to the observation only during deployment instead of the *current* time-to-go input at each timestep. Assuming that the policy network is capable of generalizing to out-of-combination inputs, inputting the observation and a constant time-to-go C can be thought of as invoking the policy with the observation and C timesteps to go. By invoking the policy with a constant time-to-go at each timestep, the policy is queried at each timestep with the information that there are C timesteps to go. In the context of receding horizon approaches, the constant time-to-go alludes to the strategy of approximating the solution to the infinite-horizon task by solving a sequence of finite-horizon tasks.

It is important to note that our work does not provide theoretical guarantees of this approximation but, rather, explores this intuition in non-episodic tasks.

3.2.6 Contributions

In this work, we consider the use of time-to-go in observations from the context of generalization beyond the training episode horizon. We reintroduce the use of time-to-go as part of the observations with this new perspective and propose a geometric time-embedding, in contrast with the prior work using a linear time-embedding. During the training, we make use of domain randomization by varying the episode lengths during the training to provide the policy with varied combinations of time-unaware observations and time-to-go embeddings. Finally, during the deployment, we introduce the use of a constant time-to-go embedding and illustrate its limited benefits in the MuJoCo suite tasks.

3.3 Methods

3.3.1 Choice of Time-to-Go Embedding

In the prior work introducing time-awareness [13], the observations were appended with $\varphi(t, T) = 1 - (t/T)$, where t represents the current timestep and T represents the maximum episode length. One possible concern regarding this time embedding is that the initial time-to-go embedding is the same regardless of the episode length, thus producing the same initial behavior regardless of the time horizon.

Our choice of time-to-go embedding is the d-dimensional vector $[\lambda_i^{(T-t)}]_{i=1}^{d}$, where $\lambda_i \in [0, 1]$. This geometric time-embedding aims to enable the agent to distinguish between horizons of varying lengths and is, thus, hypothesized to be more preferred in the task of generalizing the policy during deployment with a longer time horizon. In our implementation, during the training, we choose evenly-spaced $\lambda_i \in [0.99, 1.0]$. As $(T - t) \to \infty$, the embedding dimension $\lambda_i^{(T-t)}$ converges to 0 at a faster rate for $\lambda_i = 0.99$ than for $\lambda_i = 0.999$. Increasing d enables greater expressivity of the

time-to-go.

3.3.2 Training Procedure with Time-to-Go Embedding

During the training, we set the discount factor $\gamma = 1.0$ to increase the effective planning horizon. As previously discussed, the inclusion of the time-to-go embedding provides a theoretical guarantee that the value function will converge due to the dynamic programming procedure for updating the value function. Additionally, we use domain randomization by terminating the episodes during training, resulting in a random maximum episode length T chosen uniformly at random $T \in [T_{min}, T_{max}]$. The purpose of this domain randomization is to provide the agent with a greater number of combinations of time-unaware observations and time-embeddings, thus potentially reducing the likelihood of sampling out-of-combination during deployment. This is crucially important as our proposed method relies on the capability of the policy network to generalize across combinations of time-unaware observations and the time-to-go embedding.

3.3.3 Deployment Procedure with Time-to-Go Embedding

During the evaluation stage, as we evaluate with episode length $T_{test} \gg T_{train}$, we consider several choices for the time-embedding at time t: (1) $\varphi(t, T_{test})$, (2) $\varphi(0, C)$, and (3) $\varphi(t \mod C, C)$, where C is a positive constant integer. (1) denotes no difference between the train and test settings (but just using T_{test} to denote the episode length during the evaluation), while (2) $\varphi(0, C)$ denotes our fixed time-embedding with parameter C and (3) $\varphi(t \mod C, C)$ denotes a time embedding at time t with periodicity C. Intuitively, one can interpret $\varphi(0, C)$ as the time-embedding for an observation with C timesteps remaining in an episode.

3.4 Results

We evaluate our experiments across six MuJoCo suite tasks: Ant-v3, HalfCheetah-v3, Hopper-v3, Walker2d-v3, Swimmer-v3, and Humanoid-v3. In these experiments, we make use of the previously mentioned finetuned hyperparameters [15] for the MuJoCo suite and use the Stable-Baselines3 implementation of PPO [16] for training the RL policy. Unless otherwise mentioned, the evaluated policies use these default hyperparameters.

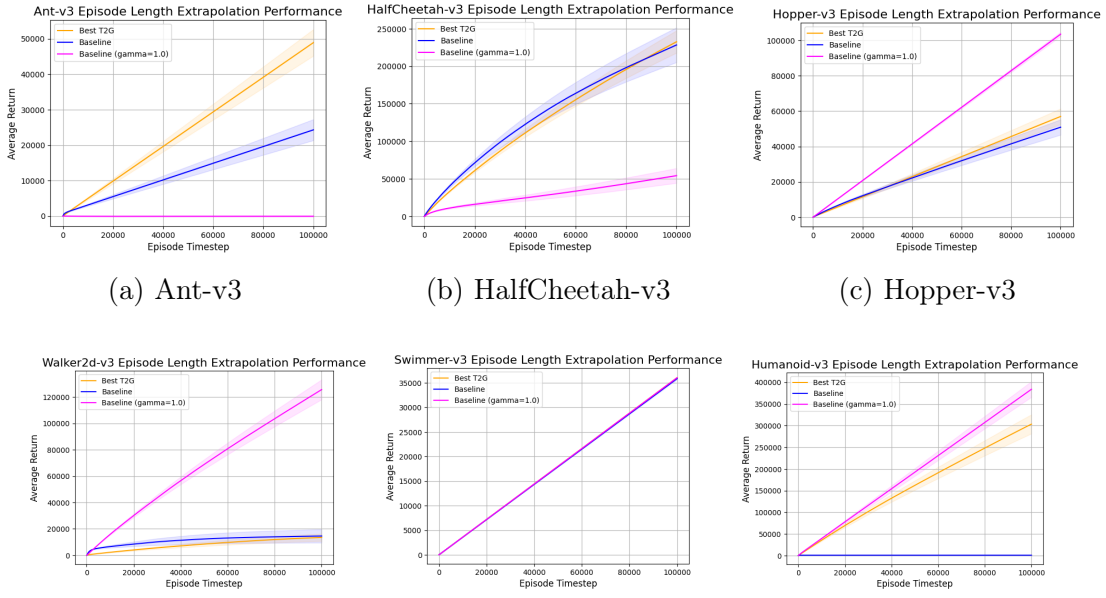
We evaluate two baselines: (1) PPO using the default finetuned γ and (2) PPO with $\gamma = 1$. In both of these two baselines, we set the maximum episode length during training as 1000 steps. Additionally, we train our time-embedding methods by randomly sampling the episode length from $T_{train} \in [200, 1000]$. During the evaluation, we evaluate each of these methods on $T_{test} = 100,000$ steps.

For our methods, we consider $\varphi(t, T_{test})$ (i.e., no change in the time-to-go embedding from training to testing), $\varphi(0, C)$ (i.e., the constant time-to-go embedding), and $\varphi(t \mod C, C)$ (i.e., using a periodic time-to-go embedding). The choices for the constant time-to-go embedding were varied from $C \in [0, 100, 200, \ldots, 1000]$, as these time-embeddings are in the training distribution of time-to-go embeddings.

Figure 3-2 illustrates the average evaluation returns across 5 train seeds and 100 evaluation seeds for the two baselines, as well as our best-performing policy using a constant time-to-go embedding.

3.5 Discussion

In this work, we explored the use of time-to-go embeddings as part of the observations and its capacity to extrapolate behavior well beyond the training episode length. In the evaluation across the six tasks in the MuJoCo suite, the results were inconclusive. In particular, in some tasks (Hopper-v3, Walker2d-v3 and Humanoid-v3), the undiscounted baseline performs best, while in others (Ant-v3 and HalfCheetah-v3) our method performs best. However, across all six tasks, our method either matches



(a) Ant-v3 (b) HalfCheetah-v3 (c) Hopper-v3

(d) Walker2d-v3 (e) Swimmer-v3 (f) Humanoid-v3

Figure 3-2: This figure illustrates the average evaluation episode returns with and without time-to-go observations.

or outperforms the performance of the discounted baseline that uses the tuned hyperparameters.

3.5.1 Limited Practical Benefits of Extrapolation in Cyclical Tasks

As we discussed, a common failure mode of typical RL is the treatment of cyclical tasks as episodic. This leads to unstable behavior towards the end of the episode, as a robot incentivized to lurch forward may not receive any penalties for its unstable behavior. In particular, the unstable behavior often occurs at the end of an episode (i.e., $t = T$) with a high positive reward. This means that the bias in the value function will lead to action sequences that render the agent unstable at the end of the episode.

In this work, we hypothesized that our time-to-go embedding, as well as the training and deployment procedures help mitigate this issue. By introducing a time-to-go embedding, the policy may learn that the unstable behavior is correlated with

the time embedding $\varphi(T, T)$, which denotes 0 timesteps remaining in the episode of length T. During deployment, we aimed to avoid this dilemma by choosing a constant time-to-go embedding. In doing so, the policy may predict actions as though there were $C > 0$ timesteps remaining at any given time.

However, our current results do not strongly support this hypothesis. We believe that the current approach of simply concatenating the observation and time-to-go embedding may not lead to generalization across inputs of the time-to-go embedding for the policy. Instead, future work may involve a different architecture for the policy and value networks for better generalization across time-to-go inputs.

3.5.2 Future Experiments

Beyond the need to illustrate the benefits of our approach across the tasks in the MuJoCo locomotion suite, additional experiments are necessary. To test on similar environments, future experiments may involve testing on the DeepMind dm_control software stack for physics-based simulation and RL environments [21]. For understanding whether our proposed method may help in other nonepisodic tasks, such as goal-reaching, we may test on other environments, such as OpenAI's ShadowHand dextrous object manipulation environment [12].

3.6 Limitations

Aside from the limitations in performance, one limitation of our proposed time-to-go embedding is that our method does not prescribe an optimal constant time-to-go embedding during evaluation for extrapolation. Instead, we require a cross-validation procedure to select the best-performing time-to-go embedding during evaluation. However, this need for cross-validation could be perceived as advantageous over the need for training across various choices of discount factors and selecting the one with the highest episode returns.

Another limitation is that, despite the objective of extrapolating in nonepisodic tasks, it is unclear whether our proposed methods may have benefits in goal-reaching

tasks, such as object reorientation and dextrous manipulation. This, as described, will require additional future experimentation.

Chapter 4

Conclusion

In this thesis, we have examined two different sets of non-episodic tasks and the failures of current RL policies, which are trained in an episodic setting.

In object manipulation, the first of these two sets of non-episodic tasks, we found a common failure mode that led to failures in setting of the Habitat Home Assistant Benchmark [18], due its solutions' open-loop task planner. Despite considering RL policy objectives that are more conducive to stabilizing behavior, we found limited success in such RL policies. Instead, it was the combination of an RL policy followed by the execution of an error corrective inverse-kinematics policy that resulted in more stable object placements in the Habitat HAB. Such solutions lead one to consider the potential for better integration of joint space planning in RL with the Cartesian space planning in inverse-kinematics, combining the advantages of these two fields.

In locomotion, we analyze the failures of extrapolating the behavior of an RL policy trained in an episodic setting. Oftentimes, such policies fail due to the limitations in the value estimation during the final steps in an episode, leading to unstable behavior and, thus, failure in locomotion tasks during an extrapolation beyond the training horizon. To address the problem of extrapolation, we considered the undiscounted finite-horizon setting and sought an approach that could continuously output actions from a policy optimized for finite-horizon. We proposed one such approach, though without theoretical guarantees, that considered the use of a time-to-go embedding in the observation input. We introduced the use of a constant time-to-go embedding

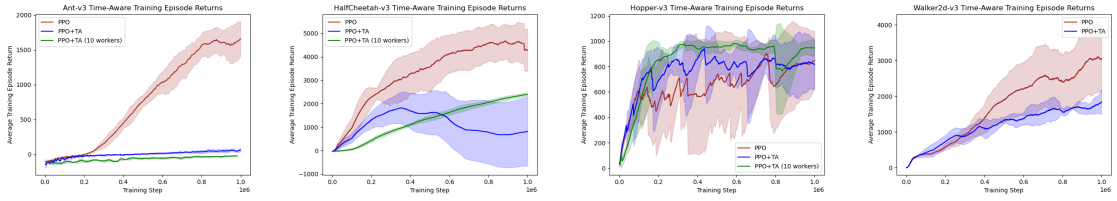
during evaluation, and found inconclusive results and limited benefits over the baseline RL policy trained using PPO in the MuJoCo suite. This finding leads to various future considerations, including the need for more robust testing across a wide variety of tasks and an analytical explanation for the utility of a constant time-aware embedding during deployment.

Appendix A

Additional Plots: Extrapolation in Locomotion

A.1 Reproducibility of Linear Time-Aware Baseline

Figure A-1 illustrates the average training episode returns across 5 seeds. Here, we illustrate the results across 3 methods: (1) PPO, (2) PPO with time-aware observations, and (3) PPO with time-aware observations and across 10 workers. Method (3) was the baseline that performed best in the prior work [13] but could not be reproduced to match that result.

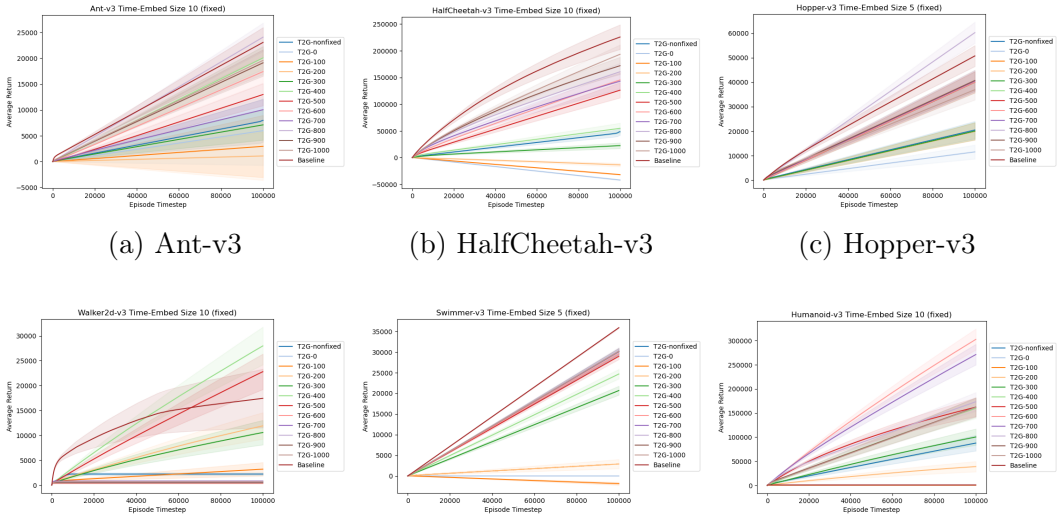


(a) Ant-v3 (b) HalfCheetah-v3 (c) Hopper-v3 (d) Walker2d-v3

Figure A-1: This figure illustrates the training episode returns with and without time-aware observations.

A.2 Plots for Time-Aware Embedding Sizes 5 and 10

Let d be the dimension of the observation space. Here, we choose the time-aware embedding size to be 10 if $10 < 2d/3$ and 5 otherwise. In this case, we set the size as 5 for Hopper-v3 and 10 otherwise.

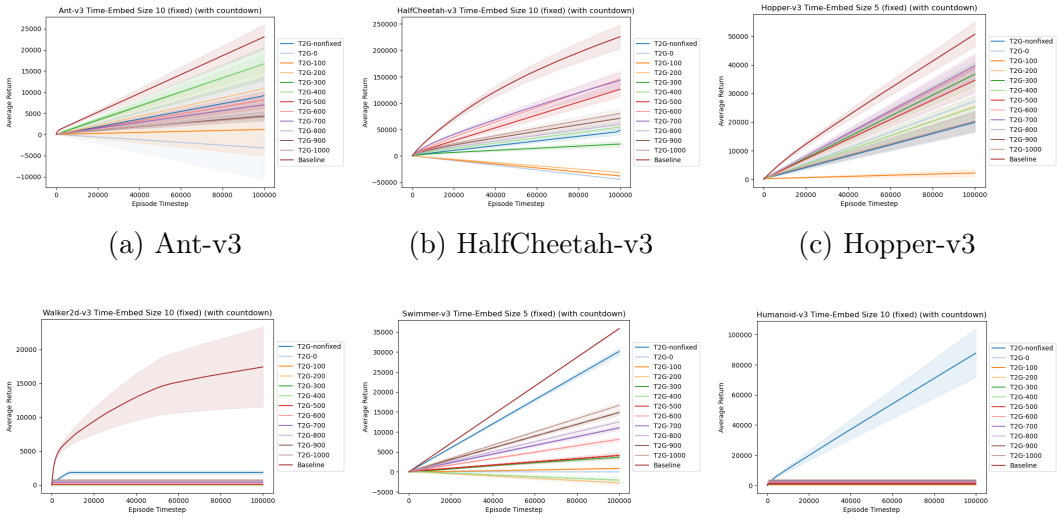


(a) Ant-v3 (b) HalfCheetah-v3 (c) Hopper-v3

(d) Walker2d-v3 (e) Swimmer-v3 (f) Humanoid-v3

Figure A-2: This figure illustrates the average evaluation performance across 5 train seeds and 100 evaluation seeds per train seed. The x-axis represents the timestep during evaluation with $T_{test} = 100,000$, and the y-axis represents the average cumulative returns. During the evaluation, we set the time embedding as $\varphi(0, C)$ for $C \in \{0, 100, 200, \ldots, 1000\}$.

Figure A-2 illustrates the results when the time embedding is $\varphi(0, C)$, while Figure A-3 illustrates the results when the time embedding is $\varphi(t \mod C, C)$.



(a) Ant-v3 (b) HalfCheetah-v3 (c) Hopper-v3

(d) Walker2d-v3 (e) Swimmer-v3 (f) Humanoid-v3

Figure A-3: This figure illustrates the average evaluation performance across 5 train seeds and 100 evaluation seeds per train seed. The x-axis represents the timestep during evaluation with $T_{test} = 100,000$, and the y-axis represents the average cumulative returns. During the evaluation, we set the time embedding as $\varphi(t \mod C, C)$ for $C \in \{0, 100, 200, \ldots, 1000\}$.

Bibliography

[1] Fetch robotics. https://fetchrobotics.com/.

[2] Dimitri Bertsekas. *Dynamic programming and optimal control: Volume I*, volume 4. Athena scientific, 2012.

[3] Dimitri P Bertsekas. Value and policy iterations in optimal control and adaptive dynamic programming. *IEEE transactions on neural networks and learning systems*, 28(3):500–509, 2015.

[4] Richard E Fikes and Nils J Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4):189–208, 1971.

[5] Pete Florence, Corey Lynch, Andy Zeng, Oscar A Ramirez, Ayzaan Wahid, Laura Downs, Adrian Wong, Johnny Lee, Igor Mordatch, and Jonathan Tompson. Implicit behavioral cloning. In *Conference on Robot Learning*, pages 158–168. PMLR, 2022.

[6] Aditya Ganapathi, Pete Florence, Jake Varley, Kaylee Burns, Ken Goldberg, and Andy Zeng. Implicit kinematic policies: Unifying joint and cartesian action spaces in end-to-end robot learning. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 2656–2662. IEEE, 2022.

[7] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021.

[8] Jiayuan Gu, Devendra Singh Chaplot, Hao Su, and Jitendra Malik. Multi-skill mobile manipulation for object rearrangement. *arXiv preprint arXiv:2209.02778*, 2022.

[9] Nan Jiang, Alex Kulesza, Satinder Singh, and Richard Lewis. The dependence of effective planning horizon on model accuracy. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, pages 1181–1189, 2015.

[10] Roberto Martín-Martín, Michelle A Lee, Rachel Gardner, Silvio Savarese, Jeannette Bohg, and Animesh Garg. Variable impedance control in end-effector space: An action space for reinforcement learning in contact-rich tasks. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1010–1017. IEEE, 2019.

[11] Toki Migimatsu and Jeannette Bohg. Object-centric task and motion planning in dynamic environments. *IEEE Robotics and Automation Letters*, 5(2):844–851, 2020.

[12] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafał Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *CoRR*, 2018.

[13] Fabio Pardo, Arash Tavakoli, Vitaly Levdik, and Petar Kormushev. Time limits in reinforcement learning. In *International Conference on Machine Learning*, pages 4045–4054. PMLR, 2018.

[14] Fabio Pardo, Arash Tavakoli, Vitaly Levdik, and Petar Kormushev. Time limits in reinforcement learning, 2018.

[15] Antonin Raffin. Rl baselines3 zoo. https://github.com/DLR-RM/rl-baselines3-zoo, 2020.

[16] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021.

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

[18] Andrew Szot, Alexander Clegg, Eric Undersander, Erik Wijmans, Yili Zhao, John Turner, Noah Maestre, Mustafa Mukadam, Devendra Singh Chaplot, Oleksandr Maksymets, et al. Habitat 2.0: Training home assistants to rearrange their habitat. *Advances in Neural Information Processing Systems*, 34:251–266, 2021.

[19] Yunhao Tang, Mark Rowland, Rémi Munos, and Michal Valko. Taylor expansion of discount factors. In *International Conference on Machine Learning*, pages 10130–10140. PMLR, 2021.

[20] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 5026–5033. IEEE, 2012.

[21] Saran Tunyasuvunakool, Alistair Muldal, Yotam Doron, Siqi Liu, Steven Bohez, Josh Merel, Tom Erez, Timothy Lillicrap, Nicolas Heess, and Yuval Tassa.

dm_control: Software and tasks for continuous control. *Software Impacts*, 6:100022, 2020.

[22] Patrick Varin, Lev Grossman, and Scott Kuindersma. A comparison of action spaces for learning manipulation tasks. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6015–6021. IEEE, 2019.