SAFE AND ROBUST REINFORCEMENT LEARNING FOR AUTONOMOUS CYBER-PHYSICAL SYSTEMS

By

Nathaniel Hamilton

Dissertation
Submitted to the Faculty of the
Graduate School of Vanderbilt University
in partial fulfillment of the requirements
for the degree of

DOCTOR OF PHILOSOPHY
in
ELECTRICAL ENGINEERING

August 12, 2022

Nashville, Tennessee

Dedicated to my wife, family, and friends. Thank you so much for all the support!

ACKNOWLEDGMENTS

This dissertation would not have been possible without the help, support, and encouragement of so many people.

First, I want to extend a large amount of gratitude to my advisor, Dr. Taylor Johnson. Thank you for all the guidance, support, and mentorship these last 5 years. Thank you to Dr. Jonathan Sprinkle: you inspired and encouraged me to pursue my PhD during the REU, and I am so glad you were involved in its completion. Thank you to Dr. Kerianne Hobbs: working with you and the Safe Autonomy Team these last few years has been a joy that revitalized my passion for the research we are doing. To the rest of my committee, thank you for taking time to review and critique my work. It has been made better because of you.

I have been so fortunate to have such a great cohort of collaborators and friends in the Verification and Validation of Intelligent and Trustworthy Autonomy Lab (VeriVITAL). Patrick Musau and Diego Manzanas Lopez, your insight, encouragement, and support over the last couple of years has been invaluable. Thank you so much for the good times at game nights, happy hours, and whiteboard discussions. I won't miss the late nights scrambling to get papers finished just before the deadline, but I wouldn't have been able to do all this without your help. Good luck in the next steps of your careers, and don't become strangers! To the others in the VeriVITAL crew, Ayana Wild, Neelanjana Pal, Preston Robinette, and Xiadong Yang, I regret not getting to work with you all more during this journey. You're all so brilliant, and I hope we get to collaborate on projects in the future or catch up at conferences.

To my Lipscomb friends, and the friends I've made since starting at Vanderbilt, thank you for making this journey enjoyable. In particular, thank you to Patrick Russel and Zach Zeiger. Our years living together are a cherished memory. I wish you all the best in your future careers and life pursuits.

To my family: Mom, Dad, Luke, Jared, and Isaac, thank you for the consistent support and encouragement throughout my life. I could not have made it to this point without all of you. Thank you for the countless hours listening to me talk about my research and asking questions for clarifications. The interest you all have shown has been invaluable for staying motivated to reach this point. Thank you!

Last, I would like to thank my wife Kendall for all her support through all the highs and lows. Together we survived completing this dissertation while also facing the pandemic. Kendall, thank you so much for all you did to keep me alive and healthy through all this. Your support was (and still is) invaluable. Thank you so much for choosing to stick with me. We did it!

Acknowledgement of Support

This material is based upon work supported by the Air Force Office of Scientific Research (AFOSR) under award number FA9550-22-1-0019, the National Science Foundation (NSF) under grant numbers 1918450, 1910017, and 2028001, the Department of Defense (DoD) through the National Defense Science & Engineering Graduate (NDSEG) Fellowship Program, and the Defense Advanced Research Projects Agency (DARPA) Assured Autonomy program through contract number FA8750-18-C-0089. Any opinions, finding, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force, DARPA, nor NSF.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
ARS	Augmented Random Search
AV	Autonomous Vehicle
BHNR	Bounded Horizon Nominal Robustness
CBF	Control Barrier Function
CPS	Cyber-Physical System
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q-Learning
DRL	Deep Reinforcement Learning
FSA	Finite State Automata
IL	Imitation Learning
IMU	Inertial Measurement Unit
LiDAR	Light Detection and Ranging
MDP	Markov Decision Process
NDSEG	National Defense Science and Engineering Graduate Fellowship
NFQ	Neural Fitted Q-iteration
NNC	Neural Network Controller
NSA	Neural Simplex Architecture
OMDP	Observed Markov Decision Process
POMDP	Partially Observable Markov Decision Process
PPO	Proximal Policy Optimization
REPS	Relative Entropy Policy Search
RL	Reinforcement Learning (can be used for DRL)
ROS	Robot Operating System
RTA	Run Time Assurance
SAC	Soft Actor-Critic
SRL	Safe Reinforcement Learning
STL	Signal Temporal Logic
$TD3$	Twin Delayed Deep Deterministic Policy Gradient
TLTL	Truncated Linear Temporal Logic

CHAPTER I

Introduction

I.1 Motivation

Cyber-Physical Systems (CPS) is a classification of intelligent systems where a physical mechanism is controlled or monitored using computer-based (i.e. cyber) algorithms. CPS includes systems such as smart power grids, autonomous vehicles, medical monitoring, autopilot, and robotics. As CPS start to become more integrated with everyday life, we want to push the limits of their capabilities and automate their use. To accomplish this desire, we look to *Artificial Intelligence* (AI), and more specifically, *Machine Learning* (ML). AI is a field of study where we develop computer systems that are able to complete tasks that require human- or at least animal-level intelligence, i.e. visual perception, decision-making, and complex control. ML is a subset of AI where we study methods and techniques that allow computer systems to learn and improve their performance at completing tasks.

Recent successes have brought *Reinforcement Learning* (RL) to the forefront of the AI and ML discussion, such as *SonyAI*'s world-champion racer in Gran Turismo [6], *OpenAI Five*'s defeat of professional-level players in the game Dota 2 [7], and *MuZero*'s mastery of Go, Chess, Shogi, and numerous Atari games with a single learned policy [8]. Additional research consistently shows RL training produces optimal results even when trained with inaccurate or incomplete models [9]. Instead of hard-coding, agents are programmed via reward and punishment without needing to specify how the task is completed. Because of this, RL techniques are extremely attractive for robotics and other cyber-physical systems applications, which have complex dynamics and environments that are difficult to model. However, the behaviors of these RL systems are often difficult to interpret and predict. A simple example of this can be seen in teaching hexapod robots to walk using RL. During the unbounded exploration used for training, some applied policies can cause legs to cross, get caught on each other, and pull apart, causing one or more legs to break. Similar scenarios and situations can occur in any system trained using RL. This is unacceptable for safety-critical systems, where unpredictable and unsafe behavior could be the difference between life and death.

To protect systems, and speed up the training process, engineers can train their RL agents in simulation and then transfer the learned control policy to the corresponding real-world system. This process is a challenging problem referred to as the *sim2real* challenge. Simulation environments abstract away a lot of the nuance and noise of the real world. As a result, RL policies trained in simulation can end up "brittle", i.e., when confronted with scenarios that differ from the examples seen in training they can fail to contextualize the situation and break. Thus, *sim2real* transfers often have catastrophic results where the real world results are drastically different from simulation [10, 11]. Therefore, if we are to take advantage of the many benefits of training in simulation, we must emphasize training more robust policies and/or formally verify their behavior before deploying on safety-critical, real-world systems.

To this end, the work completed in this dissertation builds towards the goal of making safe and robust reinforcement learning for autonomous cyber-physical systems.

I.2 Research Challenges

One field of research aimed towards making RL-trained agents safer during and after training is *Safe Reinforcement Learning* SRL. Recent SRL works demonstrate real-world online learning [9], optimal performance that does not require safety checking when deployed [12], and SRL approaches that work better than state-of-the-art DRL approaches [13]. Each new SRL paper claims to be the best, safest, most efficient, or least restrictive approach, but few prove these claims with valid demonstrations. We have tried to replicate studies with the original code, and found issues in many of their comparisons of SRL to other RL approaches. (1) Some results were invalid SRL approaches because *unsafe conditions were not used as terminal conditions*. (2) In some cases, *inconsistent hyperparameters* were used between the safe and unsafe experiments, which means the improved efficiency authors claimed as a result of their SRL approach might actually be the result of hyperparameter tuning. (3) Often, the experiments are *not repeated across multiple random seeds*. Because RL is a stochastic process, showing results from one random seed is not representative of the true performance of the algorithm. Only presenting the results of trials across one seed, allows for results to be cherry-picked from the best trial. The work in [14] highlights the importance of running experiments across at least 5 random seeds and averaging the results and showing the performance range in order to prove the trend of increased efficiency. (4) In some experiments we repeated, we found that authors *manipulated initial conditions* to improve efficiency. For example, inverted pendulum experiments that claimed increased efficiency as a result of an SRL approach could be explained by starting SRL trials closer to a vertical position than baseline trials. When we applied the same initial conditions to the baseline approach and SRL approaches, we found cases where the SRL approach learned slower than the baseline, disproving their claims.

Spurred by the inconsistencies and the desire to justify further research in SRL research, this dissertation sets out to answer these crucial questions:

- Why should we consider RL for *sim2real* scenarios?
- How can we best incorporate safety in the RL process?
- How can we assure that learning safety produces safer results?

- How can we integrate safety specifications in the reward designing/shaping process?

I.3 Research Contributions

Motivated by the existing challenges, the research contributions are presented and discussed in detail in Chapters III to VI, and can be summarized as follows:

1. We begin by performing experiments in an original transfer reinforcement learning task and evaluating two transfer learning techniques from the literature.

2. We then present a comparative analysis of two leading machine learning methods for training autonomous controller, Reinforcement Learning and Imitation Learning. This case study was evaluated on a 1/10th scale car racing scenario and helps provide justification for why we should be considering RL for sim2real scenarios.

3. Having demonstrated RL's usefulness in sim2real and demonstrating a need to integrate safety considerations, we present an ablation study where we analyze different approaches for integrating safety. Our results provide answers for a number of crucial questions in the field and point towards a greater need for developing reward functions that encourage the desired safe behavior.

4. Finally, we present our tool, STLGym, which automatically generates reward functions designed to help RL agents learn to satisfy both timed and untimed specifications. With this tool, researchers can write up how they want the RL agent to behave in the environment, and our tool will generate a reward function to help the RL agent match the desired behavior.

I.4 Organization

The remainder of this dissertation is organized as follows. Chapter II introduces the respective related work, providing background information for the core topics discussed throughout the dissertation. Chapter III introduces methods for transferring trained RL policies between tasks with different dynamics. Chapter IV focuses on a comparative analysis of two leading machine learning methods for training autonomous controllers, Reinforcement Learning and Imitation Learning. This case study was evaluated on a 1/10th scale car racing scenario and helps provide justification for why we should be considering RL for sim2real scenarios. Chapter V presents our ablation study of various Safe Reinforcement Learning approaches and how each safety enforcing modification impacts the learning process and whether they make RL agents safer after all the training is done. Our results provide answers for a number of crucial questions in the field and point towards a greater need for developing reward functions that encourage the desired safe behavior. Chapter VI introduces our tool for helping RL agents learn desired behavior, STLGym. STLGym generates reward functions designed to help RL agents learn to satisfy the desired behavior, written in the form of a Signal Temporal Logic (STL) specification. Chapter VII ends with concluding remarks, followed by the author's lists of publications and presentations in Chapters VIII and IX, respectively.

I.5 Copyright Acknowledgments

We include the following required copyright statements for permission to reprint portions of [15] and [16]

- For portions of [15] reproduced in this dissertation, we acknowledge the Society of Photo-Optical Instrumentation Engineers (SPIE) copyright : ©2020 COPYRIGHT SPIE. Reprinted with permission from Nathaniel Hamilton, Lena Schlemmer, Christopher Menart, Chad Waddington, Todd Jenkins, and Taylor T. Johnson, "Sonic to knuckles: evaluations on transfer reinforcement learning," in *Proceedings Volume 11425, Unmanned Systems Technology XXII; 114250J* (March 2020), https://doi.org/10.1117/12.2559546.

- For portions of [16] reproduced in this dissertation, we acknowledge the IEEE copyright: ©2022 IEEE. Reprinted with permission from Nathaniel Hamilton, Patrick Musau, Diego Manzanas Lopez, and Taylor T. Johnson, "Zero-Shot Policy Transfer in Autonomous Racing: Reinforcement Learning vs Imitation Learning," in *Proceedings of the 1st IEEE International Conference on Assured Autonomy (ICAA 2022)*, Fajardo, Puerto Rico, USA, Mar. 2022, pp. 11-20.

CHAPTER II

Preliminaries and Related Work

In this chapter, we describe the related works that are foundational topics to the work presented in this dissertation. Chief among these foundational topics is describing the problem formulation for model-free reinforcement learning, which is the primary focus of this work.

II.1 Problem Formulation

Reinforcement Learning is designed to solve problems modeled as *Markov Decision Processes* (MDPs), or a variant of the original MDP formulation.

Definition 1 (Markov Decision Process)**.** *A discrete-time Markov Decision Process (MDP) is represented by a tuple* $(\mathbf{S}, \mathbf{U}, \mathbf{T}, \gamma, \mathbf{R})$*, where*

- $\mathbf{S}$ *is a set of states called the* state space*,*
- $\mathbf{U}$ *is a set of actions called the* action space *(alternatively,* $\mathbf{U}_s$ *is the set of actions available from state* s*),*
- $\mathbf{T}(s,u,s') = \Pr(s_{t+1} = s' \mid s_t = s, u_t = u)$ *is the probability that action* u *in state* s *at time* t *will lead to state* s' *at time* $t+1$*,*
- $\gamma \in [0,1]$ *is the discount factor determining preference for immediate or distant reward assignments, and*
- $\mathbf{R}(s,u,s') : \mathbf{S} \times \mathbf{U} \to \mathbb{R}$ *is the reward function assigning a scalar reward value to the transition from state* s *to state* s' *using action* u*.*

The solution to an MDP is the optimal behavior function, $u = \pi^*(s)$*, that maximizes the expected cumulative sum of rewards, i.e. return,* J*:*

$$\mathbb{E}[J] = \mathbb{E}[\sum_{t=0}^{T} \gamma^t r_t], \tag{II.1}$$

where T *is a finite time-horizon. When* $\gamma = 0$*, the solution prioritizes immediate reward, and when* $\gamma = 1$ *the solution prioritizes maximizing the expected sum of future rewards.*

Formulating the problem as an MDP assumes that the state space is fully observable. While this might be true for some simulated systems, many real systems are only able to estimate some state information using sensors. Because of this, only a portion of the state space may be observable. To handle these systems, we need to account for *Partially Observable Markov Decision Processes* (POMDPs).

Definition 2 (Partially Observable Markov Decision Process)**.** *A discrete-time POMDP extends the MDP to cases where the state may not be fully observable to the agent. Formally, a POMDP is represented by a 7-tuple,* $(\mathbf{S}, \mathbf{U}, \mathbf{T}, \gamma, \mathbf{R}, \Omega, \mathbf{O})$*, which builds off the original MDP tuple by adding additional information, where*

- $\mathbf{O}$ *is a set of observations, and*
- $\Omega(s,o) : \mathbf{S} \rightarrow \mathbf{O}$ *is a probabilistic mapping of states* $s \in \mathbf{S}$ *to corresponding observations* $o \in \mathbf{O}$.

In this formulation, the behavior function maps observations, $o \in \mathbf{O}$*, to actions* $u \in \mathbf{U}$*. Thus, the optimal behavior function is* $u = \pi^*(o)$.

All model-free RL algorithms are designed to solve the POMDP formulation problem because any MDP can be reformulated as a POMDP by setting the observation set to be equal to the set of states, $\mathbf{O} = \mathbf{S}$, and defining the observation conditional probabilities to always match the observation that corresponds to the true state.

II.1.1 Observed Markov Decision Process

Model-free RL algorithms are traditionally designed to solve *Partially Observable Markov Decision Processes* (POMDPs) since all *Markov Decision Processes* (MDPs) can be described by an equivalent POMDP. While POMDP is a very useful class of systems, it includes cases where some values are unobservable. Having unobservable state values is not an issue for model-free RL, but it is an issue if there are certain state values that need to be monitored to determine the safety of the system. To this end, we propose a new MDP formulation we refer to as *Observed Markov Decision Process* (OMDP). This formulation is a subset of POMDP that requires all the values we want to keep track of are measurable. This formulation does not require that all measurable state values are included in the input observation.

Using the observation as the input for the policy is really useful because it allows us to normalize the state values and change the input dimensions in order to ignore irrelevant variables and/or increase the importance of other variables by repeating them. For example, consider the example of an aircraft doing waypoint navigation. In this example, the observation can be comprised of the relative position between the aircraft and the destination. If global position, the state, was used instead for the input, the learned policy would not transfer well for even small changes in location. In this case, the global position is necessary for computing the transition, but consciously ignored by the policy. Now add to the example a restriction on the system in the form of a geofence[1] the aircraft cannot leave. In order to monitor this restriction, we need access to the global position. By formulating the problem as an OMDP, we ensure the global position is accessible for such monitoring.

Definition 3 (Observed Markov Decision Process)**.** *A finite OMDP is a tuple* $(\mathbf{S}, \mathbf{U}, \mathbf{T}, \mathbf{R}, \mathbf{O}, \Omega, \mathbf{D})$ *where*

- $\mathbf{S}$ *is a set of states called the* state space*,*
- $\mathbf{U}$ *is a set of actions called the* action space *(alternatively,* $\mathbf{U}_s$ *is the set of actions available from state* s*),*
- $\mathbf{T}(s,u,s') = \Pr(s_{t+1} = s' \mid s_t = s, u_t = u)$ *is the probability that action* u *in state* s *at time* t *will lead to state* s' *at time* $t+1$*,*
- $\mathbf{R}(s,u,s') : \mathbf{S} \times \mathbf{U} \to \mathbb{R}$ *is the reward function assigning a scalar reward value to the transition from state* s *to state* s' *using action* u*,*
- $\mathbf{O}$ *is a set of observations,*
- $\Omega(s,o) : \mathbf{S} \to \mathbf{O}$ *is a mapping of states* $s \in \mathbf{S}$ *to corresponding observations* $o \in \mathbf{O}$*, and*
- $\mathbf{D}$ *is a definition of terminal conditions that indicate what is "done".*

Throughout this dissertation, we assume the RL problem can be represented by an OMDP. This allows us to cover a broader class of problems than those represented by an MDP, but restricts our problems to those where the state and observation values used for determining whether the system is "safe" are observable even if they are ignored by the learning agent.

II.2 Deep Reinforcement Learning

Reinforcement Learning (RL) is a form of machine learning in which an agent acts in an environment, learns through experience, and increases its performance based on rewarded behavior. *Deep Reinforcement Learning* (DRL) is a newer branch of RL in which a neural network is used to approximate the behavior function, i.e. policy π. The basic construction of the DRL approach is shown in Figure II.1. The agent consists of the *Neural Network Controller* (NNC) and RL algorithm, and the environment consists of a plant and observer model. The environment can be comprised of any dynamical system, from Atari simulations ([15, 17]) to complex robotics scenarios ([5, 9, 14, 18, 10, 19]).

Reinforcement learning is based on the *reward hypothesis* that all goals can be described by the maximization of expected return, i.e. the cumulative reward [20]. During training, the agent chooses an action, u_t,

[1]A geofence is a perimeter drawn up around a real-world geographical area.

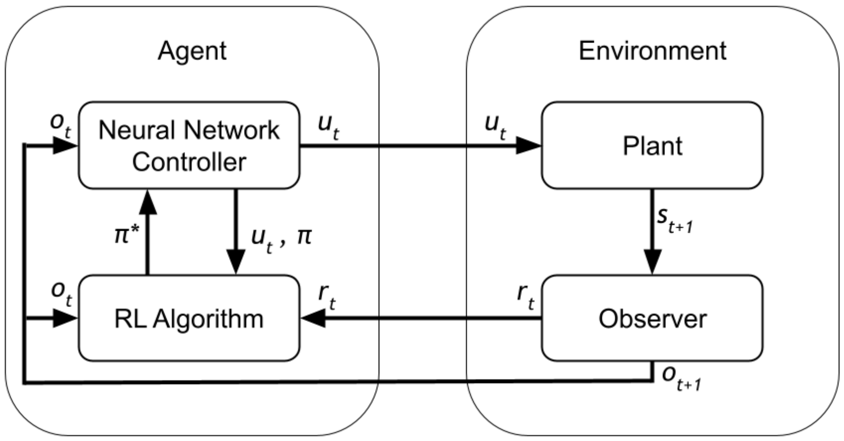


Figure II.1: Basic formulation of Deep Reinforcement Learning for a control problem.

based on the input observation, o. The action is then executed in the environment, which causes the environment to transition to state s_{t+1} with probability $\mathbf{T}(s_t, u_t, s_{t+1})$ according to the plant model. However, instead of receiving the state information, the agent receives an observation $o_{t+1} \in \mathbf{O}$ which depends on the new state of the environment, s_{t+1}, and on the just taken action, u_t, with probability $\Omega(o_{t+1} \mid s_{t+1}, u_t)$. Finally, the agent receives a reward r_t equal to $\mathbf{R}(s_t, u_t, s_{t+1})$.

In all the examples shown in this dissertation, we assume the problem is represented as an OMDP, meaning all important state values are measurable even if they are ignored by the agent. The process of executing an action and receiving a reward and next observation is referred to as a *timestep*. Relevant values, like the input observation, action, and reward are collected as a data tuple, i.e. *sample*, by the RL algorithm to update the current NNC policy, π, to an improved policy, π^*. How often these updates are done is dependent on the RL algorithm.

When comparing RL algorithms and approaches or determining how successful they are, we look at two metrics: (1) the final policy performance, usually measured by the expected return; and (2) the sample complexity, a measure of how fast the agent learns the optimal policy.

The return is the sum of all rewards collected over the course of an *episode*. An episode is a finite sequence of states, observations, actions, and rewards starting from an initial state and ending when some terminal, i.e. *done*, conditions are met. Throughout this dissertation, we refer to different elements of the episode by their corresponding timestep, t. Thus, r_t is the reward value at timestep $t \in [0, T]$, where T is the final timestep in the episode.

The sample complexity is a measure adapted from general machine learning research that represents the number of training-samples (i.e. timesteps in RL) needed to achieve a certain level of performance. The larger the sample complexity is, the more data is required to train the agent to a certain level of performance. Since RL collects data throughout the training process, sample complexity is a proxy for measuring how quickly an agent learns to complete the task. Throughout the training process, the training is halted periodically, and the policy learned so far is evaluated in a number of episodes and the return is recorded. When this information is plotted in a sample complexity plot, it shows how well the agent improves performance as it continues to learn. These plots are used in RL for comparing how well an agent learns a task. An agent with better the sample complexity, will have a higher return after fewer timesteps.

II.3 Deep Reinforcement Learning Algorithms

DRL algorithms are defined and categorized by how they use 3 distinct components and combinations of them. These components are:

- **Model**: a descriptor of the environment. With a model, we can learn or infer how the environment responds to different actions. The major components of a model are the transition probability function and reward function.
- **Policy**: the agent's behavior function, π, is a mapping from observation, o, to action, u, and can be deterministic, $\pi(o) = u$, or stochastic, $\pi(u|o) = Pr_{\pi}(u_t = u|o_t = o)$.
- **Value Function**: a measure of the "goodness" of a state, or how rewarding a state-action pair is based on the expected return computed with discount factor $\gamma \in [0,1]$: $J_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$
  - **state value**: the expected return, J_t, if the agent is in state s at time t: $V_{\pi}(s) = \mathbb{E}_{\pi}[J_t|s_t = s]$
  - **state-action value**: also known as the Q-value is the expected return of a given state-action pair: $Q_{\pi}(s,u) = \mathbb{E}_{\pi}[J_t|s_t = s, u_t = u]$. This can be used to estimate the state-value using the target policy. $V_{\pi} = \sum_{u \in \mathbf{U}} Q_{\pi}(s,u)\pi(u|s)$
  - **advantage**: the difference between the state-action value and the state value. It is defined as:

$$\hat{A}_{\pi}(s,u) = Q_{\pi}(s,u) - V_{\pi}(s) \tag{II.2}$$

Most algorithms refer to how they use those components with the following labels and combinations of labels:

- **Model-based**: these methods rely on the model of the environment; either the model is known or the algorithm learns it explicitly. Having a model of the environment allows the agent to "think ahead" by predicting outcomes and choosing the best option.

- **Model-free**: these methods have no dependency on the model during learning.
- **On-policy**: during training, the learned policy selects every action taken. These methods usually update less often and rely on a stochastic version of the policy during training.
- **Off-policy**: during training, a separate behavior policy (often random or process noise) is used to select the actions taken. Thus, the learned policy is not used during training.

In this dissertation, we focus on model-free DRL algorithms, using both *on-policy* and *off-policy* approaches. Model-free approaches have the advantage of not requiring a ground-truth model of the environment, but are still able to learn an optimal policy. We use multiple algorithms throughout this dissertation, but our work favors using *Proximal Policy Optimization* (PPO) as our on-policy[2] algorithm, and *Soft Actor-Critic* (SAC) as our off-policy[3] algorithm.

Both PPO and SAC, along with many state-of-the-art DRL algorithms, build off the foundational actor-critic methodology. Actor-critic methods train two[4] networks, an actor and critic, concurrently. The actor network learns the optimal policy, π^*, and is the NNC that is eventually deployed after training. The critic network learns the value function, either the state value or the Q-value, depending on the algorithm. The critic's learned value function is used throughout training to determine the updates to the actor network.

II.3.1 Proximal Policy Optimization

Proximal Policy Optimization (PPO) was first introduced by Schulman et al. in [26] as an improvement to their previous DRL algorithm, *Trust Region Policy Optimization* (TRPO) [22]. Both algorithms are on-policy and focus on iteratively improving the policy in small increments to prevent against making large changes in the policy that might lead to a drop in performance. They differ in how the "small increment" is calculated. TRPO solves an optimization problem to identify the largest change to the policy that can be made while remaining within a "trustworthy" region. Solving this optimization problem is computationally expensive, so PPO simplifies the process using a clipping function. Instead of defining the trustworthy region based on the optimization problem, the clipping function sets a hard, constant limit. This simplification speeds up computation, and, according to the introductory paper, improves sample complexity. Later work, [27], showed that the improvements were actually a result of code-level improvements. Despite this finding, PPO has continued to grow and is regularly improved with new features.

[2]Other on-policy RL algorithms include A2C [21], TRPO [22], and ARS [18].

[3]Other off-policy RL algorithms include DQN [3], DDPG [23], and TD3 [24].

[4]Two networks is not always the case. In Chapter III, our actor and critic share the same network body, but the actor and critic have separate final layers. Furthermore, some DRL algorithms make use of multiple critic networks. These additional critics help provide a more stable update for the actor network. Some DRL algorithms that make use of multiple critics include SAC [25] and TD3 [24].

II.3.2 Soft Actor-Critic

Soft Actor-Critic (SAC) was first introduced by Haarnoja et al. in [25]. The algorithm is an improvement over other off-policy algorithms because the learning process is more stable. One of SAC's predecessors, *Deep Deterministic Policy Gradient* (DDPG) [23], has well-documented issues with stability caused by *catastrophic forgetting* [28]. Catastrophic forgetting occurs as a result of the agent improving the learned policy and no longer visiting less-desirable states. Eventually, the optimal action in these less-desirable or early-episode states are forgotten and the result of this forgetting shows up when the policy is evaluated, and we observe the performance plummeting instead of improving. The agent then has to re-learn what to do in those states, hurting the sample complexity. SAC addresses this issue by incorporating entropy maximization in the exploration policy, i.e. the policy used during training. Because off-policy algorithms use a separate policy during training (also referred to as exploration), the exploration policy can have a different goal from the learned optimal policy.

This entropy maximization prioritizes trying new state-action combinations during training in order to better approximate the true Q-function. In this way, the agent regularly revisits any state that has been 'forgotten' and prevents any catastrophic decline in performance.

II.4 Safe Reinforcement Learning

When an RL agent explores states in a video game, the consequences of making a "wrong" move are limited. However, using RL in the real world has shown catastrophic results [9, 10]. The field of *Safe Reinforcement Learning* (SRL) was developed in response to RL's use on cyber-physical systems domain that interact with the real world in complex scenarios. In García and Fernández's comprehensive survey of SRL from 2015, they categorized the approaches into two main categories or styles: (1) modification of the optimality criterion and (2) modification of the exploration process [29]. In this dissertation, we refer to these categories under the more general terms: (1) *reward shaping* and (2) *safe exploration*. Additionally, we introduce an emerging category of approaches, (3) *adversarial training/retraining*. Each are described in more detail in this section.

II.4.1 Safe Exploration

An approach geared more towards hardware deployment, *safe exploration* approaches ensure the agent remains 100% safe throughout the duration of training. Furthermore, this approach can be redesigned for deployment, ensuring the future safety of trained policies. Safe exploration techniques can be further broken down into the following three categories.

1. **Preemptive Shielding** where the action set the agent is allowed to choose from is preemptively reduced to only allow safe actions [17, 30].

2. **Safe-by-Construction** in which verification techniques are used, often applied to an abstraction of the learned policy, to verify safe behavior before being allowed to explore and develop further [31, 32, 33]. Alternatively, correct-by-construction can also be applied to a shielded RL solution [17].

3. **Run Time Assurance (RTA) methods** filter the agent's desired actions, u_{NN}, to assure safety. In some cases, a monitor and/or decision module is used to determine whether the desired action provided by the learning agent is safe. In the event the agent's desired action is deemed unsafe, a different action that is determined to be safe is substituted and sent to the plant [34, 12, 35, 36, 9, 37, 38, 39, 40, 41].

II.4.2 Reward Shaping

Reward shaping, the process of crafting a well-designed, optimal reward function, is essential for all forms of DRL since a poorly designed reward function can lead to unexpected and/or ineffective behavior [15]. Within SRL, reward shaping is often used to reformulate the problem as a *Constrained Markov Decision Processes* (CMDP) [42].

Definition 4 (Constrained Markov Decision Process)**.** *A* Constrained Markov Decision Process *(CMDP) is denoted by the tuple* $(\mathbf{S}, \mathbf{U}, \mathbf{T}, \gamma, \mathbf{R}, D)$*. This adds to the original MDP tuple* $D : \mathbf{S} \times \mathbf{U} \to \mathbb{R}$*, which is the cost function.*

The CMDP formulation behaves the same as the MDP formulation, except at each timestep a reward, $r_t = \mathbf{R}(s_t, u_t, s_{t+1})$*, and a cost,* $d_t = D(s_t, u_t, s_{t+1})$ *are computed. With this additional cost, the objective changes to find an optimal policy,* $u = \pi^*(s)$*, that not only maximizes the expected return according to Equation II.1, but also ensures the expected cumulative cost, K, remains below a specified value,* K_{max}*.*

$$\mathbb{E}[K] = \mathbb{E}[\sum_{t=0}^{T} \gamma^t d_t] \leq K_{max} \tag{II.3}$$

Instead of optimizing performance according to a singular reward function, performance is optimized according to a task-oriented reward and a safety-focused cost [43, 44, 45, 46, 13, 47], so the agent learns a high-performing, safe policy. However, this style of SRL does not prohibit the agent from exploring unsafe behavior. Thus, it cannot be used to train on real hardware platforms. Instead, reward shaping techniques are limited to simulated environments and rely on high-quality transfer learning to be deployed in the real world.

II.4.3 Adversarial Training/Retraining

The newest category of SRL approaches, *Adversarial Training/Retraining*, focuses on identifying unsafe behavior in the agent and then correcting that behavior [48, 49, 50]. Most of the papers that use this approach focus on retraining an agent that already performs well in the environment. However, the approach can also be applied to an untrained network at the cost of requiring more training time.

II.5 Run Time Assurance

One way to enforce the safety of a system while it is operating is through the use of *Run Time Assurance* (RTA). RTA approaches filter out potentially unsafe control inputs provided by a primary controller with the intent of preserving the safety of the system.

For the dynamical system represented by an OMDP, inequality constraints $\varphi_i(s) : \mathbb{R}^n \to \mathbb{R}, \forall i \in \{1,...,M\}$ can be used to define a set of M safety constraints, where the constraint is satisfied when $\varphi_i(s) \geq 0$. With these safety constraints, we can define safe operation by its relation to the *admissible set*.

Definition 5 (Admissible Set)**.** *The admissible set $\mathbf{S}_\varphi \subseteq \mathbf{S}$, is defined as the set of states where all safety constraints are satisfied. This is represented by,*

$$\mathbf{S}_\varphi := \{s \in \mathbf{S} \mid \varphi_i(s) \geq 0, \forall i \in \{1,...,M\}\}. \tag{II.4}$$

Definition 6 (Safety)**.** *Safety and/or safe operation is achieved by always remaining within the admissible set, i.e. not violating any specified constraints. In the examples provided in this work, safety is defined on a finite time horizon, such that the operation is considered safe if $\forall t \in [t_0, T], s_t \in \mathbf{S}_\varphi$. However, the ending time bound, T can be set to infinity for other systems that operate in perpetuity.*

For RTA to ensure safe operation, we need to define a stricter subset of states to further constrain operations, known as the *control invariant* safe set, $\mathbf{S}_h$. By operating in this stricter defined set, we avoid scenarios that can arise near the boundary of the admissible set, $\mathbf{S}_\varphi$ where, no matter the action executed, the next state will be outside the admissible set.

Definition 7 (Control Invariant Safe Set)**.** *The control invariant safe set, $\mathbf{S}_h$, is a subset of the admissible set, $\mathbf{S}_\varphi$, where $\forall s \in \mathbf{S}_h, \exists u \in \mathbf{U}, \mathbf{T}(s,u,s') \in \mathbf{S}_\varphi$. This means that, within the control invariant safe set, there always exists an action that will keep the system within the control invariant safe set.*

II.5.1 Explicit vs Implicit

RTA approaches are classified by how they determine $\mathbf{S}_h$, explicitly or implicitly. Explicit approaches use a pre-defined $\mathbf{S}_h$ to determine when RTA intervention is necessary. Implicit approaches use a defined backup control law and a model of the system dynamics to compute trajectories, which are used to determine when intervention is necessary. Because computing trajectories can be computationally expensive, explicit approaches tend to be more efficient. However, implicit approaches can be easier to implement since they do not require a precise definition of $\mathbf{S}_h$. This also allows implicit approaches to be less conservative.

II.5.2 Simplex vs Active Set-Invariance Filter

RTA monitoring approaches can be further split into two classes of intervention, *simplex* and *Active Set-Invariance Filter* (ASIF). The simplex approach switches between the primary control (RL actions) and a pre-defined backup controller [51]. The backup controller is usually less efficient at the desired control task, but meets desired safety and/or human-machine teaming constraints. In contrast, ASIF approaches use barrier constraints to minimize deviations from the primary control signal while assuring safety [52]. In other words, ASIF approaches are always intervening, adjusting the primary control signal by the minimal amount necessary to ensure safety. When the primary control is safe, the adjustment approaches 0.

II.6 Summary

In this chapter, we described the foundations of research areas fundamental to the rest of this dissertation. The topics included deep reinforcement learning, safe reinforcement learning and its various approaches, and run time assurance.

CHAPTER III

Sonic2Knuckles: Evaluations on Transfer Reinforcement Learning

Deep Reinforcement Learning with real-world systems is a much greater challenge than in simulated environments and tasks, because a learner in a real-world system cannot run millions of trials or easily tolerate fatal trajectories. Therefore, the ability to train agents in simulated environments and effectively transfer their knowledge to real-world environments is a crucial, and integral part of constructing autonomous robotic systems. In this chapter, we perform experiments in an original transfer reinforcement learning task we constructed using the game "Sonic 3 & Knuckles," evaluating two transfer learning techniques from the literature[1].

III.1 Introduction

Reinforcement Learning (RL) is a branch of machine learning that focuses on a software agent taking actions in an environment to maximize rewards. Due to its generality and versatility, RL is also studied in disciplines outside of machine learning, such as game theory, control theory, simulation-based optimization, multiagent systems, and swarm intelligence. RL is so versatile because it is a way of programming agents via reward and punishment without needing to specify how the task is accomplished [53].

Reinforcement Learning and other learning-based methods have been around since the early days of computer science, but are gaining influence in the control and artificial intelligence research communities [9]. In part because RL algorithms consistently show an ability to produce optimal results despite having poor models to work with. Additionally, since Neural Networks and Deep Learning have been gaining popularity, RL algorithms have been combined with Deep Learning to create a new branch of RL called Deep Reinforcement Learning (DRL). DRL is able to better represent complicated, multidimensional systems. DRL techniques are extremely attractive for robotics applications, which have complex dynamics and environments.

Despite the gaining popularity and success of DRL, it has one fatal setback; training the learning agent is a computationally expensive process. To combat this, Transfer Learning and Meta-Learning have become increasingly impactful topics. One example of this is OpenAI's Retro Competition, which utilizes an emulation of the Sega Genesis *Sonic the Hedgehog* games as the training environment. The competition challenged competitors to develop new ways of outperforming baseline algorithms at learning how to reach the end of secret/unseen levels created for the competition. The focus of the competition was on *Meta-Learning*, a way of training a *fast learner* [2]. These fast learners are created by training an agent to play a number of different

[1]This chapter is based on [15], portions of which are reprinted here.

levels/environments and using that learned policy as the starting point for an agent learning to play on any secret test level. The focus of the competition was to highlight the value of Meta-Learning, but it also proved that the Sonic games posed an interesting test environment for learning algorithms. It is non-trivial to develop an agent that performs well in the game. Thus, we chose the Sonic environment for our experiments.

In addition to transferring learned policies between the same agent in different environments, like in the Retro Competition, there are many scenarios where it would be beneficial to share learned policies among agents that have slightly different dynamics. A common application where this problem often occurs is moving learned policies from simulation to the real world. Simulations often simplify equations for the agent's motion by using a constant friction coefficient, removing slippage, or using out-of-the-box conversion factors for rotors. These simplifications often cause agents to have poor results that do not work as expected, sometimes resulting in fatal behaviors when moving from simulation to the real world [10]. However, it would be greatly beneficial to train agents in simulated environments and then run those learned policies on real world systems with comparable results since training in the real world is extremely costly, slow, and fatal trajectories are common during the learning process.

In this chapter, we explore and test different methods for transferring learned policies between agents with different dynamics using a simpler, safer environment, "Sonic the Hedgehog" for Sega Genesis. More specifically, we are using *Proximal Policy Optimization* (PPO) to train one character, Sonic, in the "Sonic 3 & Knuckles" game environment and then transfer the learned policy to train a different character, Knuckles. The characters differ in that, Knuckles is slower, has shorter jumps, and glides if a double jump is performed. By transferring the learned policy between these two characters, we are trying to emulate the same kind of transfer between simulated and real environments faced by researchers in industry today. In this project, we wrote our own implementation of PPO and all the transfer learning methods to better understand them. Additionally, we explored two methods for executing this transfer, *jumpstart* and *imitation*.

III.2 Background and Technical Approach

III.2.1 Deep Reinforcement Learning

The goal of RL is to learn the optimal policy, mapping states or observations to action, for completing a task. Learning is done through experimental trials that have their levels of success represented as a reward. With an optimal policy, the agent is able to actively adapt to the environment, maximizing future rewards. This is done by an agent choosing actions in an environment. This is illustrated in Figure III.1.

For a more detailed background on Reinforcement Learning, see Section II.2.

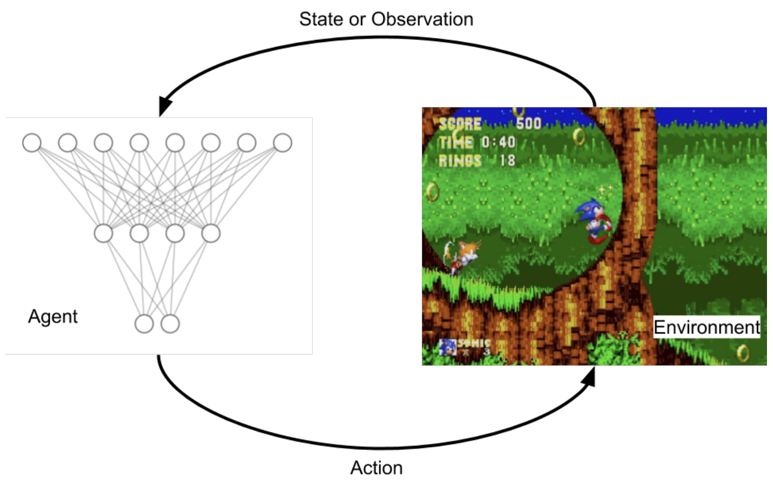


Figure III.1: Example image showing how an agent interacts with an environment. The agent is depicted as a neural network, which is used in this work and all applications of deep reinforcement learning. The environment is shown as a scene from “Sonic 3 & Knuckles”. The game is the environment, the screen capture would be the state or observation the agent uses to determine the next action.

III.2.2 Jumpstart, a Starting Point Method

The first transfer learning method we experimented with, Jumpstart, is a starting point method. That means, the target network is initialized according to knowledge from the source network [1]. In our case, the target neural network is initialized as a copy of the source neural network. This is one of the simplest transfer methods and is known for showing a large increase in the target policy’s initial performance, i.e. jumpstarting the performance. This jumpstart is highlighted in Figure III.2.a by the transfer curve starting at a higher point than the no transfer curve. We expect this method to show strong results transferring from Sonic to Knuckles if the differences between the characters are sufficiently small.

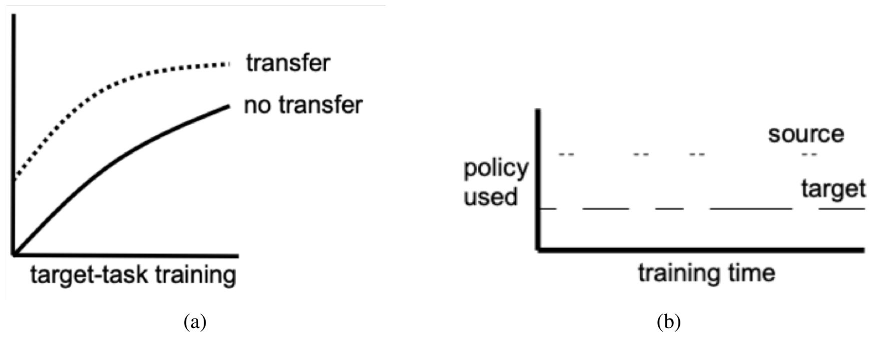


Figure III.2: The jumpstart method is highlighted in (a) as the transfer curve starts at a higher reward point. The imitation method we used is depicted in (b) showing how the agent moves between using the source and target policy to determine actions. Figures copied from Chapter 11 of *Handbook of Research On Machine Learning Applications and Trends* [1]

III.2.3 Imitation Method

Imitation methods are closely related to *imitation learning*, which allow an agent to learn by watching an expert perform a task [54]. Imitation transfer learning methods differ in how the expert is defined and the agent executes the actions determined by the expert instead of watching.

There are two main imitation methods. In the first, the agent follows the source policy at the start of the training process. This guides the target policy towards more favorable behaviors, instead of executing random actions trying to learn those good behaviors. Because the first imitation method is really similar to the jumpstart method mentioned in Section III.2.2, we decided to experiment with the second imitation method.

The second imitation method[2], depicted in Figure III.2.b, follows the source policy intermittently throughout the training process. Similar to the first imitation method, this helps guide the target policy towards favorable actions, but allows for more autonomy in learning different and potentially more beneficial behaviors. We reasoned that this would work well in the case where the source policy was trained with different dynamics than the target policy. The differences in the dynamics might make the source policy suboptimal in the new environment. So instead of learning the source policy directly, the target policy is learned from new interactions that are guided towards optimal performance by the similarities learned from the source policy.

III.3 Experimental Setup

In this section, we discuss how we designed and set up our experiments. This includes information about the environment we chose to train agents and run our experiments in, how we designed the reward function to optimize, how we structured the neural networks used, and how we set up and designed our experiments.

III.3.1 Learning Environment

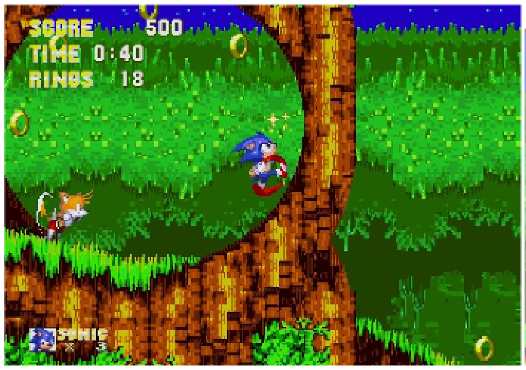
Our experiments and training were run in the "Sonic 3 & Knuckles" Sega Genesis game using the Gym Retro[3] emulator [2]. To make our results comparable, we modified the environment the same way explained in [2]. This involved restricting the available action space from every combination of the 12 available buttons being pressed (2^{12} actions) to the following 8 actions. This reduced action space contains only the useful actions for moving the Sonic character around in the game[4].

$$\{\{\}, \{LEFT\}, \{RIGHT\}, \{LEFT, DOWN\}, \{RIGHT, DOWN\}, \{DOWN\}, \{DOWN, B\}, \{B\}\}$$

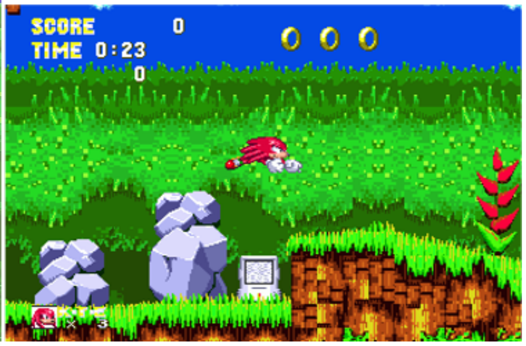
[2]This method is similar to Safe Reinforcement Learning (SRL) approaches that use Run Time Assurance (RTA) (see Section II.4.1). In the SRL case, the RTA is the source policy that is intermittently used.

[3]https://github.com/openai/retro

[4]For those not familiar with the Sonic games, B makes the character jump.



(a) Sonic



(b) Knuckles

Figure III.3: Example observations from the Sonic 3 & Knuckles video game. The screenshot on the left shows the observation for an agent playing as the Sonic character. The yellow character behind Sonic is Tails, who cannot be removed and is not controllable. The screenshot on the right shows the observation for an agent playing as Knuckles.

Additionally, we modified our environment to employ the *frame skip* and *sticky keys* modifications, which combine to make the *sticky frame skip* modification described in [2]. The first, frame skip, makes the agent's response time more similar to a humans. Instead of selecting an action for each frame of the video game occurring every $\frac{1}{60}^{th}$ of a second, each action is repeated 4 times. Now, the agent makes decisions every $\frac{1}{15}^{th}$ of a second. Each action is determined by a singular frame of the game, shown in Figure III.3. Since each timestep/interaction still occurs faster than the human reaction time, we include the second modification, sticky keys. With sticky keys, at each step, there is a 25% chance the last action executed will be used for an extra frame. An example of sticky frame skip is shown in Figure III.4.

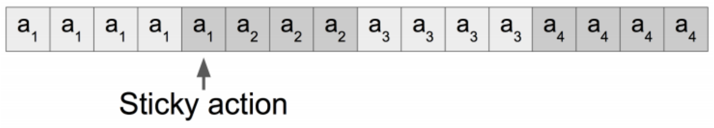

a_1 a_1 a_1 a_1 a_1 a_2 a_2 a_2 a_3 a_3 a_3 a_3 a_4 a_4 a_4 a_4
Sticky action

Figure III.4: A visualization of "sticky frame skip" described by, and image copied from, Schulman et al. [2]

Since advantages are computed backwards from the end of an episode, we had to define boundaries for an episode. We used definitions similar to Schulman et al. [2]. An episode ends if any of the following occurs:

- The character completes a level successfully. In this case, that means reaching a specified horizontal offset.
- The character loses a life. Since this results in the character starting back at the beginning, it makes sense to terminate the run.
- The agent has executed 4500 interactions in the environment. This is about 5 minutes of in-game time.

This forces the agent to complete the level within a set time limit.

III.3.2 Choosing a Reward Function

Although reinforcement learning is seeing a surge in popularity, it faces many significant challenges, one of the most notable being the problem of sparse rewards. The most effective reinforcement learning agents must be rewarded very frequently, often at every step of play or multiple times per second. Without such *dense* rewards, a reinforcement learning agent will, at the beginning of training, act at random until it stumbles across rewards or punishments to guide its behavior. Additionally, once it is actively achieving certain rewards, it will not explore the state space sufficiently to discover rewards it may not yet be aware of. When rewards are not clearly related to recent actions, distributing credit for success or failure to the appropriate actions in a long sequence may take many trials.

These challenges highlight the importance of reward function design, or *reward shaping*, for current agents. A well-designed reward function will greatly improve the speed of an agent's learning process. However, for a given environment and task, the optimal reward function is not necessarily clear. We evaluate 4 possible reward functions for our Sonic environment. We used the game's built-in rewards, known as points, the reward function provided as a baseline by OpenAI, the number of rings collected by the agent, and a *potential-based* reward using the agent's position.

Score in Sonic is mostly provided on a milestone basis, i.e. the score increases whenever certain horizontal distances in the level are reached. But these score rewards are sparse, from a reinforcement learning perspective, meaning that a reinforcement learning agent trained on this reward may not receive any feedback for some time. The ring-based reward awards the agent for collecting rings, but does not punish the agent for losing them. This was intended to encourage exploration, as rings are scattered throughout each level. The OpenAI reward provides reward to the agent for traveling further to the right than it has before, as well as a bonus for level completion, but does not punish the agent for traveling left as this is sometimes necessary. Our 'potential-based' reward was also based on position. It rewarded the agent for traveling to the right, (closer to the ultimate goal at the end of the level), but punished it for traveling left. This was intended to prevent possible cycles in the reward function that could be exploited by the agent, as outlined by [55].

We tested each reward function 3 times, each with a different random seed, and averaged results due to the high variability of training trajectories. Each time, the agent was trained for 1 million timesteps, or an average wall time of about 10 hours. Initially, we expected the training reward to follow a simple upward trend which soon saturated, indicating the agent learning an optimal policy for each reward. However, the trials were significantly more chaotic than we initially expected. Sometimes, results fluctuated wildly due to fatal trajectories. If Sonic happened to die, the reward in that episode was significantly lower than average.

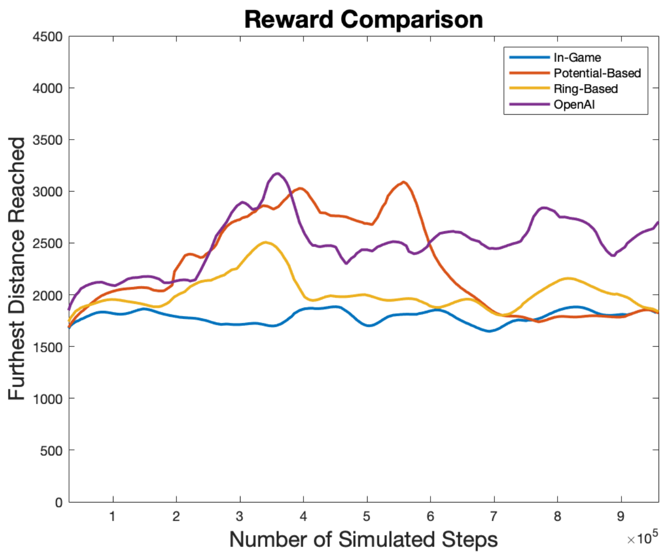


Figure III.5: A comparison of how the reward function influences the learning process. Each curve represents the average of three training runs, each with a different random seed, using the indicated reward function.

The averaged results are compiled in Figure III.5. The OpenAI reward led to the highest overall performance, in that the agent trained with this reward eventually traveled the furthest distance through the level in the shortest number of timesteps.

Other reward functions often led to unanticipated behavior. For instance, the potential-based reward, in a result which could not have been predicted in advance, led to the agent often stumbling into a hidden *bonus level*, which prevented them from traveling any further throughout the original level. This may be the primary reason it underperformed the OpenAI reward function. The ring-based reward was expected to lead to *reward farming*[5] through cycles. The agent could have learned to jump into a hazard, lose its rings, and then quickly collect them again, accumulating arbitrary amounts of reward without completing the level. However, this did not seem to occur, perhaps because there were not many rings near the beginning of the level and because learning to carry out this cycle was difficult in and of itself. We did, however, observe the agent going backwards to collect rings that it had missed.

These results are another clear demonstration of the impact reward engineering has on reinforcement learning. The OpenAI reward seemed to achieve high performance by encouraging the agent to travel right without punishing it for traveling left when necessary. Under other reward conditions, the agent sometimes got stuck against an obstacle, and did not back-track for fear of losing points. It is worth noting that the OpenAI reward function is a blend of the in-game and *potential-based* reward functions. The reward is based on more dense milestones with the maximum traveled distance rather than current location, and as such does

[5]Reward Farming is explained best at: https://openai.com/blog/gym-retro/

not contain reward cycles.

III.3.3 Neural Network Architecture

The neural network used in these experiments is split into two main components, the main body, and the split output. The main body's architecture is designed after the architecture used by OpenAI in their competition, which comes from [3]. The architecture, shown in Figure III.6, consists of three convolutional layers where the output is rolled out into a vector and passed through a fully connected ReLU layer that reduces the size. The layer dimensions were scaled down to fit our application. The dimensions of the first layer match that of $1/16^{th}$ the game's output screen image, 52x76x3 with a filter size of 8x8 and stride length 4. We reduced the image to $1/16^{th}$ its original size to help with memory management. Without this reduction, we were not able to store the information needed to complete a single horizon[6]. The dimensions of the second layer are 12x18x32 with a filter size of 4x4 and stride length 2. The dimensions of the third layer are 5x8x64 with a filter size of 3x3 and stride length 1. The output of the third convolutional layer is rolled out to 1152 inputs for the fully connected ReLU layer with 512 output nodes.

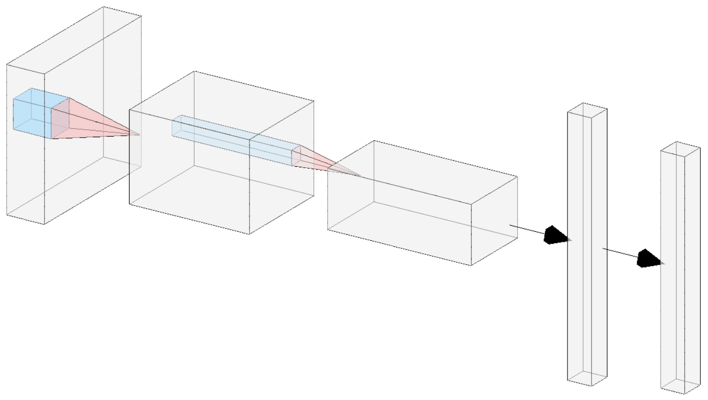


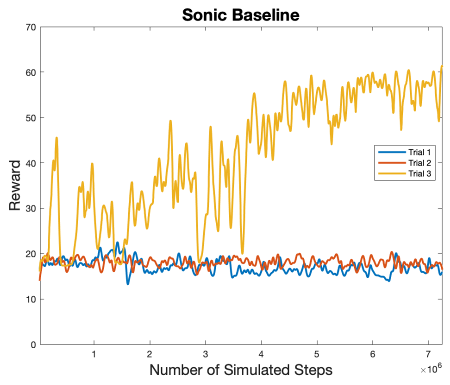
Figure III.6: The main body of the neural network used in this experiment. The structure is based on that used in [3] and [2]. The output is fully connected to two separate outputs, the actor and critic, respectively. Model generated using the web tool at http://alexlenail.me/NN-SVG/LeNet.html

The 512 output nodes of the main body are used as the input to the split output section. The final output is split to represent the actor and the critic. The actor output uses a fully connected ReLU layer to output 8 nodes, which are then scaled proportionally using the *softmax* function. Each of the 8 nodes represents an action in the reduced action space described in Section III.3.1. The critic output uses a fully connected ReLU layer to output a single node representing the estimated value of the input observation.

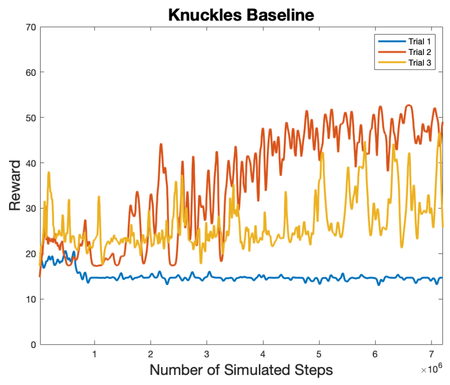
[6]A horizon is made up of one or more episodes, where the policy is not updated until after the horizon is completed. The term is often used in the literature interchangeably with the term *epoch*.

III.3.4 Experiments

We ran the following experiments on Angel Island Zone 1 Act 1 of "Sonic 3 & Knuckles". We used this level, because it is the same level and structure for both Sonic and Knuckles. Many of the other levels have different starting points or completely different structures for Sonic and Knuckles. By using the same level, we simplify the problem to focus solely on character differences and not level differences.



(a)



(b)

Figure III.7: Three training results run for ∼8 million simulated steps using the (a) Sonic character and (b) Knuckles character. The results show a large amount of variation due to the inherent randomness in using a stochastic on-policy method. The best performing trace from (a) generated the source policy for all transfer learning experiments.

To develop the *source policy*, we ran six instances of PPO training an agent. Three were trained as Sonic and the other three were trained as Knuckles. Training was done with the hyperparameters shown in Table III.1, which are the same ones used in [2]. Because of a typing error, the code ran for a longer training time than intended. The plan was to run the training for 1e6 steps instead of up to 8e6. We kept and used the result because the distance Sonic reached was better than anything we had previously seen. The last saved file of the best performing Sonic Neural Network's parameters was used in the transfer learning tests as the source policy.

The results of this training session, shown in Figure III.7, highlight the large variance that occurs in training agents with RL. Experiments conducted in [14] show how the variance in runs is enough to create statistically different distributions by varying random seeds alone. To reduce that impact and make our results more repeatable, we conducted all of our experiments using the same random seed. However, using the same random seed in every trial does not eliminate the variance. PPO uses a stochastic policy, which is inherently random. Actions are randomly chosen according to the output policy distribution. Since the random decisions made by the agent impact the future training data it obtains, the trajectories can take very different paths and converge with varying levels of success. Additionally, fatal trajectories are present in the environment. If

Table III.1: PPO hyperparameters for training the *source policy*

Hyperparameter	Value
Horizon length	8192
Updates per horizon	4
Minibatch size	8192
Discount factor (γ)	0.99
Clipping factor	0.2
Learning rate	2e-4
Episode length	4500
Total steps	8e6
Save interval	1
Load model	none

the agent selects actions that result in the character's death, the episode ends early, resulting in a drop in the accumulated reward. These random drops make the plots very difficult to understand, so we have filtered them using a Gaussian filter (length 30, $\sigma = 5$) to better highlight the reward trend.

Two transfer learning tests were run using the *jumpstart method*. Both tests used the same source policy, but the first trained the agent to play as Knuckles and the second trained the agent to play as Sonic. We tested both characters to see how the transfer changed between the two characters. We expected to see the transfer to Sonic perform better because of the matching character dynamics. Both tests were run on three instances with the same hyperparameters, shown in Table III.2, except for the character selected.

Table III.2: PPO hyperparameters for the *jumpstart method*

Hyperparameter	Value
Horizon length	8192
Updates per horizon	4
Minibatch size	8192
Discount factor (γ)	0.99
Clipping factor	0.2
Learning rate	2e-4
Episode length	4500
Total steps	1e6
Save interval	1
Load model	source policy

Two transfer learning tests were run using the *imitation method*. Both tests used the same source policy, but the first trained the agent to play as Knuckles and the second trained the agent to play as Sonic. We tested both characters to see how the transfer changed between the two characters. We expected to see the transfer to Sonic perform better because of the matching character dynamics. Both tests were run on three instances with the same hyperparameters, shown in Table III.3, except for the character selected.

Imitation length and *Imitation frequency* have no mathematical reason for their selected values. The imitation method is too theoretical to have recommended values, so we chose what we thought would work

Table III.3: PPO hyperparameters for the *imitation method*

Hyperparameter	Value
Horizon length	8192
Updates per horizon	4
Minibatch size	8192
Discount factor (γ)	0.99
Clipping factor	0.2
Learning rate	2e-4
Episode length	4500
Total steps	1e6
Save interval	1
Load model	source policy
Imitation length	10
Imitation frequency	0.1

well. The imitation length refers to how many steps the agent follows the actions determined by the source policy in succession. Imitation frequency determines how often the agent follows the source policy. With these particular settings, the agent follows the source policy for the first 10 steps out of every 100 steps simulated.

III.4 Results

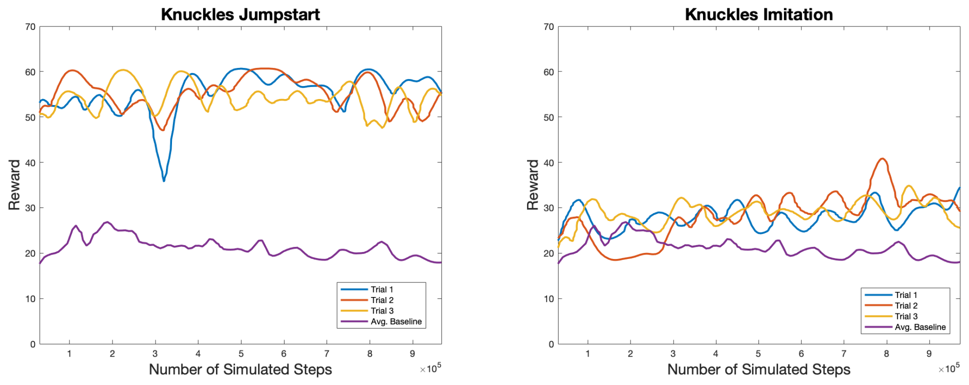
Our results show an increase in learning speed for all transfer learning tests. Additionally, we saw consistent transfers for both characters, which suggests the differences in the character dynamics were not large enough to drastically affect the learning process. These results are discussed more in-depth below.

III.4.1 Jumpstart Method

The jumpstart method showed us the expected large increase in performance at the start of the training process, despite being used for a different character. The large dips seen in Figure III.8.a occur when the character ends the episode prematurely by losing a life in a region with multiple traps. This behavior was also seen in the source policy, so it was expected. The agent does not appear to improve the policy, but instead simply maintains the results. The ending point reached when simulating the final policy of these learning curves shows that the character gets stuck at the same point as the source policy.

III.4.2 Imitation Method

The imitation method showed an increased learning speed, but was not allowed to run for long enough to show if it might eventually surpass the source policy. The results in Figure III.8.b show the policy changes regularly increase the episode reward.



(a) (b)

Figure III.8: Three training curves showing the result of transferring the source policy to an agent playing as Knuckles using (a) the jumpstart method and (b) the imitation method. The average Knuckles baseline result is provided in both plots for comparison. The jumpstart method (a) maintains the end results seen using the source policy with no improvement. The large dips show the learned policy is not very safe because the character keeps dying early, resulting in a much lower reward. The imitation method (b) shows a steady reward increase with each update to the policy.

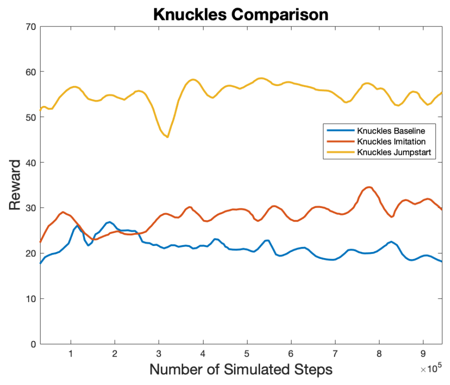
III.4.3 Knuckles Results

Comparing the results of the transfer learning methods to the results of training without transfer, shown in Figure III.9, indicate both transfer learning methods increased the final result. The jumpstart method had a significant increase in the early results, but did not increase after that. The imitation method did not see as large of an initial increase, but the learning speed was noticeably steeper than the baseline without transfer. The jumpstart method resulted in the largest final result after running for one million steps.

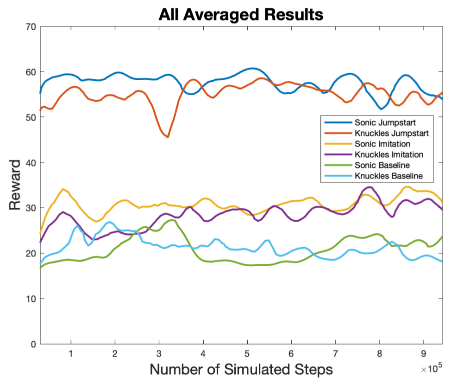
III.4.4 Different Dynamics vs. Same Dynamics

In our final results comparison, Figure III.9, we looked at how transferring between systems with the same dynamics (Sonic $\rightarrow$ Sonic) compares to transferring between systems with different dynamics (Sonic $\rightarrow$ Knuckles). The grouping in the results suggest that the change in dynamics does not create a noticeable effect. The lack of a noticeable effect suggests the slight differences in speed and jump height had very little impact on the learning process.

One interesting thing to note is transferring Sonic $\rightarrow$ Sonic using the jumpstart method is essentially continuing the training session. Picking up right where it left off. Since the policy had already converged, we expected the agent to maintain the same level of performance and decrease the time it takes the agent to reach the final position. Figure III.9.b shows the Sonic $\rightarrow$ Sonic transfer roughly maintains the same level of performance, and our viewing of the playback supports the increased speed with which Sonic reaches the final position. There were more fatal trajectories than we had anticipated, but the number should decrease



(a)



(b)

Figure III.9: (a) Averaged results of all tests run (three trials each) using the Knuckles character. (b) Averaged results of all tests run (three trials each) using both the Sonic and Knuckles characters. In both plots, the results are limited to fewer than 1 million steps. The results suggest either the character differences were not large enough to affect the transfer learning methods, or the transfer methods were effective at reducing the impact of the character differences.

with more training.

III.5 Summary

This project successfully implemented PPO and tested the desired Transfer Learning methods. The results suggest there is a noticeable improvement using the transfer learning methods. More specifically, Jumpstart had the largest effect, but did not improve upon the source policy significantly, and the imitation method increased the learning speed gradually. In conclusion, the differences between the characters were not enough to hinder the transfer learning methods, however, more tests with the platform and with more characters are needed to form any definitive conclusions.

CHAPTER IV

Zero-Shot Policy Transfer in Autonomous Racing: Reinforcement Learning vs Imitation Learning

There are few technologies that hold as much promise in achieving safe, accessible, and convenient transportation as autonomous vehicles. However, as recent years have demonstrated, safety and reliability remain the most obstinate challenges, especially in complex domains. Autonomous racing has demonstrated unique benefits in that researchers can conduct research in controlled environments, allowing for experimentation with approaches that are too risky to evaluate on public roads. In this chapter, we compare two leading methods for training neural network controllers, Reinforcement Learning and Imitation Learning, for the autonomous racing task. We compare their viability by analyzing their performance and safety when deployed in novel scenarios outside their training via zero-shot policy transfer. Our evaluation is made up of many experiments in simulation and on our real-world hardware platform that analyze whether these algorithms remain effective when transferred to the real-world. Our results show reinforcement learning outperforms imitation learning in most scenarios[1].

IV.1 Introduction

Autonomous Racing is a growing topic of interest, ranging from small-scale academic competitions (e.g. F1/10 [56]) to full-scale competitions (e.g. Roborace, AWS DeepRacer [57] and the Indy Autonomous Challenge [58]). These racing competitions are integral to the development of *Autonomous Vehicles*, (AVs) as they help promote general confidence and societal acceptance of a novel emerging technology. Moreover, they allow researchers to conduct explorations of possible solutions to difficult scenarios such as high-speed obstacle avoidance and other risky maneuvers that may be too dangerous to consider in urban settings [4].

Within this realm, one classical approach of constructing these systems involves a decomposition of tasks into four main areas: perception, planning, control, and system supervision [59]. Confining our focus to the control of these vehicles, many platforms favor classical or model predictive control techniques for their predictably safe performance. However, in recent years, many researchers have proposed the use of machine learning for control tasks, as these methods have shown significant potential in solving optimal control problems for highly nonlinear systems with varying degrees of uncertainty [59]. This prowess has made these types of regimes particularly attractive for autonomous vehicle development.

One of the most successful frameworks for solving machine learning control problems has been *Reinforcement Learning* (RL). RL is a branch of machine learning that focuses on software agents learning to

[1]This chapter is based on [16], portions of which are reprinted here.

maximize rewards in an environment through experience. The general idea is similar to training a dog to do tricks by giving it treats when it performs the desired task. Thus, an optimal controller can be synthesized using data evaluated by key performance criteria through trial and error [60]. Many RL approaches leverage neural networks due to their advantages in dealing with complex data. These approaches can be referred to as *Deep Reinforcement Learning* (also referred to as RL) techniques, and recent successes such as OpenAI's *OpenAI Five* outperforming pro-level players at Dota 2 [7], and Microsoft's *MuZero* [8] mastering Atari, Go, Chess and Shogi have helped bring RL to the forefront of AI discussion.

Despite their success in numerous realms, RL approaches can be costly to train, especially as systems become more complex and dynamic. Additionally, RL allows agents to learn via trial and error, exploring *any behavior* during the learning process. In many realistic domains, this level of freedom is unacceptable, thus training in simulation is standard. Therefore, the challenge becomes how to minimize the inherent mismatches between real-world settings, and the simulation environments used to train RL agents [61].

Training agents in simulation and then deploying them on real-world hardware platforms, known as a *sim2real* transfer, is a challenging problem. In many cases, the agents do not perform as expected in the real world, sometimes resulting in unsafe or catastrophic behavior [10, 11]. Their performance can be improved with further training in the new environment, but that is only possible if the behavior policy is safe from the outset. Transferring a learned policy and evaluating before any additional training is done is referred to as a *zero-shot policy transfer*.

In this chapter we focus on zero-shot policy transfer since active learning, i.e. learning during evaluation, is impractical for real-time systems because updates to the neural network control policy are computationally expensive and time-consuming. Instead, we evaluate trained policy networks as they are. This is standard practice in industry, to deploy a trained model and release updates intermittently.[2]

One way to achieve high performance with a zero-shot policy transfer is by leveraging external or expert knowledge. *Imitation Learning* (IL) utilizes expert demonstrations to train an agent to mimic the behavior. Using IL, an agent can be trained to mimic a human or a complicated array of computationally intensive classical control methods that perform optimally in different scenarios. In this way, complicated algorithms and/or human experience can be boiled down to one neural network capable of replicating their behaviors.

While the last several years have witnessed a significant number of approaches for addressing these challenges, there have been few in-depth empirical studies comparing the efficacy of different learning frameworks for learning robust agent behavior [62]. In [62], Gros et al. note that RL approaches generally outperform IL. However, this performance comes at a cost of significant reward shaping. While this work provides an enlightening discussion, the authors consider only discrete environments and do not address *sim2real*

[2]The rate at which these updates occur depends highly on the application.

challenges.

In light of the lack of empirical comparisons of IL and RL, in this chapter, we experiment with and compare *Neural Network Controllers* (NNCs) trained using these approaches for the control of a 1/10 scale autonomous vehicle. The performance of these trained NNCs are compared through a number of experiments, testing their ability to handle scenarios outside their training environment via zero-shot policy transfer. These experiments include changing the vehicle's constant speed, adding unknown obstacles to the track, and evaluating on different tracks. These experiments culminate in a *sim2real* transfer and evaluation of the controllers on our hardware platform.

In summary, the contributions of this chapter are:

1. We train 2 NNCs using IL to imitate a path following algorithm that effectively balances efficiency and safety.
2. We train 2 NNCs using state-of-the-art RL algorithms, DDPG and SAC.
3. We compare their performance in a series of zero-shot policy transfer experiments in simulation.
4. We compare their performance in a *sim2real* zero-shot policy transfer experiment.

IV.2 Preliminaries

IV.2.1 Imitation Learning

Imitation Learning (IL) seeks to replicate the behavior of a human or other expert on a given task [63, 64]. These approaches fall within the field of *Expert Systems* in Artificial Intelligence, and in recent years the demand for these approaches has increased substantially. The surge in interest is spurred on by two main motivations. (1) The number of possible actions needed to execute a complex task is too large to cover by explicit programming. (2) Demonstrations show that having prior knowledge provided by an expert is more efficient than learning from scratch [63].

In this chapter, we employ one of the most common methods of IL, *Behavior Cloning*, which was first introduced to train a modified van to navigate paths at speeds up to 20 miles per hour [65, 66]. The work was later replicated with an updated convolutional neural network architecture in [67] with great success.

IV.2.2 Reinforcement Learning

Reinforcement Learning (RL) is described in greater detail in Section II.2. For the work described in this chapter, we utilize two well-known, state-of-the-art off-policy deep reinforcement learning algorithms *Soft Actor-Critic* (SAC) [25], and its predecessor *Deep Deterministic Policy Gradient* (DDPG) [23].

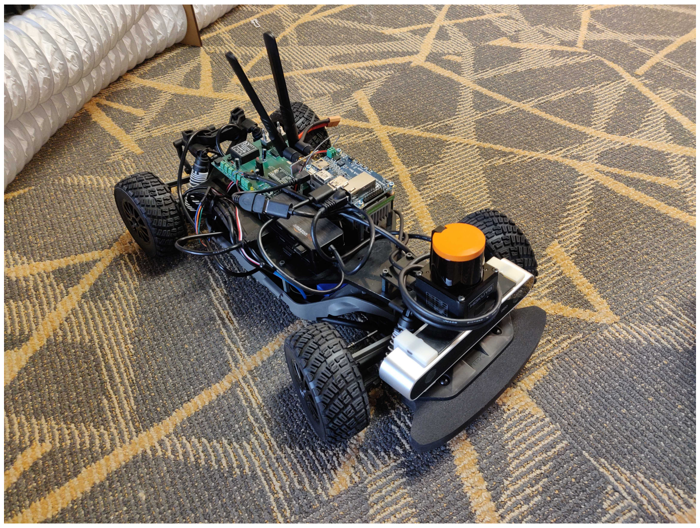


Figure IV.1: Visualization of our experimental F1/10 hardware platform. This platform is a one-tenth scale RC car that has been altered to operate autonomously with the support of a sensor and compute architecture for autonomous decision-making [4].

IV.2.3 F1/10

For our experiments, we utilize the F1/10 simulation and hardware platform [56]. The platform was designed to replicate the hardware and software capabilities of full scale autonomous vehicles. The hardware platform is equipped with a standard suite of sensors including stereo cameras, LiDAR (light detection and ranging), and inertial measurement units (IMU). The car is controlled by an NVIDIA Jetson TX2, and its software stack is built on the *Robot Operating System* (ROS) [68]. In the Gazebo simulation environment, all the sensors are replicated, so the transition from simulation to the real-world and back is straightforward without hours worth of re-configuring.

IV.3 Experimental Setup

In order to make the comparisons as fair as possible, all the controllers we trained have the same neural network architecture and are trained on the *Porto* track shown in Figure IV.2 unless otherwise specified. The trained NNCs selected for our experimental evaluations are the best performing of at least 3 NNCs trained the same way using different random seeds[3]. Additionally, the control output has been limited to only steering and the car travels at a constant speed of $1m/s$ during training.

[3]The random seed used for training has a large impact on the training process and resulting policy, as demonstrated in [14, 18]

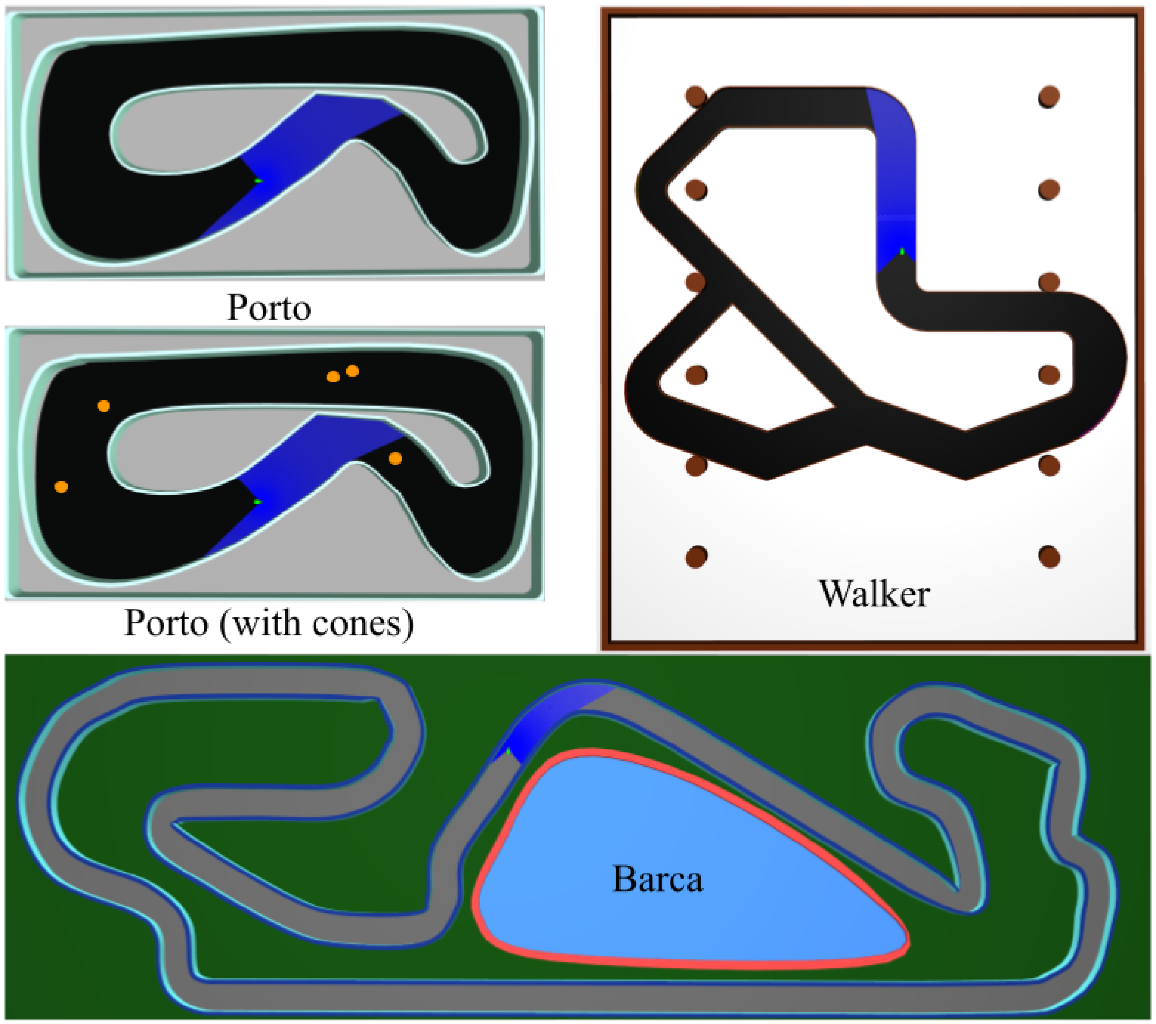


Figure IV.2: The different tracks and the corresponding starting positions we used in our simulation experiments. The bright green rectangle is the simulated car, and the blue region around it represents the full set of range values collected by the LiDAR sensor.

IV.3.1 Neural Network Architecture

In this chapter, we utilize a common architecture found in RL literature. The simple multi-layer perceptron network consists of an input layer, 2 fully connected hidden layers of 64 nodes with ReLU activation functions, and a fully connected output layer with a hyperbolic tangent, *tanh*, activation function. The input layer accepts nine range values collected from the LiDAR at $-90°$, $-60°$, $-45°$, $-30°$, $0°$, $30°$, $45°$, $60°$, and $90°$ from forward. The range values are clipped between $[0m, 10m]$. The output layer provides a single value between $[-1, 1]$, which is scaled up linearly for the desired steering angle between $[-34°, 34°]$.

IV.3.2 Training the Agents

IV.3.2.1 Imitation Learning

We trained the imitation learning agent using a procedure that is a simplification of the seminal work by Dean Pomerleau, in which a neural network was trained to control an autonomous vehicle [66]. The agents in this chapter were trained on sensor-action pairs collected during experiments where the vehicle was controlled using a path following algorithm on the racetrack. The path we used for training lead around the middle of the track, ensuring safe operation. The path following algorithm we utilized, *Pure Pursuit* [69], does a quick search for a waypoint that it can safely reach governed by a specified look-ahead horizon, it then steers the car towards that waypoint. The pure pursuit algorithm has been used in numerous contexts and has been shown to be a robust method for efficiently and accurately following a path. However, the pure pursuit algorithm requires access to accurate localization, which we did not have access to on our real-world racetrack. Using imitation learning, we wanted to bring that same level of performance to a controller that used forward-facing LiDAR ranges instead of localization. This was our main motivation in using this controller.

The first imitation learning agent, IL, was trained only on data collected from the *Porto* track. This makes the training process more like what the RL agents will see, since they are also only trained on the *Porto* track. The second agent, IL-3, was trained using data collected from the *Porto* track as well as the two other tracks, *Walker* and *Barca* shown in Figure IV.2. We include IL-3 to highlight one of the main advantages of using IL to train NNCs: any recorded data of the expert can be used for training. In both cases, the dataset of state-action pairs was split into 70% training data and 30% validation data. The policy was then trained with supervised learning, using Adam [70] with minibatches of 128 examples, until validation error stops decreasing.

IV.3.2.2 Deep Reinforcement Learning

Both RL controllers, DDPG and SAC, were trained using common hyperparameters, which are provided in Section IV.8. The agents optimize performance according to a dense reward function that assigns a positive reward for counterclockwise progress around the track. The reward is calculated using a reference path that runs through the middle of the track. The value of the reward is the positive arc length between the previous and current closest point along the path. This reward function encourages the agent to complete as many laps as possible as quickly as possible.

We trained the agents according to their respective algorithms. We halted the training process to evaluate performance after every 500 training steps. The performance is measured by how many laps the agent can complete within 100 seconds. This is more than enough time to complete 2 laps in the training track (*Porto*). We chose 2 laps because completing 1 lap is not enough to show the controller is capable of completing multiple laps. The car always starts in the same position, but may not return to the same position at the end of the first lap. However, the starting position of laps 2+ will be about the same. Thus, if the controller is able to complete 2 laps, it is likely capable of completing any number of laps.

The evaluation is repeated up to 10 times, and training stops when the agent is able to complete at least 2 laps 10 times in a row. Once the agent is able to complete at least 2 laps 10 times, the training process is halted and control policy is saved for our experiments.

IV.3.3 Evaluating Performance

We evaluate the controllers through a variety of scenarios that test their ability to maintain optimal performance in scenarios outside their training environment. These scenarios include changing the constant speed value, adding obstacles to the track, evaluating on a different track, and a real-world evaluation on our hardware platform. We compare the performance of the controllers according to three metrics we refer to as *track distance*, *efficiency*, and *safety*.

Efficiency is calculated as the distance the car travels around the track divided by the amount of time it took to get there. Each test runs for a maximum of 60 seconds and cuts off sooner if the car collides with a wall or obstacle. We refer to this as a measure of efficiency because the distance is not measured by the direct distance the car traveled. Instead, the distance is measured in relation to the arc length of a path going through the center of the track, which we refer to as the *track distance*. The closer the car stays to following the center path, the closer the efficiency value will match the constant speed. However, if the car takes sharp turns around the corners, the efficiency will increase since the car covers the same track distance in less time.

Safety is a measure of how prone to collisions the controller is at a specific track. The safety value corresponds to the percentage of runs that ended with no collision regardless of the time or distance traveled, i.e. if safety = 100%, there were no collisions encountered in the experiments.

Table IV.1: Performance on Porto with and without obstacles

	No Obstacles			Obstacles		
Algorithm	Track Distance	Efficiency	Safety	Track Distance	Efficiency	Safety
IL	121.44 ± 14.41	1.94 ± 0.25	100%	41.80 ± 0.88	1.93 ± 0.02	0%
IL-3	125.23 ± 0.31	2.01 ± 0.00	100%	40.79 ± 0.26	1.92 ± 0.02	0%
DDPG	142.74 ± 10.52	2.29 ± 0.16	100%	40.33 ± 0.37	2.23 ± 0.03	0%
SAC	144.04 ± 0.47	2.31 ± 0.01	100%	182.98 ± 2.92	2.94 ± 0.01	96.66%

Table IV.2: Performance on Porto varying constant speed

	0.5 m/s			1.0 m/s			1.5 m/s		
Algorithm	Track Distance	Efficiency	Safety	Track Distance	Efficiency	Safety	Track Distance	Efficiency	Safety
IL	64.73 ± 0.36	1.04 ± 0.01	100%	121.44 ± 14.41	1.94 ± 0.25	100%	171.24 ± 41.24	2.90 ± 0.06	93.33%
IL-3	64.60 ± 0.09	1.04 ± 0.00	100%	125.23 ± 0.31	2.01 ± 0.00	100%	178.54 ± 20.94	2.92 ± 0.03	96.67%
DDPG	73.60 ± 0.22	1.18 ± 0.00	100%	142.74 ± 10.52	2.29 ± 0.16	100%	18.09 ± 0.29	2.57 ± 0.08	0%
SAC	72.97 ± 10.05	1.20 ± 0.03	93.33%	144.04 ± 0.47	2.31 ± 0.01	100%	174.11 ± 62.58	3.21 ± 0.15	80.0%

IV.4 Experiments and Results

Our experiments were designed to test the performance of both the RL and IL controllers in challenging scenarios. The first experiment demonstrates the ideal test conditions, evaluating in the same environment the controllers were trained in. The following three experiments introduce changes to the environment that test the robustness of the learned control policies, building up towards the final experiment, deploying on the real-world hardware platform.[4]

All simulation experiments test each controller 30 times in the designated scenario. Each test lasts for a maximum of 60 seconds, stopping early in the event of a collision.[5]

IV.4.1 Training Environment (Porto)

Our first experiment evaluates the performance of the controllers in the environment they were trained in. This provides a baseline that we can compare to as we test these controllers in scenarios outside their training. The results in Table IV.1 show all the controllers operate safely without any recorded collisions. Additionally, the results show both RL controllers operate more efficiently and travel further than the IL controllers.

IV.4.2 Varying Speed

In our second experiment, we explore how changing the constant speed of the car impacts performance. This subtle change tests the robustness of the controllers with respect to a change in speed. The control policies were trained with the assumption the car moves at $1.0m/s$. Moving at different speeds, especially faster than expected, might reveal unsafe behaviors. Additionally, this experiment provides some insight into how well

[4]The hardware experiments are summarized at:
https://youtu.be/rgVb46RMMvE

[5]A video summarizing the simulation experiments can be found at: **https://tinyurl.com/2bjwpcxs**

the controllers will handle a *sim2real* transfer. Unlike in simulation, the hardware platform can experience fluctuations in speed caused by a poorly-tuned speed regulator, wheel slippage, etc.

We tested the controllers on the *Porto* track with constant speeds $0.5m/s$ and $1.5m/s$. We expected the efficiency and track distance of the controllers to be cut in half when run at half speed. We also expected the controllers would remain safe at half speed. For the tests at a faster speed, we expected the efficiency to increase by a factor of 1.5, but experience more collisions.

The results in Table IV.2 show that cutting the speed in half leads the efficiency and track distance to be reduced by about half for every controller. Since the efficiencies and recorded track distances of the controllers at $0.5m/s$ are slightly above the expected half, the controller's efficient behaviors are more impactful at slower speeds. Every controller except for SAC maintained their safe performance. In the one trial that the SAC controller collided with the wall, it was during the first left turn. The controller turned too early while driving close to the wall, resulting in a collision.

Furthermore, the results in Table IV.2 show that increasing the speed reduces the safety of all the controllers. In our experiments, none of the controllers were safe for all evaluated runs. In particular, DDPG was unable to complete any runs without colliding after the first curve. However, despite the increase in collisions, all the controllers operated more efficiently. IL, IL-3, DDPG, and SAC saw a $1.5x$, 1.45, $1.12x$, and $1.39x$ increase respectively. The IL controller was the only one able to meet the $1.5x$ increase we expected to match the speed increase.

IV.4.3 Obstacles

Our third experiment introduces unknown obstacles, orange traffic cones, to the *Porto* track as shown in Figure IV.2. This experiment tests the controllers beyond what they were trained to do. Not only does the controller have to steer the car along the optimal path while avoiding the walls, there are now additional obstacles to avoid. Thus, it provides a measure of each controller's ability to mimic the driving task, rather than robust pattern matching. The IL controllers failed to generalize to this scenario, and failed to complete a single lap without a collision failing around the last cone. DDPG was similar in nature, however, it maintained its higher level of efficiency over the IL controllers. SAC was the only controller able to handle obstacles and successfully navigated the cones in 96.66% of our evaluations. Interestingly, the obstacles improved SAC's performance. The last cone on the track was positioned just right to direct the controller to steer sooner, finding a more optimal path.

Table IV.3: performance across different racetracks

	Porto (57.5m)			Walker (73.25m)			Barca (221.14m)		
Algorithm	Track Distance	Efficiency	Safety	Track Distance	Efficiency	Safety	Track Distance	Efficiency	Safety
IL	121.44 ± 14.41	1.94 ± 0.25	100%	33.50 ± 1.55	1.89 ± 0.02	0%	108.54 ± 33.26	1.96 ± 0.03	83.33%
IL-3	125.23 ± 0.31	2.01 ± 0.00	100%	123.38 ± 0.33	1.98 ± 0.01	100%	124 ± 0.29	2.00 ± 0.00	100%
DDPG	142.74 ± 10.52	2.29 ± 0.16	100%	130.08 ± 0.34	2.09 ± 0.00	100%	31.54 ± 0.03	1.80 ± 0.01	0%
SAC	144.04 ± 0.47	2.31 ± 0.01	100%	62.64 ± 38.31	2.11 ± 0.25	0%	24.27 ± 4.09	1.76 ± 0.02	0%

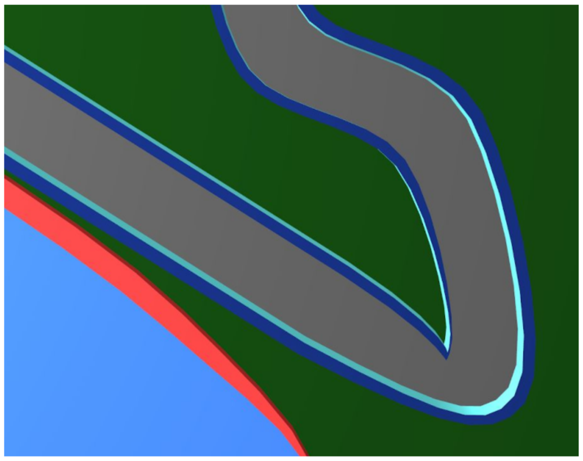


Figure IV.3: Difficult sharp turn on Barca track.

IV.4.4 Alternate Race Tracks (Walker and Barca)

In our fourth experiment, we examined how well the controllers perform when used on two different, more complicated tracks, *Walker* and *Barca* shown in Figure IV.2. *Walker* introduces a choice between two paths, which we anticipated would cause issues because none of the controllers, except IL-3, have experience with that scenario. We also anticipated that *Barca*'s long straightaways and sharp turns would cause more collisions for controllers trying to cut corners. The results for this experiment, and the lengths of the tracks for comparison, are shown in Table IV.3

On the *Walker* track, both the IL and SAC controllers were unable to consistently navigate the junction. Instead of picking a direction to pursue, the IL controller drove the car directly into the corner of the junction in every test while the SAC controller managed to avoid that fatal mistake in some cases, but rarely passed it in the second lap. In contrast, the DDPG controller was able to successfully navigate the divergent track without prior experience. Additionally, both RL controllers navigated the track more efficiently than the IL-3 controller, which had prior experience on the track.

On the *Barca* track, only the IL controllers were able to safely navigate the sharp turn highlighted in Figure IV.3. Both DDPG and SAC collide with the track wall at the sharp turn by either turning too soon or not turning at all.

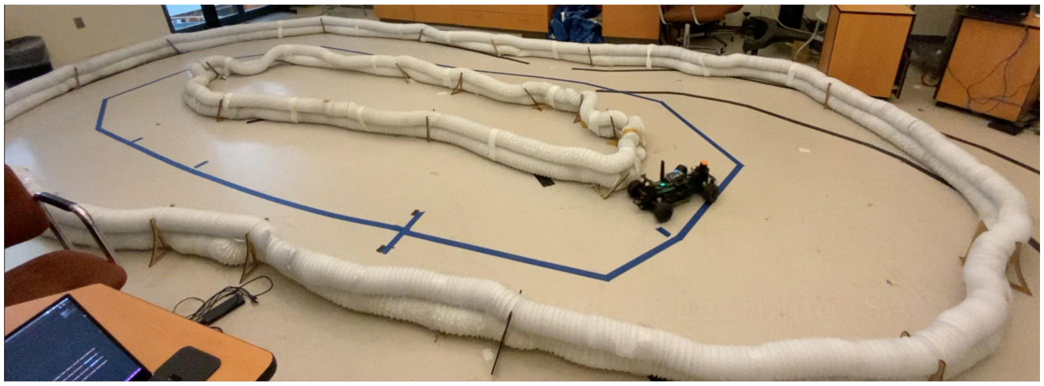


Figure IV.4: Our real-world track with the reference path used for measuring the distance traveled marked in blue.

Table IV.4: performance on hardware platform

Algorithm	Track Distance	Bumps	Collisions
IL	53.86 ± 0.47	0	0
IL-3	5.81 ± 0.00	0	10
DDPG	2.00 ± 0.00	0	10
SAC	61.83 ± 0.37	4.7 ± 0.67	0

IV.4.5 Real-World, Hardware Platform

In our final experiment, we test how well these controllers handle an actual *sim2real* transfer on our hardware platform. The experiments were conducted on our track, shown in Figure IV.4, which has a middle-of-the-track path length of 13.08m. Because we could not reliably record the time for runs that resulted in a collision, we do not compare the controllers' efficiency. Instead, we compare the distance traveled around the track in 60 seconds. We averaged the results across 10 runs and kept a count of how often the controllers drove the car along the side of the track, bumping into it (Bump), as well as how many times it drove directly into the side of the track (Collision). We halted the run in the event of a collision and recorded the final position as the total distance traveled. While bumps in our simulated results counted as collisions, we decided to allow them in the hardware experiments because they did not harm the track and would have been allowed in the F1/10 competition.

The results in Table IV.4 show DDPG and IL-3 were unable to complete a lap, instead colliding with the side of the track before completing the first turn. However, both IL and SAC were able to complete over 4 laps in the allotted 60 seconds.

IV.5 Discussion

Our experiments highlight two main challenges to the *sim2real* problem, *model mismatch* and *domain mismatch*. Model mismatch centers around the output not having the expected outcome. We highlight this challenge in our experiments with varying speed. Domain mismatch centers around the input being out of scope, or not what is expected. In other words, the real inputs do not match the training inputs. We highlight this challenge in our experiments with the obstacles and alternate racetracks. In this section, we discuss which controllers handled each type of mismatch best and theorize why that might be the case.

IV.5.1 Model Mismatch

Model mismatch is a result of the output not having the expected outcome. This could be the result of noisy actuators, inaccurate model dynamics, etc. In our experiments, we highlight this challenge by testing the controllers at varying speeds in Table IV.2.

Our results show the IL controllers handled this challenge better than the RL controllers. We attribute this result to how the controllers were trained. The IL controllers were trained to imitate the behavior of our expert control that balanced safety and efficiency by trending towards the middle of the track. In contrast, the RL controllers were trained solely to optimize efficiency. As a result, the RL controllers cut corners sharply and drove close to the walls. Because of this, changes to the speed had a larger impact on safety. Turning close to the walls has a smaller margin for error than turning in the middle of the track. Despite this greater challenge, the SAC controller was almost as safe as the IL controllers, with much better performance. If we compare the track distance of only safe trials, the SAC controller traveled an average of $201.25m$ and the IL controllers traveled an average of $182.3m$. The difference between the two is about $1/3$ of a lap.

IV.5.2 Domain Mismatch

Domain mismatch is a result of the input data not matching the training input. This could be the result of noisy sensors, unexpected obstacles, or a change in environment. We highlight this challenge in our experiments by introducing obstacles (Table IV.1) and testing on alternate racetracks (Table IV.3).

For this challenge, there is not a clear victor since the results were more varied. The IL-3 controller performed well across all the racetracks, but failed at obstacle avoidance. The SAC controller, on the other hand, successfully avoided colliding with obstacles in almost all our tests, but struggled when evaluated on alternate racetracks.

IV.5.3 sim2real

The experiments on our hardware platform help emphasize why *sim2real* is such a challenging problem. While the IL-3 controller maintained performance across all three racetracks and was the safest controller when we varied the speed, it failed to complete a single lap in the real world. Meanwhile, the IL controller, which had similar results with varied speed but struggled more on the different racetracks, successfully navigated our real world track. Because the IL-3 controller failed despite the IL controller's success, we theorize IL-3's failure was a result of overfitting. Overfitting occurs when the learned policy too closely or exactly matches the training data, and fails to generalize well to new data reliably. Training the IL-3 controller across multiple racetracks helped it perform well on all three tracks and improved its performance on *Porto*. However, all the extra training data in simulation, across varied racetracks, caused the controller to overfit to the simulation domain where the car can safely maintain a $1m$ distance from the left wall without colliding.

The varied training data that negatively impacted the IL-3 controller is likely what caused the SAC controller to succeed. While DDPG and SAC are similar RL approaches, they differ greatly in how they collect training data. In DDPG, new data is collected by adding random noise to the output of the learned policy. As the learned policy improves, the data collected starts to repeat. This repetition can cause undesirable effects on the learned policy, like *catastrophic forgetting* [28]. In contrast, SAC collects new data using an entropy maximizing function. This means that throughout the training process, new, unique, and varied data is prioritized. The result is a more robust learned policy with optimal performance.

IV.5.4 Lessons Learned

IV.5.4.1 Reinforcement Learning vs Imitation Learning

From the data and observations we collected throughout the training and evaluation processes, we found that reinforcement learning has a greater potential to learn robust and optimal control policies. However, the potential is lost without a well-defined reward function. Since RL focuses solely on optimizing performance, when we changed the track, many of the optimal performance strategies backfired and lead the car into collisions. We expect that this problem could be mitigated if we defined a reward function that incorporated an additional aspect, like a punishment for moving away from the center of the track. The result would be a more robust control policy that avoids colliding with walls, even in new tracks.

On the other hand, IL is still a valuable method, particularly when creating a well-defined reward function is not possible. However, one of the main challenges with imitation learning lies in synthesizing a dataset that allows the agent to truly mimic the expert behavior. Although the training regime for these approaches resembles standard supervised learning regimes, the i.i.d assumption may no longer be valid [71]. Often, the current state of the system prompts the next state. Thus, if the agent makes a mistake in carrying out an action, it may eventually reach a state that the agent has never been trained on. For example, if the training data only contained state-action pairs where the agent was following a path in the center of the track, any deviation from this path could result in states outside the training data and suboptimal actions that lead the car straight into a wall. Therefore, while imitation learning is extremely effective in numerous applications, it can also fail spectacularly, like shown in our *sim2real* experiments.

IV.5.4.2 Low Error is Not Necessarily a Good Indicator of Success

One commonly held principle within machine learning is that accuracy alone is generally a poor measure of evaluating a model's performance. In classification tasks, this can be addressed by using a metric such as an F1-Score, which balances the precision and recall of a model. However, it is not as straightforward for imitation learning tasks. In our experiments, we utilized mean-squared error to measure the effectiveness of our controllers. Curiously, some of the models that had very low error-rates, both on the test and validation set, could not complete a single lap. While other models, with a lower measured performance, did better on the driving task. This illustrates the need for better metrics for evaluating imitation learning tasks. There has been a large body of work towards this end over the last several years [72].

IV.5.4.3 General Recommendations

We recognize that it is difficult to issue broad recommendations on a limited set of experiments. However, we believe the observations we made will translate to other platforms. Thus, we propose the following suggestions for those who wish to apply these techniques to other platforms:

- In general, we believe that RL approaches will fare better at *sim2real* tasks, since their inspiration is more conducive to exploring a wide range of state-action pairs than those considered in behavior cloning paradigms. However, reward shaping for these approaches is still extremely challenging. Therefore, one needs to weigh the cost of reward shaping against synthesizing expansive datasets for imitation learning models.

- Our experiments did not evaluate training or fine-tuning models in the real world. This choice was motivated by a desire to ensure fairness in the evaluation process between the two approaches. While it is straightforward to train imitation learning models on real-world data, training RL approaches in the real world remains a challenge within the machine learning literature [73]. Our future work would like to consider an analysis of training and/or fine-tuning RL- and IL-trained models in the real world.

While imitation learning and reinforcement learning approaches are not widely used within production-ready, state-of-the-art autonomous vehicles, they have enjoyed significant success within industrial robotics applications. One such example of this success, is the rise of robotics companies leveraging these approaches, such as Alphabet's Intrinsic AI, Veo Robotics, Symbio, and Covariant. Still, there are few works comparing the success of imitation learning versus reinforcement learning approaches within these contexts. The work in this chapter serves to motivate these types of studies in the community at large.

IV.6 Related Work

There is a large body of work developing methods to improve reinforcement learning and overcome its shortcomings. These methods include ways to cut back on costly data collection and training time, reduce over-specialization, and improve the safety of the system during and after training is over. In this section, we highlight some of the promising methods we found in the literature. For an in-depth overview of how RL is being used in autonomous driving, we recommend Kiran et al.'s 2021 survey [74].

IV.6.1 Offline Reinforcement Learning and Inverse Reinforcement Learning

Offline Reinforcement Learning, which is best described in [75], and Inverse Reinforcement Learning [76], are both similar to a combination of imitation learning and reinforcement learning.

Like IL, the training data is collected once and goes unaltered during the training process. Additionally, the agent does not interact with the environment at all during the training process until it is deployed after training is complete. This method is very beneficial to settings where data collection is slow, expensive, and/or dangerous like in robotics, autonomous driving, or healthcare.

The key aspect that allows both of these approaches to perform better than the policy used to collect the data is the use of a reward function. The reward function allows the agent to better infer what should be done in unexplored states, guiding the agent to perform optimally. In Offline RL, the reward function is known, but in Inverse RL, the reward function is inferred from observing expert behavior.

IV.6.2 Meta Reinforcement Learning

Meta Reinforcement Learning (Meta-RL), best represented by model-agnostic meta-learning [77] and RL^2 [78], seeks to reduce over-specialization by training across multiple environments. In addition to making the agent more robust, the agent is able to learn to solve new tasks quickly. Some promising works in the area include *Joint PPO* [2] and *POET* [79].

IV.6.3 Run Time Assurance

The most effective way to ensure safety after training, no matter the learning algorithm, is with *Run Time Assurance* (RTA)[6]. Especially in safety critical settings like autonomous driving, it is imperative that system designers prevent catastrophic failures that can result from biased or limited training data [80]. In recent years, numerous RTA approaches have been proposed, ranging from approaches that are statistical in nature [81, 82, 83, 84, 48], to more rigorous formal proof regimes [85, 86, 87, 88, 89, 90, 91]. Formally demonstrating the correctness of modern machine learning models is a difficult task that often suffers from the well-known

[6]This approach is explained and experimented with in the following chapter, Chapter V.

state explosion problem [92]. While there has been a recent influx of formal methods capable of being run in real-time, statistical methods are the current leaders at circumventing scalability issues, though without the formal guarantees.

IV.7 Summary

In this chapter, we experimented with neural network controllers trained using imitation and reinforcement learning to compete in autonomous racing. We compared how the trained networks performed in new scenarios via zero-shot policy transfers. These scenarios tested the controllers' performance despite changes made to the operation of the vehicle and the track it was racing on. These changes were then combined by testing the controllers on our hardware platform.

The results show the RL controllers had more efficient performance even in new environments. SAC in particular was robust to the introduced static obstacles as well as the *sim2real* transfer. However, the RL controllers' more efficient performance led to more collisions. Therefore, developing RL for use on real-world systems needs to consider safety constraints. In the next chapter, Chapter V, we look at existing methods for incorporating safety in the RL training process and how utilizing run time assurance impacts the training and performance of the RL agents.

IV.8 Hyperparameters for Experiments

IL and IL-3 Hyperparameters:

- Network Architecture : (64, relu, 64, relu, tanh)
- Optimizer: Stochastic Gradient Descent, Nesterov Momentum
- Learning Rate (LR): 0.01
- Decay: 0.002
- Epochs: 100
- Loss: Mean Average Error

DDPG Hyperparameters:

- Policy Network (Actor): (64, relu, 64, relu, tanh)
- Q Network (Critic): (64, relu, 64, relu, linear)
- Actor LR: 0.0001

- Critic LR: 0.001
- Noise type: Ornstein-Uhlenbeck Process Noise $\sigma = 0.3$, $\theta = 0.15$
- Soft target update: $\tau = 0.001$
- $\gamma = 0.99$
- Critic L2 reg: 0.01
- buffer size: 10^6
- batch size: $B = 64$
- episode length: $T = 500$
- maximum number of steps: 45000

SAC Hyperparameters:

- Policy Network (Actor): (64, relu, 64, relu, tanh)
- Q Networks (Critics): (64, relu, 64, relu, relu)
- learning rate: 0.0001
- Soft target update: $\tau = 0.001$
- $\gamma = 0.99$
- $\alpha = 0.01$
- buffer size: 10^6
- batch size: $B = 64$
- episode length: $T = 500$
- maximum number of steps: 45000

CHAPTER V

Ablation Study of How Run Time Assurance Impacts the Training and Performance of Reinforcement Learning Agents

Reinforcement Learning (RL) has become an increasingly important research area as the success of machine learning algorithms and methods grows. To combat the safety concerns surrounding the freedom given to RL agents while training, there has been an increase in work concerning *Safe Reinforcement Learning* (SRL). However, these new and safe methods have been held to less scrutiny than their unsafe counterparts. For instance, comparisons among safe methods often lack fair evaluations across similar initial condition bounds and hyperparameter settings, use poor evaluation metrics, and cherry-pick the best training runs rather than averaging over multiple random seeds. In this chapter, we conduct an *ablation study* using evaluation best practices to investigate the impact of *run time assurance* (RTA), which monitors the system state and intervenes to assure safety, on effective learning. By studying multiple RTA approaches in both on-policy and off-policy RL algorithms, we seek to understand which RTA methods are most effective, whether the agents become dependent on the RTA, and the importance of reward shaping versus safe exploration in RL agent training. Our conclusions shed light on the most promising directions of SRL, and our evaluation methodology lays the groundwork for creating better comparisons in future SRL work[1].

V.1 Introduction

Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL) are fast-growing fields with growing impact, spurred by success in agents that learn to beat human experts in games like Go [94] and Starcraft [95]. However, these successes are predominantly limited to virtual environments. An RL agent learns a behavior policy that is optimized according to a reward function. The policy is learned through interacting with/in the environment, making training on real-world hardware platforms prohibitively expensive and time-consuming. Additionally, RL allows agents to learn via trial and error, exploring *any behavior* during the learning process. In many cyber-physical domains, this level of freedom is unacceptable. Consider the example of an industrial robot arm learning to place objects in a factory. Some behaviors could cause the robot to damage itself, other elements in the factory, or nearby workers. To mitigate these set-backs, most training is done in simulation. After the training is completed in simulation, the learned policy can then be transferred to the real world via a *sim2real* transfer. However, this approach can result in poor performance and undesirable behavior [10, 16].

[1]This chapter is based on prior work currently under review for the Journal on Artificial Intelligence and available on arXiv as a preprint [93]. This work has been approved for public release: distribution unlimited. Case Number AFRL-2022-0550.

To counteract these issues, the field of *Safe Reinforcement Learning* (SRL) has grown. Recent works demonstrate real-world online learning [9], optimal performance that does not require safety checking when deployed [12], and SRL approaches that work better than state-of-the-art DRL approaches [13]. Each new SRL paper claims to be the best, safest, most efficient, or least restrictive approach, but few prove these claims with valid demonstrations. For example, the work done in Cheng et al.'s AAAI 2019 paper, [40], claims their SRL approach "attains much greater sample efficiency in learning than other state-of-the-art algorithms and maintains safety during the entire learning process." The authors prove their claims by applying their new approach on top of two RL algorithms, TRPO and DDPG, and testing in two environments, the inverted pendulum and car following. We have tried to replicate studies with the original code, and found issues in many of their comparisons of SRL to other RL approaches. (1) Some results were invalid SRL approaches because *unrecoverable unsafe conditions*[2] *were not used as terminal conditions*. The consequence is these approaches learn to recover from unrecoverable unsafe conditions (e.g. a collision) and complete the task rather than completely avoiding unsafe states. (2) In some cases, *inconsistent hyperparameters* were used between the safe and unsafe experiments, which means the improved efficiency authors claimed as a result of their SRL approach might actually be the result of hyperparameter tuning. (3) Often, the experiments are *not repeated across multiple random seeds*. Because RL is a stochastic process, showing results from one random seed is not representative of the true performance of the algorithm. Only presenting the results of a singular trial allows for results to be cherry-picked from the best trial. The work in [14] and [96] highlight the importance of running experiments across at least 5 random seeds and averaging the results and showing the performance range in order to prove the trend of increased efficiency. (4) In some experiments we repeated, we found that authors *manipulated initial conditions* to improve efficiency. For example, inverted pendulum experiments that claimed increased efficiency as a result of an SRL approach could be explained by starting SRL trials closer to a vertical position than baseline trials. When we applied the same initial conditions to the baseline approach and SRL approaches, we found the SRL approach often learned slower than the baseline, disproving their claims.

These issues in SRL publications bring rise to the need for better comparative studies and more *ablation studies*. An ablation study involves singling out and removing individual components of a complex system to understand their impact on the system as a whole. Ablation studies are used to determine causality and can prove which aspects of a system are actually the most important. In this work, we outline a better standard for comparing SRL approaches as we conduct a thorough ablation study on SRL approaches that use *Run Time Assurance* (RTA), an approach that monitors the output of the control policy for unsafe control actions

[2]An example of this would include crashing the vehicle being controlled, while a recoverable unsafe condition might include violating a set speed limit.

and intervenes by modifying the output to assure system safety. RTA can be applied during training and after the training is complete.

Our contributions. This chapter presents an in-depth investigation on how RTA configuration and usage choices impact RL training and final agent performance. The key contributions presented in this chapter are as follows.

1. Evaluation across four different RTA approaches in addition to training with no RTA.
2. Evaluation of five different RTA training configurations that adapt how penalties are assigned during training and whether the RL agent has knowledge of a corrected action.
3. Evaluation of (1) and (2) on two different classes of RL: off-policy (SAC) and on-policy (PPO).
4. Evaluation of the true performance of each combination by training across 10 random seeds and averaging the results.
5. A large-scale (880 unique agents trained) experimental ablation study that covers (1), (2), and (3).
6. Analysis of the experimental results to provide practical insights and recommendations for training RL agents with RTA. In particular, answering these important questions:
   (a) (V.5.1) Do agents learn to become dependent on RTA?
   (b) (V.5.2) Which RTA configuration is most effective?
   (c) (V.5.3) Which RTA approach is most effective?
   (d) (V.5.4) Which works better with RTA, off-policy (SAC) or on-policy (PPO)?
   (e) (V.5.5) Which is more important, Reward Shaping or Safe Exploration?

V.2 Preliminaries

V.3 Deep Reinforcement Learning

Reinforcement Learning (RL) is a form of machine learning in which an agent acts in an environment, learns through experience, and increases its performance based on rewarded behavior. *Deep Reinforcement Learning* (DRL) is a newer branch of RL in which a neural network is used to approximate the behavior function, i.e. policy π. The basic construction of the DRL approach is shown in Figure V.1. The agent consists of the *Neural Network Controller* (NNC) and RL algorithm, and the environment consists of a plant and observer model. The environment can be comprised of any dynamical system, from Atari simulations ([15, 17]) to complex robotics scenarios ([5, 9, 14, 18, 10, 19]).

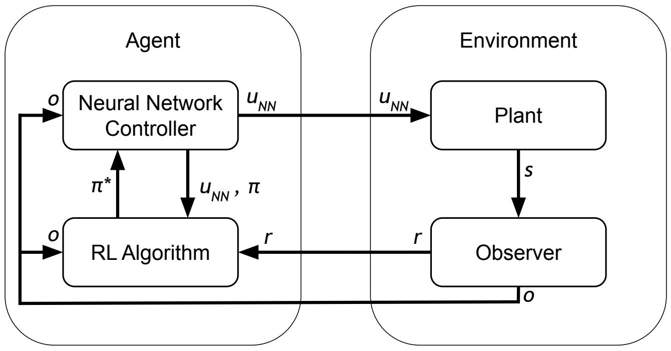


Figure V.1: DRL training interactions between agent and environment without RTA.

Reinforcement learning is based on the *reward hypothesis* that all goals can be described by the maximization of expected return, i.e. the cumulative reward [20]. During training, the agent chooses an action, u_{NN}, based on the input observation, o. The action is then executed in the environment, updating the internal state, s, according to the plant dynamics. The updated state, s', is then assigned a scalar reward, r, and transformed into the next observation vector. In all the examples shown in this chapter, we assume the task can be represented as an *Observed Markov Decision Process*[3] (OMDP). The process of executing an action and receiving a reward and next observation is referred to as a *timestep*. Relevant values, like the input observation, action, and reward are collected as a data tuple, i.e. *sample*, by the RL algorithm to update the current NNC policy, π, to an improved policy, π^*. How often these updates are done is dependent on the RL algorithm.

In this chapter, we focus on model-free DRL algorithms, meaning the agent has no dependency on the environment model during training. Within model-free DRL algorithms, there are two main categories of training, *on-policy* and *off-policy*. On-policy algorithms use the learned policy to select the actions taken during training, while off-policy algorithms use a separate policy. This distinction will cause the RTA to have a varied impact on the learning process. Thus, we repeat our experiments on two state-of-the-art DRL algorithms representing these two categories of training. *Proximal Policy Optimization* (PPO) is our on-policy algorithm[4] and *Soft Actor-Critic* (SAC) is our off-policy algorithm[5].

V.3.1 Safe Reinforcement Learning

When an RL agent explores states in a video game, the consequences of making a "wrong" move are limited. However, using RL in the real world has shown catastrophic results [9, 10]. The field of *Safe Reinforcement Learning* (SRL) was developed in response to RL's use on cyber-physical systems domain that interact with the real world in complex scenarios. In García and Fernández's comprehensive survey of SRL from 2015,

[3]See Section II.1.1 for more details.
[4]Other on-policy RL algorithms include A2C [21], TRPO [22], and ARS [18].
[5]Other off-policy RL algorithms include DQN [3], DDPG [23], and TD3 [24].

they categorized the approaches into two main categories or styles: (1) modification of the optimality criterion and (2) modification of the exploration process [29]. In this work, we refer to these categories under the more general terms: (1) *reward shaping* and (2) *safe exploration*. Additionally, we introduce an emerging category of approaches, (3) *adversarial training/retraining*. Each are described in more detail in this section.

V.3.1.1 Reward Shaping

Reward shaping, the process of crafting a well-designed, optimal reward function, is essential for all forms of DRL since a poorly designed reward function can lead to unexpected and/or ineffective behavior [15]. Within SRL, reward shaping is used to reformulate the problem as a *Constrained Markov Decision Processes*[6] (CMDP) [42]. Instead of optimizing performance according to a singular reward function, performance is optimized according to a task-oriented reward and a safety-focused cost [43, 44, 45, 46, 13, 47], so the agent learns a high-performing, safe policy. However, this style of SRL does not prohibit the agent from exploring unsafe behavior. Thus, it cannot be used to train on real hardware platforms. Instead, reward shaping techniques are limited to simulated environments and rely on high-quality transfer learning to be deployed in the real world.

V.3.1.2 Safe Exploration

Safe exploration approaches, which are often geared towards hardware deployment, ensure the agent remains 100% safe throughout the duration of training. Furthermore, this approach can be redesigned for deployment, ensuring the future safety of a static neural network that has completed training. Safe exploration techniques can be further broken down into the following three categories.

1. **Preemptive Shielding** where the action set the agent is allowed to choose from is preemptively reduced to only allow safe actions [17, 30].

2. **Safe-by-Construction** in which verification techniques are used, often on an abstraction of the learned policy, to verify safe behavior before being allowed to explore and develop further [31, 32, 33]. Alternatively, correct-by-construction can also be applied to a shielded RL solution [17].

3. **Run Time Assurance (RTA) methods** filter the agent's desired actions, u_{NN}, to assure safety. In some cases, a monitor and/or decision module is used to determine whether the desired action provided by the learning agent is safe. In the event the agent's desired action is deemed unsafe, a different action that is determined to be safe is substituted and sent to the plant [34, 12, 35, 36, 9, 37, 38, 39, 40, 41].

[6]The definition for CMDP can be found back in Section II.4.

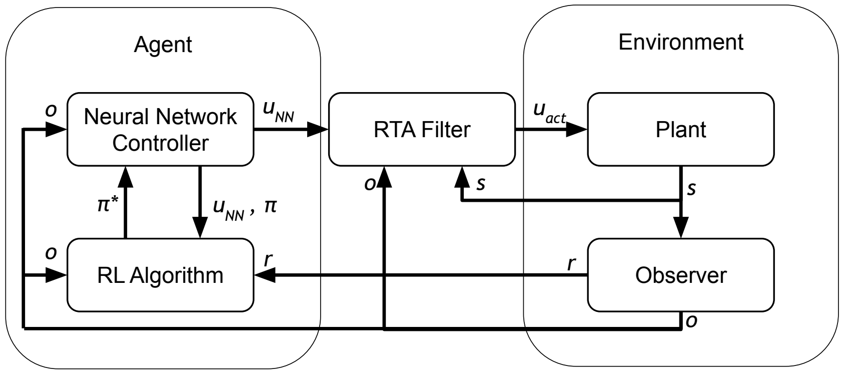


Figure V.2: DRL training interactions between the agent and the environment with RTA.

In this work, we focus solely on RTA methods for ensuring safe exploration, since they work across more examples with fewer scalability issues. An example of a general setup for safe exploration via RTA is shown in Figure V.2.

V.3.1.3 Adversarial Training/Retraining

The newest category of SRL approaches, *Adversarial Training/Retraining*, focuses on identifying unsafe behavior in the agent and then generating data to learn from and correct that behavior [48, 49, 50]. Most of the papers that use this approach focus on retraining an agent that already performs well in the environment. However, the approach can also be applied to an untrained agent.

V.3.2 Run Time Assurance

One of the main contributions of this work is investigating how the RL training process is impacted by RTA approaches that filter unsafe control inputs to preserve system safety. For this paper, we focus on dynamical system plant models sampled discretely given by $s_{t+1} = f(s_t, u_t)$ where $s_t \in \mathbf{S}$ is the state of the plant at timestep t, $\mathbf{S} \subseteq \mathbb{R}^n$ is the real-valued state space, $u_t \in \mathbf{U}$ is the control input to the plant at timestep t, with $\mathbf{U} \subseteq \mathbb{R}^m$ the action space, and f is a function describing the state evolution from current state and control action.

For the dynamical system, inequality constraints $\varphi_i(s) : \mathbb{R}^n \to \mathbb{R}$, $\forall i \in \{1, ..., M\}$ can be used to define a set of M safety constraints, where the constraint is satisfied when $\varphi_i(s) \geq 0$. The admissible set $\mathbf{S}_\varphi \subseteq \mathbf{S}$, which is defined as the set of states where all constraints are satisfied, is then given by,

$$\mathbf{S}_\varphi := \{s \in \mathbf{S} \mid \varphi_i(s) \geq 0, \forall i \in \{1, ..., M\}\}. \tag{V.1}$$

Definition 8. *Safety and/or safe operation is achieved by always remaining within the admissible set, i.e. not* *violating any specified constraints. In the examples provided in this work, safety is defined on a finite time horizon, such that the operation is considered safe if $\forall t \in [t_0, T], s_t \in \mathbf{S}_\varphi$. However, the ending time bound, T can be set to infinity for other systems that operate in perpetuity.*

For RTA to ensure safe operation, we need to define a stricter subset of states to further constrain operations, known as the *control invariant* safe set, $\mathbf{S}_h$. By operating in this stricter defined set, we avoid scenarios that can arise near the boundary of the admissible set, $\mathbf{S}_\varphi$ where, no matter the action executed, the next state will be outside the admissible set.

Definition 9. *The control invariant safe set, $\mathbf{S}_h$, is a subset of the admissible set, $\mathbf{S}_\varphi$, where $\forall s \in \mathbf{S}_h, \exists u \in \mathbf{U}, f(s,u) \in \mathbf{S}_\varphi$.*

In this work, we first focus on two classes of RTA monitoring approaches, *explicit* and *implicit*, which define $\mathbf{S}_h$ differently. Explicit approaches use a pre-defined $\mathbf{S}_h$, to determine when RTA intervention is necessary. To define $\mathbf{S}_h$ explicitly, we first define a set of M control invariant inequality constraints $h_i(s) : \mathbb{R}^n \to \mathbb{R}, \forall i \in \{1, ..., M\}$, where the constraints are satisfied when $h_i(s) \geq 0$. $\mathbf{S}_h$ is then given by,

$$\mathbf{S}_h := \{s \in \mathbf{S} \mid h_i(s) \geq 0, \forall i \in \{1, ..., M\}\}. \tag{V.2}$$

Implicit approaches use a defined backup control policy and the system dynamics to compute trajectories, which are used to determine when intervention is necessary. Implicitly, the $\mathbf{S}_h$ is defined as,

$$\mathbf{S}_h := \{s \in \mathbf{S} \mid \forall k \in [t_0, T], \mathbb{P}_k^{u_\text{b}}(s) \in \mathbf{S}_\varphi\}, \tag{V.3}$$

where $\mathbb{P}_k^{u_\text{b}}$ represents a prediction of the state s for k timesteps under the backup control policy u_b. Because computing trajectories can be computationally expensive, explicit approaches tend to be more efficient. However, implicit approaches can be easier to implement since they do not require a precise definition of the control invariant safe set, which is difficult to define without being overly conservative.

Additionally, we split the RTA monitoring approaches further with two classes of intervention, *simplex* and *Active Set-Invariance Filter* (ASIF). The simplex approach switches from the primary control to a predefined backup controller if the system is about to leave the control invariant safe set [51]. The backup controller is usually less efficient at the desired control task, but meets desired safety and/or human-machine teaming constraints. One possible implementation for a simplex RTA filter is constructed as follows,

Simplex Filter

$$u_{\text{act}}(s) = \begin{cases} u_{\text{NN}}(o(s)) & \text{if} \quad \mathbb{P}_k^{u_{\text{NN}}}(s) \in \mathbf{S}_h \\ u_{\text{b}}(s) & \text{otherwise} \end{cases} \tag{V.4}$$

Here, $\mathbb{P}_k^{u_{\text{NN}}}(s)$ represents the predicted state if u_{NN} is applied for k discrete time intervals.

ASIF approaches use barrier constraints to minimize deviations from the primary control signal while assuring safety [52]. One possible implementation for an ASIF RTA filter is constructed using a quadratic program as follows,

Active Set-Invariance Filter

$$\begin{aligned} u_{\text{act}}(s, u_{\text{NN}}) = \text{argmin}\, &\|u_{\text{NN}} - u_{\text{b}}\| \\ \text{s.t.} \quad & BC_i(s, u_{\text{b}}) \geq 0, \quad \forall i \in \{1, ..., M\} \end{aligned} \tag{V.5}$$

Here, $BC_i(s, u_{\text{b}})$ represents a set of M barrier constraints [97] used to assure safety of the system. The purpose of barrier constraints is to enforce Nagumo's condition [98] and ensure $\dot{h}_i(s)$ is never decreasing $\forall i \in \{1, ..., M\}$ along the boundary of $\mathbf{S}_h$. The function argmin finds the value of u_{b} closest to u_{NN} that still satisfies the barrier constraints. In this way, ASIF approaches apply the minimal change necessary to keep the system within $\mathbf{S}_\varphi$ at each timestep.

Using these defined approaches, we categorize our experiments in this paper according to the four derived classes of RTA monitoring approaches: *Explicit Simplex*, *Implicit Simplex*, *Explicit ASIF*, and *Implicit ASIF*.

V.4 Experiments

In order to answer the questions posed in the introduction, we have designed 88 experiments across multiple environments, RTA configurations, RTA approaches, and random seeds[7].

For each environment and DRL algorithm, we use established hyperparameters to limit the impact of tuning. We use 10 random seeds to generate our traces [14]. The evaluations run during training halt and freeze the NNC for the duration of the evaluation in order to better represent the performance of the agent at that point. After training is completed, the final learned policy is evaluated on the task 100 times to better approximate the expected performance if deployed. All evaluations are done in environments with and

[7]Code is currently undergoing the public release process and will be made available as soon as it is completed.

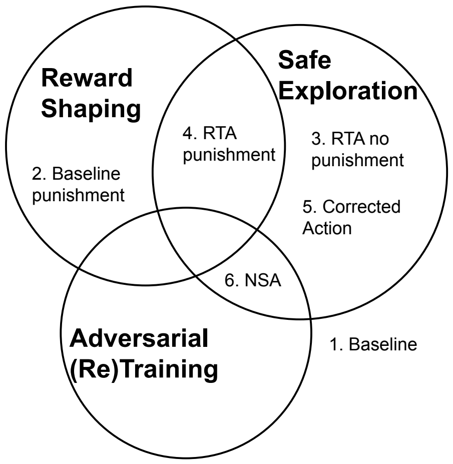


Figure V.3: The RTA configurations used in our experiments represented within the three main categories of Safe Reinforcement Learning outlined in Section V.3.1

without the RTA active in order to identify any dependence on the RTA forming.

V.4.1 Run Time Assurance Configurations

In Figure V.2, we show a general method for including RTA in the training loop and purposefully left it vague. In the literature, there are many ways of connecting RTA. These range from treating it as an unknown feature of the environment, to using it for generating additional training data. In this work, we have separated these various ways of connecting the RTA into the 6 configurations shown in Figure V.3. While we were only able to experiment with the first 5, the sixth is presented for future work. The configurations are listed in order of increasing complexity. Each configuration builds on the previous ones, helping us observe the impact of each addition.

The configurations are explained in detail below. Because SAC and PPO collect different data tuples during training, we must define the configurations using different terms, $data_{SAC}$ and $data_{PPO}$, respectively.

V.4.1.1 (1) Baseline (no RTA)

This configuration, demonstrated in Figure V.1, is used as a baseline to compare all the RTA configurations against. In this configuration, the agent is learning according to the RL algorithm without any modifications. Note that for this comparison to be fair, the environment must be the same with no alterations to the initial set or terminal conditions.

$$data_{SAC} = \{o, u_{\mathrm{NN}}, r, o'\}$$

$$data_{PPO} = \{o, u_{\mathrm{NN}}, r, v, logp(u_{\mathrm{NN}})\}$$

o is the input observation that led to the agent providing the output action, u_{NN}. o' is the observation of the state reached after taking action u_{NN} and r is the reward value associated with it. v, the estimated value of the reached state, and $logp(u_{\text{NN}})$, the log-probability of selecting u_{NN} given the current policy, are terms specific to PPO.

V.4.1.2 (2) Baseline punishment

In this configuration, we assign a negative reward, i.e. punishment p, if $unsafe?$ returns true, meaning at least one safety constraint was violated. This configuration adds SRL-style reward shaping to the problem. Instead of only maximizing the reward, the problem has two goals: (1) complete the task and (2) minimize the punishment, or cost, incurred from violating constraints. The remaining configurations cannot factor in this kind of punishment because they rely on safe exploration, which does not allow any violations of the safety constraints.

$$data_{SAC} = \begin{cases} \{o, u_{\text{NN}}, r+p, o'\}, & \text{if } unsafe? \\ \{o, u_{\text{NN}}, r, o'\}, & \text{otherwise} \end{cases}$$

$$data_{PPO} = \begin{cases} \{o, u_{\text{NN}}, r+p, v, logp(u_{\text{NN}})\}, & \text{if } unsafe? \\ \{o, u_{\text{NN}}, r, v, logp(u_{\text{NN}})\}, & \text{otherwise} \end{cases}$$

V.4.1.3 (3) RTA no punishment

This configuration is the simplest form of safe exploration. Nothing changes from the baseline configuration, except the agent remains safe throughout the training process because of the RTA.

$$data_{SAC} = \begin{cases} \{o, u_{\text{NN}}, r, o'\}, & \text{if } intervening? \\ \{o, u_{\text{NN}}, r, o'\}, & \text{otherwise} \end{cases}$$

$$data_{PPO} = \begin{cases} \{o, u_{\text{NN}}, r, v, logp(u_{\text{NN}})\}, & \text{if } intervening? \\ \{o, u_{\text{NN}}, r, v, logp(u_{\text{NN}})\}, & \text{otherwise} \end{cases}$$

V.4.1.4 (4) RTA punishment

This configuration adds an element of reward shaping to the previous configuration. Since we want the agent to learn the correct action to take in a scenario without the help of an RTA, we assign a punishment for having the RTA intervene. By adding this punishment, p, when the RTA intervenes, the agent should learn to make a distinction between safe and unsafe actions since safe actions will not incur a punishment.

$$data_{SAC} = \begin{cases} \{o, u_{\text{NN}}, r+p, o'\}, & \text{if } intervening? \\ \{o, u_{\text{NN}}, r, o'\}, & \text{otherwise} \end{cases}$$

$$data_{PPO} = \begin{cases} \{o, u_{\text{NN}}, r+p, v, logp(u_{\text{NN}})\}, & \text{if } intervening? \\ \{o, u_{\text{NN}}, r, v, logp(u_{\text{NN}})\}, & \text{otherwise} \end{cases}$$

V.4.1.5 (5) RTA Corrected Action

In this configuration, we build on the idea of helping the agent identify the correct action to take in states near violating the safety constraints. Instead of punishing the agent for having the RTA intervene, we correct the agent's output to match that of the RTA's. In this manner, the agent only learns the actions actually taken in the environment.

$$data_{SAC} = \{o, u_{\text{act}}, r, o'\}$$

$$data_{PPO} = \{o, u_{\text{act}}, r, v, logp(u_{\text{act}})\}$$

V.4.1.6 (6) Neural Simplex Architecture (NSA)

This configuration (not used in this work but planned for future work) is based on the SRL approach first published in [48]. In the authors' original implementation, NSA is used for retraining a learned policy. However, the retraining is done in an online approach, which can be easily adjusted to train a control policy from scratch.

This configuration estimates the result of taking the unsafe action and adds that estimate to the training data. This allows the agent to learn about unsafe actions without actually executing them. The additional data should help the agent develop a much better understanding of the environment and, thus, learn a more optimal policy after fewer timesteps. Currently, this configuration is limited to off-policy RL algorithms, but we are actively working on extending it for use in on-policy algorithms.

$$data_{SAC} = \begin{cases} \begin{cases} \{o, u_b, r, o'\} \text{ and} \\ \{o, u_{\text{NN}}, est\_r, est\_o'\} \end{cases}, & \text{if } intervening? \\ \{o, u_{\text{NN}}, r, o'\}, & \text{otherwise} \end{cases}$$

Here, est_r and est_o' are the estimated reward and estimated next observation, respectively. If an implicit

RTA is used, the estimates can be computed using the internal simulation, $\mathbb{P}_1^{u_{NN}}$, which is used for determining if an action is safe.

V.4.2 Environments

We ran our experiments in three environments with varying levels of complexity. By running our experiments in environments with different levels of complexity, we can observe whether the trends remain the same. If they do, then we can reasonably assume the trends will remain in even more complex environments. Currently, and to the best of our knowledge, these three environments are the only ones provided with accompanying RTA[8].

V.4.2.1 Inverted Pendulum

This environment was previously used in [40] as a good indicator of the effectiveness of SRL over standard Deep RL. We use the same initial conditions and constraints described in their work, explained below.

The goal of the agent in this environment is to use an actuator to keep the frictionless pendulum upright and within the bounds of $\pm 1 rad \approx \pm 46°$. Thus, the inequality constraint the RTA is designed to uphold can be written as,

$$\varphi_1(s) := 1 - |\theta|, \tag{V.6}$$

where θ is the displacement angle of the pendulum measured from the upright position.

The interior plant model changes according to the discrete dynamics

$$\begin{aligned} \omega_{t+1} &= \omega_t + (\frac{-3g}{2l}\sin(\theta_t + \pi) + \frac{3u_t}{ml^2})\Delta t \\ \theta_{t+1} &= \theta_t + \omega_t \Delta t + (\frac{-3g}{2l}\sin(\theta_t + \pi) + \frac{3u_t}{ml^2})\Delta t^2, \end{aligned} \tag{V.7}$$

where $g = 10$, $l = 1$, $m = 1$, $\Delta t = 0.05$, and u_t is the control from the neural network in the range $[-15, 15]$. Additionally, within the environment the pendulum's angular velocity, ω, is clipped within the range $[-60, 60]$, and the angle from upright, θ, is aliased within $[-\pi, \pi]$ radians. θ is measured from upright and increases as the pendulum moves clockwise. These values, θ and ω, are then used to determine the input values for the neural network controller. The input observation is

$$o = [\cos(\theta), \sin(\theta), \omega]^T. \tag{V.8}$$

The pendulum is randomly initialized within a subset of the safe region in order to ensure it has enough time to intervene before violating the safety constraint. θ_0 is randomly initialized between $\pm 0.8 rad$. ω_0 is

[8]As more RL environments are released with RTA, we hope to include them in our study to ensure our results continue to hold true.

randomly initialized between $\pm 1.0 rad/s$.

In the event the safety constraint is violated, the episode is deemed a failure and immediately terminated. If the simulation were allowed to continue, the problem goal would change from simply keeping the pendulum upright, to also include learning how to swing back up. Additionally, this termination helps us identify safety violations later on in our analyses, since any safety violations would result in an episode length less than 200 timesteps.

The reward function was modified as well by adding a constant, 5. 5 was chosen in order to make a majority of the reward values positive. By keeping the reward positive, the agent is encouraged to not terminate the episode early. If the reward were mostly negative, the agent might learn the fastest way to terminate the episode in order to maximize the cumulative reward. The resulting reward function, Equation V.9, has a cumulative maximum of 1000 instead of 0.

$$r_t = 5 - (\theta_t^2 + 0.1\omega_t^2 + 0.001u_t^2) \tag{V.9}$$

We use this reward function, Equation V.9, for all the evaluations we conducted. The punishment value used in various configurations when the RTA intervenes is $p = -1$.

The RTA design implemented in this environment is a simple implicit simplex design. The backup controller, described by Equation V.10, intervenes if the desired control action is not recoverable using the backup controller. To determine whether the desired action, u_{NN} is recoverable, the desired action is simulated internally. If the simulated next state, $\bar{s}_{t+1} = \mathbb{P}_1^{u_{NN}}(s_t)$, violates the safety constraint, then the backup controller intervenes. In the event the simulated next state, $\bar{s}_{t+1}$, is safe, a trajectory of up to 100 timesteps is simulated from $\bar{s}_{t+1}$ using the backup controller. If any simulated state in the trajectory is unsafe, the desired action, u_{NN}, is determined unsafe, and the backup controller intervenes. If the trajectory remains safe or a simulated next state is within the initial conditions, the desired action is determined to be safe, and the backup controller does not intervene.

$$u_b(s) = \min(\max(\frac{-32}{\pi}\theta, -15), 15) \tag{V.10}$$

V.4.2.2 Spacecraft Docking 2D & 3D

The cost of building and sending spacecraft into orbit is on the order of hundreds of millions of dollars. Therefore, it is in everyone's best interest to keep spacecraft in orbit operational and prevent collisions. Spacecraft docking is a common and challenging problem with a high risk for failure in the event an error occurs in the docking procedure and the two spacecraft collide. Here, we describe the problem with 3-dimensional dynamics, but repeated our experiments in a 2-dimensional environment where all z values are held to a constant 0.

The goal of the agent in these environments is to use mounted thrusters that move the *deputy* spacecraft in the x, y, and z directions to a docking region around the *chief* spacecraft located at the origin. The state and observation are the same,

$$s = o = [x, y, z, \dot{x}, \dot{y}, \dot{z}]^T.$$

The action vector consists of the net force produced by the thrusters in each direction,

$$u = [F_x, F_y, F_z]^T,$$

where each net force is a real value bounded in the range $\mathbf{U} = [-1, 1] m/s^2$.

In this environment, the relative motion dynamics between the deputy and chief spacecraft are given by Clohessy-Wiltshire equations [99]. These equations are a first-order approximation represented by

$$s_{t+1} = As_t + Bu_t, \tag{V.11}$$

where

$$A = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \\ 3n^2 & 0 & 0 & 1 & 2n & 0 \\ 0 & 0 & 0 & -2n & 1 & 0 \\ 0 & 0 & -n^2 & 0 & 0 & 1 \end{bmatrix}, \; B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{m} & 0 & 0 \\ 0 & \frac{1}{m} & 0 \\ 0 & 0 & \frac{1}{m} \end{bmatrix}. \tag{V.12}$$

In these equations, $n = 0.001027 rad/s$ is the spacecraft mean motion and $m = 12kg$ is the mass of the deputy spacecraft.

The agent, i.e. the deputy spacecraft, is randomly initialized in a stationary position ($\nu_{\mathrm{H}} = 0m/s$) around the chief so the distance from the chief, d_{H} is in the range $[100, 150]m$. From there, the deputy successfully docks if the distance between the deputy and chief, $d_{\mathrm{H}} = (x^2 + y^2 + z^2)^{1/2}$, is less than $20m$ and the deputy's relative speed, $\nu_{\mathrm{H}} = (\dot{x}^2 + \dot{y}^2 + \dot{z}^2)^{1/2}$, is less than $0.2m/s$. If the deputy is traveling faster than $0.2m/s$ within the docking region, then a crash occurs and the agent failed the task.

RTA is used in these environments to enforce a distance dependent speed limit and maximum velocity limits[9]. Together, these constraints keep the deputy spacecraft controllable and prevent collisions caused by

[9]More information on how and why these constraints were chosen can be found in [100, 101].

the deputy approaching too fast. The distance dependent speed limit is defined as,

$$\varphi_1(s) := v_D - v_{\rm H} + c d_{\rm H}, \tag{V.13}$$

where $v_D = 0.2m/s$ defines the maximum allowable docking velocity and $c = 0.002054s^{-1}$ is a constant. The maximum velocity, $v_{\max} = 10m/s$, limits can be written as inequality constraints,

$$\varphi_2(s) := v_{\max}^2 - \dot{x}^2, \quad \varphi_3(s) := v_{\max}^2 - \dot{y}^2, \quad \varphi_4(s) := v_{\max}^2 - \dot{z}^2. \tag{V.14}$$

Table V.1: Spacecraft 2D Spacecraft Docking & 3D reward function components

Terminal Rewards: All Configurations	
Successfully Docked ($d_{\rm H} \leq 20m$ and $v_{\rm H} \leq 0.2m/s$)	+1
Crashed ($d_{\rm H} \leq 20m$ with a velocity $v_{\rm H} > 0.2m/s$)	-1
Out of Bounds ($d_{\rm H} > 200m$)	-1
Over Max Time/Control	-1
Dense Reward: All Configurations	
Proximity	$0.0125(\Delta d_{\rm H})$
Safety Rewards: Punishment Configurations	
If RTA is Intervening	-0.001
Over Max Velocity	$-0.1 - 0.1(v_{\rm H} - v_{\max})$

The reward functions for these environments are defined by sparse and dense components defined in Table V.1[10]. The sparsely defined terminal and safety reward components are only applied if the agent meets the specified requirements. In contrast, the dense reward component is computed after each timestep. In our experiments, the evaluation returns are computed using all the components defined in Table V.1. However, during training, the safety components are ignored unless the punishment is required by the configuration.

V.4.3 Hyperparameters

Providing the hyperparameters used in RL experiments is crucial for recreating the results. In all of our experiments, we train 10 agents using the following random seeds,

$$[1630, 2241, 2320, 2990, 3281, 4930, 5640, 8005, 9348, 9462].$$

Additionally, in this section, we provide the remaining hyperparameters used for training. No matter the configuration, the following hyperparameters were used. PPO hyperparameters are provided in Table V.2. SAC hyperparameters are provided in Table V.3.

[10]These values were provided by the authors of the environments during early development and do not match those published in [102]. Additionally, these values were chosen with PPO as the target RL algorithm, which helps explain why SAC struggled with learning to complete the task.

Table V.2: PPO Hyperparameters

	Inverted Pendulum	Docking 2D	Docking 3D
actor architecture	64 tanh, 64 tanh, 1 linear	64 tanh, 64 tanh, 2 linear	64 tanh, 64 tanh, 3 linear
critic architecture	64 tanh, 64 tanh, 1 linear	64 tanh, 64 tanh, 1 linear	64 tanh, 64 tanh, 1 linear
epoch length	4000	10564	10564
epochs	100	100	100
discount factor γ	0.0	0.988633	0.988633
clip ratio	0.2	0.2	0.2
actor learning rate	0.0003	0.001344	0.001344
critic learning rate	0.001	0.001344	0.001344
updates per epoch	80	34	34
target kl	0.01	0.01	0.01
GAE-λ	0.0	0.904496	0.904496
max episode length	200	1000	1000

Table V.3: SAC Hyperparameters

	Inverted Pendulum	Docking 2D	Docking 3D
actor architecture	64 ReLU, 64 ReLU, 1 tanh	64 ReLU, 64 ReLU, 2 tanh	64 ReLU, 64 ReLU, 3 tanh
critic architecture	64 ReLU, 64 ReLU, 1 ReLU	64 ReLU, 64 ReLU, 1 ReLU	64 ReLU, 64 ReLU, 1 ReLU
epoch length	400	1000	1000
epochs	40	1000	1000
replay buffer size	10000	10000	10000
discount factor γ	0.99	0.99	0.99
polyak	0.995	0.995	0.995
entropy coefficient α	0.2	0.2	0.2
actor learning rate	0.001	0.001	0.001
critic learning rate	0.001	0.001	0.001
minibatch size	256	256	256
update after _ step(s)	1	1	1
max episode length	200	1000	1000

V.5 Results and Discussion

In this section, we try to answer the questions posed in the introduction by analyzing the overarching trends found in our experiments. We include select results that highlight the trends we found and provide all the results in Section V.7

V.5.1 Do agents learn to become dependent on RTA?

Answer: Sometimes. Training RL agents with run time assurance *always* runs the risk of forming dependence. Furthermore, this phenomenon is more prevalent in our on-policy results than our off-policy results.

An agent is dependent on the RTA if the RTA is necessary for safe and successful behavior during deployment. In cases where optimal performance fits well within the safety specifications, like our Inverted Pendulum example, dependence is not an issue. However, in cases like our docking examples where optimal performance is restricted by the safety specifications, dependence is more crucial. Because the agent will regularly encounter the boundary between safe and unsafe when deployed, the agent should understand how to best navigate those scenarios and not rely on RTA, which usually has worse performance[11]

We can determine if an agent has learned to be dependent on the RTA by evaluating performance with and without the RTA. We can identify when an agent has learned to form a dependence if the return and success drop significantly when evaluated without the RTA. If the agent is independent of the RTA, the performance metrics should be consistent when evaluated with and without the RTA.

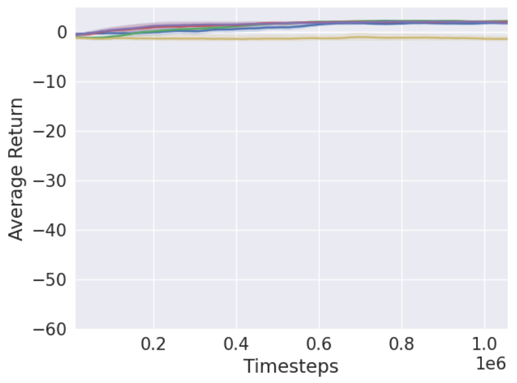
Table V.4: This table shows final policy evaluation results across all test environments trained using the PPO algorithm with the Implicit Simplex RTA approach. We show the recorded performance measured by the reward function (Return) and whether the agent was successful at completing the task (Success). Rows highlighted in gray indicate a learned dependency.

Environment	Configuration	RTA	Return	Success
Inverted Pendulum	RTA no punishment	on	987.84 ± 10.86	1.00 ± 0.00
		off	987.59 ± 10.38	1.00 ± 0.00
Inverted Pendulum	RTA punishment	on	987.57 ± 11.20	1.00 ± 0.00
		off	987.85 ± 11.18	1.00 ± 0.00
2D Spacecraft Docking	RTA no punishment	on	2.02 ± 0.39	0.92 ± 0.27
		off	-22.39 ± 15.39	0.34 ± 0.47
2D Spacecraft Docking	RTA punishment	on	1.82 ± 0.60	0.73 ± 0.44
		off	-17.99 ± 5.16	0.46 ± 0.50
3D Spacecraft Docking	RTA no punishment	on	-33.29 ± 22.18	0.50 ± 0.50
		off	-23.51 ± 8.54	0.44 ± 0.50
3D Spacecraft Docking	RTA punishment	on	2.04 ± 0.39	0.89 ± 0.31
		off	2.07 ± 0.34	0.92 ± 0.27

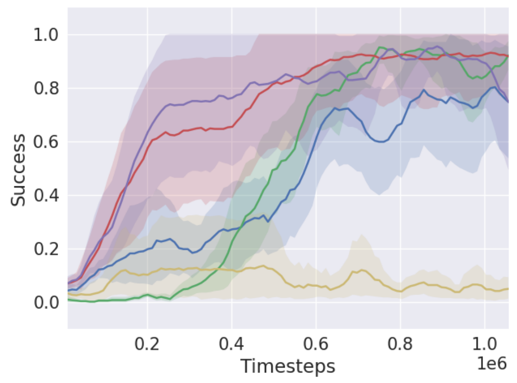
We use gray to highlight examples where agents learned to form a dependency in Table V.4, which contains the final policy evaluations of our on-policy results across all the environments using the implicit

[11]Examples of this trade-off in reduced performance for safety can be found in the following texts: [103, 104, 105].

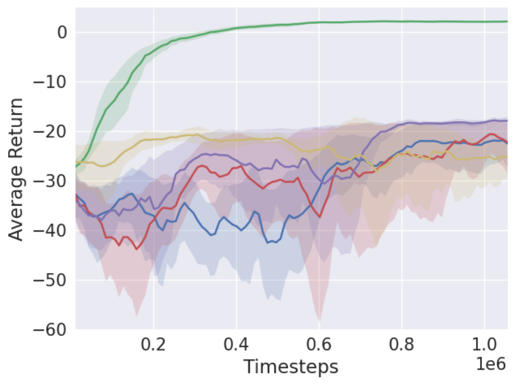
simplex RTA approach. Note that not all agents in the highlighted rows (differentiated by the random seed used for training) learned a dependency, as evidenced by the increased standard deviation about the mean values. This further reinforces our answer that *sometimes* agents learn to become dependent on the RTA they are trained with. Instead of “always” or “never,” whether the agent learns to become dependent on the RTA is a matter of chance, i.e. which random seed is used. Mania et al. show a great visualization in [18] of just how large an impact the random seed has on whether the agent learns a successful policy. The same is true here. If we had selected different random seeds, we would likely see different results for which agents learn to become dependent. However, it is the case that this can only happen if the agent is trained with RTA, and we have seen it is less likely to occur if the agent is punished for using the RTA as in the *RTA punishment* configuration. Note that the impact of the level/scale of punishment on whether dependence forms is left for future work.



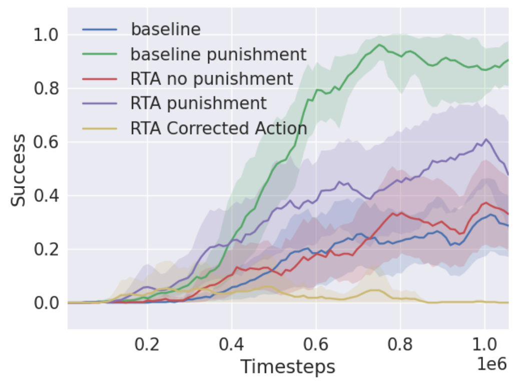
(a) PPO evaluated with RTA Average Return



(b) PPO evaluated with RTA Average Success



(c) PPO evaluated without RTA Average Return



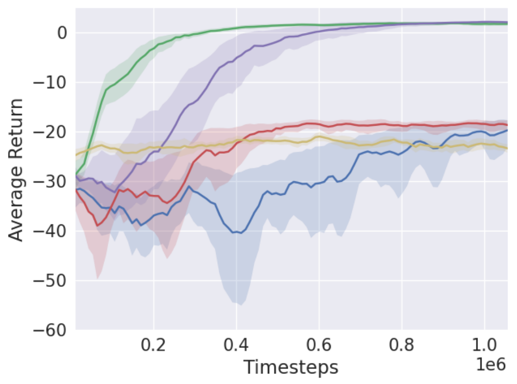
(d) PPO evaluated without RTA Average Success

Figure V.4: Results collected from experiments run in the 2D Spacecraft Docking environment with an implicit simplex RTA. Each curve represents the average of 10 trials, and the shaded region is the 95% confidence interval about the mean. The large difference in return and success that is recorded with (a & b) and without (c & d) RTA shows that all agents trained with RTA learned to depend on it.

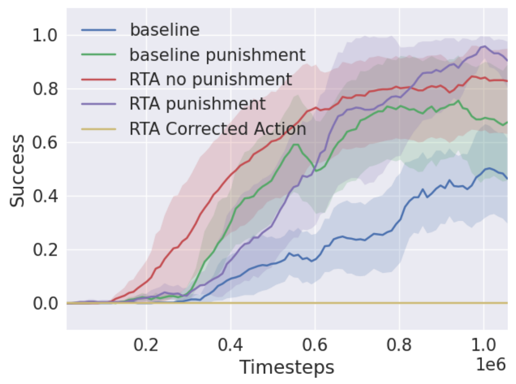
To further demonstrate issues with agents forming a dependence on RTA, observe the drop in performance and success between evaluating with RTA (a & b) and without RTA (c & d) in Figure V.4. While the RTA helps all the agents reach success throughout the training process, the agents trained with RTA (*RTA no punishment*, *RTA punishment*, and *RTA Corrected Action*) do not maintain that same level of performance when evaluated without the RTA. In contrast, the *baseline punishment* agents learn successful behavior that works with and without the RTA.

V.5.2 Which RTA configuration is most effective?

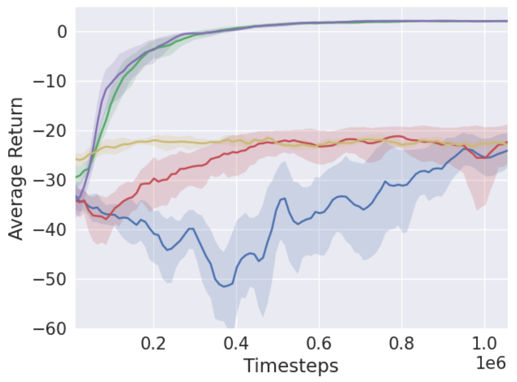
Answer: *Baseline punishment* is the most effective. However, if safe exploration is necessary, *RTA punishment* is the most effective.



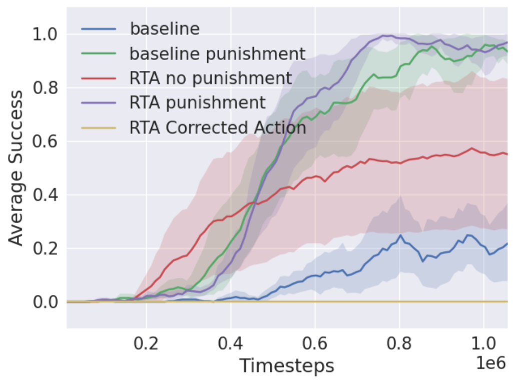
(a) 2D Docking, PPO evaluated without RTA Average Return



(b) 2D Docking, PPO evaluated without RTA Average Success



(c) 3D Docking, PPO evaluated without RTA Average Return



(d) 3D Docking, PPO evaluated without RTA Average Success

Figure V.5: Results collected from experiments run in the 2D (a & b) and 3D (c & d) Spacecraft Docking environment with an explicit simplex RTA. Each curve represents the average of 10 trials, and the shaded region is the 95% confidence interval about the mean. All plots show the *baseline punishment* and *RTA punishment* configurations learn at a similar rate and converge to similar levels of success and return.

The most effective RTA configuration is the one that consistently trains the best performing agent evaluated without RTA. In the case of a tie and the final performance is comparable across multiple configurations, the best configuration is the one that learns the optimal performance quicker, i.e. requiring fewer samples. Across all our experiments, the *baseline punishment* configuration was consistently among the best performing agents. The next best performer was the *RTA punishment* configuration, which often outperformed the *baseline punishment* configuration in our PPO experiments, but did not do as well in all of our SAC experiments. We discuss why this might be the case in Section V.5.4. To demonstrate this conclusion, we show the training curves in Figure V.5 from our experiments training agents across all configurations in both the 2D and 3D Spacecraft Docking environments using the PPO algorithm and the explicit simplex RTA approach.

In these particular examples, Figure V.5, both *baseline punishment* and *RTA punishment* have similar training curves that converge about the same return and success. This is similar across most of our experiments, except in some experiments when *RTA punishment* has a noticeably lower return because a dependence on the RTA formed.

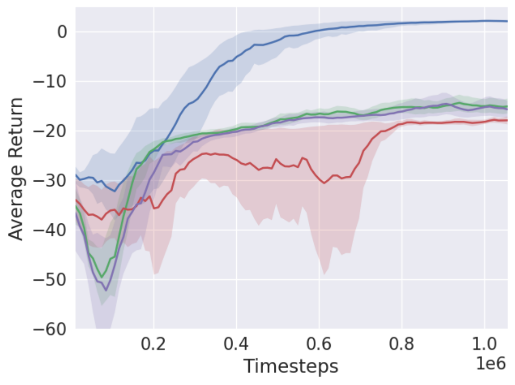
V.5.3 Which RTA approach is most effective?

Answer: The explicit simplex is the most effective RTA approach for training agents that consistently perform well and do not learn to depend on the RTA to maintain safety.

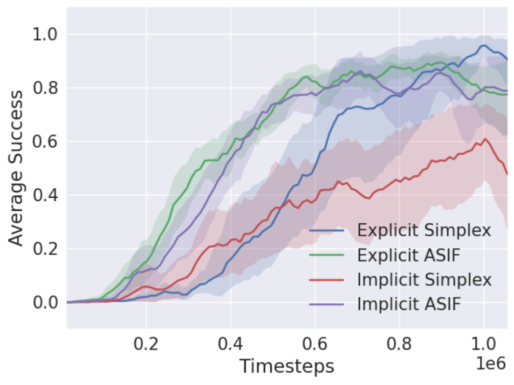
Figure V.6 shows the training curves for PPO agents trained in our 2D and 3D Spacecraft Docking environments with four different RTA approaches. All the training curves represent the PPO agents trained in the *RTA punishment* configuration and evaluated without the RTA. The curves broadly show ASIF RTA's guide the agent to success earlier on, but at the cost of increased sample complexity. The simplex approaches instead have a reduced sample complexity achieving a higher return sooner, which then leads to a greater chance of success.

We attribute these results to the differences between simplex and ASIF approaches. With simplex, the RTA does not intervene until the last moment, which allows for more agent-guided exploration, leading to more unique data samples. More unique data samples leads to a better approximation of the value- and/or Q-function, which reduces sample complexity. In contrast, ASIF approaches apply minimal corrections intended to guide the agent away from boundary conditions. This applies a greater restriction on agent-guided exploration, which can lead to more duplicated data samples, but increases the successfully completing the task during a training episode.

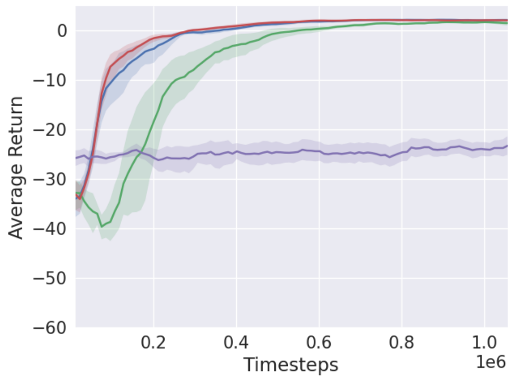
The implicit RTA approaches were less effective than the explicit approaches and were less consistent. In the 2D Spacecraft Docking environment, both ASIF training curves had similar return and success. However, in the 3D Spacecraft Docking environment, the explicit ASIF curve maintained the trend of earlier success with reduced return while the implicit ASIF curve failed to improve throughout the entire training process. Similarly, in the 3D Spacecraft Docking environment, the simplex curves had similar return and success, but



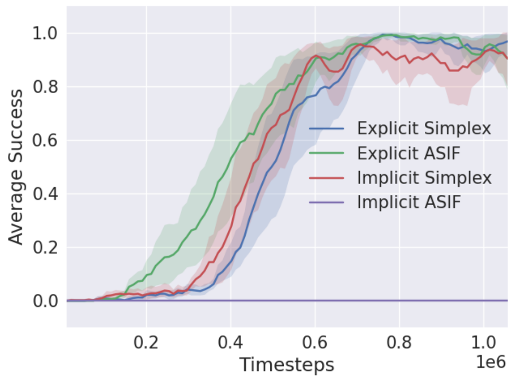
(a) PPO Average Return evaluated without RTA in the 2D Spacecraft Docking environment



(b) PPO Average Success evaluated without RTA in the 2D Spacecraft Docking environment



(c) PPO Average Return evaluated without RTA in the 3D Spacecraft Docking environment



(d) PPO Average Success evaluated without RTA in the 3D Spacecraft Docking environment

Figure V.6: Training curves collected from experiments run in the 2D and 3D Spacecraft Docking environments, training with PPO in the *RTA punishment* configuration across all four RTA approaches. Each curve represents the average of 10 trials, and the shaded region is the 95% confidence interval about the mean.

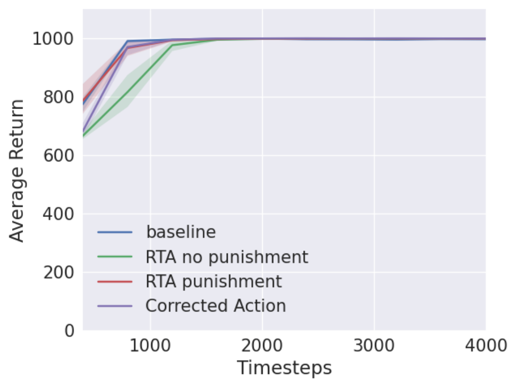
in the 2D Spacecraft Docking environment the implicit simplex curve showed a large drop in both return and success.

Therefore, we reason explicit RTA approaches are better for training. Additionally, simplex approaches lead to a better performing agent in the long run.

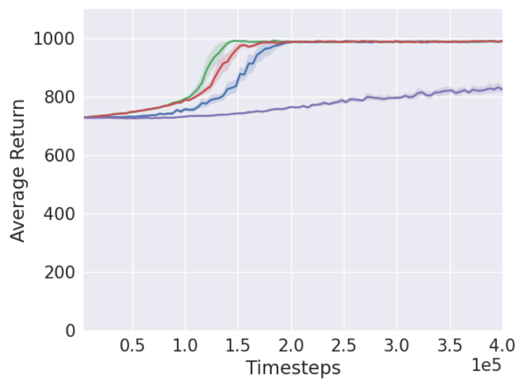
V.5.4 Which works better with RTA, off-policy (SAC) or on-policy (PPO)?

Answer: On-policy methods see a greater benefit from training with RTA.

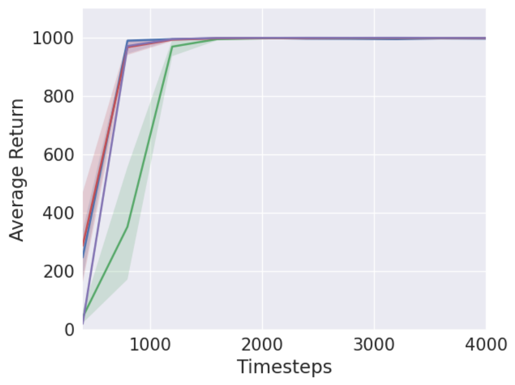
Our results showed PPO sees a greater benefit from training with RTA than SAC. This is likely a result of how the methods approach the *exploration versus exploitation* problem. Too much exploitation, using only known information (i.e. the current policy) too strictly, and the agent may never find the optimal policy. However, too much exploration and the agent may never learn what the goal is, particularly if the rewards are sparse. In general, on-policy methods leverage more exploitation than off-policy methods through their use of the learned policy.



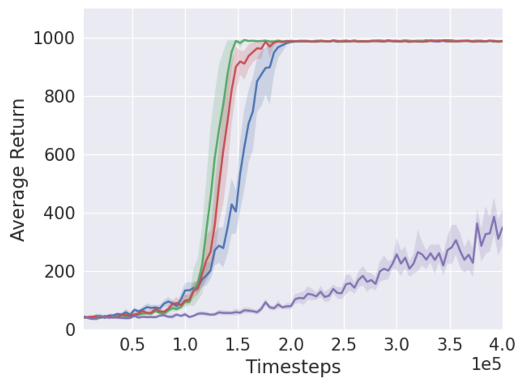
(a) SAC Average Return evaluated with RTA



(b) PPO Average Return evaluated with RTA



(c) SAC Average Return evaluated no RTA



(d) PPO Average Return evaluated no RTA

Figure V.7: Results collected from experiments run in the Inverted Pendulum environment. Each curve represents the average of 10 trials, and the shaded region is the 95% confidence interval about the mean. Note: maximum possible return in the environment is 1000.

On-policy methods exploit the learned policy. Therefore, guiding the agent to success and away from unsafe behavior helps the agent learn that behavior. As a result, the sample complexity is reduced. Figure V.7 (b & d) highlights this effect well. However, these benefits are hindered if the wrong configuration is chosen. Across almost all of our PPO experiments, the *RTA Corrected Action* configuration prevented the agents from improving the learned policy as shown in Figure V.5. This is likely a result of too much exploitation from the RTA intervening around boundary conditions, which prevented the agents from exploring other options to better define the optimal policy.

In contrast, **off-policy** methods have a larger focus on exploration. In particular, SAC maximizes entropy, assigning a higher value to unexplored state-action combinations. By restricting the actions taken near boundary conditions, the "unsafe" actions are never explored in actuality. Without making some distinction when the RTA intervenes makes the patterns harder to learn. This is shown in Figure V.8 where both *RTA no punishment* and *RTA Corrected Action* have a noticeably worse training curve than the *baseline* configuration, failing to improve at all through training. We see a similar trend in Figure V.7 (a & c) when SAC is used to learn the optimal policy for controlling our inverted pendulum. However, in this environment, the agent is still able to learn a successful policy.

V.5.5 Which is more important, Reward Shaping or Safe Exploration?

Answer: Reward shaping is generally more important for training. While safe exploration can improve sample complexity in some cases, a well-defined reward function is imperative for training successful RL agents.

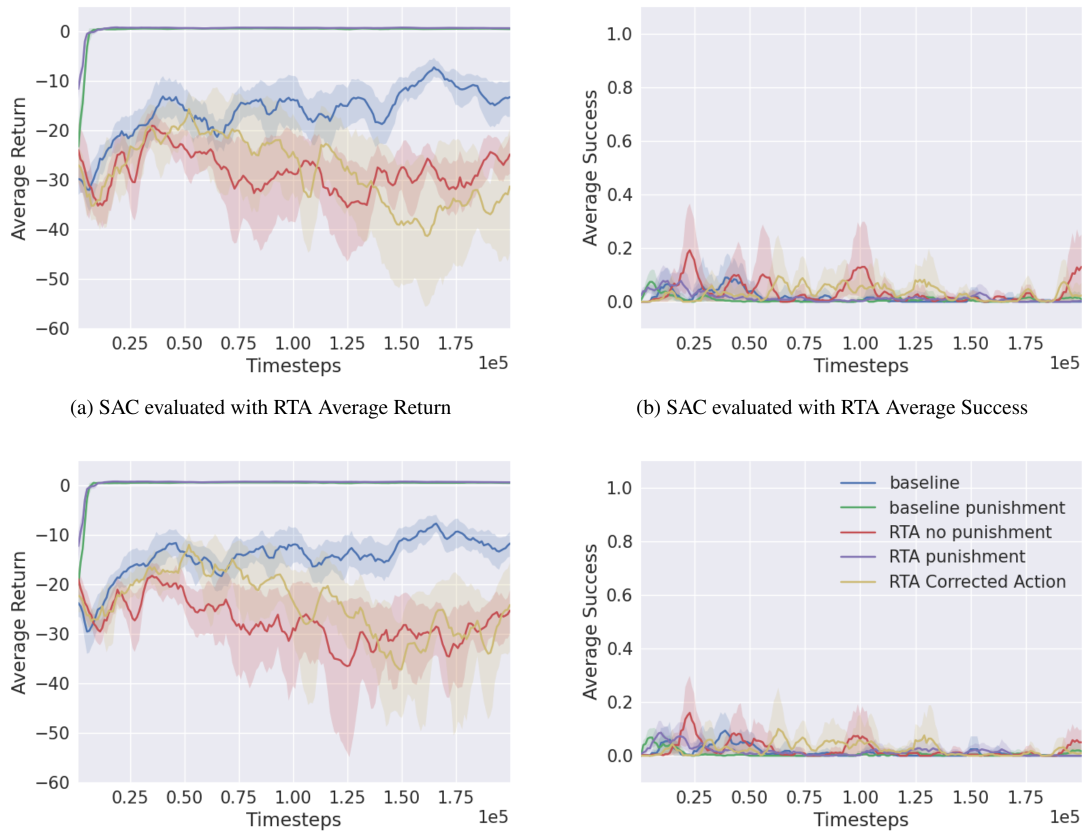
In every experiment where reward shaping was applied, we saw a more consistent and improved training curve. For example, note in Figure V.5 that the 95% confidence interval about *baseline punishment* and *RTA punishment* is smaller than *baseline* and *RTA no punishment* respectively. Additionally, the return and success tend to be much greater. The same trend shows in all of our experiments.

That said, safe exploration does improve sample complexity for on-policy RL, but the improvements are much greater when reward shaping is also applied in the *RTA punishment* configuration. Additionally, the punishment helps prevent the agent from becoming dependent on the RTA.

However, safe exploration on its own is no substitute for a well-defined/tuned reward function, as evidenced in our SAC experiments in the docking environments shown in Figure V.8. In these experiments, the agents with the *baseline punishment* and *RTA punishment* configurations quickly converged to an optimal performance, but the optimal performance did not result in success.

V.6 Summary

In summary, we trained 880 RL agents in 88 experimental configurations in order to answer some important questions regarding the use of RTA for training safe RL agents. Our results showed that **(1)** agents sometimes learn to become dependent on the RTA if trained with one, **(2)** *baseline punishment* and *RTA punishment* are the most effective configurations for training safe RL agents, **(3)** the explicit simplex RTA approach is most effective for consistent training results that do not depend on the RTA for safety, **(4)** PPO saw a greater benefit from training with RTA than SAC, suggesting that RTA may be more beneficial for on-policy than off-policy RL algorithms, and **(5)** effective reward shaping is generally more important than safe exploration for training safe RL agents.



(a) SAC evaluated with RTA Average Return

(b) SAC evaluated with RTA Average Success

(c) SAC evaluated without RTA Average Return

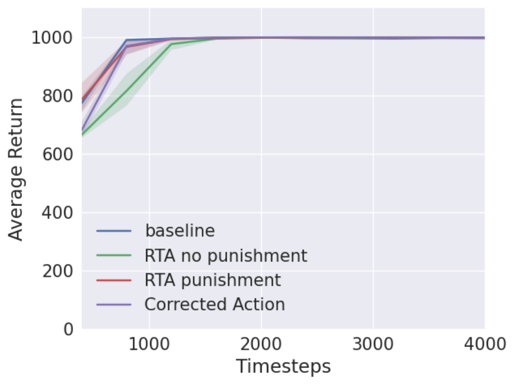
(d) SAC evaluated without RTA Average Success

Figure V.8: Results collected from experiments run in the 2D Spacecraft Docking environment with explicit simplex RTA. Each curve represents the average of 10 trials, and the shaded region is the 95% confidence interval about the mean.

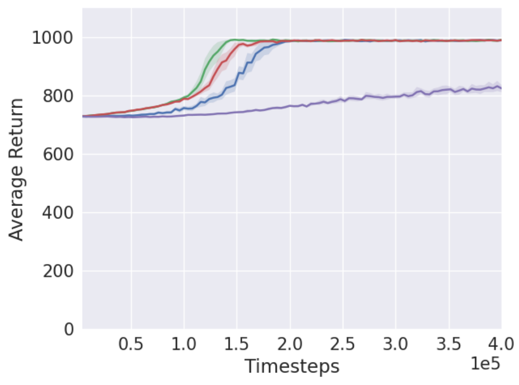
V.7 All Experimental Results From Ablation Study

In this section, we show the results collected from all of our experiments. The subsections are broken up according to environment and RTA approach. Experiments in the pendulum environment are in Section V.7.1. Experiments in the 2D Spacecraft Docking environment are in Sections V.7.2, V.7.3, V.7.4, and V.7.5. Experiments in the 3D Spacecraft Docking environment are in Sections V.7.6, V.7.7, V.7.8, and V.7.9.

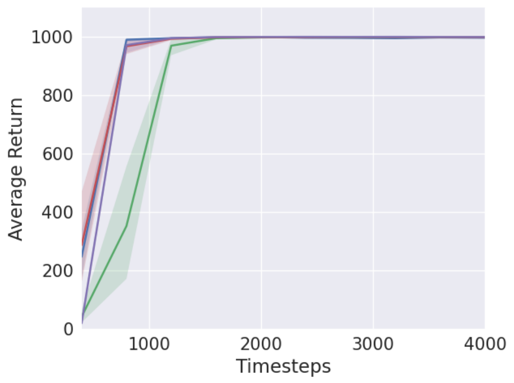
V.7.1 Pendulum Implicit Simplex



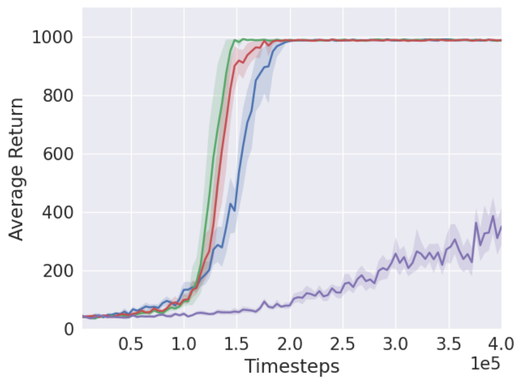
(a) SAC with RTA



(b) PPO with RTA



(c) SAC no RTA



(d) PPO no RTA

Figure V.9: Results collected from experiments run in the Pendulum environment. Each curve represents the average of 10 trials, and the shaded region is the 95% confidence interval about the mean. Note: maximum possible return in the environment is 1000.

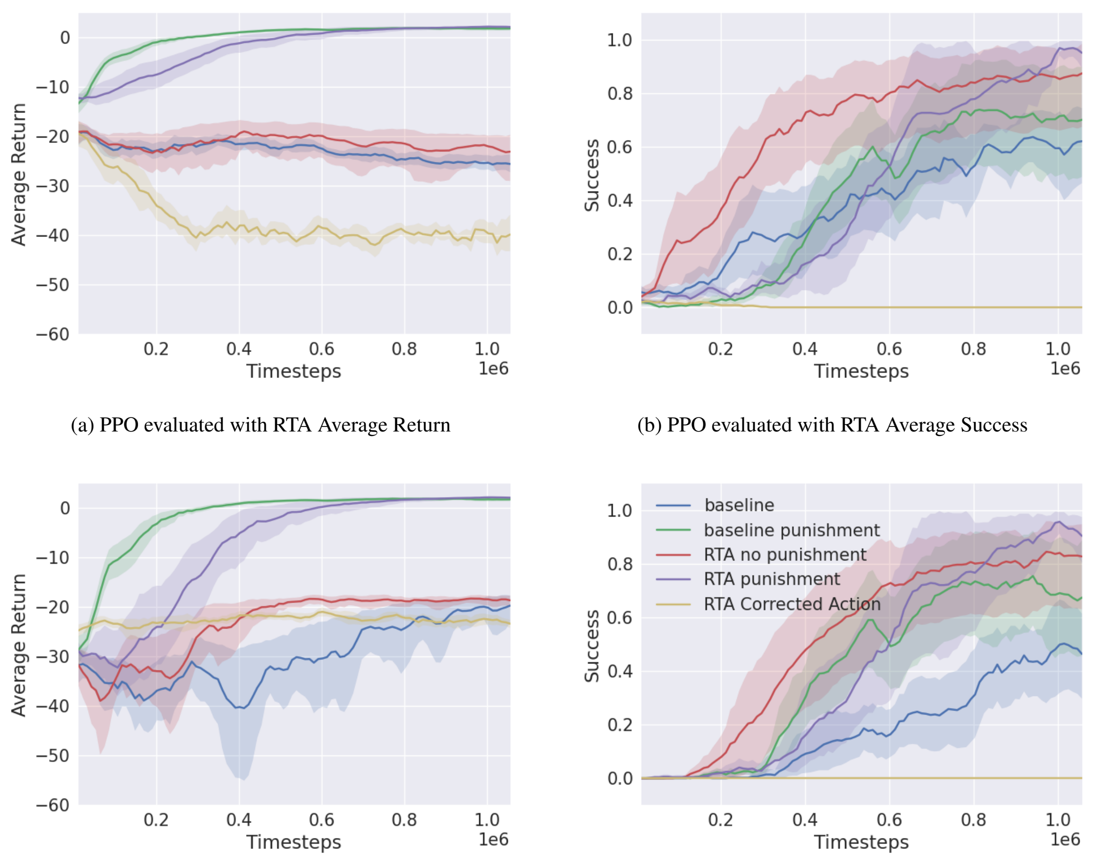
Table V.5: SAC Pendulum

Configuration	RTA	Return	Length	Interventions
baseline	on	983.1007 ± 3.1389	200.0 ± 0.0	0.2940 ± 0.7250
	off	982.8630 ± 3.3692	200.0 ± 0.0	-
RTA no punishment	on	983.9275 ± 2.0795	200.0 ± 0.0	0.0370 ± 0.2401
	off	982.8445 ± 30.7715	199.8070 ± 6.1001	-
RTA punishment	on	984.1265 ± 2.0494	200.0 ± 0.0	0.0 ± 0.0
	off	984.0623 ± 1.9408	200.0 ± 0.0	-
Corrected Action	on	984.5683 ± 2.1403	200.0 ± 0.0	0.0030 ± 0.0547
	off	984.5059 ± 2.0753	200.0 ± 0.0	-

Table V.6: PPO Pendulum

Configuration	RTA	Return	Length	Interventions
baseline	on	987.9677 ± 10.9786	200.0 ± 0.0	0.0 ± 0.0
	off	987.3375 ± 11.2540	200.0 ± 0.0	-
RTA no punishment	on	987.8363 ± 10.8634	200.0 ± 0.0	0.0 ± 0.0
	off	987.5939 ± 10.3785	200.0 ± 0.0	-
RTA punishment	on	987.5672 ± 11.2048	200.0 ± 0.0	0.0 ± 0.0
	off	987.8488 ± 11.1780	200.0 ± 0.0	-
Corrected Action	on	850.7139 ± 51.6744	200.0 ± 0.0	27.7470 ± 11.6062
	off	306.7720 ± 281.0874	64.7700 ± 56.5273	-

V.7.2 2D Spacecraft Docking Explicit Simplex



(a) PPO evaluated with RTA Average Return

(b) PPO evaluated with RTA Average Success

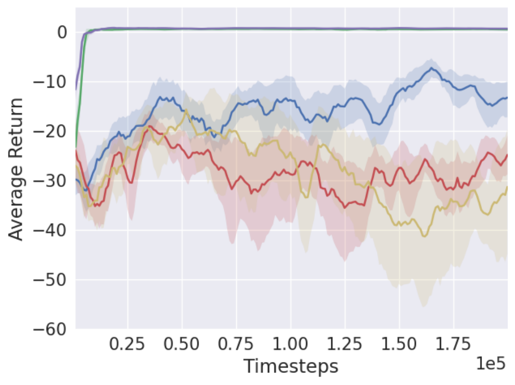
(c) PPO evaluated without RTA Average Return

(d) PPO evaluated without RTA Average Success

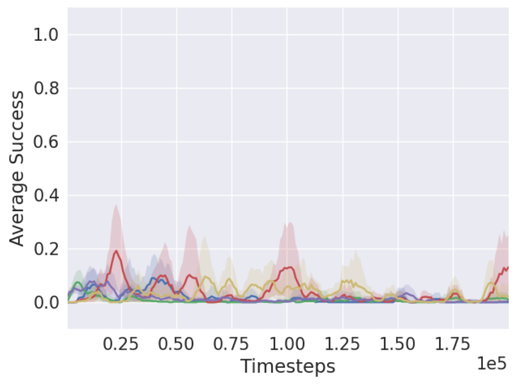
Figure V.10: Results collected from experiments run in the 2D Spacecraft Docking environment with an explicit simplex RTA. Each curve represents the average 10 trials and the shaded region is the 95% confidence interval about the mean.

Table V.7: PPO 2D Spacecraft Docking Explicit Simplex

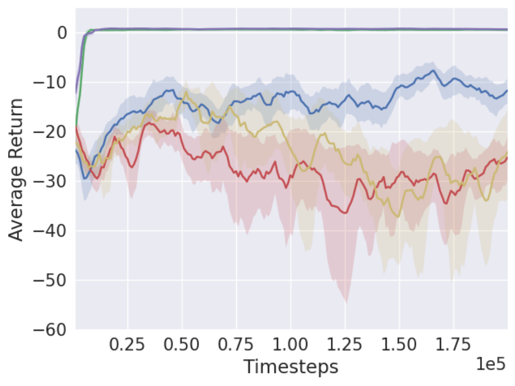
Configuration	RTA	Return	Length	Success	Interventions/Violations	Correction
baseline	on	-25.5058 ± 7.0824	451.3160 ± 228.9647	0.5960 ± 0.4907	269.1270 ± 68.7516	0.8521 ± 0.2730
	off	-19.5788 ± 7.5443	261.0470 ± 255.9112	0.4740 ± 0.4993	132.0560 ± 59.6531	-
baseline punishment	on	1.7415 ± 0.7302	763.2820 ± 197.3288	0.6720 ± 0.4695	1.0440 ± 2.4430	0.0501 ± 0.0980
	off	1.7350 ± 0.7505	758.7400 ± 198.3093	0.6830 ± 0.4653	1.1420 ± 2.7484	-
RTA no punishment	on	-22.5140 ± 10.7681	348.6550 ± 85.4651	0.8530 ± 0.3541	242.0030 ± 100.1411	0.9448 ± 0.3900
	off	-18.3351 ± 4.1843	230.2570 ± 68.7432	0.8320 ± 0.3739	154.8820 ± 42.6876	-
RTA punishment	on	2.0770 ± 0.3609	710.4490 ± 157.6271	0.8820 ± 0.3226	1.0830 ± 1.9236	0.0595 ± 0.0981
	off	2.0637 ± 0.3659	703.4900 ± 155.3529	0.8920 ± 0.3104	1.2450 ± 2.2545	-
RTA Corrected Action	on	-38.8204 ± 23.3258	370.8200 ± 230.2876	0.0 ± 0.0	370.8060 ± 230.2876	12.8179 ± 3.2593
	off	-22.7426 ± 9.9165	18.0930 ± 4.9153	0.0 ± 0.0	18.0720 ± 4.8995	-



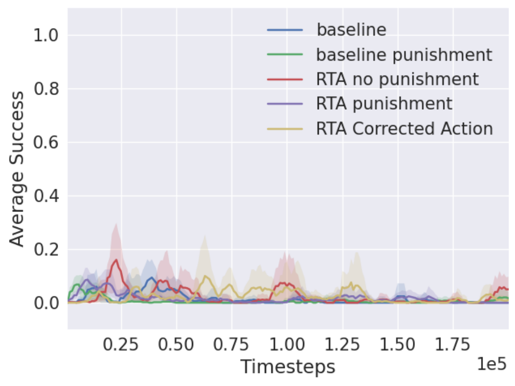
(a) SAC evaluated with RTA Average Return



(b) SAC evaluated with RTA Average Success



(c) SAC evaluated without RTA Average Return



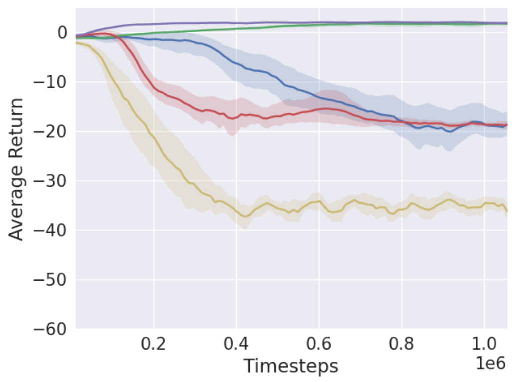
(d) SAC evaluated without RTA Average Success

Figure V.11: Results collected from experiments run in the 2D Spacecraft Docking environment with an explicit simplex RTA. Each curve represents the average 10 trials and the shaded region is the 95% confidence interval about the mean.

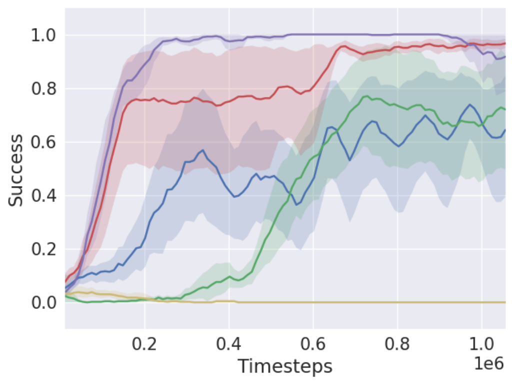
Table V.8: SAC 2D Spacecraft Docking Explicit Simplex

Configuration	RTA	Return	Length	Success	Interventions/Violations	Correction
baseline	on	-12.5070 ± 5.3815	980.3370 ± 95.3160	0.0170 ± 0.1293	129.1350 ± 54.4923	0.4160 ± 0.0389
	off	-44.4387 ± 18.4755	936.0760 ± 201.4800	0.0 ± 0.0	375.4500 ± 150.4911	-
baseline punishment	on	-0.7163 ± 1.1067	998.6170 ± 18.4751	0.0030 ± 0.0547	10.8460 ± 12.2911	0.2622 ± 0.1204
	off	-1.4651 ± 2.3989	995.6790 ± 41.1139	0.0050 ± 0.0705	17.8680 ± 24.0389	-
RTA no punishment	on	-14.5861 ± 4.6865	943.9070 ± 160.2614	0.0380 ± 0.1912	148.0320 ± 50.7698	0.4289 ± 0.0300
	off	-55.0964 ± 28.7844	639.9500 ± 320.4161	0.0080 ± 0.0891	397.9490 ± 210.2359	-
RTA punishment	on	-2.2407 ± 1.5318	994.1500 ± 53.0907	0.0010 ± 0.0316	25.4220 ± 16.9404	0.3336 ± 0.0755
	off	-5.9313 ± 5.0539	996.6070 ± 35.4610	0.0040 ± 0.0631	59.2540 ± 48.6995	-
RTA Corrected Action	on	-19.8202 ± 14.5191	927.6350 ± 179.7466	0.0580 ± 0.2337	199.4710 ± 144.4070	0.4516 ± 0.0666
	off	-43.6724 ± 20.6484	588.6050 ± 361.0214	0.0050 ± 0.0705	295.2280 ± 154.5340	-

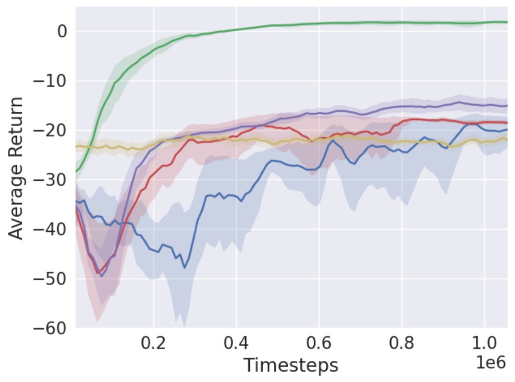
V.7.3 2D Spacecraft Docking Explicit ASIF



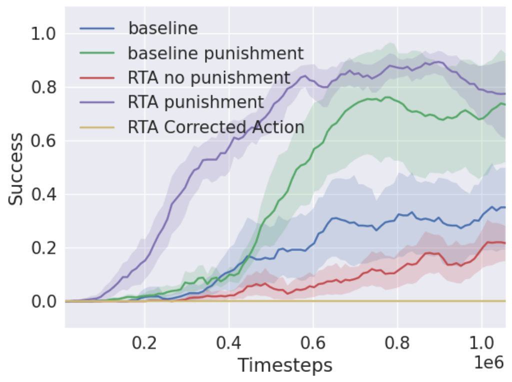
(a) PPO evaluated with RTA Average Return



(b) PPO evaluated with RTA Average Success



(c) PPO evaluated without RTA Average Return

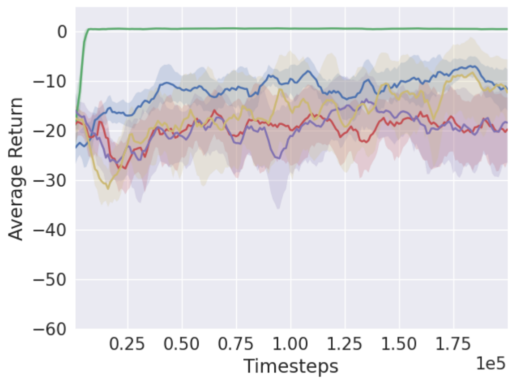


(d) PPO evaluated without RTA Average Success

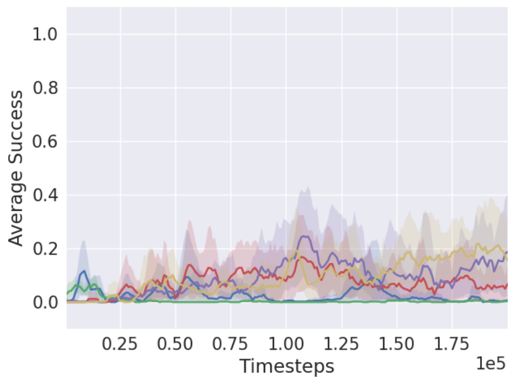
Figure V.12: Results collected from experiments run in the 2D Spacecraft Docking environment with an explicit ASIF RTA. Each curve represents the average 10 trials and the shaded region is the 95% confidence interval about the mean.

Table V.9: PPO 2D Spacecraft Docking Explicit ASIF

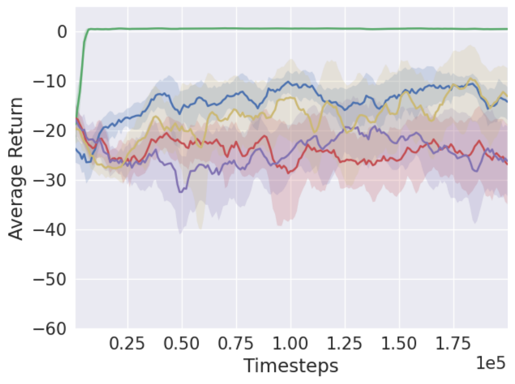
Configuration	RTA	Return	Length	Success	Interventions/Violations	Correction
baseline	on	-19.3937 ± 8.3859	496.2690 ± 276.8729	0.6100 ± 0.4877	320.1170 ± 76.3203	0.9750 ± 0.3698
	off	-19.6417 ± 8.6105	288.4830 ± 318.4605	0.3710 ± 0.4831	126.4670 ± 64.2468	-
baseline punishment	on	1.6885 ± 0.7824	794.0 ± 174.0669	0.6830 ± 0.4653	135.3380 ± 47.6325	0.2399 ± 0.0411
	off	1.7063 ± 0.8210	770.9680 ± 187.9315	0.6920 ± 0.4617	1.1970 ± 2.4409	-
RTA no punishment	on	-18.8412 ± 3.9873	327.1660 ± 35.9863	0.9570 ± 0.2029	287.7570 ± 32.6175	0.9100 ± 0.1651
	off	-18.4948 ± 2.2503	137.2860 ± 31.5151	0.2320 ± 0.4221	126.7610 ± 19.5246	-
RTA punishment	on	1.8265 ± 0.7208	512.1800 ± 150.3579	0.9330 ± 0.2500	198.6380 ± 45.1417	0.2718 ± 0.0470
	off	-15.3082 ± 5.7940	391.2710 ± 193.5872	0.7970 ± 0.4022	152.7720 ± 47.8269	-
RTA Corrected Action	on	-35.2004 ± 20.0159	321.1060 ± 181.7095	0.0 ± 0.0	321.1060 ± 181.7095	9.7873 ± 3.5195
	off	-22.4530 ± 9.9569	20.7170 ± 6.3494	0.0 ± 0.0	20.5450 ± 6.2174	-



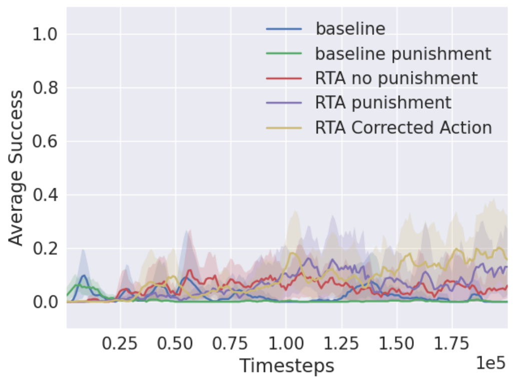
(a) SAC evaluated with RTA Average Return



(b) SAC evaluated with RTA Average Success



(c) SAC evaluated without RTA Average Return



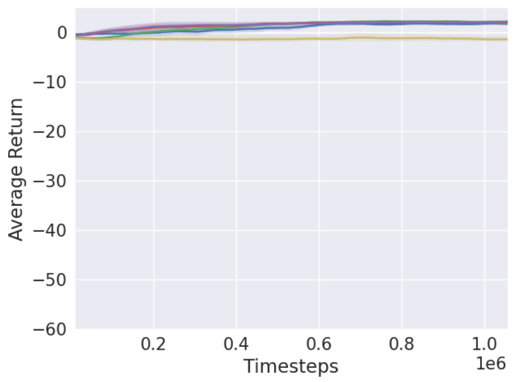
(d) SAC evaluated without RTA Average Success

Figure V.13: Results collected from experiments run in the 2D Spacecraft Docking environment with an explicit ASIF RTA. Each curve represents the average 10 trials and the shaded region is the 95% confidence interval about the mean.

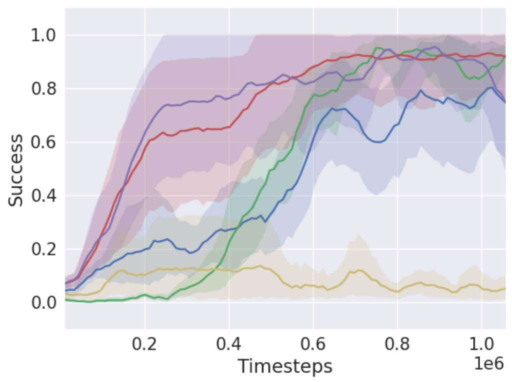
Table V.10: SAC 2D Spacecraft Docking Explicit ASIF

Configuration	RTA	Return	Length	Success	Interventions/Violations	Correction
baseline	on	0.0216 ± 0.9744	968.7900 ± 127.7141	0.0320 ± 0.1760	280.0150 ± 76.8260	0.3916 ± 0.0454
	off	-45.9935 ± 21.5150	890.9630 ± 269.6588	0.0010 ± 0.0316	376.5710 ± 170.1958	-
baseline punishment	on	0.1026 ± 0.2979	999.5220 ± 10.2286	0.0 ± 0.0	151.7450 ± 25.1203	0.3134 ± 0.0211
	off	-0.9064 ± 1.0850	995.0900 ± 51.4919	0.0 ± 0.0	12.0070 ± 11.5531	-
RTA no punishment	on	-0.1290 ± 0.6667	974.0790 ± 107.3582	0.0010 ± 0.0316	257.1470 ± 44.1387	0.3755 ± 0.0204
	off	-37.6844 ± 22.5865	374.3540 ± 225.3516	0.0 ± 0.0	250.7620 ± 155.8610	-
RTA punishment	on	-0.0374 ± 0.5376	987.6740 ± 73.9589	0.0020 ± 0.0447	264.8470 ± 47.4221	0.3788 ± 0.0252
	off	-41.9909 ± 23.4668	428.7950 ± 266.7232	0.0020 ± 0.0447	282.9180 ± 171.1889	-
RTA Corrected Action	on	0.2143 ± 0.5982	990.1150 ± 56.8130	0.0280 ± 0.1650	252.5090 ± 53.4003	0.3716 ± 0.0291
	off	-36.1541 ± 18.2568	598.6270 ± 325.5361	0.0120 ± 0.1089	269.8170 ± 136.6618	-

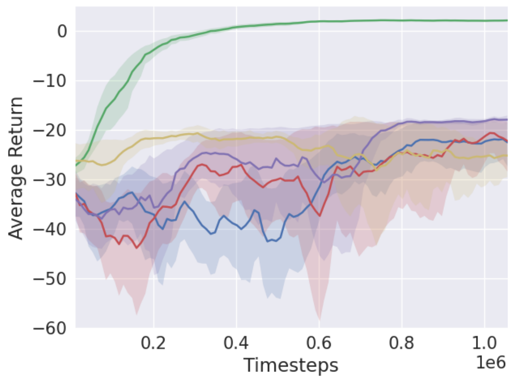
V.7.4 2D Spacecraft Docking Implicit Simplex



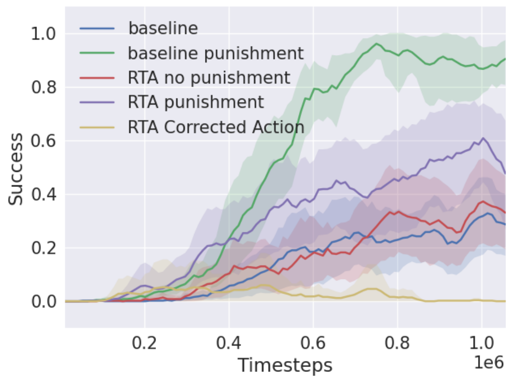
(a) PPO evaluated with RTA Average Return



(b) PPO evaluated with RTA Average Success



(c) PPO evaluated without RTA Average Return

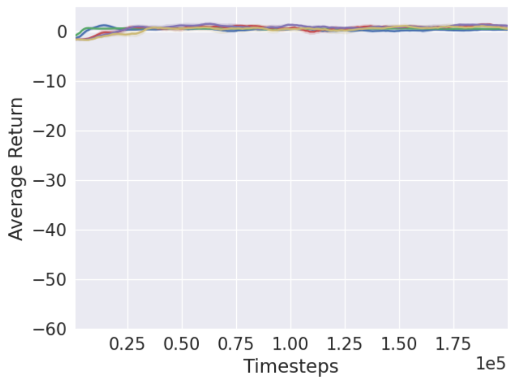


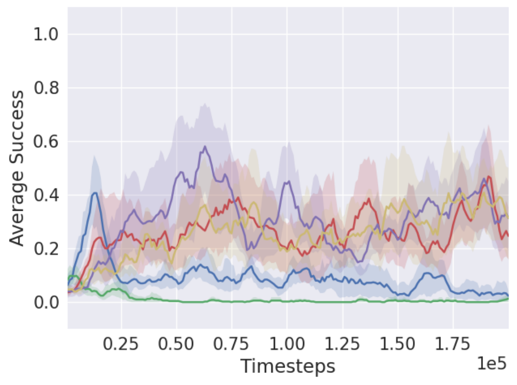
(d) PPO evaluated without RTA Average Success

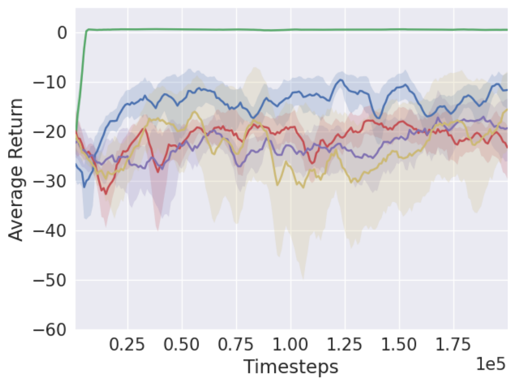
Figure V.14: Results collected from experiments run in the 2D Spacecraft Docking environment with an implicit simplex RTA. Each curve represents the average 10 trials and the shaded region is the 95% confidence interval about the mean.

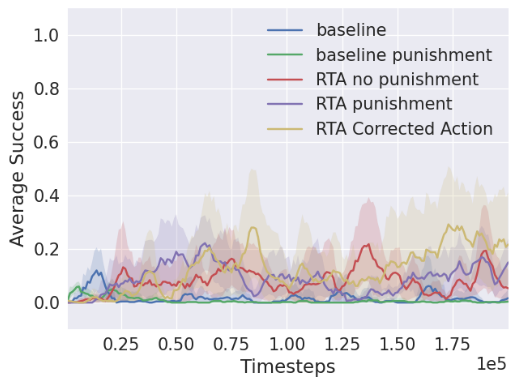
Table V.11: PPO 2D Spacecraft Docking Implicit Simplex

Configuration	RTA	Return	Length	Success	Interventions/Violations	Correction
baseline	on	1.7324 ± 0.5743	628.6310 ± 233.1921	0.7340 ± 0.4419	190.7160 ± 87.9536	2.5410 ± 0.4201
	off	-22.1098 ± 10.9814	320.5170 ± 360.2277	0.2660 ± 0.4419	143.9450 ± 79.5162	-
baseline punishment	on	2.2181 ± 0.3315	709.3360 ± 158.4366	0.9100 ± 0.2862	0.6080 ± 1.1218	0.6372 ± 0.9141
	off	2.1394 ± 0.3291	697.2640 ± 151.6598	0.9270 ± 0.2601	0.9660 ± 2.2147	-
RTA no punishment	on	2.0223 ± 0.3853	483.4820 ± 159.6581	0.9200 ± 0.2713	137.0720 ± 37.9305	2.1929 ± 0.2068
	off	-22.3856 ± 15.3886	176.1880 ± 147.2843	0.3410 ± 0.4740	147.8420 ± 102.7799	-
RTA punishment	on	1.8153 ± 0.5999	604.7220 ± 252.2543	0.7310 ± 0.4434	108.0690 ± 54.2185	2.0565 ± 0.2571
	off	-17.9896 ± 5.1619	415.3470 ± 341.2807	0.4600 ± 0.4984	147.5900 ± 40.1991	-
RTA Corrected Action	on	-1.3983 ± 1.2348	706.2610 ± 303.1083	0.0230 ± 0.1499	456.0030 ± 333.3891	4.8626 ± 2.1863
	off	-27.0715 ± 19.0662	132.6590 ± 234.5261	0.0 ± 0.0	90.0710 ± 154.2038	-


(a) SAC evaluated with RTA Average Return


(b) SAC evaluated with RTA Average Success


(c) SAC evaluated without RTA Average Return



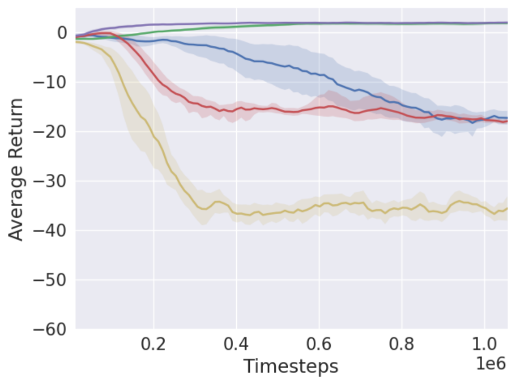
(d) SAC evaluated without RTA Average Success

Figure V.15: Results collected from experiments run in the 2D Spacecraft Docking environment with an implicit simplex RTA. Each curve represents the average 10 trials and the shaded region is the 95% confidence interval about the mean.

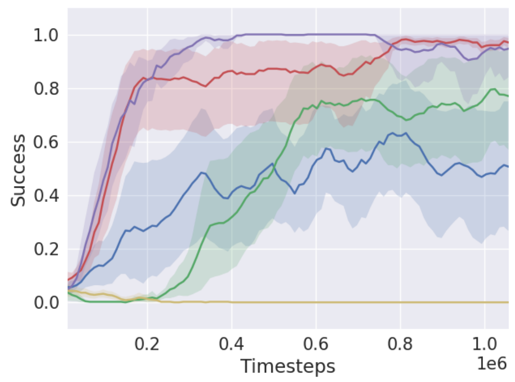
Table V.12: SAC 2D Spacecraft Docking Implicit Simplex

Configuration	RTA	Return	Length	Success	Interventions/Violations	Correction
baseline	on	0.4104 ± 0.4594	992.1540 ± 61.4690	0.0170 ± 0.1293	40.1990 ± 29.2195	2.0015 ± 0.1018
	off	-36.9676 ± 17.9372	963.3310 ± 160.9444	0.0010 ± 0.0316	312.0830 ± 141.1573	-
baseline punishment	on	0.2988 ± 0.4151	996.6570 ± 34.9140	0.0 ± 0.0	5.1930 ± 4.0115	1.8304 ± 0.5606
	off	-1.1718 ± 1.6220	994.9620 ± 50.3511	0.0 ± 0.0	14.5570 ± 16.7634	-
RTA no punishment	on	0.5268 ± 0.5937	985.8930 ± 77.3721	0.0260 ± 0.1591	45.5030 ± 14.1846	2.0088 ± 0.0616
	off	-49.2378 ± 26.5212	575.5730 ± 312.6387	0.0090 ± 0.0944	356.9840 ± 198.7087	-
RTA punishment	on	0.4419 ± 0.6550	980.7460 ± 98.2139	0.0050 ± 0.0705	43.3130 ± 12.2646	2.0111 ± 0.0616
	off	-47.3890 ± 26.1510	525.6140 ± 298.1644	0.0090 ± 0.0944	341.2960 ± 193.6923	-
RTA Corrected Action	on	0.6113 ± 0.6937	982.4190 ± 82.0556	0.0410 ± 0.1983	46.1820 ± 21.0099	2.0038 ± 0.0626
	off	-38.0627 ± 19.2859	598.8810 ± 311.1165	0.0350 ± 0.1838	286.5530 ± 145.0794	-

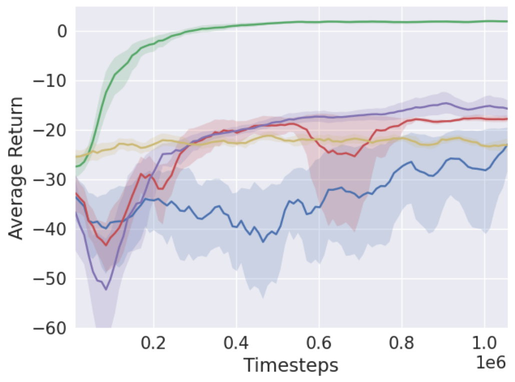
V.7.5 2D Spacecraft Docking Implicit ASIF



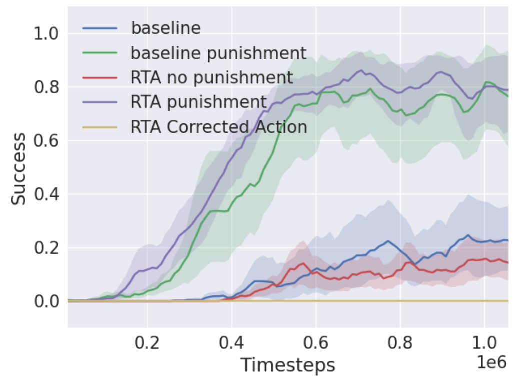
(a) PPO evaluated with RTA Average Return



(b) PPO evaluated with RTA Average Success



(c) PPO evaluated without RTA Average Return

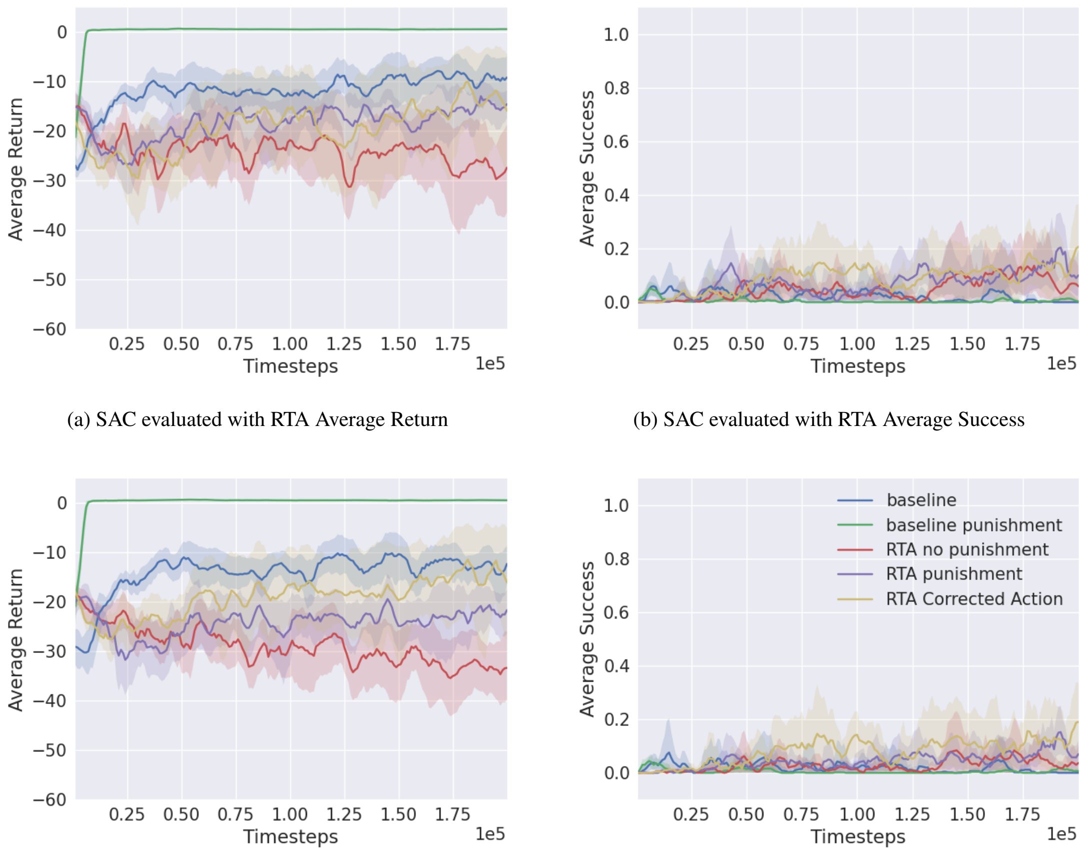


(d) PPO evaluated without RTA Average Success

Figure V.16: Results collected from experiments run in the 2D Spacecraft Docking environment with an implicit ASIF RTA. Each curve represents the average 10 trials and the shaded region is the 95% confidence interval about the mean.

Table V.13: PPO 2D Spacecraft Docking Implicit ASIF

Configuration	RTA	Return	Length	Success	Interventions/Violations	Correction
baseline	on	-17.7031 ± 8.5565	601.9800 ± 318.1612	0.4870 ± 0.4998	337.4270 ± 80.0928	0.8393 ± 0.2897
	off	-23.4315 ± 13.5040	385.0710 ± 393.7963	0.1980 ± 0.3985	143.9770 ± 99.2892	-
baseline punishment	on	1.8864 ± 0.5720	751.0750 ± 157.2639	0.7810 ± 0.4136	138.1540 ± 44.0523	0.2437 ± 0.0398
	off	1.8927 ± 0.5791	733.8830 ± 175.6035	0.7550 ± 0.4301	0.9690 ± 1.8868	-
RTA no punishment	on	-17.5369 ± 4.5902	329.2390 ± 34.7500	0.9610 ± 0.1936	290.2000 ± 32.0983	0.8294 ± 0.1533
	off	-17.8740 ± 2.3042	123.9470 ± 36.0346	0.1880 ± 0.3907	114.9080 ± 25.0518	-
RTA punishment	on	1.9883 ± 0.3713	502.9990 ± 143.2604	0.9480 ± 0.2220	204.7170 ± 37.7863	0.2781 ± 0.0398
	off	-15.8753 ± 5.1271	364.4110 ± 181.5286	0.7690 ± 0.4215	156.4490 ± 41.2289	-
RTA Corrected Action	on	-36.3506 ± 21.1084	335.6310 ± 192.6612	0.0 ± 0.0	335.6310 ± 192.6612	9.6171 ± 3.2334
	off	-22.0632 ± 9.6757	21.0460 ± 7.2165	0.0 ± 0.0	20.8590 ± 7.0182	-



(a) SAC evaluated with RTA Average Return

(b) SAC evaluated with RTA Average Success

(c) SAC evaluated without RTA Average Return

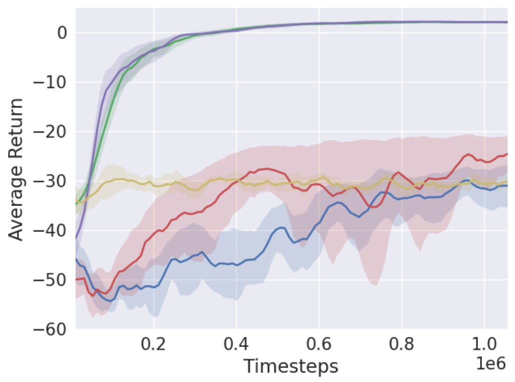
(d) SAC evaluated without RTA Average Success

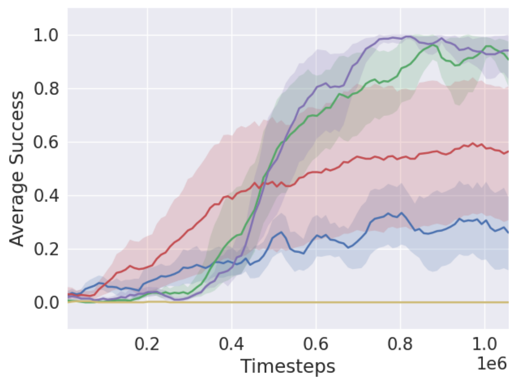
Figure V.17: Results collected from experiments run in the 2D Spacecraft Docking environment with an implicit ASIF RTA. Each curve represents the average 10 trials and the shaded region is the 95% confidence interval about the mean.

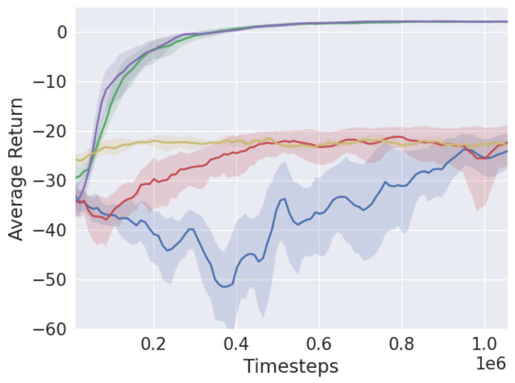
Table V.14: SAC 2D Spacecraft Docking Implicit ASIF

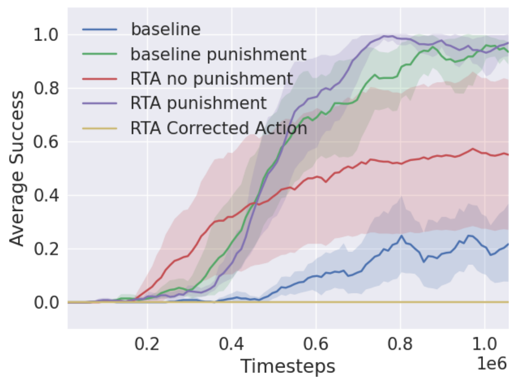
Configuration	RTA	Return	Length	Success	Interventions/Violations	Correction
baseline	on	0.0170 ± 0.6159	988.4880 ± 66.0788	0.0220 ± 0.1467	327.1520 ± 78.9583	0.3659 ± 0.0418
	off	-40.4752 ± 19.6528	945.6220 ± 187.7012	0.0030 ± 0.0547	335.9500 ± 154.2188	-
baseline punishment	on	-0.0926 ± 0.3824	998.7810 ± 21.0513	0.0 ± 0.0	242.3540 ± 94.6843	0.2916 ± 0.0280
	off	-0.6723 ± 1.0088	998.8650 ± 18.3413	0.0 ± 0.0	9.5740 ± 10.5377	-
RTA no punishment	on	0.0072 ± 0.5972	989.0330 ± 63.0413	0.0120 ± 0.1089	337.4620 ± 70.4624	0.3637 ± 0.0308
	off	-40.3231 ± 22.8186	388.8060 ± 237.8237	0.0030 ± 0.0547	266.6250 ± 160.5296	-
RTA punishment	on	-0.0248 ± 0.5306	992.9850 ± 54.3038	0.0020 ± 0.0447	322.7700 ± 69.0018	0.3560 ± 0.0272
	off	-45.2013 ± 24.9335	500.5040 ± 285.1929	0.0020 ± 0.0447	319.9860 ± 182.5422	-
RTA Corrected Action	on	0.0461 ± 0.7008	984.5460 ± 75.3451	0.0210 ± 0.1434	317.3800 ± 76.3632	0.3630 ± 0.0405
	off	-38.1003 ± 20.0277	474.9430 ± 292.3608	0.0110 ± 0.1043	261.0650 ± 143.3972	-

V.7.6 3D Spacecraft Docking Explicit Simplex


(a) PPO evaluated with RTA Average Return


(b) PPO evaluated with RTA Average Success


(c) PPO evaluated without RTA Average Return

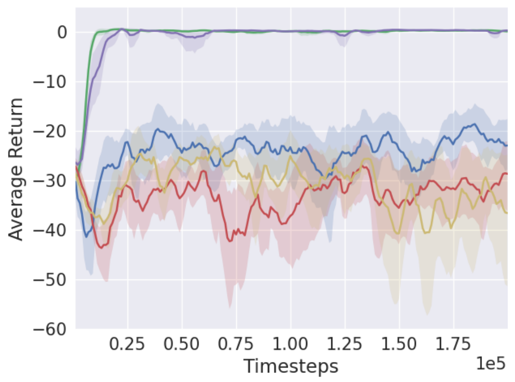


(d) PPO evaluated without RTA Average Success

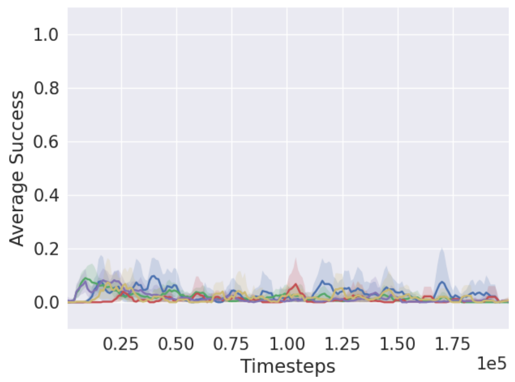
Figure V.18: Results collected from experiments run in the Docking3D environment with an explicit simplex RTA. Each curve represents the average 10 trials and the shaded region is the 95% confidence interval about the mean.

Table V.15: PPO 3D Spacecraft Docking Explicit Simplex

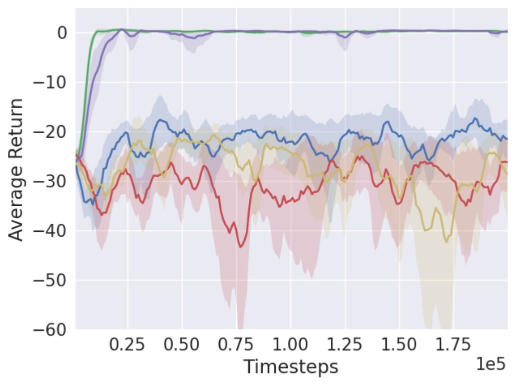
Configuration	RTA	Return	Length	Success	Interventions/Violations	Correction
baseline	on	-31.0188 ± 10.6629	518.1790 ± 321.9829	0.2820 ± 0.4500	193.3420 ± 67.8229	0.8483 ± 0.2922
	off	-23.6606 ± 10.0484	387.3470 ± 368.7029	0.2320 ± 0.4221	153.5120 ± 81.3324	-
baseline punishment	on	2.0348 ± 0.3485	700.8320 ± 150.7818	0.8990 ± 0.3013	0.0 ± 0.0	0.0 ± 0.0
	off	2.0182 ± 0.3761	714.7290 ± 147.2671	0.8870 ± 0.3166	0.9160 ± 1.9034	-
RTA no punishment	on	-24.4788 ± 10.7840	280.2370 ± 88.3712	0.5980 ± 0.4903	149.2880 ± 101.5997	0.9711 ± 0.8016
	off	-21.8075 ± 10.9196	190.2490 ± 111.3772	0.5510 ± 0.4974	134.1560 ± 72.1069	-
RTA punishment	on	2.0870 ± 0.3968	687.3150 ± 135.2182	0.9360 ± 0.2448	0.0 ± 0.0	0.0 ± 0.0
	off	2.0874 ± 0.3910	685.5820 ± 136.5485	0.9330 ± 0.2500	1.0390 ± 2.3608	-
RTA Corrected Action	on	-30.8903 ± 14.7595	230.2190 ± 122.1902	0.0 ± 0.0	230.2190 ± 122.1902	19.4718 ± 3.4466
	off	-22.6890 ± 8.2845	14.7530 ± 3.3308	0.0 ± 0.0	14.7530 ± 3.3308	-



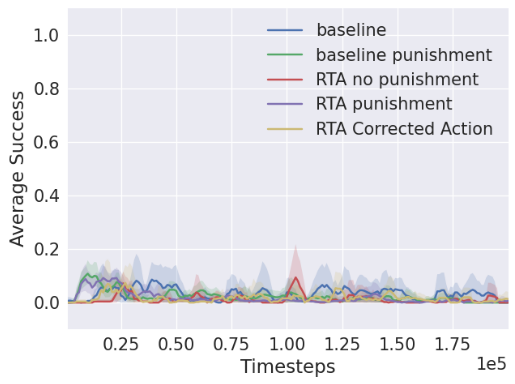
(a) SAC evaluated with RTA Average Return



(b) SAC evaluated with RTA Average Success



(c) SAC evaluated without RTA Average Return



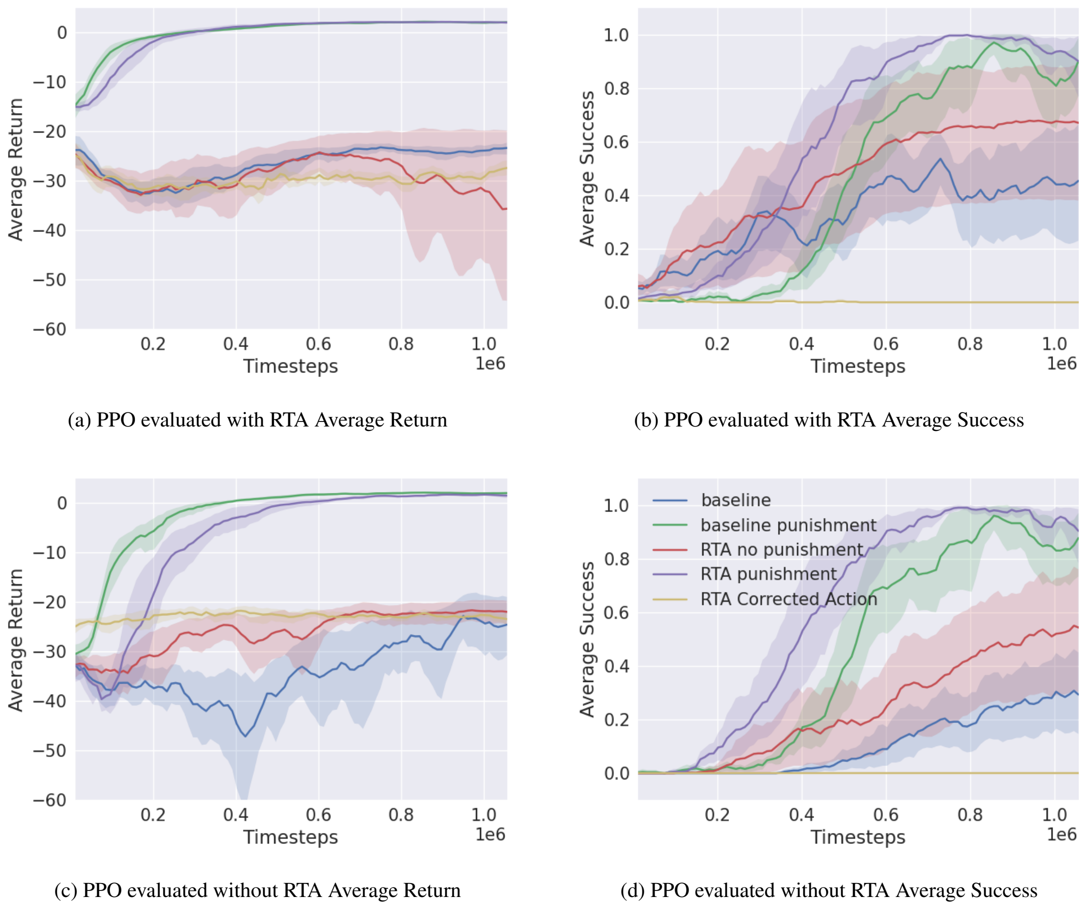
(d) SAC evaluated without RTA Average Success

Figure V.19: Results collected from experiments run in the Docking3D environment with an explicit simplex RTA. Each curve represents the average 10 trials and the shaded region is the 95% confidence interval about the mean.

Table V.16: SAC 3D Spacecraft Docking Explicit Simplex

Configuration	RTA	Return	Length	Success	Interventions/Violations	Correction
baseline	on	-45.5176 ± 17.8207	889.6050 ± 233.6027	0.0210 ± 0.1434	81.5790 ± 65.0452	0.4107 ± 0.1040
	off	-51.7261 ± 22.6411	793.0560 ± 312.3463	0.0050 ± 0.0705	404.6600 ± 179.4067	-
baseline punishment	on	-3.1201 ± 3.3789	984.0890 ± 78.1228	0.0 ± 0.0	0.5910 ± 2.4831	0.0378 ± 0.1168
	off	-3.0951 ± 3.7081	985.4280 ± 66.5032	0.0010 ± 0.0316	29.8100 ± 35.5730	-
RTA no punishment	on	-56.2666 ± 18.2059	894.0790 ± 228.7942	0.0340 ± 0.1812	113.1790 ± 63.4759	0.4365 ± 0.0638
	off	-65.2064 ± 30.6350	749.4010 ± 336.3577	0.0020 ± 0.0447	478.7300 ± 229.1926	-
RTA punishment	on	-3.2459 ± 4.9680	997.0710 ± 27.9703	0.0050 ± 0.0705	1.6550 ± 6.6895	0.0429 ± 0.1162
	off	-3.5462 ± 6.0729	995.9100 ± 36.4424	0.0050 ± 0.0705	35.5580 ± 58.1211	-
RTA Corrected Action	on	-57.1892 ± 23.8735	874.9890 ± 254.3602	0.0230 ± 0.1499	134.5720 ± 111.9885	0.4541 ± 0.1048
	off	-53.6893 ± 28.7788	639.5200 ± 380.3689	0.0030 ± 0.0547	379.5430 ± 226.4612	-

V.7.7 3D Spacecraft Docking Explicit ASIF



(a) PPO evaluated with RTA Average Return

(b) PPO evaluated with RTA Average Success

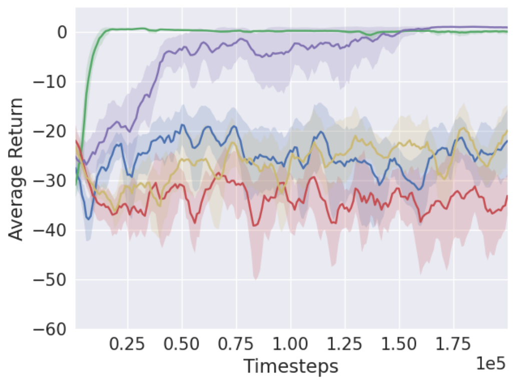
(c) PPO evaluated without RTA Average Return

(d) PPO evaluated without RTA Average Success

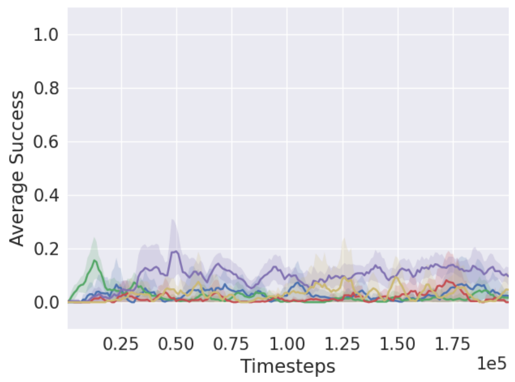
Figure V.20: Results collected from experiments run in the Docking3D environment with an explicit ASIF RTA. Each curve represents the average 10 trials and the shaded region is the 95% confidence interval about the mean.

Table V.17: PPO 3D Spacecraft Docking Explicit ASIF

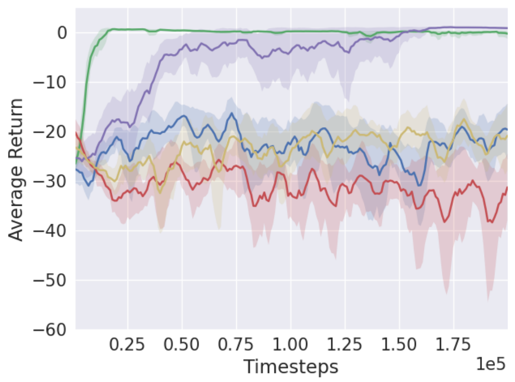
Configuration	RTA	Return	Length	Success	Interventions/Violations	Correction
baseline	on	-23.5395 ± 5.5692	520.2150 ± 335.7678	0.4380 ± 0.4961	262.1080 ± 81.3717	0.7496 ± 0.2788
	off	-25.9051 ± 15.2182	395.0710 ± 393.6692	0.2810 ± 0.4495	174.5970 ± 127.5959	-
baseline punishment	on	2.0252 ± 0.4779	719.1760 ± 154.5015	0.8760 ± 0.3296	61.9650 ± 14.8473	0.1813 ± 0.0324
	off	1.9796 ± 0.5287	699.8420 ± 156.6811	0.8890 ± 0.3141	1.2990 ± 3.2802	-
RTA no punishment	on	-35.7737 ± 31.4269	385.1380 ± 255.2133	0.6760 ± 0.4680	295.9280 ± 214.9787	0.8228 ± 0.4336
	off	-22.1710 ± 6.5638	161.8100 ± 77.1825	0.5380 ± 0.4986	133.3550 ± 59.4011	-
RTA punishment	on	1.9614 ± 0.5390	678.1530 ± 155.3601	0.8860 ± 0.3178	65.2410 ± 17.8011	0.1843 ± 0.0306
	off	1.5387 ± 1.0994	667.7730 ± 160.3803	0.8910 ± 0.3116	6.0220 ± 10.9016	-
RTA Corrected Action	on	-28.6656 ± 12.8821	196.4600 ± 92.5018	0.0 ± 0.0	196.4600 ± 92.5018	18.3608 ± 2.8570
	off	-22.9620 ± 8.5660	14.2520 ± 3.0771	0.0 ± 0.0	14.2440 ± 3.0875	-



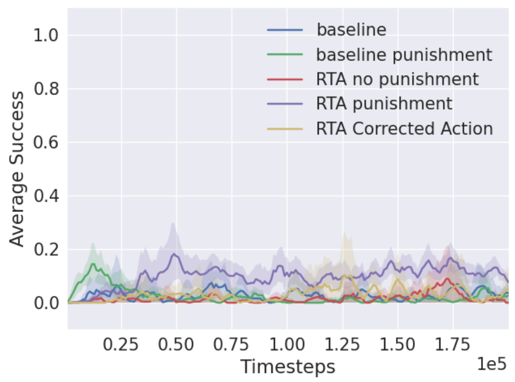
(a) SAC evaluated with RTA Average Return



(b) SAC evaluated with RTA Average Success



(c) SAC evaluated without RTA Average Return



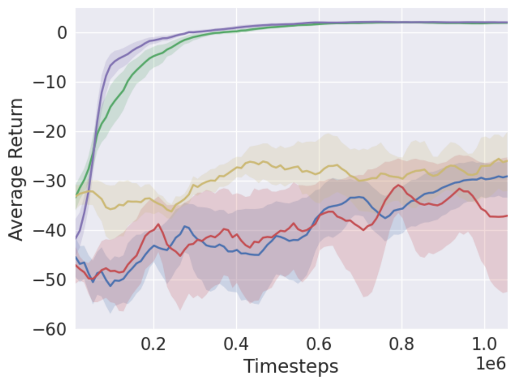
(d) SAC evaluated without RTA Average Success

Figure V.21: Results collected from experiments run in the Docking3D environment with an explicit ASIF RTA. Each curve represents the average 10 trials and the shaded region is the 95% confidence interval about the mean.

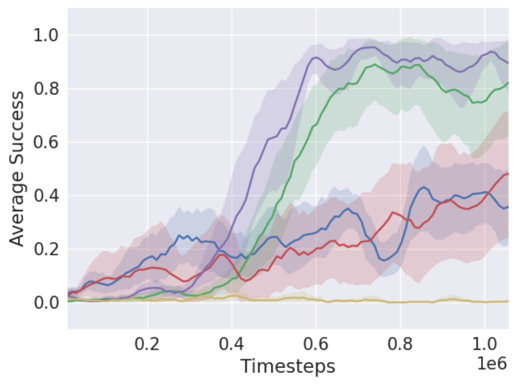
Table V.18: SAC 3D Spacecraft Docking Explicit ASIF

Configuration	RTA	Return	Length	Success	Interventions/Violations	Correction
baseline	on	-12.7579 ± 10.1785	915.5070 ± 202.0146	0.1110 ± 0.3141	204.1420 ± 90.2193	0.3788 ± 0.0620
	off	-38.0083 ± 20.4644	854.8270 ± 283.0951	0.0410 ± 0.1983	304.2500 ± 160.4455	-
baseline punishment	on	-0.1637 ± 0.6890	987.1430 ± 68.1994	0.0010 ± 0.0316	60.4100 ± 32.5684	0.2518 ± 0.0390
	off	-2.8927 ± 3.6149	986.2250 ± 70.9115	0.0050 ± 0.0705	29.0850 ± 35.7144	-
RTA no punishment	on	-14.5421 ± 7.3016	940.2470 ± 159.9236	0.0780 ± 0.2682	252.0550 ± 65.3287	0.3840 ± 0.0382
	off	-56.6587 ± 29.6890	568.9540 ± 316.5479	0.0050 ± 0.0705	399.5180 ± 211.1678	-
RTA punishment	on	-2.7094 ± 1.9604	999.2260 ± 16.0357	0.0 ± 0.0	156.4860 ± 42.5640	0.3229 ± 0.0292
	off	-29.8980 ± 14.0208	947.3790 ± 146.0030	0.0180 ± 0.1330	270.2530 ± 120.9608	-
RTA Corrected Action	on	-12.3856 ± 9.0203	905.9600 ± 213.5613	0.0410 ± 0.1983	217.5670 ± 78.7771	0.3668 ± 0.0465
	off	-48.8981 ± 23.9236	687.4480 ± 355.5327	0.0080 ± 0.0891	375.6960 ± 195.4146	-

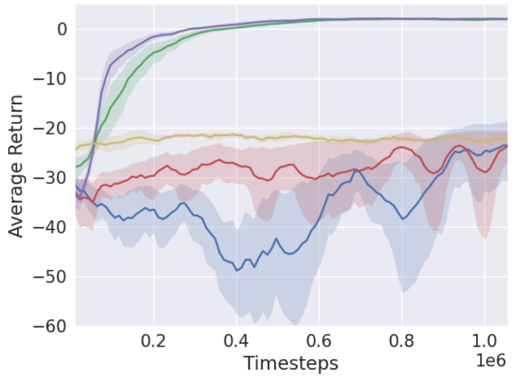
V.7.8 3D Spacecraft Docking Implicit Simplex



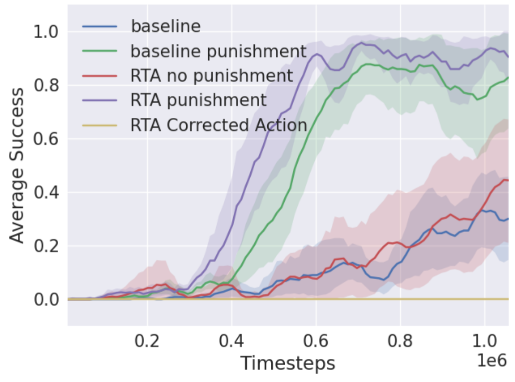
(a) PPO evaluated with RTA Average Return



(b) PPO evaluated with RTA Average Success



(c) PPO evaluated without RTA Average Return

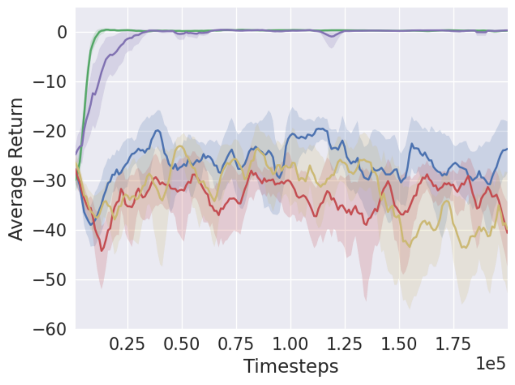


(d) PPO evaluated without RTA Average Success

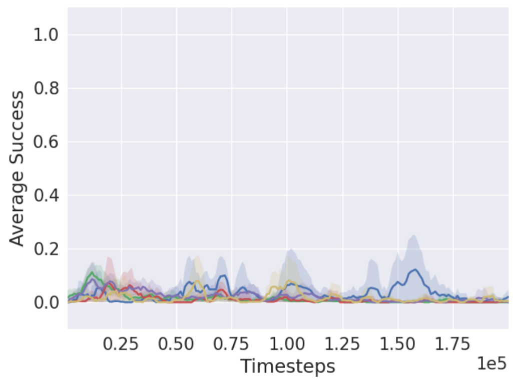
Figure V.22: Results collected from experiments run in the Docking3D environment with an implicit simplex RTA. Each curve represents the average 10 trials and the shaded region is the 95% confidence interval about the mean.

Table V.19: PPO 3D Spacecraft Docking Implicit Simplex

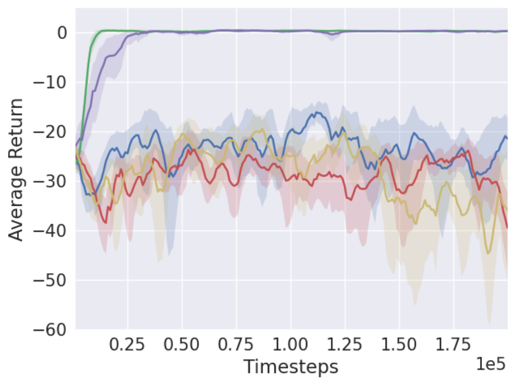
Configuration	RTA	Return	Length	Success	Interventions/Violations	Correction
baseline	on	-28.8368 ± 9.1682	444.4520 ± 311.7484	0.3520 ± 0.4776	177.5300 ± 58.5945	1.2214 ± 0.2807
	off	-23.7356 ± 14.1796	339.6470 ± 368.3187	0.2710 ± 0.4445	148.9460 ± 106.5690	-
baseline punishment	on	1.9298 ± 0.4878	729.8170 ± 164.2315	0.8050 ± 0.3962	0.0 ± 0.0	0.0 ± 0.0
	off	1.9062 ± 0.4930	741.4000 ± 163.1066	0.7880 ± 0.4087	0.8780 ± 1.9645	-
RTA no punishment	on	-33.2941 ± 22.1803	388.1860 ± 222.9841	0.5020 ± 0.5000	152.7050 ± 104.8435	1.2013 ± 0.4864
	off	-23.5098 ± 8.5427	214.6090 ± 125.9420	0.4380 ± 0.4961	152.0270 ± 57.3112	-
RTA punishment	on	2.0360 ± 0.3864	722.1020 ± 156.9345	0.8900 ± 0.3129	0.0010 ± 0.0316	0.0008 ± 0.0247
	off	2.0734 ± 0.3357	703.6070 ± 152.0538	0.9190 ± 0.2728	1.1600 ± 2.1805	-
RTA Corrected Action	on	-12.6732 ± 15.6686	705.8490 ± 305.7374	0.0290 ± 0.1678	705.4710 ± 305.5783	18.1370 ± 3.7118
	off	-22.2881 ± 8.1576	15.2590 ± 3.6905	0.0 ± 0.0	15.2590 ± 3.6905	-



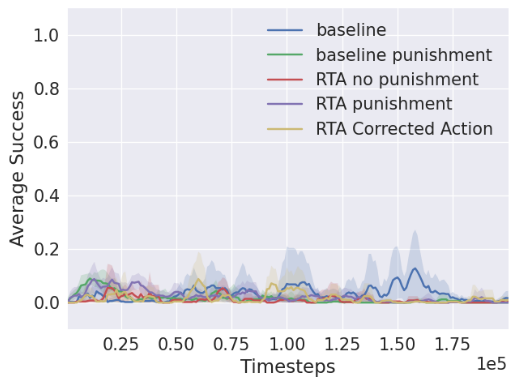
(a) SAC evaluated with RTA Average Return



(b) SAC evaluated with RTA Average Success



(c) SAC evaluated without RTA Average Return



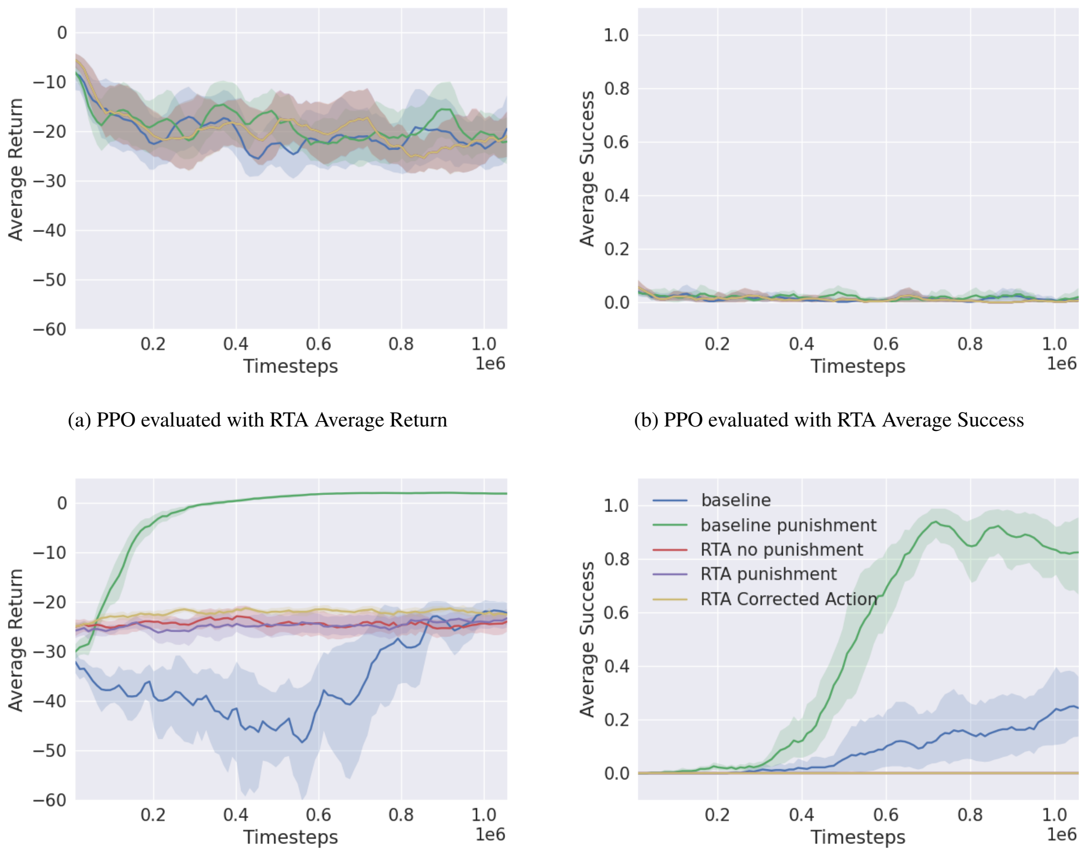
(d) SAC evaluated without RTA Average Success

Figure V.23: Results collected from experiments run in the Docking3D environment with an implicit simplex RTA. Each curve represents the average 10 trials and the shaded region is the 95% confidence interval about the mean.

Table V.20: SAC 3D Spacecraft Docking Implicit Simplex

Configuration	RTA	Return	Length	Success	Interventions/Violations	Correction
baseline	on	-39.8408 ± 15.1098	853.8830 ± 271.8064	0.0480 ± 0.2138	49.7620 ± 37.1512	1.0700 ± 0.2004
	off	-49.9209 ± 23.8915	772.6130 ± 329.1146	0.0090 ± 0.0944	384.6810 ± 189.9644	-
baseline punishment	on	-1.7770 ± 2.1333	996.0120 ± 39.0965	0.0010 ± 0.0316	0.1110 ± 0.7673	0.0623 ± 0.2595
	off	-1.7795 ± 2.1777	996.9430 ± 31.9320	0.0 ± 0.0	18.1090 ± 21.9634	-
RTA no punishment	on	-56.7316 ± 18.9754	886.5070 ± 231.2872	0.0530 ± 0.2240	81.5040 ± 47.9373	1.0850 ± 0.1024
	off	-69.4304 ± 35.3073	701.8620 ± 334.5834	0.0010 ± 0.0316	489.0240 ± 244.4675	-
RTA punishment	on	-2.3084 ± 2.7257	997.5530 ± 30.4823	0.0020 ± 0.0447	0.4140 ± 1.8554	0.1206 ± 0.3443
	off	-2.5016 ± 3.2310	998.3320 ± 22.5890	0.0030 ± 0.0547	25.6890 ± 32.5553	-
RTA Corrected Action	on	-54.9041 ± 21.5688	818.4390 ± 297.3325	0.0260 ± 0.1591	119.9790 ± 103.9273	1.1138 ± 0.0633
	off	-52.5477 ± 28.3996	510.3110 ± 374.1590	0.0 ± 0.0	332.9860 ± 220.6356	-

V.7.9 3D Spacecraft Docking Implicit ASIF



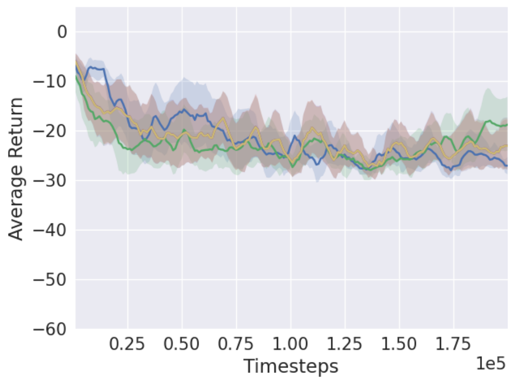
(a) PPO evaluated with RTA Average Return (b) PPO evaluated with RTA Average Success

(c) PPO evaluated without RTA Average Return (d) PPO evaluated without RTA Average Success

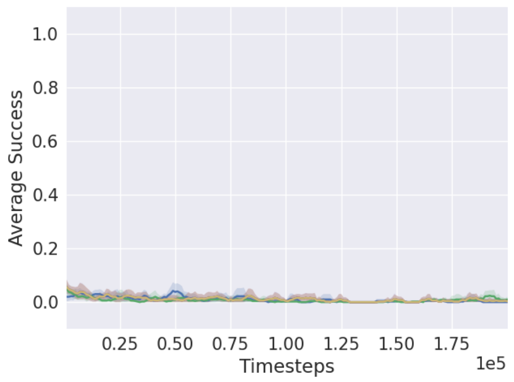
Figure V.24: Results collected from experiments run in the Docking3D environment with an implicit ASIF RTA. Each curve represents the average 10 trials and the shaded region is the 95% confidence interval about the mean.

Table V.21: PPO 3D Spacecraft Docking Implicit ASIF

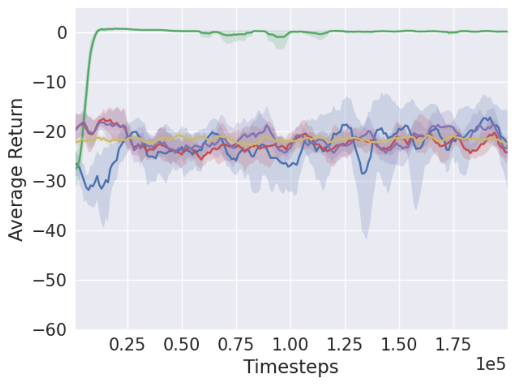
Configuration	RTA	Return	Length	Success	Interventions/Violations	Correction
baseline	on	-11.1833 ± 14.6859	670.6230 ± 303.9664	0.0260 ± 0.1591	670.6230 ± 303.9664	2.1712 ± 0.6998
	off	-21.7117 ± 8.4841	358.8620 ± 359.6777	0.2550 ± 0.4359	134.7320 ± 69.5541	-
baseline punishment	on	-11.2320 ± 15.6139	723.6400 ± 297.1046	0.0340 ± 0.1812	723.6400 ± 297.1046	2.0077 ± 0.7648
	off	1.8275 ± 0.5826	740.5620 ± 177.9170	0.7960 ± 0.4030	1.2820 ± 2.5820	-
RTA no punishment	on	-10.4772 ± 14.6715	727.0780 ± 304.3238	0.0290 ± 0.1678	727.0770 ± 304.3229	0.9596 ± 0.1598
	off	-24.7680 ± 8.6542	101.9490 ± 38.2915	0.0 ± 0.0	89.9680 ± 33.4132	-
RTA punishment	on	-10.4772 ± 14.6715	727.0780 ± 304.3238	0.0290 ± 0.1678	727.0740 ± 304.3218	0.7655 ± 0.1558
	off	-24.4860 ± 8.6466	88.4590 ± 32.2835	0.0 ± 0.0	77.6300 ± 30.2741	-
RTA Corrected Action	on	-10.4772 ± 14.6715	727.0780 ± 304.3238	0.0290 ± 0.1678	727.0780 ± 304.3238	5.3637 ± 2.2937
	off	-21.8312 ± 8.9370	27.9090 ± 8.6748	0.0 ± 0.0	27.3040 ± 8.4486	-



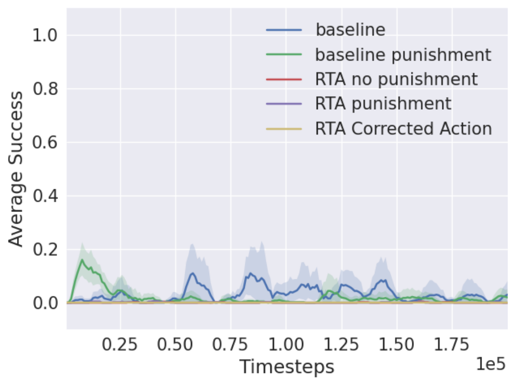
(a) SAC evaluated with RTA Average Return



(b) SAC evaluated with RTA Average Success



(c) SAC evaluated without RTA Average Return



(d) SAC evaluated without RTA Average Success

Figure V.25: Results collected from experiments run in the Docking3D environment with an implicit ASIF RTA. Each curve represents the average 10 trials and the shaded region is the 95% confidence interval about the mean.

Table V.22: SAC 3D Spacecraft Docking Implicit ASIF

Configuration	RTA	Return	Length	Success	Interventions/Violations	Correction
baseline	on	-10.1428 ± 14.3804	721.7680 ± 289.5521	0.0350 ± 0.1838	721.7680 ± 289.5521	1.0391 ± 0.0697
	off	-49.4634 ± 24.9526	805.9750 ± 312.1980	0.0050 ± 0.0705	384.3810 ± 188.4363	-
baseline punishment	on	-13.8266 ± 16.6181	643.0790 ± 310.5455	0.0220 ± 0.1467	643.0790 ± 310.5455	1.0190 ± 0.0795
	off	-3.7864 ± 5.3072	974.0050 ± 109.7366	0.0020 ± 0.0447	36.7950 ± 50.6100	-
RTA no punishment	on	-9.2941 ± 13.4528	740.2390 ± 294.4016	0.0440 ± 0.2051	740.2390 ± 294.4016	0.9780 ± 0.0124
	off	-28.8167 ± 13.9128	233.3260 ± 108.2606	0.0 ± 0.0	180.0350 ± 88.3977	-
RTA punishment	on	-9.2941 ± 13.4528	740.2390 ± 294.4016	0.0440 ± 0.2051	740.2390 ± 294.4016	0.9796 ± 0.0123
	off	-29.0804 ± 13.9550	232.2120 ± 107.8407	0.0 ± 0.0	180.9220 ± 87.4997	-
RTA Corrected Action	on	-9.2941 ± 13.4528	740.2390 ± 294.4016	0.0440 ± 0.2051	740.2390 ± 294.4016	1.6682 ± 0.1922
	off	-21.4260 ± 7.9279	48.9950 ± 12.3313	0.0 ± 0.0	46.4630 ± 11.9167	-

CHAPTER VI

Training Agents to Satisfy Timed and Untimed Signal Temporal Logic Specifications with Reinforcement Learning

Reinforcement Learning (RL) depends critically on how reward functions are designed to capture intended behavior. However, traditional approaches are unable to represent temporal behavior, such as "do task 1 before doing task 2" or "do task 1 while avoiding region O." In the event they can represent temporal behavior, these reward functions are handcrafted by researchers and often require long hours of trial and error to shape the reward function just right to get the desired behavior. In these cases, the desired behavior and constraints are already known, the problem is generating a reward function to train the RL agent to satisfy that behavior. To address this issue, we present our tool, STLGym, for automatically converting timed and untimed specifications into a reward function. In this work, we show how STLGym can be used to train RL agents to satisfy specifications better than traditional approaches and to refine previously learned behavior to better match the specification[1].

VI.1 Introduction

Reinforcement Learning (RL) and Deep Reinforcement Learning (DRL) are fast-growing fields with growing impact, spurred by success in training agents to beat human experts in games like Go [94], Starcraft [95], and Gran Turismo [6]. These results support the claims from [106] that "reward is enough to drive behavior that exhibits abilities studied in natural and artificial intelligence."

However, traditional reward functions are Markovian by nature; mapping states, or states and actions, to scalar reward values without considering previous states or actions [107]. This Markovian nature is in direct conflict with designing reward functions that describe complex, temporally-extended behavior. For example, the task of opening a freezer door, taking something out, and then closing the freezer door cannot be represented by Markovian reward functions, because the success of taking something out of the freezer is dependent on opening the freezer door first. This problem also extends to the context of safety-critical systems, where the desired behavior might include never entering some region or responding to a situation within a specified amount of time.

Therefore, if we want to use RL and DRL to solve complex, temporally-extended problems, we need a new way of writing and defining reward functions. This is a challenging problem with growing interest as RL

[1]This chapter is based on prior work currently under review for the 2022 International Conference on Software Engineering and Formal Methods (SEFM).

research looks into new ways to formulate the reward function to solve these kinds of problems. The most promising approaches look at using temporal logic to write specifications describing the desired behavior, and then generating complex reward functions that help agents learn to satisfy the specifications. Temporal logics are formalism for specifying the desired behavior of systems that evolve over time [108]. Some approaches, like the one presented in this work, take advantage of quantitative semantics [109, 110, 111], while others construct reward machines that change how the reward function is defined depending on which states have been reached [112, 107, 13, 113].

Despite the many successes of these approaches, only one is able to incorporate timing constraints ([111]) and many only work with a few RL algorithms that require researchers to write up the problem in a custom format to work with the implementation provided. By ignoring timing constraints, the approaches leave out reactive specifications where systems need to respond within a specified amount of time, like in power systems.

Our contributions. In this chapter, we introduce STLGym, our tool for training RL agents to satisfy complex, temporally-extended problems with and without timing constraints using RL. To the best of our knowledge, and compared to related works discussed in Section VI.6, our approach is the first that allows users to train agents to satisfy timed and untimed specifications, evaluate how well their agents satisfy those specifications, and retrain agents that do not already satisfy the specifications. We demonstrate the features of our tool and explore some best practices in five interesting example case studies. Our results show STLGym is an effective tool for training RL agents to satisfy a variety of timed and untimed temporal logic specifications.

VI.2 Preliminaries

Reinforcement Learning (RL) is a form of machine learning in which an agent acts in an environment, learning through experience to increase its performance based on rewarded behavior. *Deep Reinforcement Learning* (DRL) is a newer branch of RL in which a neural network is used to approximate the behavior function, i.e. policy π. The environment can be comprised of any dynamical system, from video game simulations ([15, 95, 6]) to complex robotics scenarios ([5, 18, 16]). In this work, and to use our tool STLGym, the environment must be constructed using OpenAI's Gym API[2] [5].

Reinforcement learning is based on the *reward hypothesis* that all goals can be described by the maximization of expected *return*, i.e. the cumulative reward. During training, the agent chooses an action, u, based on the input observation, o. The action is then executed in the environment, updating the internal state, s, according to the plant dynamics. The agent then receives a scalar r, and the next observation vector, o'.

[2]Information on the specifics of the Gym API can be found at https://gym.openai.com/. Because the Gym API is widely used and often the required environment API for using open-source RL libraries, we do not consider this restriction a limitation of our work, but instead a good feature.

The process of executing an action and receiving a reward and next observation is referred to as a *timestep*. Relevant values, like the input observation, action, and reward are collected as a data tuple, i.e. *sample*, by the RL algorithm to update the current policy, π, to an improved policy, π^*. How often these updates are done is dependent on the RL algorithm.

The return is the sum of all rewards collected over the course of an *episode*. An episode is a finite sequence of states, observations, actions, and rewards starting from an initial state and ending when some terminal, i.e. *done*, conditions are met. In this work, we refer to different elements of the episode by their corresponding timestep, t. Thus, r_t is the reward value at timestep $t \in [0, T]$, where T is the final timestep in the episode.

VI.2.1 Signal Temporal Logic

Signal Temporal Logic (STL) was first introduced in [108] as an extension of previous temporal logics that allows for formalizing control-theoretic properties, properties of path-planning algorithms, and expressing timing constrains and causality relations.

STL specifications are defined recursively according to the *syntax*:

$$\phi := \psi | \neg\phi | \phi \wedge \varphi | \phi \vee \varphi | \phi \implies \psi | F\phi | G\phi | \phi U \psi | N\phi | F_{[a,b]}\phi | G_{[a,b]}\phi | \phi U_{[a,b]}\psi, \quad \text{(VI.1)}$$

where $a, b \in \mathbb{R}_{\geq 0}$ are finite non-negative time bounds; ϕ and φ are STL formulae; and ψ is a predicate in the form $f(w) < d$. In the predicate, $w : \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ is a signal, $f : \mathbb{R}^n \to \mathbb{R}$ is a function, and $d \in \mathbb{R}$ is a constant. The Boolean operators $\neg$, $\wedge$, $\vee$, $\veebar$, and $\implies$ are negation, conjunction, disjunction, and implication respectively; and the temporal operators F, G, U, and N refer to *Finally* (i.e. eventually), *Globally* (i.e. always), *Until*, and *Next* respectively.

w_t denotes the value of w at time t and (w, t) is the part of the signal that is a sequence of $w_{t'}$ for $t' \in [t, |w|)$, where $|w|$ is the end of the signal. The Boolean semantics of STL are recursively defined as follows:

$$
\begin{aligned}
(w,t) &\models (f(w) < d) && \Leftrightarrow && f(w_t) < d, \\
(w,t) &\models \neg(f(w) < d) && \Leftrightarrow && \neg((w,t) \models (f(w) < d), \\
(w,t) &\models \phi \wedge \varphi && \Leftrightarrow && (w,t) \models \phi \text{ and } (w,t) \models \varphi, \\
(w,t) &\models \phi \vee \varphi && \Leftrightarrow && (w,t) \models \phi \text{ or } (w,t) \models \varphi, \\
(w,t) &\models \phi \implies \varphi && \Leftrightarrow && \text{if } (w,t) \models \phi, \text{ then } (w,t) \models \varphi, \\
(w,t) &\models F\phi && \Leftrightarrow && \exists t' \in [t, |w|) \text{ s.t. } (w,t') \models \phi, \\
(w,t) &\models G\phi && \Leftrightarrow && (w,t') \models \phi \ \forall t' \in [t, |w|), \\
(w,t) &\models \phi U \varphi && \Leftrightarrow && \exists t_u \in [t, |w|) \text{ s.t. } (w,t_u) \models \varphi \wedge \forall t' \in [t, t_u)(w,t') \models \phi, \\
(w,t) &\models N\phi && \Leftrightarrow && (w,t+1) \models \phi, \\
(w,t) &\models F_{[a,b]}\phi && \Leftrightarrow && \exists t' \in [t+a, t+b] \text{ s.t. } (w,t') \models \phi, \\
(w,t) &\models G_{[a,b]}\phi && \Leftrightarrow && (w,t') \models \phi \ \forall t' \in [t+a, t+b], \\
(w,t) &\models \phi U_{[a,b]} \varphi && \Leftrightarrow && \exists t_u \in [t+a, t+b) \text{ s.t. } (w,t_u) \models \varphi \\
&&&&& \wedge \forall t' \in [t+a, t_u)(w,t') \models \phi.
\end{aligned}
$$

For a signal $(w,0)$, i.e. the whole signal starting at time 0, satisfying the timed predicate $F_{[a,b]}\phi$ means that "there exists a time within $[a,b]$ such that ϕ will eventually be true", and satisfying the timed predicate $G_{[a,b]}\phi$ means that "ϕ is true for all times between $[a,b]$". Satisfying the timed predicate $\phi U_{[a,b]}\varphi$ means "there exists a time within $[a,b]$ such that φ will be true, and *until* then, ϕ is true." Satisfying the untimed predicates have the same description as their timed counterpart, but with $a = 0$ and $b = |w|$.

VI.2.1.1 Quantitative Semantics

STL has a metric known as *robustness degree* or "degree of satisfaction" that quantifies how well a given signal w satisfies a given formula ϕ. The robustness degree is calculated recursively according to the *quantitative* *semantics*:

$$
\begin{aligned}
\rho(w,(f(w)<d),t) &= d-f(w_t),\\
\rho(w,(f(w)>d),t) &= f(w_t)-d,\\
\rho(w,(\phi\wedge\varphi),t) &= \min\big(\rho(w,\phi,t),\rho(w,\varphi,t)\big),\\
\rho(w,(\phi\vee\varphi),t) &= \max\big(\rho(w,\phi,t),\rho(w,\varphi,t)\big),\\
\rho(w,(\phi\implies\varphi),t) &= \max\big(-\rho(w,\phi,t),\rho(w,\varphi,t)\big),\\
\rho(w,F\phi,t) &= \max_{t'\in[t,|w|]}\rho(w,\phi,t'),\\
\rho(w,G\phi,t) &= \min_{t'\in[t,|w|]}\rho(w,\phi,t'),\\
\rho(w,\phi U\varphi,t) &= \max_{t_u\in[t,|w|]}\Big(\min\{\rho(w,\varphi,t_u),\min_{t'\in[t,t_u)}\big(\rho(w,\phi,t')\big)\}\Big),\\
\rho(w,N\phi,t) &= \rho(w,\phi,t+1),\\
\rho(w,F_{[a,b]}\phi,t) &= \max_{t'\in[t+a,t+b]}\rho(w,\phi,t'),\\
\rho(w,G_{[a,b]}\phi,t) &= \min_{t'\in[t+a,t+b]}\rho(w,\phi,t'),\\
\rho(w,\phi U_{[a,b]}\varphi,t) &= \max_{t_u\in[t+a,t+b]}\Big(\min\{\rho(w,\varphi,t_u),\min_{t'\in[t,t_u)}\big(\rho(w,\phi,t')\big)\}\Big).
\end{aligned}
$$

VI.2.1.2 Horizon Length

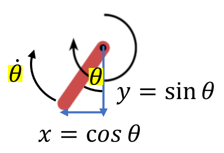
The horizon length of a specification, $hrz(\Phi)$ is the minimum signal length necessary to compute the robustness/degree of satisfaction. This is important for timed specifications where time bounds are provided. If the specification covers 10 timesteps, then the signal must have at least 10 values to analyze. The horizon length is computed recursively according to the equations:

$$
\begin{aligned}
hrz(\psi) &= 0, \\
hrz(\phi) &= b \text{ if} \phi = G_{[a,b]}\psi \text{ or } F_{[a,b]}\psi, \\
hrz(\neg\phi) &= hrz(\phi), \\
hrz(\phi \wedge \varphi) &= \max(hrz(\phi), hrz(\varphi)), \\
hrz(\phi \vee \varphi) &= \max(hrz(\phi), hrz(\varphi)), \\
hrz(F_{[a,b]}\phi) &= b + hrz(\phi), \\
hrz(G_{[a,b]}\phi) &= b + hrz(\phi), \\
hrz(\phi U_{[a,b]}\varphi) &= b + max(hrz(\phi), hrz(\varphi)), and \\
hrz(N\phi) &= hrz(\phi) + 1.
\end{aligned}
$$

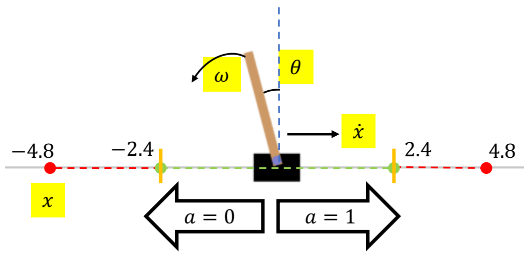
where $a, b \in \mathbb{R}_{\geq 0}$, ψ is a predicate in the form $f(w) < d$, and ϕ and φ are STL formulae. The final results is then determined as the maximum of the value computed using the formulas above and 1, thus $hrz(\Phi) \geq 1$. In order to calculate the robustness of a signal, the signal must exist, i.e. have a length greater than 0.

VI.3 Examples

In the remaining sections, we will be referring to these two example RL environments, *Pendulum* and *CartPole*, in order to explain how STLGym works and differs from other approaches. Figure VI.1 shows annotated screenshots of the simulated environments.



(a) *Pendulum-v0*



(b) *CartPole-v0*

Figure VI.1: Annotated screenshots showing the simulated environments, *Pendulum* (left) and *CartPole* (right), from the OpenAI Gym benchmarks [5].

VI.3.1 Pendulum

The Pendulum[3] environment, shown in Figure VI.1.a, consists of an inverted pendulum attached to a fixed point on one side. The agent's goal in this environment is to swing the free end of the pendulum to an upright position, $\theta = 0$, and maintain the position.

The interior plant model changes the state, $s = [\theta, \omega]$, according to the discrete dynamics

$$\begin{aligned} \omega_{t+1} &= \omega_t + (\frac{3g}{2l}\sin(\theta_t) + \frac{3u_t}{ml^2})\Delta t \\ \theta_{t+1} &= \theta_t + \omega_t \Delta t + (\frac{3g}{2l}\sin(\theta_t) + \frac{3u_t}{ml^2})\Delta t^2, \end{aligned} \tag{VI.2}$$

where $g = 10$, $l = 1$, $m = 1$, $\Delta t = 0.05$, and u_t is the control from the RL agent in the range $[-2, 2]$ applied as a torque about the fixed end of the pendulum. Additionally, within the environment the pendulum's angular velocity, ω, is clipped within the range $[-8, 8]$, and the angle from upright, θ, is aliased within $[-\pi, \pi]$ radians. θ is measured from upright and increases as the pendulum moves clockwise. The values θ, ω, and u are used to determine the observation, $o = [\cos(\theta), \sin(\theta), \omega]^T$ and the reward,

$$r_t = -\theta_t^2 - 0.1(\omega_t)^2 - 0.001(u_t)^2. \tag{VI.3}$$

For each episode, the pendulum is initialized according to a uniform distribution with $\theta \in [-\pi, \pi]$ and $\omega \in [-1, 1]$. The episode ends when 200 timesteps have occurred. That means T is always 200.

VI.3.2 CartPole

In the CartPole environment[4], a pole is attached to a cart moving along a frictionless track. The agent's goal in this environment is to keep the pole upright, $-12^\circ \leq \theta \leq 12^\circ$, and the cart within the bounds $-2.4 \leq x \leq 2.4$ until the time limit, $t = 200$, is reached. The agent accomplishes this goal by applying a leftward or rightward force to move the cart along the track. The agent's actions are discretized for a "bang-bang" control architecture that moves the cart left when $u = 0$ and right when $u = 1$.

The interior plant model changes the state, $s = [x, \dot{x}, \theta, \dot{\theta}]$, until a terminal condition is met. These terminal conditions are

1. the cart's position leaves the bounds $-2.4 \leq x \leq 2.4$,

2. the pole's angle is outside the bounds $-12^\circ \leq \theta \leq 12^\circ$, and

[3]This inverted pendulum environment is different from the inverted pendulum environment used in Chapter V. This is the original version produced in [5], while the other chapters use a modified version meant to make it more difficult for maintaining safety introduced in [40].

[4]The environment is based on the classic cart-pole system implemented for [114], where more information on the dynamics can be found.

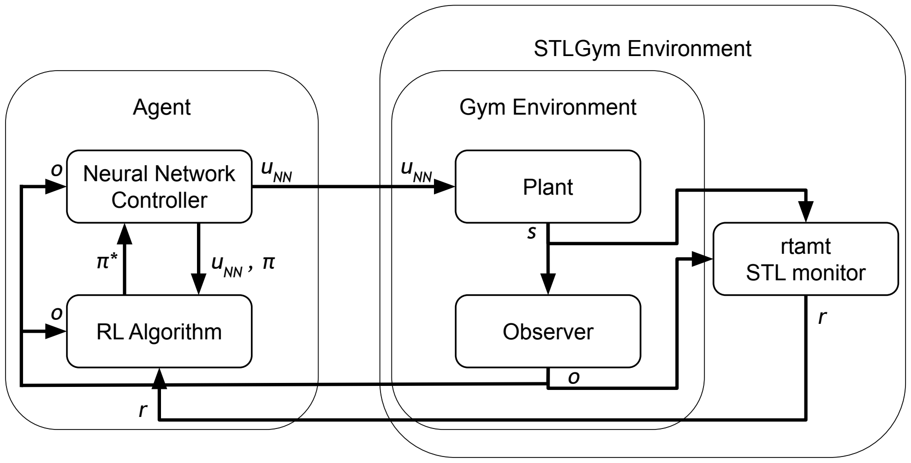


Figure VI.2: A representation of how STLGym wraps around the user's environment to record signals and replace the reward function.

3. the goal time limit is reached, i.e. $t = 200$.

The original, baseline reward function with this environment gives the agent $+1$ for every timestep the first two terminal conditions are not violated. Thus, the return for an episode is the same as the episode's length. To ensure the agent has a chance to complete at least one timestep successfully, each state element is initialized according to a uniform distribution in the range $[-0.05, 0.05]$. In this implementation, the observation is equivalent to the state, $o = s$.

VI.4 Our Approach: STLGym

Our approach focuses solely on augmenting the environment side of the RL process to add an STL monitor and replace the existing reward output with the calculated *robustness degree* as it relates to the desired specification(s), as shown in Figure VI.2. This process maintains the standards of the Gym API, so no changes to the RL algorithm are necessary to facilitate its use. As a result, our approach is *algorithm-agnostic*, since no modifications to the RL algorithm are required. Furthermore, since our approach makes use of existing environments, there is great potential for *retraining* learned policies to better optimize satisfying specifications. Our approach is implemented as the tool STLGym[5]

To use the tool, a user provides a YAML file[6] that defines the variable(s) that need to be recorded for the multivariate signal, w, and the specification(s) that the signal needs to satisfy. Additionally, the user must provide the underlying Gym environment that will be augmented. Provided these two inputs, STLGym generates a new Gym environment where the specified variables are recorded so RTAMT can monitor the

[5]STLGym implementation is available at https://github.com/nphamilton/stl-gym

[6]Example YAML files can be found at https://github.com/nphamilton/stl-gym/tree/main/examples and https://github.com/nphamilton/spinningup/tree/master/spinup/examples/sefm2022/configs.

defined STL specification and return the robustness degree as the reward function.

VI.4.1 Computing the Robustness Degree

To compute the robustness degree, we make use of RTAMT [115], a tool for monitoring STL specifications on recorded data. Given the recorded signal and specification, RTAMT computes the robustness degree according to the quantitative semantics described in Section VI.2.1.

VI.4.2 Allowable Specifications

Our approach is amenable to a wide range of specifications and supports the full range of semantics described in Section VI.2.1 in addition to any described in RTAMT's readme[7]. This includes both timed and untimed operators, adding more options than in the existing TLTL. Furthermore, our approach allows for specifications to be broken up into individual parts. For example, consider the CartPole example from Section VI.3.2. The desired behavior ("Keep the pole upright between $\pm 12^\circ$ and the cart within ± 2.4 units") can be written as

$$\Phi = G((|\theta| < 0.20944) \wedge (|x| < 2.4)) \tag{VI.4}$$

or it can be broken up into the individual components[8] and combined with a conjunction,

$$\begin{aligned} \phi_{angle} &= G(|\theta| < 0.20944) \\ \phi_{position} &= G(|x| < 2.4) \\ \Phi &= \phi_{angle} \wedge \phi_{position}. \end{aligned} \tag{VI.5}$$

These specifications, Equation VI.4 and Equation VI.5, are equivalent and allowable in both TLTL and STL-Gym. However, STLGym allows you to treat ϕ_{angle} and $\phi_{position}$ as individual specifications and automatically applies the conjunction. Any number of individual specifications can be defined, and the resulting specification the RL agent will learn to satisfy is the conjunction of all of them. Thus, if n specifications are provided, the RL agent will learn to satisfy

$$\Phi = \bigwedge_{i=0}^{n} \phi_i. \tag{VI.6}$$

VI.4.3 Calculating Reward

STLGym replaces any existing reward function in the environment with the *robustness degree* calculated using the provided specification(s) and RTAMT. If the user defines n specifications, $\phi_0, \phi_1, ..., \phi_n$ with corre-

[7]The RTAMT code is available at https://github.com/nickovic/rtamt

[8]These individual components were designed to allow users to write goal specifications and safety specifications as separate entities.

sponding weight values[9], $c_0, c_1, ..., c_n$, the reward function is constructed as

$$r_t = \sum_{i=0}^{n} c_i \rho(s, \phi_i, 0). \tag{VI.7}$$

We include optional weights to add more versatility. This allows for users to write specifications that build on each other, i.e. a specification is defined using another specification, but remove one from the reward function if desired by setting its weight to 0. Additionally, weights can help establish priorities in learning specifications. For example, we go back to the CartPole specification Equation VI.5. The reward function generated, according to the quantitative semantics described in Section VI.2.1, for the specification is

$$r_t = c_{angle} \min_{t' \in [0,t]} (0.20944 - |\theta_{t'}|) + c_{position} \min_{t' \in [0,t]} (2.4 - |x_{t'}|). \tag{VI.8}$$

If both $c_{angle} = c_{position} = 1$, then the maximum possible reward for satisfying both specifications is 2.60944. However, because the environment was designed to terminate if either specification is violated, if the agent only satisfies $\phi_{position}$ and lets the pole fall, the maximum possible reward is 2.4. Since the gain from keeping the pole upright is so small, we found it was often ignored. In contrast, if we make the weights $c_{angle} = 4.7746$ and $c_{position} = 0.41666$, then the maximum possible reward for satisfying both specifications is 2. If either of the specifications are ignored, the maximum possible reward drops to 1. Thus, we have enforced equal priority for satisfying the specifications.

VI.4.3.1 Dense vs Sparse

In addition to adding optional weights for each specification, STLGym allows users to specify if the reward function should be calculated densely or sparsely. This design decision was spurred on by the existing RL literature, where there are two main types of rewards utilized: dense and sparse. In the literature, dense rewards are returned at every timestep and are often a scalar representation of the agent's progresses toward the goal. For example, the baseline reward function in the Pendulum environment (Equation VI.3) is a dense reward. In contrast, sparse rewards are not returned at each timestep, but instead are only returned if certain conditions are met. For example, an agent receiving $+1$ for passing a checkpoint would be considered a sparse reward. Each of these reward types have their advantages for different tasks and algorithms. However, we make use of these terms to make our own definitions of dense and sparse reward as they relate to frequency.

Definition 10 (Dense Reward)**.** *In this setting, the robustness degree is computed at every allowable timestep. Thus, at each timestep, the reward returned to the agent is the robustness degree of the episode from the*

[9]If a weight is not defined by the user, the default is 1.

beginning to the current time step.

Definition 11 (Sparse Reward)**.** *In this setting, the robustness degree is only computed once at the end of the episode. In all timesteps before that, the reward is 0. Thus, the return is the robustness degree for the entire episode.*

From our experiments, we found dense rewards works better for training agents, while a sparse reward is better for evaluating their performance and understanding if they have successfully learned to satisfy the specification or not. An example is provided in Section VI.5.1.

VI.5 Example Case Studies

In this section, we describe 5 case studies we conducted using the environments described in Section VI.3.1 and Section VI.3.2[10]. In all of our case studies, we use the Proximal Policy Optimization (PPO) [26] algorithm for training, unless otherwise specified. These case studies were designed to highlight features of STLGym and try to identify some potential "best practices" for future use in other environments.

VI.5.1 Sparse vs Dense Reward

In this case study, we demonstrate why having the ability to swap between sparse and dense versions of our STL reward function is important. To this end, we train 30 agents in the pendulum environment from Section VI.3.1 to swing the pendulum upright and stay upright. Written as an STL specification, that is

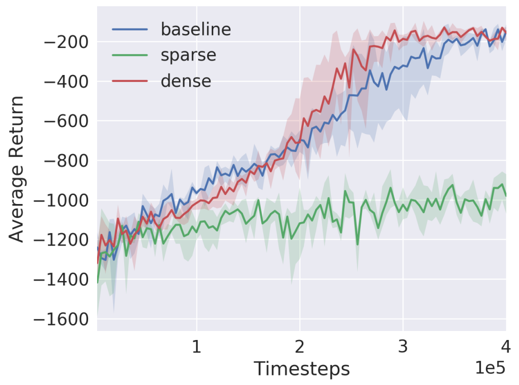
$$\Phi = F(G((|\theta| < 0.5))). \tag{VI.9}$$

Ten agents are trained using the baseline reward function (Equation VI.3), ten agents are trained with the sparse version of our STL reward function, and ten agents are trained with the dense version of our STL reward function. Using the quantitative semantics from Section VI.2.1, our tool automatically generates the reward function,

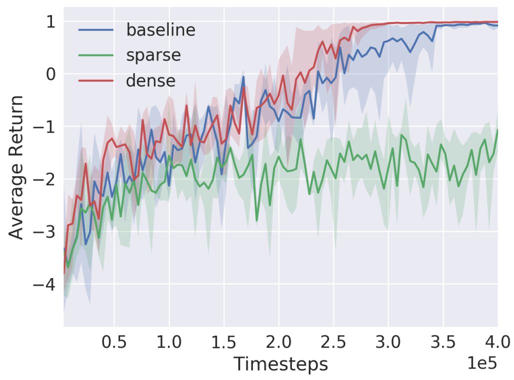
$$r_t = \max_{t' \in [0,t]} \big(\min_{t'' \in [t',t]} (0.5 - |\theta_{t''}|) \big). \tag{VI.10}$$

We show the sample complexity plots of training these 30 agents with the 3 different reward functions in Figure VI.3. Sample complexity is a measure of how quickly an RL agent learns optimal performance. Throughout training, the process is halted, and the agent is evaluated to see how well it performs with the policy learned so far. The policy is evaluated in ten episodes, and the performance, measured by the return, is recorded for the plot. A better sample complexity is shown by a higher return earlier in training. In

[10]All training scripts are available at https://github.com/nphamilton/spinningup/tree/master/spinup/examples/sefm2022



(a) Sample complexity of PPO agents trained in Pendulum environment. Evaluations done with Equation VI.3 for the reward function.



(b) Sample complexity of PPO agents trained in Pendulum environment. Evaluations done with Equation VI.10 defines sparesly for the reward function.

Figure VI.3: Plots comparing the sample complexity from training in the Pendulum environment using three reward functions: (baseline) the baseline reward function, Equation VI.3; (sparse) the STLGym reward function, Equation VI.10, defined sparsely; and (dense) the STLGym reward function defined densely. Each curve represents the average return from 10 agents trained the same way. The shaded region around each curve shows the 95% confidence interval.

Figure VI.3, we show sample complexity measured by the (a) baseline reward function and (b) the sparse STL reward function to highlight how the agents trained with the dense STL reward have a better sample complexity than agents trained with the baseline reward function even according to the baseline metric.

While the agents trained using the sparse STL reward function failed to learn an optimal policy, using the sparse STL reward function for evaluating performance was very beneficial. Using the dense reward function for evaluating performance is very similar to the baseline reward function, in that neither provide any insight into whether or not the learned policy satisfies the desired behavior. In contrast, using the sparse STL reward function in Figure VI.3(b), we see the exact point where the learned policies are successfully able to satisfy the specification when the return is greater than 0.

VI.5.2 STLGym is Algorithm-Agnostic

In this case study, we demonstrate that our approach is algorithm-agnostic by using multiple RL algorithms for the Pendulum example explained in Section VI.3.1. All algorithms are used to learn the optimal policy for satisfying the specification in Equation VI.9. We demonstrate the following RL algorithms successfully learning to satisfy the specification using STLGym: Proximal Policy Optimization (PPO) [26], Soft Actor-Critic (SAC) [25], and Twin Delayed Deep Deterministic Policy Gradient (TD3) [24]. The sample complexity plot in Figure VI.4 shows all RL algorithms successfully learn to satisfy the specification. While the results suggest SAC and TD3 work better with our STL reward function, these algorithms are known to learn the optimal policy for this environment very quickly. More examples, across different environments, are needed

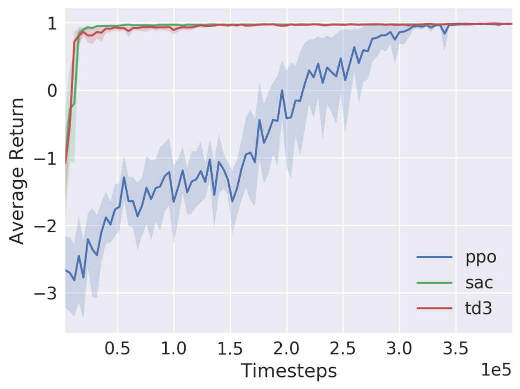


Figure VI.4: The sample complexity of multiple RL algorithms using STLGym to learn the Pendulum specification, Equation VI.9. Each curve represents the average of 10 agents trained the same way. The shaded region around each curve shows the 95% confidence interval.

to make that claim.

VI.5.3 On Separating Specifications and Scaling

The goal of the agent in the CartPole environment is to learn how to keep the pole upright so the angle, θ, is between $\pm 12^\circ$ and the cart's position, x remains within the boundary of ± 2.4 for 200 timesteps. As explained in Section VI.4.2, this specification can be written as a singular specification, Equation VI.4, or as the conjunction of individual components, Equation VI.5.

Using STL's quantitative semantics, STLGym would generate the reward function for Φ_{single} as

$$r_t = \min_{t' \in [0,t]} \Big(\min \big((0.20944 - |\theta_{t'}|), (2.4 - |x_{t'}|) \big) \Big). \tag{VI.11}$$

Similarly, STLGym would generate the reward function for Φ_{split} as Equation VI.8

In this case study, we look at how splitting up the specification into its individual components to create a different reward function impacts the training. We compare the sample complexity of learning Φ_{single} against learning Φ_{split} with and without weights. The results are shown in Figure VI.5.

The results shown in Figure VI.5 indicate splitting the specification is a hindrance for learning. The agents that were trained to satisfy Φ_{single} (single), converged to a more optimal policy faster than both the weighted (stlgym) and unweighted (split) options of Φ_{split}. We believe this is a result of trying to satisfy Φ_{single}, where the robustness degree is always the worst-case of satisfying both the angle and positions specifications. There is no credit awarded for satisfying one better than the other, like in the Φ_{split} definition. We believe that, while splitting the specification in this case study was more of a hindrance, in more complicated systems with more specifications, splitting could be more beneficial than shown here. In those cases, the option for

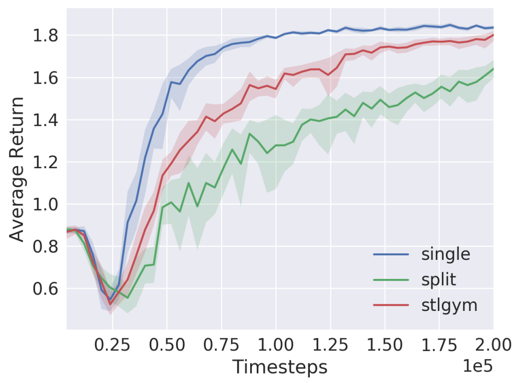


Figure VI.5: Plot comparing the sample complexity of the three options presented in Section VI.5.3. Each curve represents the average of 10 trained agents, and the shaded region shows the 95% confidence interval. The return is calculated using the sparse definition of Φ_{split} (reward function represented by Equation VI.8) with $c_{angle} = 4.7746$ and $c_{position} = 0.41666$ so the maximum possible return is 2.0.

weighting the individual specifications will be very helpful as the weighted and split option (stlgym), which is only supported in STLGym, learned faster than and outperformed the unweighted option.

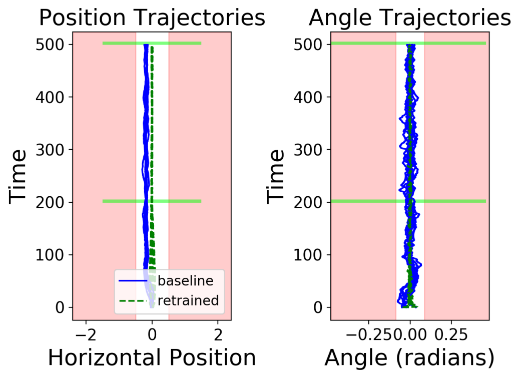
VI.5.4 Retraining With New Goal

There are many cases where the traditional reward functions successfully train agents to complete the desired behavior, but we want to refine/improve/augment that behavior to some other desired behavior. Instead of designing a new reward function and training a new agent from scratch, our tool can be leveraged to retrain the agent to satisfy the new desired behavior. This also makes our tool amenable to curriculum learning [116], an RL training strategy that trains agents in progressively harder environments or constraints. Similar to a learning curriculum used to teach students in a class, by starting with easier constraints and building upon what is learned from the easier tasks, the agent is better able to learn more complex behaviors.

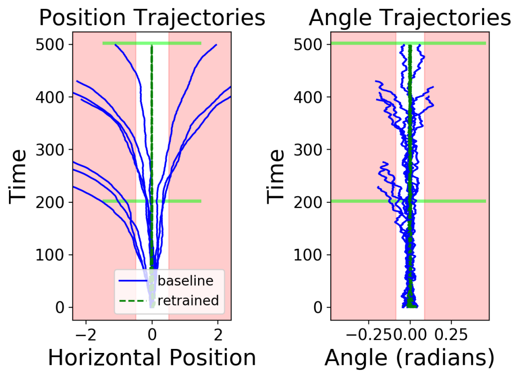
In this case study, we look at an example with the CartPole environment described in Section VI.3.2. The baseline reward function trains agents to keep the pole upright very efficiently, but as [111] point out in their work, many of the learned policies are unstable. When they evaluated the policies for longer than 200 timesteps, they found many learned policies failed shortly after 200 timesteps. We saw similar results, which are shown in Figure VI.6. To counteract this issue, we retrain the agents to maximize the measured robustness of the specifications

$$\begin{aligned} \phi_{position} &= F(G(|x| < 0.5)), \text{ and} \\ \phi_{anlge} &= F(G(|\theta| < 0.0872665)). \end{aligned} \tag{VI.12}$$

In plain English, the specifications translate to “eventually the cart will always be within ±0.5 units of the



(a) 10 example episodes where the policy learned using the baseline reward function is stable after $t = 200$.



(b) 10 example episodes where the policy learned using the baseline reward function is unstable after $t = 200$.

Figure VI.6: Episodes recorded from trained policies evaluated in the CartPole environment. The policies trained using only the baseline reward function can learn unstable policies that fail shortly after $t = 200$, while policies retrained with STLGym are able to continue satisfying the specification past $t = 200$. The red marks the region outside the specification and the horizontal green lines mark the goal during training at 200, and the goal at evaluation 500.

center of the track" and "eventually, the pole's angle will always be within $\pm 5°$."[11]

After some retraining, Figure VI.6 shows the retrained policies converged to more stable and consistent behavior. In particular, Figure VI.6.b shows our approach corrects the unstable behavior.

VI.5.5 Learning a Timed Specification

In this case study, we look at one of the features of our tool that sets it apart from almost all existing approaches in the literature—the ability to learn timed specifications. Here we return to the Pendulum environment described in Section VI.3.1. This time, the specification is to "eventually the angle will be between $\pm 45°$ for 10 timesteps." In STL, the desired behavior is written as,

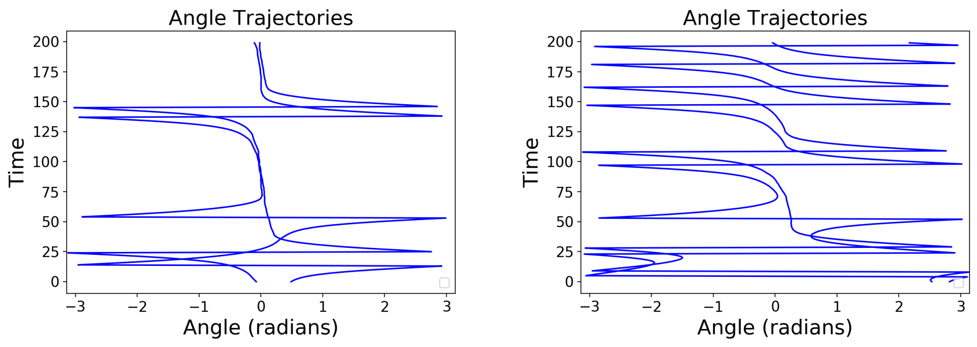
$$\Phi = F(G_{[0:10]}(|\theta| < 0.5)). \tag{VI.13}$$

And is converted by our tool to the reward function,

$$r_t = \max_{t' \in [0,t]} \Big(\min_{t'' \in [t', t'+10]} (0.5 - |\theta_{t''}|) \Big). \tag{VI.14}$$

The results of learning the specification in Equation VI.13 are highlighted in Figure VI.7 where we show a few example episodes. When we first wrote this specification, we believed the resulting behavior would closely match that of the agents in Section VI.5.1. Instead, the learned policies were more varied. Some

[11]These specifications came from [111].



(a) One trained policy.

(b) A different trained policy.

Figure VI.7: Episodes of policies trained to satisfy the specification in Equation VI.13.

stay close to the upright position for longer than others, but they always return. We believe this is a result of the circular state space, which puts the agent back in a starting position after it moves away from upright. This result shows STLGym can successfully train agents to satisfy timed specifications. However, it also highlights a limitation of our approach: we have no way of overwriting the terminal conditions. We would see more similar results if we were able to stop the episode once the specification was satisfied, but that is a feature left for future work.

VI.6 Related Work

Our work is not the first to use temporal logic specifications to create reward functions. The previous works can be grouped into two categories, (1) quantitative semantics and (2) reward machines. We describe the related works in greater detail below and provide a general comparison of our approach with others in Table VI.1. The RL algorithms listed in Table VI.1 are the following: Augmented Random Search (ARS) [18], Deep Deterministic Policy Gradient (DDPG) [23], Deep Q-Learning (DQN) [3], Neural Fitted Q-iteration (NFQ) [117], Relative Entropy Policy Search (REPS) [118], Q-Learning (Q) [119], and Twin Delayed Deep Deterministic Policy Gradient (TD3) [24].

VI.6.1 Quantitative Semantics

This category is where our work resides. These works, [109, 110, 111], generate reward functions based on the quantitative semantics of the temporal logics used to write the specifications the RL agents are tasked with learning to satisfy. In Truncated Linear Temporal Logic (TLTL), presented in [109], the authors create a new specification language, TLTL, that consciously removes the time bounds from STL to only have untimed operators. They made this decision, so specifications do not have to account for robotic limitations. In contrast, our STLGym is designed to handle both timed and untimed specifications, thus handling all TLTL

Table VI.1: A comparison of our tool to similar tools in the literature, separated by category, filled in to the best of our knowledge. $\times$ indicates the feature is not supported, ✓ indicates the feature is supported, and ? indicates it should be supported, but we cannot say so with confidence.

Name	Env-API	Sparse/Dense	RL Algorithms	Retraining	Timed	Sequential
TLTL [109]	?	Dense	REPS	?	$\times$	✓
BHNR [111]	Custom	Dense	DQN	?	✓	?
STLGym (ours)	Gym	Both	Any	✓	✓	✓
QRM [112]	Gym	Both	Q, DQN	$\times$	$\times$	✓
LCRL [120]	Custom	Both	Q, DDPG, NFQ	$\times$	$\times$	✓
SPECTRL [13]	Custom	Dense	ARS	$\times$	$\times$	✓
DIRL [113]	Gym	Dense	ARS, TD3	$\times$	$\times$	✓

problems and more.

Another work, [111], uses timed and untimed STL specifications similar to our STLGym. Their approach, Bounded Horizon Nominal Robustness (BHNR), computes a normalized robustness value over bounded horizons, i.e. small segments, of the episode, creating a reward vector. By only analyzing the robustness over smaller segments of the episode, their approach is able to speed up the robustness degree calculation for dense reward computation. However, because only a small portion of the episode is analyzed, their approach cannot be used to determine the robustness degree across an entire episode like our sparse reward function is able to do. Additionally, their implementation limits user's specifications to be defined only by variables in the environment's observation space. Thus, their tool cannot train our pendulum example without re-writing to specification in terms of x and y instead of θ.

VI.6.2 Reward Machines

Reward machine approaches, [112, 107, 13, 113], use finite state automata (FSA) to handle context switching in the reward function. Temporal logic specifications are used to generate FSA that monitor the trace for satisfaction. Additionally, depending on which state of the FSA is in, the reward function changes in order to guide the agent towards satisfying the next specification. This approach is really great for solving sequential tasks because it allows the user to specify "go to the fridge; open the door; take something out; close the door; return to home" and the reward function changes depending on which part of the task is being done. To the best of our knowledge, however, none of these approaches can handle timed specifications yet.

VI.7 Summary

This chapter presents our tool, STLGym for training agents to satisfy timed and untimed STL specification using RL. To demonstrate the features of our tool and explore some best practices for learning to satisfy STL specifications, we trained over 130 different RL agents in our 5 case studies. From these case studies we observed (1) RL agents learned STLGym's dense rewards better than sparse rewards, (2) STLGym is algorithm-agnostic and works with any RL algorithm designed to integrate with Gym environments, (3) leaving specifications combined is better for RL agents than splitting them into individual parts, (4) STLGym is effective for retraining RL agents to better satisfy specifications, and (5) STLGym is effective for training RL agents to satisfy timed STL specifications.

VI.8 Reinforcement Learning Hyperparameters

Providing the hyperparameters used in RL experiments is crucial for recreating the results. In all of our experiments, we train 10 agents using the following random seeds,

$$[1630, 2241, 2320, 2990, 3281, 4930, 5640, 8005, 9348, 9462].$$

The algorithm hyperparameters used for each environment are specified in the following subsections. Unless otherwise specified, the hyperparameters are consistent across all case studies. All hyperparameter values, except the one marked with a $*$, are the default values provided in OpenAI's SpinningUp library available at https://github.com/openai/spinningup.

VI.8.1 Pendulum

In Table VI.2 we provide the hyperparameters used for training agents with the PPO, SAC, and TD3 algorithms in the *Pendulum-v0* environment.

Table VI.2: Pendulum Hyperparameters

	PPO	SAC & TD3
actor architecture	64 tanh, 64 tanh, 1 linear	64 ReLU, 64 ReLU, 2 tanh
critic architecture	64 tanh, 64 tanh, 1 linear	64 ReLU, 64 ReLU, 1 ReLU
epoch length	4000	4000
epochs	100	100
discount factor γ	0.99	0.99
polyak	N/A	0.995
entropy coefficient α	N/A	0.2
clip ratio	0.2	N/A
actor learning rate	0.0003	0.001
critic learning rate	0.001	0.001
updates per epoch	80	N/A
target kl	0.01	0.01
GAE-λ	0.97	N/A
minibatch size	N/A	100
start steps	N/A	10000
update after _ step(s)	N/A	1000
update every _ step(s)	N/A	50
max episode length	200	200

VI.8.2 CartPole

In Table VI.3 we provide the hyperparameters we used to train our PPO agents in the *CartPole-v0* environment.

Table VI.3: CartPole Hyperparameters

	Separating Specifications	Train/Retrain
actor architecture	64 tanh, 64 tanh, 1 linear	64 tanh, 64 tanh, 1 linear
critic architecture	64 tanh, 64 tanh, 1 linear	64 tanh, 64 tanh, 1 linear
epoch length	4000	4000
epochs	100	50$*$
discount factor γ	0.99	0.99
clip ratio	0.2	0.2
actor learning rate	0.0003	0.0003
critic learning rate	0.001	0.001
updates per epoch	80	80
target kl	0.01	0.01
GAE-λ	0.97	0.97
max episode length	200	200

CHAPTER VII

Conclusions

This dissertation presents our contributions in the field of safe and robust reinforcement learning for cyber-physical systems, and this chapter concludes the dissertation with a brief summary and directions for future research.

VII.1 Summary

In Chapter I, we established the goal of this dissertation to answer the crucial questions

1. Why should we consider RL for *sim2real* scenarios?
2. How can we best incorporate safety in the RL process?
3. How can we assure that learning safety produces safer results?
4. How can we integrate safety specifications in the reward designing/shaping process?

We answered the first question in Chapters III and IV. The experiments in Chapter III showed RL agents are robust to small changes in the model and environment and can quickly learn to overcome the differences, provided the goal is the same. The experiments with our 1/10th scale autonomous racecar in Chapter IV supported this conclusion. Additionally, Chapter IV extended our examples to include a sim2real demonstration highlighting RL's effectiveness at bridging this gap, and its key shortcoming: if the agent is trained without safety in mind, it will not be safe in the real world. Identifying this shortcoming helped spur our research towards answering the remaining questions.

We answered the second and third questions in Chapter V by training 880 RL agents in 88 experimental configurations. The results from the ablation study answered many questions about SRL approaches. Key among these answers was the best method to incorporate safety in the training process, and ensure the RL agents learn safe behavior is through the reward function. Agents trained using reward functions that include a measure of safety consistently learned safe, high-performing policies. The agents trained using safe exploration methods were not as consistent when it came to learning safe policies that did not rely on having the RTA as a backup.

Designing reward functions is a challenging problem. Designing reward functions that integrate safety specifications in a way that does not prevent the agent from learning to complete the goal task is even more challenging. To assist the reward function design process, we developed our tool, STLGym, described in

Chapter VI. STLGym allows designers to automatically generate complex reward functions that help agents learn to satisfy the written behavior specifications. This allows researchers to write specifications describing the desired safe behavior, and the agent will learn to satisfy it.

VII.2 Future Work

This section describes future research directions based on the results of this dissertation.

VII.2.1 Continuing and Expanding Ablation Study

The ablation study we conducted in Chapter V was large, but not exhaustive, and we want to expand our study to include more complex environments. Furthermore, new SRL approaches and ways of integrating RTA in the learning structure are still in development. The Neural Simplex Architecture (NSA) [48] is just one example of a newer approach we would like to add to our study. Additionally, we have identified the following questions that we would like to explore:

- **Soft vs hard constraints**: soft constraints are things like speed limits. Violating soft constraints is undesirable, but violating these constraints is not immediately catastrophic. In contrast, hard constraints cannot be violated without catastrophic results. Should we be treating these types of constraints differently during training?
- **Scaled punishment for violating safety constraints**: in our study, the punishment for violating a safety constraint was always constant. How might it impact training if we scaled the punishment according to how bad the violation was?
- **Quantifying risk as an input for the agent**: in this case, risk would measure the probability of success or failure[1]. How can we leverage this valuable information in the input space to improve agent safety?

In addition to the new SRL approaches being developed, we have recently come across new approaches in the DRL field that have not been identified as SRL approaches but share a lot of similarities to SRL approaches. This opens up even more literature to look through for potential improvement to existing SRL algorithms. For example, the recently developed work "Exploring With Sticky Mittens: Reinforcement Learning With Expert Interventions" [121] takes advantage of existing expert control systems to learn how to complete low-level tasks from demonstration. They relate this to how infants are better able to learn grasping tasks from using sticky mittens that simplify the grasping task, allowing the infants to more quickly realize the value of grasping and prioritize learning the particulars of it. This shares a lot of similarities with the SRL

[1]This is a newer field of study that, to the best of our knowledge, has not been solved yet and could be an interesting research direction all its own.

approach of training with RTA, except in SRL we want to avoid using the RTA instead of leveraging it to learn from demonstration.

VII.2.2 Further Development with STLGym

The case studies presented in Chapter IV are only a fraction of what STLGym is potentially capable of. In future work, we hope to explore the following STLGym uses and demonstrate STLGym's effectiveness in more test environments.

More environments: One of the key features of STLGym is its adaptability to integrating with any existing Gym environments, with only minor modifications to make sure all the variables that need to be tracked are readily accessible. Once we are able to make these modifications on the Spacecraft Docking environments from Chapter V, we want to train agents that are able to satisfy the safety and goal specifications:

$$\begin{aligned}
\phi_{safety} &= G((\bar{r} < \bar{r}_{dock}) \implies (\dot{\bar{r}} < \dot{\bar{r}}_{crash}) \\
\phi_{goal} &= F(\bar{r} < \bar{r}_{dock}) \\
\Phi &= \phi_{safety} \wedge \phi_{goal}
\end{aligned} \tag{VII.1}$$

where $\bar{r}$ is the radial distance between the deputy and chief spacecraft and $\dot{\bar{r}}$ is the relative speed the deputy is traveling towards the chief. By learning to satisfy these specifications, the agent will learn to eventually guide the deputy within the docking region and when it does, the speed should be below the crashing speed limit.

Sequential tasks: Writing STL specifications that describe sequential tasks is possible, but requires a lot of nuances to ensure one is done before the other. We have already started work on an example demonstrating STLGym's ability to train agents to satisfy sequential specifications. However, we have not yet figured out the particulars for writing sequential specifications that can generate dense reward functions using STLGym.

Monte Carlo analysis: Because STLGym does not change the environment's dynamics in any way, STLGym could be very useful for Monte Carlo analysis. Monte Carlo analysis is a straightforward verification technique that relies on repeated random sampling in simulation to identify uncertainty. This usually requires hundreds to millions of simulations to cover as much of the operating space as possible. Using STLGym, we can write the specifications that need to be satisfied and, using the sparse reward setting, run all of those simulations. Since the return is the measure of satisfaction across the entire episode, we can quickly determine if the agent regularly satisfies the specification by looking at the average, maximum, and minimum return.

Curriculum learning: One of the features of STLGym we noted in Chapter VI is its amenability to curriculum learning. In future work, we would like to explore this in greater detail. We hope to uncover the advantages of increasing the strictness of safety specifications throughout the learning process.

VII.2.3 More sim2real Demonstrations

The primary motivation of the work in this dissertation was geared towards improving RL in order to more confidently deploy RL-trained agents in the real world. However, the majority of our work was limited to purely simulation examples. Having established more of the theory for building safe and robust autonomous controls using (S)RL, we want to put that theory into practice on real-world systems to verify that the theory holds up in practice.

VII.2.4 Other Uses for RL

This dissertation focused solely on utilizing RL for learning optimal control policies. However, RL can be used for more problems than just control. Recent works have shown RL is effective for predicting the distances between pairs of amino acid residues, helping better predict protein structures and, from this information, their purpose [122]. Other work has demonstrated RL's usefulness in designing computer chips by identifying the best locations for connecting chip blocks, including unseen blocks the agent has no prior knowledge of. The uses of RL are so much greater than simply autonomous control. If I had to start my PhD over, I would want to spend more time learning about these other RL formulations. I believe I still would have pursued using RL for safe and robust control, but learning about the other formulations while I was learning how to implement RL would have helped me better understand the potential for RL. Additionally, I think these other directions would have been fascinating to at least look into more

CHAPTER VIII

List of Publications

1. Nathaniel Hamilton, Kyle Dunlap, Taylor T Johnson, and Kerianne L Hobbs. "Ablation Study of How Run Time Assurance Impacts the Training and Performance of Reinforcement Learning Agents." arXiv, July 2022.

2. Patrick Musau, Nathaniel Hamilton, Diego Manzanas Lopez, Preston Robinette, and Taylor T Johnson. "An Empirical Analysis of the Use of Real-Time Reachability for the Safety Assurance of Autonomous Vehicles." arXiv, May 2022.

3. Nathaniel Hamilton, Patrick Musau, Diego Manzanas Lopez, and Taylor T Johnson. "Zero-Shot Policy Transfer in Autonomous Racing: Reinforcement Learning vs Imitation Learning." In International Conference on Assured Autonomy (ICAA), March 2022.

4. Patrick Musau, Nathaniel Hamilton, Diego Manzanas Lopez, Preston K Robinette, and Taylor T Johnson. "On Using Real-Time Reachability for the Safety Assurance of Machine Learning Controllers." In IEEE International Conference on Assured Autonomy (ICAA), March 2022.

5. Hoang-Dung Tran, Neelanjana Pal, Patrick Musau, Diego Manzanas Lopez, Nathaniel Hamilton, Xiaodong Yang, Stanley Bak, and Taylor T Johnson. "Robustness verification of semantic segmentation neural networks using relaxed reachability." In International Conference on Computer Aided Verification (20 July 2021)

6. Diego Manzanas Lopez, Patrick Musau, Nathaniel Hamilton, Hoang-Dung Tran, and Taylor T. Jonhson "Case Study: Safety Verification of an Unmanned Underwater Vehicle." In 2020 IEEE Security and Privacy Workshops (SPW), pp. 189-195. IEEE, 2020.

7. Nathaniel Hamilton, Lena Schlemmer, Christopher Menart, Chad Waddington, Todd Jenkins, and Taylor T. Johnson "Sonic to knuckles: evaluations on transfer reinforcement learning" In Unmanned Systems Technology XXII, vol. 11425, p. 114250J. International Society for Optics and Photonics, 2020.

8. Hoang-Dung Tran, Luan Viet Nguyen, Nathaniel Hamilton, Weiming Xiang, and Taylor T. Johnson "Reachability analysis for high-index linear differential algebraic equations (daes)." In 17th International Conference on Formal Modeling and Analysis of Timed Systems (FORMATS'19). Springer

International Publishing (August 2019). 2019.

9. Weiming Xiang, Patrick Musau, Ayana A. Wild, Diego Manzanas Lopez, Nathaniel Hamilton, Xiaodong Yang, Joel Rosenfeld, Taylor T. Johnson "Verification for Machine Learning, Autonomy, and Neural Networks Survey." arXiv, October 2018.

10. Raphael E. Stern, Shumo Cui, Maria Laura Delle Monache, Rahul Bhadani, Matt Bunting, Miles Churchill, Nathaniel Hamilton, R'mani Haulcy, Hannah Pohlmann, Fangyu Wu, Benedetto Piccoli, Benjamin Seibold, Jonathan Sprinkle, Daniel B Work "Dissipation of stop-and-go waves via control of autonomous vehicles: Field experiments." Transportation Research Part C: Emerging Technologies 89 (2018): 205-221.

CHAPTER IX

List of Presentations

IX.1 Poster Presentations

1. N Hamilton, L Schlemmer, C Waddignton, C Menart, T Jenkins. “Deep, Reinforcement, and Transfer Learning Applied to Sonic the Hedgehog” (Poster and Short Presentation). *Autonomy Technology Research Center’s Summer Review*, Dayton, OH, August 2019.

2. L Schlemmer, N Hamilton, C Menart, T Jenkins, C Waddignton. “Exploration in Reward Shaping” (Poster and Short Presentation). *Autonomy Technology Research Center’s Summer Review*, Dayton, OH, August 2019.

3. N Hamilton, TT Johnson. “Architecture for An Indoor Distributed Cyber-Physical System Composed of Mobile Robots and Fog Computing Nodes” (Poster). *Air Force Research Laboratory’s Safe & Secure Systems and Software Symposium (S5)*, Dayton, OH, July 2017. [PDF]

IX.2 Research Talks

1. N Hamilton, P Musau, D Manzanas Lopez, TT Johnson. Zero-Shot Policy Transfer in Autonomous Racing: Reinforcement Learning vs Imitation Learning. Research Presentation, *IEEE International Conference on Assured Autonomy*, March 2022.

2. N Hamilton, T Jenkins, K Hobbs, TT Johnson. Improving Sample Complexity and Performance of Safe Reinforcement Learning Using Kaleidoscope Experience Replay. Invited Speaker, *IEEE Dayton, OH Chapter*, October 2021.

BIBLIOGRAPHY

[1] E. S. Olivas, J. D. M. Guerrero, M. M. Sober, J. R. M. Benedito, and A. J. S. Lopez, *Handbook Of Research On Machine Learning Applications and Trends: Algorithms, Methods and Techniques - 2 Volumes*. Hershey, PA: Information Science Reference - Imprint of: IGI Publishing, 2009.

[2] A. Nichol, V. Pfau, C. Hesse, O. Klimov, and J. Schulman, "Gotta learn fast: A new benchmark for generalization in rl," *arXiv preprint arXiv:1804.03720*, 2018.

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

[4] M. O'Kelly, H. Zheng, D. Karthik, and R. Mangharam, "F1tenth: An open-source evaluation environment for continuous control and reinforcement learning," in *Post Proceedings of the NeurIPS 2019 Demonstration and Competition Track* (H. J. Escalante and R. Hadsell, eds.), Proceedings of Machine Learning Research, PMLR, 2020.

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," *arXiv preprint arXiv:1606.01540*, 2016.

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

[7] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.

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

[9] J. F. Fisac, A. K. Akametalu, M. N. Zeilinger, S. Kaynama, J. Gillula, and C. J. Tomlin, "A general safety framework for learning-based control in uncertain robotic systems," *IEEE Transactions on Automatic Control*, vol. 64, no. 7, pp. 2737–2752, 2018.

[10] K. Jang, E. Vinitsky, B. Chalaki, B. Remer, L. Beaver, A. A. Malikopoulos, and A. Bayen, "Simulation to scaled city: zero-shot policy transfer for traffic control via autonomous vehicles," in *Proceedings of the 10th ACM/IEEE International Conference on Cyber-Physical Systems*, pp. 291–300, 2019.

[11] A. Kadian, J. Truong, A. Gokaslan, A. Clegg, E. Wijmans, S. Lee, M. Savva, S. Chernova, and D. Batra, "Are we making real progress in simulated environments? measuring the sim2real gap in embodied visual navigation," *arXiv preprint arXiv:1912.06321*, 2019.

[12] N. Wagener, B. Boots, and C.-A. Cheng, "Safe reinforcement learning using advantage-based intervention," *arXiv preprint arXiv:2106.09110*, 2021.

[13] K. Jothimurugan, R. Alur, and O. Bastani, "A composable specification language for reinforcement learning tasks," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019*, 2019.

[14] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, 2018.

[15] N. Hamilton, L. Schlemmer, C. Menart, C. Waddington, T. Jenkins, and T. T. Johnson, "Sonic to knuckles: evaluations on transfer reinforcement learning," in *Unmanned Systems Technology XXII*, vol. 11425, p. 114250J, International Society for Optics and Photonics, 2020.

[16] N. Hamilton, P. Musau, D. M. Lopez, and T. T. Johnson, "Zero-shot policy transfer in autonomous racing: Reinforcement learning vs imitation learning," in *2022 IEEE International Conference on Assured Autonomy (ICAA*, pp. 11–20, 2022.

[17] M. Alshiekh, R. Bloem, R. Ehlers, B. Könighofer, S. Niekum, and U. Topcu, "Safe reinforcement learning via shielding," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[18] H. Mania, A. Guy, and B. Recht, "Simple random search of static linear policies is competitive for reinforcement learning," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 1805–1814, 2018.

[19] N. Bernini, M. Bessa, R. Delmas, A. Gold, E. Goubault, R. Pennec, S. Putot, and F. Sillion, "A few lessons learned in reinforcement learning for quadcopter attitude control," in *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*, pp. 1–11, 2021.

[20] D. Silver, "Lectures on reinforcement learning." URL: https://www.davidsilver.uk/teaching/, 2015.

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

[22] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *International conference on machine learning*, pp. 1889–1897, PMLR, 2015.

[23] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning.," in *ICLR*, 2016.

[24] S. Fujimoto, H. Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *International Conference on Machine Learning*, pp. 1587–1596, PMLR, 2018.

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

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

[27] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, "Implementation matters in deep rl: A case study on ppo and trpo," in *International conference on learning representations*, 2019.

[28] R. M. French, "Catastrophic forgetting in connectionist networks," *Trends in cognitive sciences*, vol. 3, no. 4, pp. 128–135, 1999.

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

[30] M. Wu, J. Wang, J. Deshmukh, and C. Wang, "Shield synthesis for real: Enforcing safety in cyber-physical systems," in *2019 Formal Methods in Computer Aided Design (FMCAD)*, pp. 129–137, IEEE, 2019.

[31] C. Neary, C. Verginis, M. Cubuktepe, and U. Topcu, "Verifiable and compositional reinforcement learning systems," *arXiv preprint arXiv:2106.05864*, 2021.

[32] L. Zhang, R. Zhang, T. Wu, R. Weng, M. Han, and Y. Zhao, "Safe reinforcement learning with stability guarantee for motion planning of autonomous vehicles," *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–10, 2021.

[33] Y. Wang, C. Huang, Z. Wang, Z. Wang, and Q. Zhu, "Verification in the loop: Correct-by-construction control learning with reach-avoid guarantees," *arXiv preprint arXiv:2106.03245*, 2021.

[34] N. Hunt, N. Fulton, S. Magliacane, T. N. Hoang, S. Das, and A. Solar-Lezama, "Verifiably safe exploration for end-to-end reinforcement learning," in *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*, pp. 1–11, 2021.

[35] Z. Xiong and S. Jagannathan, "Scalable synthesis of verified controllers in deep reinforcement learning," *arXiv preprint arXiv:2104.10219*, 2021.

[36] Y. Li, N. Li, H. E. Tseng, A. Girard, D. Filev, and I. Kolmanovsky, "Safe reinforcement learning using robust action governor," in *Learning for Dynamics and Control*, pp. 1093–1104, PMLR, 2021.

[37] N. Fulton and A. Platzer, "Safe reinforcement learning via formal methods: Toward safe control through proof and learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, 2018.

[38] A. Desai, S. Ghosh, S. A. Seshia, N. Shankar, and A. Tiwari, "Soter: a runtime assurance framework for programming safe robotics systems," in *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pp. 138–150, IEEE, 2019.

[39] A. Murugesan, M. Moghadamfalahi, and A. Chattopadhyay, "Formal methods assisted training of safe reinforcement learning agents," in *NASA Formal Methods Symposium*, pp. 333–340, Springer, 2019.

[40] R. Cheng, G. Orosz, R. M. Murray, and J. W. Burdick, "End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, pp. 3387–3395, 2019.

[41] H. Zhao, X. Zeng, T. Chen, Z. Liu, and J. Woodcock, "Learning safe neural network controllers with barrier certificates," in *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*, pp. 177–185, Springer, 2020.

[42] E. Altman, *Constrained Markov decision processes*, vol. 7. CRC Press, 1999.

[43] J. Achiam, D. Held, A. Tamar, and P. Abbeel, "Constrained policy optimization," in *International Conference on Machine Learning*, pp. 22–31, PMLR, 2017.

[44] A. Wachi and Y. Sui, "Safe reinforcement learning in constrained markov decision processes," in *International Conference on Machine Learning*, pp. 9797–9806, PMLR, 2020.

[45] D. Ding, X. Wei, Z. Yang, Z. Wang, and M. Jovanovic, "Provably efficient safe exploration via primal-dual policy optimization," in *International Conference on Artificial Intelligence and Statistics*, pp. 3304–3312, PMLR, 2021.

[46] A. HasanzadeZonuzy, D. M. Kalathil, and S. Shakkottai, "Learning with safety constraints: Sample complexity of reinforcement learning for constrained mdps," *arXiv preprint arXiv:2008.00311*, 2020.

[47] H. Satija, P. Amortila, and J. Pineau, "Constrained markov decision processes via backward value functions," in *International Conference on Machine Learning*, pp. 8502–8511, PMLR, 2020.

[48] D. T. Phan, R. Grosu, N. Jansen, N. Paoletti, S. A. Smolka, and S. D. Stoller, "Neural simplex architecture," in *NASA Formal Methods Symposium*, pp. 97–114, Springer, 2020.

[49] X. Wang, S. Nair, and M. Althoff, "Falsification-based robust adversarial reinforcement learning," in *2020 19th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 205–212, IEEE, 2020.

[50] X. Yang, T. Yamaguchi, H.-D. Tran, B. Hoxha, T. T. Johnson, and D. Prokhorov, "Neural network repair with reachability analysis," *arXiv preprint arXiv:2108.04214*, 2021.

[51] J. G. Rivera, A. A. Danylyszyn, C. B. Weinstock, L. R. Sha, and M. J. Gagliardi, "An architectural description of the simplex architecture.," tech. rep., CARNEGIE-MELLON UNIV PITTSBURGH PA SOFTWARE ENGINEERING INST, 1996.

[52] T. Gurriet, *Applied Safety Critical Control*. PhD thesis, California Institute of Technology, 2020.

[53] L. P. Kaelbling, M. L. Littman, and A. W. Moore, "Reinforcement learning: A survey," *Journal of artificial intelligence research*, vol. 4, pp. 237–285, 1996.

[54] M. E. Taylor and P. Stone, "Transfer learning for reinforcement learning domains: A survey.," *Journal of Machine Learning Research*, vol. 10, no. 7, 2009.

[55] M. Grześ, "Reward shaping in episodic reinforcement learning," in *Proceedings of the 16th Conference on Autonomous Agents and MultiAgent Systems*, pp. 565–573, 2017.

[56] M. O'Kelly, V. Sukhil, H. Abbas, J. Harkins, C. Kao, Y. V. Pant, R. Mangharam, D. Agarwal, M. Behl, P. Burgio, *et al.*, "F1/10: An open-source autonomous cyber-physical platform," *arXiv preprint arXiv:1901.08567*, 2019.

[57] B. Balaji, S. Mallya, S. Genc, S. Gupta, L. Dirac, V. Khare, G. Roy, T. Sun, Y. Tao, B. Townsend, E. Calleja, S. Muralidhara, and D. Karuppasamy, "Deepracer: Educational autonomous racing platform for experimentation with sim2real reinforcement learning," *CoRR*, vol. abs/1911.01562, 2019.

[58] D. Murphy, "Tum autonomous motorsport wins the indy autonomous challenge powered by cisco at the indianapolis motor speedway and the $1 million grand prize," Oct 2021.

[59] G. Velasco-Hernandez, D. J. Yeong, J. Barry, and J. Walsh, "Autonomous driving architectures, perception and data fusion: A review," in *2020 IEEE 16th International Conference on Intelligent Computer Communication and Processing (ICCP)*, pp. 315–321, 2020.

[60] M. Han, Y. Tian, L. Zhang, J. Wang, and W. Pan, "Reinforcement learning control of constrained dynamic systems with uniformly ultimate boundedness stability guarantee," *Automatica*, vol. 129, p. 109689, 2021.

[61] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: a survey," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 737–744, 2020.

[62] T. P. Gros, D. Höller, J. Hoffmann, and V. Wolf, "Tracking the race between deep reinforcement learning and imitation learning," in *International Conference on Quantitative Evaluation of Systems*, pp. 11–17, Springer, 2020.

[63] A. Hussein, M. M. Gaber, E. Elyan, and C. Jayne, "Imitation learning: A survey of learning methods," *ACM Computing Surveys (CSUR)*, vol. 50, no. 2, pp. 1–35, 2017.

[64] Y. Pan, C.-A. Cheng, K. Saigol, K. Lee, X. Yan, E. A. Theodorou, and B. Boots, "Imitation learning for agile autonomous driving," *The International Journal of Robotics Research*, vol. 39, no. 2-3, pp. 286–302, 2020.

[65] D. A. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation," *Neural computation*, vol. 3, no. 1, pp. 88–97, 1991.

[66] D. A. Pomerleau, "Alvinn: An autonomous land vehicle in a neural network," tech. rep., Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, 1989.

[67] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[68] J. Kerr and K. Nickels, "Robot operating systems: Bridging the gap between human and robot," in *Proceedings of the 2012 44th Southeastern Symposium on System Theory (SSST)*, pp. 99–104, IEEE, 2012.

[69] R. C. Coulter, "Implementation of the pure pursuit path tracking algorithm," tech. rep., Carnegie-Mellon UNIV Pittsburgh PA Robotics INST, 1992.

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

[71] K. Judah, A. P. Fern, T. G. Dietterich, and P. Tadepalli, "Active imitation learning: Formal and practical reductions to iid learning," *Journal of Machine Learning Research*, vol. 15, no. 120, pp. 4105–4143, 2014.

[72] R. Memmesheimer, I. Kramer, V. Seib, and D. Paulus, "Simitate: A hybrid imitation learning benchmark," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5243–5249, IEEE, 2019.

[73] G. Dulac-Arnold, D. Mankowitz, and T. Hester, "Challenges of real-world reinforcement learning," *arXiv preprint arXiv:1904.12901*, 2019.

[74] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. Al Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[75] S. Levine, A. Kumar, G. Tucker, and J. Fu, "Offline reinforcement learning: Tutorial, review, and perspectives on open problems," 2020.

[76] A. Y. Ng, S. J. Russell, *et al.*, "Algorithms for inverse reinforcement learning.," in *Icml*, vol. 1, p. 2, 2000.

[77] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International Conference on Machine Learning*, pp. 1126–1135, PMLR, 2017.

[78] Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel, "Rl^2: Fast reinforcement learning via slow reinforcement learning," *arXiv preprint arXiv:1611.02779*, 2016.

[79] R. Wang, J. Lehman, J. Clune, and K. O. Stanley, "Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions," *arXiv preprint arXiv:1901.01753*, 2019.

[80] A. Amini, W. Schwarting, G. Rosman, B. Araki, S. Karaman, and D. Rus, "Variational autoencoder for end-to-end control of autonomous driving with novelty detection and training de-biasing," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 568–575, IEEE, 2018.

[81] O. Pettersson, "Execution monitoring in robotics: A survey," *Robotics and Autonomous Systems*, vol. 53, no. 2, pp. 73–88, 2005.

[82] A. Desai, T. Dreossi, and S. A. Seshia, "Combining model checking and runtime verification for safe robotics," in *International Conference on Runtime Verification*, pp. 172–189, Springer, 2017.

[83] A. K. Akametalu, J. F. Fisac, J. H. Gillula, S. Kaynama, M. N. Zeilinger, and C. J. Tomlin, "Reachability-based safe learning with gaussian processes," in *53rd IEEE Conference on Decision and Control*, pp. 1424–1431, IEEE, 2014.

[84] S. Mitsch and A. Platzer, "Modelplex: Verified runtime validation of verified cyber-physical system models," *Formal Methods in System Design*, vol. 49, no. 1, pp. 33–74, 2016.

[85] S. L. Herbert, S. Bansal, S. Ghosh, and C. J. Tomlin, "Reachability-based safety guarantees using efficient initializations," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 4810–4816, IEEE, 2019.

[86] A. Bajcsy, S. Bansal, E. Bronstein, V. Tolani, and C. J. Tomlin, "An efficient reachability-based framework for provably safe autonomous navigation in unknown environments," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, pp. 1758–1765, IEEE, 2019.

[87] S. Bansal, A. Bajcsy, E. Ratner, A. D. Dragan, and C. J. Tomlin, "A hamilton-jacobi reachability-based framework for predicting and analyzing human motion for safe planning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7149–7155, IEEE, 2020.

[88] S. Bansal, V. Tolani, S. Gupta, J. Malik, and C. Tomlin, "Combining optimal control and learning for visual navigation in novel environments," in *Conference on Robot Learning*, pp. 420–429, PMLR, 2020.

[89] A. Gattami, A. Al Alam, K. H. Johansson, and C. J. Tomlin, "Establishing safety for heavy duty vehicle platooning: A game theoretical approach," *IFAC Proceedings Volumes*, vol. 44, no. 1, pp. 3818–3823, 2011.

[90] M. Chen, Q. Hu, J. F. Fisac, K. Akametalu, C. Mackin, and C. Tomlin, "Guaranteeing safety and liveness of unmanned aerial vehicle platoons on air highways," *CoRR*, 2016.

[91] A. Dhinakaran, M. Chen, G. Chou, J. C. Shih, and C. J. Tomlin, "A hybrid framework for multi-vehicle collision avoidance," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*, pp. 2979–2984, IEEE, 2017.

[92] A. Valmari, "The state explosion problem," in *Advanced Course on Petri Nets*, pp. 429–528, Springer, 1996.

[93] N. Hamilton, K. Dunlap, T. T. Johnson, and K. L. Hobbs, "Ablation study of how run time assurance impacts the training and performance of reinforcement learning agents," 2022.

[94] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, pp. 484–489, Jan. 2016.

[95] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, "Grandmaster Level in StarCraft II using Multi-Agent Reinforcement Learning," *Nature*, vol. 575, pp. 350–354, Oct. 2019.

[96] R. Agarwal, M. Schwarzer, P. S. Castro, A. C. Courville, and M. Bellemare, "Deep reinforcement learning at the edge of the statistical precipice," *Advances in Neural Information Processing Systems*, vol. 34, 2021.

[97] A. D. Ames, S. Coogan, M. Egerstedt, G. Notomista, K. Sreenath, and P. Tabuada, "Control barrier functions: Theory and applications," in *2019 18th European Control Conference (ECC)*, pp. 3420–3431, IEEE, 2019.

[98] M. Nagumo, "Über die lage der integralkurven gewöhnlicher differentialgleichungen," *Proceedings of the Physico-Mathematical Society of Japan. 3rd Series*, vol. 24, pp. 551–559, 1942.

[99] W. Clohessy and R. Wiltshire, "Terminal guidance system for satellite rendezvous," *Journal of the Aerospace Sciences*, vol. 27, no. 9, pp. 653–658, 1960.

[100] K. Dunlap, M. Mote, K. Delsing, and K. L. Hobbs, "Run time assured reinforcement learning for safe satellite docking," in *2022 AIAA SciTech Forum*, pp. 1–20, 2022.

[101] K. Dunlap, M. Hibbard, M. Mote, and K. Hobbs, "Comparing run time assurance approaches for safe spacecraft docking," *IEEE Control Systems Letters*, 2021.

[102] U. J. Ravaioli, J. Cunningham, J. McCarroll, V. Gangal, K. Dunlap, and K. L. Hobbs, "Safe reinforcement learning benchmark environments for aerospace control systems," in *2022 IEEE Aerospace Conference*, pp. 1–18, IEEE, 2022.

[103] P. Musau, N. Hamilton, D. M. Lopez, P. Robinette, and T. T. Johnson, "On using real-time reachability for the safety assurance of machine learning controllers," in *2022 IEEE International Conference on Assured Autonomy (ICAA)*, pp. 1–10, IEEE, 2022.

[104] T. T. Johnson, S. Bak, M. Caccamo, and L. Sha, "Real-time reachability for verified simplex design," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 15, no. 2, pp. 1–27, 2016.

[105] S. Bak, T. T. Johnson, M. Caccamo, and L. Sha, "Real-time reachability for verified simplex design," in *2014 IEEE Real-Time Systems Symposium*, pp. 138–148, IEEE, 2014.

[106] D. Silver, S. Singh, D. Precup, and R. S. Sutton, "Reward is enough," *Artificial Intelligence*, vol. 299, p. 103535, 2021.

[107] A. Camacho, R. T. Icarte, T. Q. Klassen, R. A. Valenzano, and S. A. McIlraith, "Ltl and beyond: Formal languages for reward function specification in reinforcement learning.," in *IJCAI*, vol. 19, pp. 6065–6073, 2019.

[108] O. Maler and D. Nickovic, "Monitoring temporal properties of continuous signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pp. 152–166, Springer, 2004.

[109] X. Li, C.-I. Vasile, and C. Belta, "Reinforcement learning with temporal logic rewards," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3834–3839, IEEE, 2017.

[110] D. Aksaray, A. Jones, Z. Kong, M. Schwager, and C. Belta, "Q-learning for robust satisfaction of signal temporal logic specifications," in *2016 IEEE 55th Conference on Decision and Control (CDC)*, pp. 6565–6570, IEEE, 2016.

[111] A. Balakrishnan and J. V. Deshmukh, "Structured reward shaping using signal temporal logic specifications," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3481–3486, IEEE, 2019.

[112] R. T. Icarte, T. Klassen, R. Valenzano, and S. McIlraith, "Using reward machines for high-level task specification and decomposition in reinforcement learning," in *International Conference on Machine Learning*, pp. 2107–2116, PMLR, 2018.

[113] K. Jothimurugan, O. Bastani, and R. Alur, "Abstract value iteration for hierarchical reinforcement learning," in *International Conference on Artificial Intelligence and Statistics*, pp. 1162–1170, PMLR, 2021.

[114] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE transactions on systems, man, and cybernetics*, no. 5, pp. 834–846, 1983.

[115] D. Ničković and T. Yamaguchi, "Rtamt: Online robustness monitors from stl," in *International Symposium on Automated Technology for Verification and Analysis*, pp. 564–571, Springer, 2020.

[116] X. Wang, Y. Chen, and W. Zhu, "A survey on curriculum learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

[117] M. Riedmiller, "Neural fitted q iteration–first experiences with a data efficient neural reinforcement learning method," in *European conference on machine learning*, pp. 317–328, Springer, 2005.

[118] J. Peters, K. Mulling, and Y. Altun, "Relative entropy policy search," in *Twenty-Fourth AAAI Conference on Artificial Intelligence*, 2010.

[119] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3, pp. 279–292, 1992.

[120] M. Hasanbeig, A. Abate, and D. Kroening, "Logically-constrained reinforcement learning code repository." https://github.com/grockious/lcrl, 2020.

[121] S. Dutta, K. Sridhar, O. Bastani, E. Dobriban, J. Weimer, I. Lee, and J. Parish-Morris, "Exploring with sticky mittens: Reinforcement learning with expert interventions via option templates," *arXiv preprint arXiv:2202.12967*, 2022.

[122] A. W. Senior, R. Evans, J. Jumper, J. Kirkpatrick, L. Sifre, T. Green, C. Qin, A. Žídek, A. W. Nelson, A. Bridgland, *et al.*, "Improved protein structure prediction using potentials from deep learning," *Nature*, vol. 577, no. 7792, pp. 706–710, 2020.