Université de Montréal

Fear prediction for training robust RL agents

par

Charlie Gauthier

Département d'informatique et de recherche opérationelle
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Informatique

22 mars 2023

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

Fear prediction for training robust RL agents

présenté par

Charlie Gauthier

a été évalué par un jury composé des personnes suivantes :

Pierre-Luc Bacon

(président-rapporteur)

Liam Paull

(directeur de recherche)

Gauthier Gidel

(membre du jury)

Résumé

Les algorithmes d'apprentissage par renforcement conditionné par les buts apprennent à accomplir des tâches en interagissant avec leur environnement. Ce faisant, ils apprennent à propos du monde qui les entourent de façon graduelle et adaptive. Parmi d'autres raisons, c'est pourquoi cette branche de l'intelligence artificielle est une des avenues les plus prometteuses pour le contrôle des robots généralistes de demain. Cependant, la sûreté de ces algorithmes de contrôle restent un champ de recherche actif. La majorité des algorithmes "d'apprentissage par renforcement sûr" tâchent d'assurer la sécurité de la politique de contrôle tant durant l'apprentissage que pendant le déploiement ou l'évaluation. Dans ce travail, nous proposons une stratégie complémentaire.

Puisque la majorité des algorithmes de contrôle pour la robotique sont développés, entraînés, et testés en simulation pour éviter d'endommager les vrais robots, nous pouvons nous permettre d'agir de façon dangereuse dans l'environnement simulé. Nous démontrons qu'en donnant des buts dangereux à effectuer à l'algorithme d'apprentissage durant l'apprentissage, nous pouvons produire des populations de politiques de contrôle plus sûres au déploiement ou à l'évaluation qu'en sélectionnant les buts avec des techniques de l'état de l'art. Pour ce faire, nous introduisons un nouvel agent à l'entraînement de la politique de contrôle, le "Directeur". Le rôle du Directeur est de sélectionner des buts qui sont assez difficiles pour aider la politique à apprendre à les résoudre sans être trop difficiles ou trop faciles. Pour aider le Directeur dans sa tâche, nous entraînons un réseau de neurones en ligne pour prédire sur quels buts la politique de contrôle échouera. Armé de ce "réseau de la peur" (nommé d'après la peur de la politique de contrôle), le Directeur parviens à sélectionner les buts de façon à ce que les politiques de contrôles finales sont plus sûres et plus performantes que les politiques entraînées à l'aide de méthodes de l'état de l'art, ou obtiennent des métriques semblables. De plus, les populations de politiques entraînées par le Directeur ont moins de variance dans leur comportement, et sont plus résistantes contre des attaques d'adversaires sur les buts qui leur sont issus.

Mots-clés : apprentissage par renforcement, robotique, apprentissage adversariel

Abstract

By learning from experience, goal-conditioned reinforcement learning methods learn from their environments gradually and adaptively. Among other reasons, this makes them a promising direction for the generalist robots of the future. However, the safety of these goal-conditioned RL policies is still an active area of research. The majority of “Safe Reinforcement Learning” methods seek to enforce safety both during training and during deployment and/or evaluation. In this work, we propose a complementary strategy.

Because the majority of control algorithms for robots are developed, trained, and tested in simulation to avoid damaging the real hardware, we can afford to let the policy act in unsafe ways in the simulated environment. We show that by tasking the learning algorithm with unsafe goals during its training, we can produce populations of final policies that are safer at evaluation or deployment than when trained with state-of-the-art goal-selection methods. To do so, we introduce a new agent to the training of the policy that we call the “Director”. The Director’s role is to select goals that are hard enough to aid the policy’s training, without being too hard or too easy. To help the Director in its task, we train a neural network online to predict which goals are unsafe for the current policy. Armed with this “fear network” (named after the policy’s own fear of violating its safety conditions), the Director is able to select training goals such that the final trained policies are safer and more performant than policies trained on state-of-the-art goal-selection methods (or just as safe/performant). Additionally, the populations of policies trained by the Director show decreased variance in their behaviour, along with increased resistance to adversarial attacks on the goals issued to them.

Keywords: reinforcement learning, robotics, adversarial training

Contents

Résumé 5

Abstract 7

List of Tables 13

List of Figures 15

List of Acronyms and Abbreviations 19

Remerciements 21

Chapter 1. Introduction 23

Chapter 2. Background 27

2.1. Preliminaries 27
2.1.1. Control 27
2.1.2. Markov Decision Process 29
2.1.2.1. **Goal-conditioned Markov Decision Process** 30
2.1.3. Bandits 31
2.1.3.1. **Thompson Sampling** 32
2.1.4. Reinforcement Learning 32
2.1.5. Deep Learning and Deep Reinforcement Learning 34
2.1.5.1. **Proximal Policy Optimization** 35
2.1.6. Curriculum Learning 35
2.1.7. Adversarial Training 36

2.2. Related Work 37
2.2.1. Intrinsic Motivation 37
2.2.1.1. **Random Network Distillation** 38
2.2.2. Reinforcement Learning Safety 38
2.2.2.1. **Recovery** 39
2.2.2.2. **Reward Shaping for Safety** 40

2.2.2.3. **Encoding Safety in the Optimization Process** 40
2.2.3. Goal Selection 40
2.2.3.1. **Value Disagreement Sampling** 41

Chapter 3. Dread Enforced Interactions for More Optimal Sampling 43

3.1. Problem setting 43
3.1.1. Optimization objectives 44

3.2. Method intuition 44

3.3. Method: the Director's strategy 46

3.4. Disempowering the Director 47

Chapter 4. Experiments 49

4.1. Environments 49
4.1.1. Toy problem: *Particle Gym* 49
4.1.2. *Legged Gym* 50
4.1.3. Evaluation and goal selection 52

4.2. Implementing DEIMOS 53
4.2.1. Ablations 54

4.3. Baseline methods 54

4.4. Terminology 55

4.5. Results 56
4.5.1. Training results for *Particle Gym* 57
4.5.2. Evaluation results for Absolute 58
4.5.3. Evaluation results for Relative 60
4.5.4. Evaluation results for Directional 62
4.5.5. Evaluation results for Quad 63
4.5.6. Evaluation results for Quad+arm 65
4.5.7. Discussion 67

4.6. Informing goal selection using a single network 68

Chapter 5. Conclusion 71

References 73

Appendix A. Additional results for *Legged Gym* 79

List of Tables

4.1 Frames per second across all parallel environments during training, rounded to the nearest thousand, ± STD Error. 69

List of Figures

2.1 Visual explanation of an instance of the Brachistochrone problem. There exists an infinity of possible curves that would lead a marble from point A to point B (three possible curves are represented in blue, red, and green). Before letting the ball fall and roll down the curve according to gravity, the agent tasked with solving the problem sets the shape of the curve. Which curve enables the marble to reach point B the fastest? The optimal curve (in red) can be derived analytically using calculus, or namely through methods derived from the control field. *Image obtained and modified from the Brachistochrone Wiki entry* [**63**]. 28

2.2 *Minigrid* is a very popular environment for benchmarking control policies [**11**]. The agent must reach the goal represented by a green cell. In some variants, there are coloured keys that must be used to unlock corresponding doors. 30

2.3 Visual explanation of Thompson Sampling. Every possible action has a corresponding true success probability: 0.3 for blue, 0.7 for orange, and 0.8 for green. At first, each possible action has never been picked, and so the estimated success probability is equal for each. The policy thus randomly picks an action. Every control episode thereafter, the control policy picks actions randomly, weighted by their historical success rate (shown in the legend of each plot: success/trials). By the end of the experimentation, the policy has correctly converged to almost always selecting the green action. *Plots obtained from a slight modification of BabyRobot* [**49**]. 33

3.1 Visualization of our method's main feedback loop. The Director emits goals sampled from the fear network. Upon receiving and acting upon the goals, the agent crashes or survives the goals. We then update the Director's belief about the value of goals by changing the fear network's parameters. Then, the loop begins anew. 47

4.1 In *Particle Gym*, the particle starts every episode at (0,0,0) (in black). Three variations exist. In the **directional** case, goals are desired velocities (visualization in red). In the **relative** case, goals are the distance from the particle's current position to a set point in space (visualization in green). In the **absolute** case, goals are an absolute position in space that the particle must reach (visualization in blue). In all variations, should the particle exit the $[-1000,1000]^3$ cube, the episode terminates. 51

4.2 Visualization of the *Legged Gym* environment. Many robots are simulated in parallel, all controlled by a single policy. The policy must learn to issues commands such that the robots walk on flat terrain, noised terrain, sloped terrain, and stairs without crashing. *Picture credit: Legged Gym* [**51**]. 52

4.3 Pictures of Quad and Quad+arm, modelled after the *Unitree Go1* robot. Quad+arm additionally has a *Interbotix ReactorX* arm onboard. 53

4.4 Crashes during training for10 proximal policy optimization (PPO) random seeds for all three variants of *Particle Gym*. Most p values were omitted for readability. On the left are the results obtained from the different ablations applied to **D**read **E**nforced **I**nteractons for **M**ore **O**ptimal **S**ampling (DEIMOS). On the right, we compare one representative p value for DEIMOS to the results obtained from the baseline methods. 57

4.5 Evaluation results for 10 PPO seeds in the *Particle Gym* Absolute environment. . 59

4.6 Evaluation results for 10 PPO seeds in the *Particle Gym* Relative environment. Coop omitted because of scaling issues. Very small boxes for crashes are not mistakes or visual artefacts; some methods achieved ≈ 0 crashes. 60

4.7 Evaluation results for 10 PPO seeds in the *Particle Gym* Directional environment. 62

4.8 Results for 30 PPO seeds in the Quad environment with the $\mathcal{T}_2$ goal resampling timing. 63

4.9 Results for 30 PPO seeds in the Quad environment with the $\mathcal{T}_{10}$ goal resampling timing. 64

4.10 Results for T_2 for Quad+arm. All DEIMOS versions produce better policy populations that the baselines, except for value disagreement sampling (VDS). The VDS+p75 schedule does best overall, with almost identical Naive results as VDS and better Adversary results. 65

4.11 Results for T_{10} for Quad+arm. Again, almost all DEIMOS versions produce better policy populations that the baselines, except for VDS. The VDS+p75 schedule does best overall, with almost identical Naive results as VDS and better Adversary results. 66

A.1 Reward and safety training curves for Quad. 79

A.2 Reward and safety training curves for Quad+arm. 80

List of Acronyms and Abbreviations

CL	Curriculum learning
CVaR	Conditional value at risk
DAGGER	Dataset Aggregation
DEIMOS	Dread Enforced Interactions for More Optimal Sampling
DQN	Deep Q-Network
DRL	Deep reinforcement learning
FPS	Frames per second
GAN	Generative adversarial network
GMDP	Goal-conditioned Markov decision process
MDP	Markov decision process

MPC	Model predictive control
PAIRED	Protagonist-Antagonist Induced Regret Environment Design
PID	Proportional-integral-derivative
POMDP	Partially observable Markov decision process
PPO	Proximal policy optimization
RL	Reinforcement learning
RND	Random network distillation
VDS	Value disagreement sampling

Remerciements

I would like to thank my advisor, Liam Paull, for his priceless mentorship. I am so grateful for you taking a chance on me. You were my introduction to research, and without you, the research accomplishments I have today would still firmly seem to belong to the realm of fantasy. Under your tutelage, I had the freedom to explore various research topics and problem settings; thanks to this, I have sharpened my insights into both what I am interested in and what is and isn't feasible. Thank you for challenging my ideas and driving me to focus on what is most important. Our discussions and collaborations on various applied projects shaped me to understand the full picture of robotics, something that I consider to be invaluable. Thank you.

To Florian Golemo, thank you for taking me under your wing. Thank you for collaborating with me on this project, for listening to my ramblings, and for your friendship. You waltzed into my life when I needed you most.

To Glen Berseth, thank you for your profound insights. Your wisdom and our discussions truly shaped both this work and my future in research.

To Mila and IVADO, I am deeply grateful for your financial support. I am also thankful for the bustling research environment that you foster.

To Geňa Hahn, thank you for pushing me towards research. Our discussions drove me to rethink my future, and I am glad for them.

To my parents and my sister, thank you for always supporting me, and for bearing with me and my strange schedule. You made my resolve to learn more about the world stronger, for I know I can always count on you to have my back.

To David Dobre and Léo Gagnon, thank you for making every day spent at Mila during the redaction of this work enjoyable. Both of you are a little special to me.

To the Star Walkers and their enthralling Weaver, thank you for countless hours of much-needed relaxation.

To Jun "Nujabes" Seba, Tash Sultana, Björk Guðmundsdóttir, Alfa Mist, and the members of L'Impératrice and Moon Hooch, thank you. Your music was the soundtrack to every experiment, every analysis, and every written word. To Hayao Miyazaki and Tsubasa Yamaguchi, thank you for setting me upright through your art when I fell.

Chapter 1

Introduction

In deep reinforcement learning (DRL), policies iteratively adapt as they explore their environments, making them one of the most promising areas of research for control policies for generalizable robots. Of particular interest are goal-conditioned DRL policies. These policies have the ability to change their behaviour based on a goal. For example, one might send the robot's control policy a string of text such as "fetch the ball" or "open the door", or a vector of numbers describing desired velocities, etc. Because any task can be specified using the goal-condition, such policies are a type of *universal* policy. These policies are especially desirable for robotic control because their goal-conditioning offers some modicum of interpretability to the agent's behaviour, and because human operators (or other DRL policies) can set the goal dynamically to change the robot's behaviour without changing the policy in control of the robot.

While DRL is promising, it is often prohibitively costly to train in the real world. For example, one of the state-of-the-art DRL techniques requires $[\approx 3M\text{timesteps}]$ to train a single policy on a simple idealized legged robotic locomotion task [**22**]. Using generous estimates (a control frequency of 60Hz), this corresponds to $\left[\approx 3M\text{timesteps} * \frac{1}{60}\text{seconds} \approx 139\text{hours}\right]$, without taking into account the large amount of time required to iteratively update the three neural networks involved, to reset the robot's environment (and the robot itself), or even to search for adequate hyperparameters for the policy's training. In fact, optimizing DRL on real robots is often deemed intractable unless given access to vast monetary support and a large amount of space and person-hours.

A common solution is training DRL in simulation, allowing for vast parallelization and to run things faster than in the real world. But this often requires fine-tuning a sub-optimal final policy on the final hardware, which can then result in damaging the robot. To take full advantage of the generality of learning from experience, we require methods that allow us to be confident that the learned policy will perform well and act *safely*.

Safety for deep reinforcement learning is not a novel concern. Many methods already exist to guide the policy's training towards safer behaviour. Most works that focus on this problem seek safety guarantees through convergence proofs. In particular, "Safe Reinforcement Learning" methods focus on safety both during training and deployment and/or evaluation. Such methods are particularly desirable when training DRL on the real hardware, precisely because the policy is incentivized to act safely at all times.

In this work, we consider a complementary approach to safety. For the same reason practitioners train in simulation before deploying on the real hardware, we can afford to act in unsafe ways in simulation. For our problem setting, the only safety consideration is to avoid undesirable terminal states called *crash states*. It is known DRL can act in overly conservative ways during training because of early termination problems, leading to sub-optimal exploration behaviour [**29**]. In this work, we show that by incentivizing unsafe behaviour during training, we can improve the optimization process of the policy and obtain increased safety after training.

As part of this work, **solely through goal selection during training**, we seek to produce goal-conditioned policies that are safer at evaluation time while still accomplishing their goals to the best of their ability. We do so by using an adversarial training goal distribution. This is motivated by the intuition that by giving the policy more experiences of unsafe states, given an adequate reward function (as defined in Chapter 3.1), the policy will implicitly learn to avoid such states.

We propose a novel automatic goal selection method that is simple and easy to implement. We introduce an adversarial goal selector to the policy's training, the "Director". In parallel to the policy's training, we train online a failure predictor that learns to predict if the policy will visit a crash state on a given goal. This failure predictor parameterizes the Director. By querying the failure predictor, the Director can retrieve goals that are adequately hard and better shape the policy's training. This strategy requires no explicit constraints, input space augmentation, computing auxiliary safety violation optimization terms, or additional policy neural parameters (all common requirements for "Safe DRL" methods), all of which can make training more brittle [**56**][**34**]. This makes our method ideal for robotic learning.

The main contributions of our work are as follows:

(1) We propose a novel training-goal selector that we consider to be simpler to implement and to understand than comparable state-of-the-art methods. To our knowledge, our method is the first training-goal-selection method that optimizes for and is shown to improve the agent's safety.

(2) Adversarial training for DRL agents is still not well understood. Overly adversarial adversaries often lead to sub-optimal final policies [**14**]. Overly cooperative adversaries lead to coddled policies that are ill-adapted to the rigors of their evaluation

or deployment environments. We offer an ablation study over the “cruelty” of our method’s adversarial agent. We identify a desirable balance between cooperative and adversarial behaviours.

(3) We evaluate our strategy on a series of tasks, including a series of toy (generalist) problems, and two very challenging quadruped locomotion and manipulation tasks. Empirically, our strategy reduces trained policy population variance and median for total evaluation crashes (or produces agents that are just as good as the best baselines in study). It similarly improves or maintains the final agent’s ability to perform the task (quantified through evaluation reward).

(4) During evaluation, we train a fully adversarial goal-selector agent. We then pit the control policy against these adversarial goals. We show that our strategy greatly improves the policy’s resistance to these goals. Against them, compared to baseline methods, the policies trained using our method show increased reward performance and decreased number of crashes, along with marked variance reduction for these metrics for populations of trained policies.

Chapter 2

Background

In this section, we discuss the necessary background aspects of the problem of sub-goal generation for safer trained final policies that we tackle. We discuss in Chapter 2.1 the preliminary knowledge necessary to understand the problem setting. We outline related work in Chapter 2.2.

2.1. Preliminaries

First, we define what a control problem is and we summarily discuss various classical control methods in Chapter 2.1.1. In Chapter 2.1.2, we discuss Markov Decision Processes, an overwhelmingly common model used to represent control tasks. We then detail Bandit problems in Chapter 2.1.3. In Chapter 2.1.4, we summarize a category of control methods called "reinforcement learning". In Chapter 2.1.5, we summarize "deep reinforcement learning". In Chapter 2.1.6, we offer a brief overview of curriculum learning, or the idea that some meaningful order of tasks is more conducive than some other for learning a target task. Finally, we briefly summarize adversarial training in Chapter 2.1.7.

2.1.1. Control

Control is the task of selecting actions to issue to an agent in such a way that it solves a problem statement while minimizing some notion of cost over the problem's states and actions.

A controller acts upon control variables. For example, for a self-driving car, the control variables could be the steering angle and the pressure put upon the gas and brake pedals. For a robotic arm, some control variables could be the joint angles and end-effector positions.

One of the earliest examples of a control problem is Johann Bernoulli's 1696 Brachistochrone problem [**27**], wherein one must find the optimal curve for a marble to go from point A to point B in the least amount of time. Here, the control variable is simple: one can only influence the shape of the curve, and one can only change the curve once, before the marble even starts rolling. See Figure 2.1 for a visual explanation of the problem.

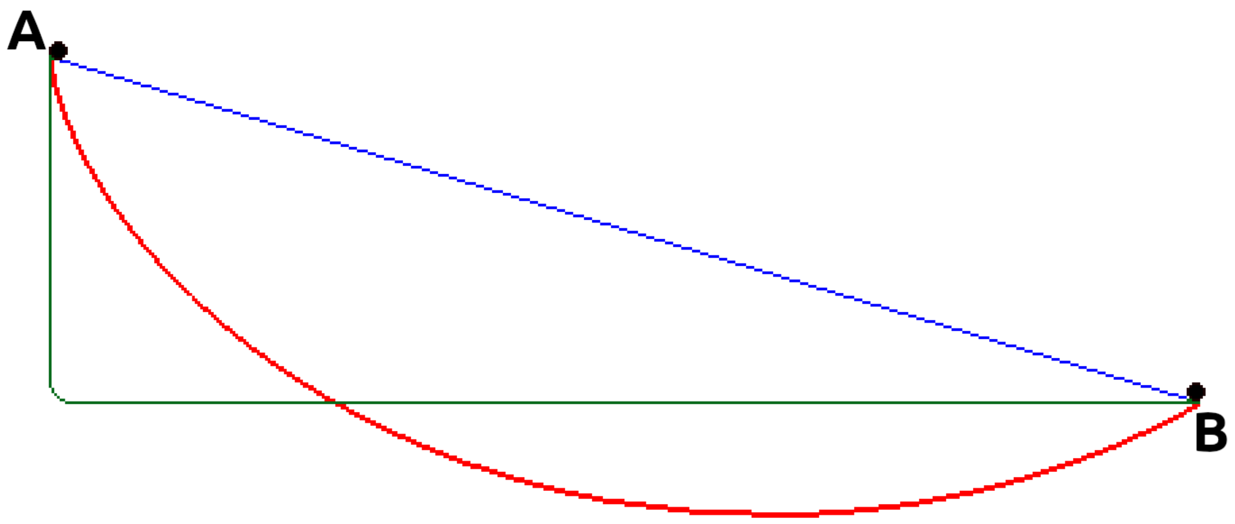


Figure 2.1 – Visual explanation of an instance of the Brachistochrone problem. There exists an infinity of possible curves that would lead a marble from point A to point B (three possible curves are represented in blue, red, and green). Before letting the ball fall and roll down the curve according to gravity, the agent tasked with solving the problem sets the shape of the curve. Which curve enables the marble to reach point B the fastest? The optimal curve (in red) can be derived analytically using calculus, or namely through methods derived from the control field. *Image obtained and modified from the Brachistochrone Wiki entry* [**63**].

Another early example of a control problem comes from the famous *On Governors* by Mr. J. C. Maxwel [**38**], where the basis of a unifying theory for the analog ship-governing and windmill-regulating devices of the time is put forth. The main problem in study is the task of issuing commands for a ship's governor such that the ship stays on course and maintains a requested speed. Contrarily to the Brachistochrone problem, in this case, the control variables (steering, engine speed or sail angles, etc.) can be influenced at any instant. *On Governors* subsequently led to the discovery in the 1920's of what is possibly the most famous general-purpose controller, the Proportional-integral-derivative (PID) controller [**39**].

A PID controller is defined using three different terms. The proportional term (P) produces an action for the control variables based on the "error" term (e), the difference between the current state of the system and the desired state. The integral term (I) seeks to eliminate historical error terms. The derivative term (D) evaluates the rate of change of the error and produces actions that try to eliminate the anticipated error.

$$u(t) = Pe(t) + I\int_0^t e(\tau)d_\tau + D\frac{d_e(t)}{d_t}$$

Notice that the PID controller aims to minimize the cost function $e(t)$. The modern optimal control field poses *control* as a decision policy for the control variables and uses the cost function to tune this policy. Finding the optimal control law amounts to solving the optimization problem defined by the relationship between the control equations and the cost function. A few different methods can be used to solve the optimization problem, from simple heuristics to classical dynamic programming to more recent methods. State-of-the-art model predictive control (MPC) essentially involves solving a simpler control problem at every single timestep to find the best current action while still accounting for the future [**18**].

In all of the above cases, the system can be described by a state s_t which is given to the controller, which then produces action a_t in response. The system then transitions to state s_{t+1}, and, if it is allowed to, the controller again takes some action in response. There exist many ways to model this feedback loop. One of the most common, and the most relevant for this work, are Markov decision processes.

2.1.2. Markov Decision Process

A Markov decision process (MDP) is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \rho_0)$, where $\mathcal{S}$ is the set of possible system states, $\mathcal{A}$ is the set of possible actions one can apply to the system's control variables, $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ describes the state-action transition dynamics, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ describes the possibly-stochastic state-action reward function (*cost function* in traditional optimal control settings), and ρ_0 describes the initial state distribution. As their name implies, MDPs have the *Markov property*: one only needs the current state and action to describe the future state (conditioned on some transition stochasticity encapsulated in $\mathcal{P}$).

Here is a list of terms that we utilize in this work to refer to different concepts related to MDPs:

— A "timestep" t is the current step count of the problem.

— A "transition" is a tuple describing the change in the MDP's state and the reward issued in reaction to the policy's action, i.e. (s_t, a_t, r_t, s_{t+1}).

— An "episode" refers to a single "rollout" of the policy in the MDP, i.e. every (s_t, a_t, r_t, s_{t+1}) tuple from a given initial state $s_0 \sim \rho_0$ until the final state of the episode. The final state either belongs to some subset $\mathcal{S}_T \subset \mathcal{S}$ of terminal states, or is simply the result of an upper bound on the number of possible timesteps.

— An "infinite-horizon" problem is an MDP where no timestep limit exists, i.e. t is unbounded.

— The entity taking actions in the MDP can be referred to as the "decision policy" π ("policy", for short). The task of $\pi : \mathcal{S} \to \mathcal{A}$ is to take actions such that it maximizes (minimizes) a function of some kind over the reward (cost) terms encountered during an episode. This function is usually the expected cumulative sum of the reward signals r_t encountered during the episode, discounted by the "discount fator" $\gamma \in [0,1]$.

— A "rollout" refers to any data collected from policy-MDP interactions. In practice, it mostly refers to the latest few generated transitions or episodes.

In MDPs, the reward function $\mathcal{R}$ habitually describes a *task* that must be accomplished. A simple example of a commonly-used MDP research environment with a single task is *Minigrid* [**11**], in which π must issue movement commands such that the agent reaches a goal in a maze. This might include more complex navigation sub-tasks such as grabbing keys to unlock doors. A more complex example of a task would be asking a robot arm "make me a cup of tea", which would also include many sub-tasks ("boil water", "grab a cup", "grab a tea bag", etc.), all described in the singular reward function.

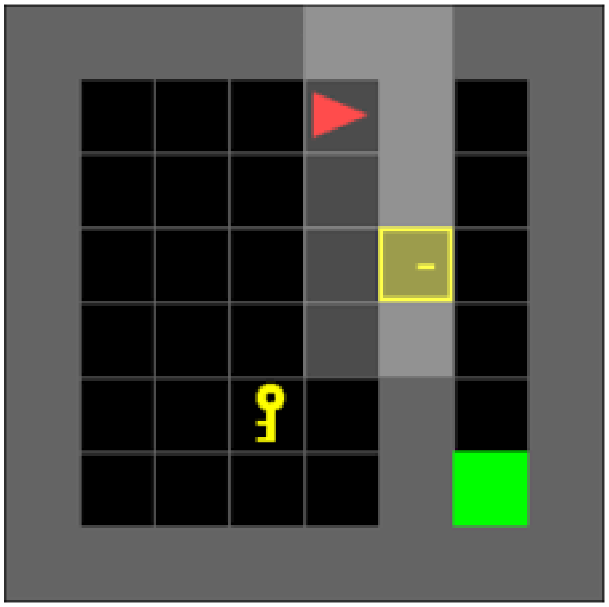


Figure 2.2 – *Minigrid* is a very popular environment for benchmarking control policies [**11**]. The agent must reach the goal represented by a green cell. In some variants, there are coloured keys that must be used to unlock corresponding doors.

2.1.2.1. **Goal-conditioned Markov Decision Process**.

In a goal-conditioned Markov decision process (GMDP), a *goal condition* is added to the system description.

Generally, GMDPs are defined by a tuple $(\mathcal{S}, \mathcal{G}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \rho_0, \rho_g)$, with $\mathcal{G}$ the set of possible goals and ρ_g the distribution from which goals are sampled (which can be a simple probability distribution, or be controlled by some higher level decision process of some kind). Because of the addition of the goals, the reward structure changes to $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{G} \to \mathbb{R}$.

Going back to the examples given above for MDPs, a simple example of goal-conditions in *Minigrid* [**11**] would be text prompts that ask the agent to accomplish some sub-tasks necessary to solve the maze ("grab yellow key", "unlock yellow door", "reach goal"). A more complex example of a series of goals would be sequentially asking a robot arm "boil water", "grab a cup", "grab a tea bag", etc., with possibly a different reward function for each. In practice, the addition of the goal-condition as input to the reward function can allow us to more precisely describe the task at hand to the policy.

Some care must be given to tie the GMDP formulation to robotic control. In the field of robotics, specific terms are used to describe different control problems. For example, a *command* normally describes a velocity target or a requested trajectory or a set of target joint angles. A *goal* habitually presupposes a navigation task of some sort, where the agent must reach some particular location in the environment. A *task* is normally some more complex set of motions, often requiring complex maneuvers that could be described as a combination of commands and goals. From the point of view of GMDPs, all of the above are simply *goals*. In this work, for the sake of simplifying our notation, we follow the GMDP formulation.

One popular generalization of MDPs and GMDPs is the Partially observable Markov decision process (POMDP) formulation, where the observations given to π are not sufficient to describe the full state of the system. In other words, there exists some *observation function* that alters the state s_t to create observation o_t. This observation is what is then given to π.

It is obviously impossible to fully describe the real world's state with our current technology. For this reason, defining or simulating robotic control problems often requires POMDPs. In this work, we only consider goal-conditioned POMDPs, but to ease our notation, we still denote them as standard GMDPs without affecting the validity of our results.

2.1.3. Bandits

Going back to the historical examples of control problems described in Chapter 2.1.1, whereas one might use an MDP with very small timestep increments between state transitions to model the task of controlling a ship's governor (because actions can be issued at any given instant), modelling the Brachistochrone control problem using an MDP seems impractical. After all, the only instant in which the controller can issue an action is before the marble even begins its descent down the curve. Why go through all of the state transitions required to fully model the marble's trajectory when all of this could be described as a singular reaction to the shape of the curve issued by the controller? Instead, one might want to use a Bandit to represent the problem.

For the purposes of this work, Bandits can simply be interpreted as single-step MDPs, i.e. MDPs in which the policy in play can only issue a singular action before the control episode ends. As such, the tuple defining a bandit is much simpler than the one used to define an MDP: (C, A, R), where C is the context given to the decision policy (some bandit problems provide no context, i.e. $C = \emptyset$), A is the set of actions available to the policy, and $R : S \times A \to \mathbb{R}$ is the reward function.

A subclass of bandits of particular interest are Bernoulli bandits, with binary reward functions $R : C \times A \to \mathbb{1}\{\text{success}\}$, where $\mathbb{1}$ is the indicator function. While seemingly restrictive, Bernoulli bandits are actually very common and hold important relevance for this work.

2.1.3.1. **Thompson Sampling**.

A common heuristic for finding decision policies for Bernoulli bandits with discrete action spaces was formulated in 1933 by Canadian entomologist William R. Thompson [**60**]. In *Thompson sampling*, a state-action function predicting success probability is represented by a Beta distribution parameterized by the historical success/failure of each action [**53**]. During the initial decision episode, this corresponds to a uniform policy over the actions. Every episode thereafter, the agent samples an arbitrary number of possible actions, rates them by the success prediction function, and selects the highest-valued. This offers a good exploration-exploitation trade-off. With adequate tuning, such a strategy is close to being optimal [**10**]. See Figure 2.3 for a visual intuition into Thompson sampling.

2.1.4. Reinforcement Learning

Reinforcement learning (RL) is perhaps the most famous area of research when it comes to defining control policies for both MDPs and Bandits. In reinforcement learning, a search in policy space is performed to find the best policy to operate in the MDP, i.e. the π that maximizes the expected γ-discounted sum of rewards. The *value function* $v^\pi(s)$ describes the *value* of being in a state s for policy π.

$$v^\pi(s) = \mathbb{E}\left[\sum_{t=0}^{\infty} \gamma^t \mathcal{R}(s_t, \pi(s_t)) \middle| s_0 = s\right]$$

By finding the policy with the maximal v^π $\forall s \in \mathcal{S}$, we find the optimal policy for the MDP.

A great many number of methods exist to perform this search. One such method is the eponymous *REINFORCE* [**57**], in which a policy π parameterized by weights θ collects data from rollouts. Every episode, the gradient term $\nabla_\theta \log \pi_\theta(s_t, a_t)$ is computed for all (s_t, a_t) in the episode. The gradient terms are then multiplied by the discounted sum of *future* rewards gathered from the transitions *after* their respective timestep t. The resulting summation over the gradient-future-rewards terms results in a gradient that allows us to update the

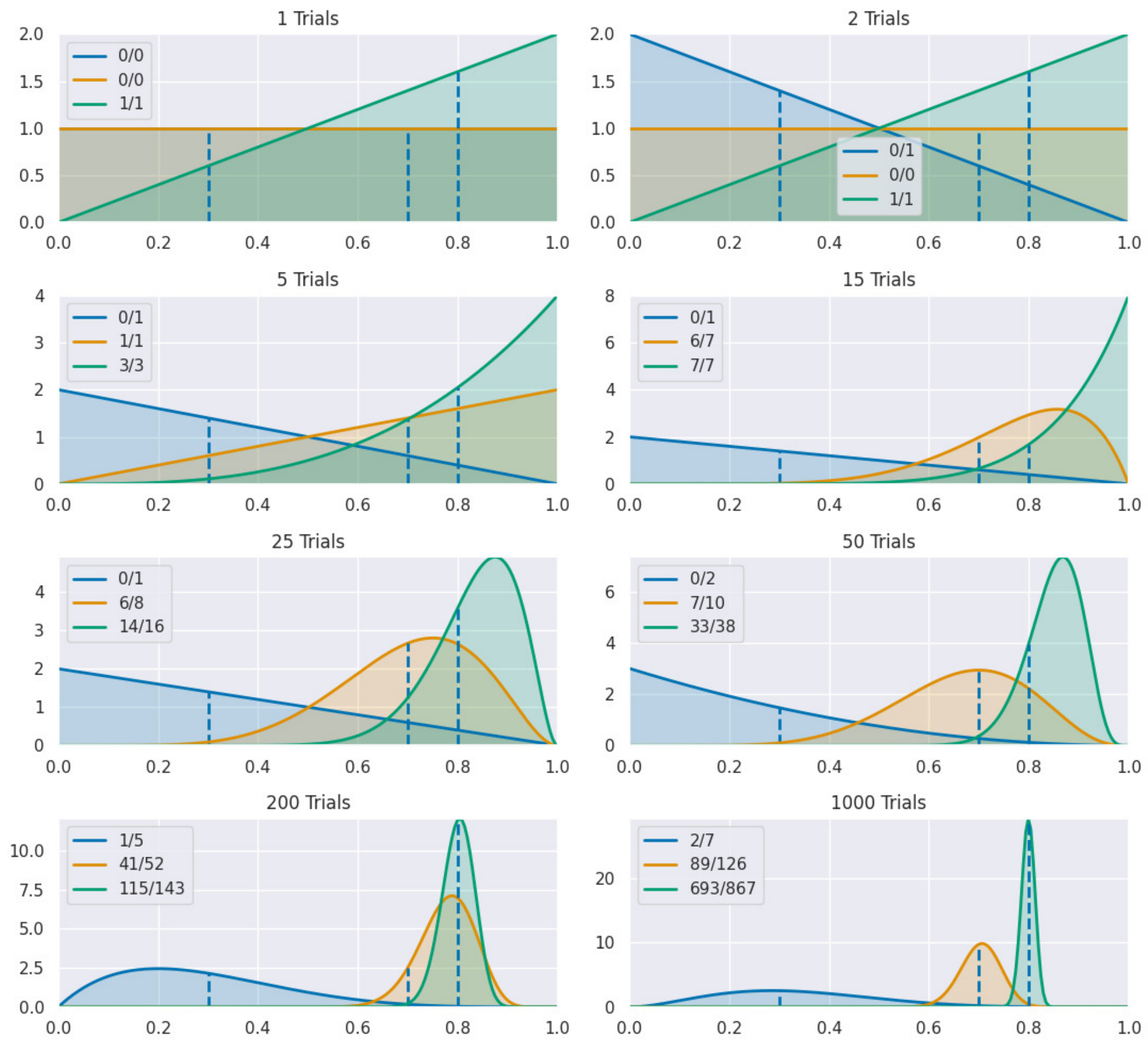


Figure 2.3 – Visual explanation of Thompson Sampling. Every possible action has a corresponding true success probability: 0.3 for blue, 0.7 for orange, and 0.8 for green. At first, each possible action has never been picked, and so the estimated success probability is equal for each. The policy thus randomly picks an action. Every control episode thereafter, the control policy picks actions randomly, weighted by their historical success rate (shown in the legend of each plot: success/trials). By the end of the experimentation, the policy has correctly converged to almost always selecting the green action. *Plots obtained from a slight modification of BabyRobot* [**49**].

policy π_θ's weights θ towards a policy with a better value function. Iteratively, we find the optimal policy.

For more complex problems, such as MDPs with large and/or continuous states and action spaces, classical reinforcement learning is ill-fitted. For example, classical Q-learning RL methods learn a $\mathcal{S} \times \mathcal{A}$ table where the cells describe the value of taking action a_t for a given state s_t [**62**]. The final policy selects the best action for its current state according to the table. This simple approach works well for small discrete action and state spaces, but it is an obviously intractable technique for continuous spaces, or even for large discrete spaces (one of the pioneers of RL, R. Bellman, calls this the "curse of dimensionality" [**3**]). Other classical RL techniques have similar issues. For example, training classical RL on visual inputs or comparatively-simple games with large discrete spaces such as *Go* is considered fairly intractable [**41**].

2.1.5. Deep Learning and Deep Reinforcement Learning

In deep learning [**31**], neural networks are used to tractably learn complex functions from data. Put simply, using neural networks, we can learn a mapping from input to output from demonstrations of (input, output) pairs.

To do so, we define a differentiable *loss function* that describes how far the network's current outputs are from the expected outputs for a given input. Because the loss function is differentiable (and so is the neural network), we can compute the gradient of the loss function for the network's parameters with regards to a batch of (input, output) pairs. This very costly computation is rendered tractable by using a now-standard technique called *back-propagation* [**52**][**19**]. Armed with the gradients, we can then tune the network's parameters to reduce the loss for each batch of data. By applying this procedure multiple times for all data points, we iteratively find the network parameters that minimize the loss [**7**]. Even though this optimization process generally has no analytical solution, and the optimization landscape is often non-convex, neural networks empirically converge to very reasonable functions, often outperforming convex relaxations of the problem [**26**].

Deep neural networks have successfully been applied to the problem of reinforcement learning. For example, for the case of intractable Q-learning described in Chapter 2.1.4, a deep neural network-enabled equivalent called Deep Q-Network (DQN) was shown to suitably handle continuous spaces and large discrete spaces [**42**]. Deep reinforcement learning was shown to beat *Atari* games [**41**], offer superhuman performance in games such as *Go* or *chess* [**54**], and even beat human champions at challenging multiplayer real-time video games such as *Dota 2* [**5**].

DRL has also successfully been applied to very challenging robotic control tasks. In one of the most famous examples, a control policy for a *Shadow Dextrous Hand*, a particularly lifelike robotic reproduction of a human hand, was learned in simulation and successfully deployed in the real world [**43**]. Some examples among many others include learning control policies for quadrupedal robots from scratch [**51**], or learning how to control robot arms [**21**].

2.1.5.1. **Proximal Policy Optimization**.

Of particular interest to this work is a method called proximal policy optimization (PPO) [**55**]. In this method, the policy learns from data that incorporates transitions that were recently collected from policy rollouts (PPO is an *on-policy* algorithm). The details and tricks required to make PPO work are both numerous and out of scope for this work (a subset of such tricks are outlined in the aptly-named *The 37 Implementation Details of Proximal Policy Optimization* [**24**]). In general, the main contribution of PPO is the idea of moving the policy's weights only slightly after each update. This is done using a measure of the difference between the action distribution of the current policy, and what the new policy's distribution would be after the update to its parameters. Given this difference, one can then alter the gradient update step of the policy in order not to change the policy too much on a single update. As such, the policy's behaviour before and after an update step are remarkably similar.

PPO enables the use of massive parallelization during policy rollouts, something that very few other methods support. As outlined in Chapter 1, the use of simulation to learn robotic control policies is widespread in order not to damage the real hardware with sub-optimal policies, so this potential for paralelization is greatly desirable. PPO and learning for robotic control go hand-in-hand [**43**][**51**][**35**].

2.1.6. Curriculum Learning

To humans, teaching complex tasks is instinctively a sequential process of some kind. All of our (functional) schooling systems follow a pattern, no matter the country or historical period. Inevitably, the schooling system starts with very small and simple tasks that build up to some larger task. As the students get better, the tasks get harder and harder. The specific optimal sequence of tasks is not known; every schooling system uses a different curriculum of tasks. But one constant is present in every (functional) human schooling system: some meaningful possibly-stochastic order of tasks exists that is touted to be more conducive to learning.

For the machine learning community, the act of ordering tasks in some meaningful way in order to guide training is called curriculum learning (CL). The earliest (and eponymous) work on this subject showed that CL can enable more rapid convergence and better generalization [**4**]. Traditionally, such curricula were hand-made by the human practitioner. We compare against one such hand-made curriculum as a baseline method, detailed in Section 4.3.

Since, many different automated CL methods were developed. Most common examples consider CL for supervised learning, where some better order of classification or regression tasks are shown to the network during training to improve its convergence and/or performance.

Other methods seek to generate or tune environments for reinforcement learning problems. One of the more famous and recent methods in this area is Protagonist-Antagonist Induced Regret Environment Design (PAIRED) [**14**], where two agents are introduced to the policy's training. The "antagonist" and the "adversary" form a team against the policy (the "protagonist"). The three agents in play are all DRL policies. The adversary's task is to generate environments. Both the protagonist and the antagonist seek to solve these environments. The adversary's reward signal comes from the *regret* of the protagonist, i.e. how much better the antagonist does compared to the protagonist. This leads to a natural progression in the complexity of environments generated by the adversary, and helps shape the training of both the protagonist and antagonist.

Of particular interest to us are curriculum learning methods that seek to generate curricula on the space of goals for a GMDP. Since such methods are the most similar to our own, we detail them separately in Section 2.2.3.

2.1.7. Adversarial Training

Common methods to automatically generate curricula for machine learning utilize adversarial training. By pitting two agents against each another, they bootstrap their strategies, forcing each other to get better and better as training goes on. Most commonly, in these methods a *zero-sum* game is formulated. In such games, one agent's gain is the other's loss. Think of playing rock-paper-scissors: when you win, the other loses.

A more complex example of a zero-sum game is the training setup of a basic Generative Adversarial Network (GAN)[**20**]. In this setup, a generator learns to generate synthetic data that resembles data from a ground truth distribution and a discriminator learns to tell the synthetic data from the ground truth data. When the generator fools the discriminator, it wins and the discriminator loses, and vice-versa. This corresponds to two opposed optimization objectives. A common formulation for such training is a *minimax* game, where one agent maximizes the objective that the other agent minimizes. Here is the *minimax* game used to train GANs [**20**]:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1 - D(G(\mathbf{z})))] \tag{2.1.1}$$

In Eq 2.1.1, D is the discriminator, G is the generator, ρ_{data} is the target data, and p_z is the noise used to condition the generator to generate an image. As mentioned above, when the discriminator wins, the generator loses. This generates a learning signal that updates the generator.

Since the original formulation of the GAN method, many similar works were developed. Some variants include conditioning the network to produce specific modes of data [**40**] or transferring styles between images [**64**]. We detail adversarial training methods that are adapted to the problem of training-goal generation that we tackle in Section 2.2.3.

2.2. Related Work

As discussed in Chapter 1, we are interested in the influence of the selection of training goals on the deployment or evaluation safety of a DRL policy. In so doing, we propose a novel and complementary approach to safety that has, to our knowledge, not been investigated before. In Chapter 2.2.1, we detail methods that utilize data intrinsic to a problem to guide a DRL agent's training. In Chapter 2.2.2 we detail the established methods normally used to attain DRL safety, and detail how such methods are still fully compatible with ours. In Chapter 2.2.3, we discuss various methods related to the task of goal selection during training.

2.2.1. Intrinsic Motivation

Intrinsic motivation methods seek to find some principled way to construct reward signals for DRL agents. In this sub-field of DRL, the reward signal offered by the MDP is seen as an *extrinsic* signal. One then seeks to learn or otherwise generate *intrinsic* reward signals from the behaviour of the agent in the MDP. A high-level human-centric example would be learning how to play the guitar using both extrinsic signals (a teacher that guides one's learning, and the fact that one's music sounds "off" compared to the target song) and intrinsic signals (learning the guitar to impress people, or spending long hours learning it from a perfectionist mindset).

Many intrinsic motivation methods are named after emotions or concepts that echo human intrinsic motivations. In the "Intrinsic **Curiosity** Module", a forward model learns to predict the future state s_{t+1} from the current state-action pair (s_t, a_t). Some notion of difference between the predicted next state and the true next state is then used as a reward signal for the agent [**44**]. In "**Diversity** is all you need", a discriminator learns to identify individual agents in a population by their behaviour. The error of the discriminator is used a reward signal for the agents [**16**]. In "**Surprise** Minimizing RL", a model is learned that predicts the environment's reaction to an action a_t in the presence of entropy extrinsic to the agent. The intrinsic reward signal is based on the minimization of the surprise of the model. By minimizing the surprise, the agent is incentivized to exert control upon its environment in such a way that it seeks predictable transitions, avoiding the undesirable entropy of the environment [**6**].

Even in the absence of extrinsic signals, intricate behaviour emerges from the intrinsic motivation. This leads some to believe that intrinsic motivation methods have the potential to replace extrinsic signals in large parts, especially in sparse reward settings [**2**].

2.2.1.1. **Random Network Distillation**.

Again inspired by human emotions, random network distillation (RND) gives the agent reward signals based on maximizing *surprise* (effectively targeting the inverse of "Surprise Minimizing RL") [**8**]. To do so, RND utilizes two networks with differently initialized weights. One of these networks is *frozen* with the initial weights it is instantiated with (the network receives no training whatsoever). Whenever the agent takes a step in the environment, the two states s_t and s_{t+1} are concatenated and passed through the frozen network, producing output o_{frozen}. The inputs are then passed through the other network, producing output $o_{trained}$. The reward given to the agent is based on the difference between o_{frozen} and $o_{trained}$.

The *surprise* terminology comes from the prediction failure of the trained network. When it outputs values that are far from the frozen network's outputs, the error term comparing the two outputs is big, and so it is said that the trained network is *surprised*.

2.2.2. Reinforcement Learning Safety

"Safety" is a difficult thing to define in the scope of DRL. For example, one might define a chess-playing robot arm's task as some compound function composed of its ability to grasp and move chess pieces and its ability to win chess games. This hypothetical reward function would fully describe the robot's task (win chess games). However, hidden within this task are safety considerations. The arm shouldn't grasp the pieces so hard that they break, it shouldn't throw the pieces off the board, and it should also **never** grasp things that are not chess pieces. Unfortunately, such concerns are hard to account for ahead of time, and accidents do happen [**37**] [1].

RL safety is the name given to the consideration of safety for RL agents. This includes both safety considerations known ahead of time and safety considerations that emerge upon deployment. "Safe RL" seeks to ensure the agents act safely at all times, both during training and during deployment. This safety guarantee is normally achieved though convergence proofs that show the agent's training *should* converge to a policy that is both optimally performant (achieves the optimal value function) and optimally safe. In practice, due to the highly non-convex optimization problem that DRL training tries to solve, not all convergence-proven "Safe RL" methods actually converge to an optimal policy on complex tasks [**56**]. In

1. While the robot involved in the cited accident was not necessarily controlled using reinforcement learning, this event still exemplifies the difficulty of coupling *task specification* and *safety considerations*.

contrast, while our method offers no safety guarantee, we empirically show that it improves both safety and performance on a series of very complex legged robotic locomotion tasks.

We elect not to compare against “Safe RL” baselines in this work. Our method is explicitly not a “Safe RL” method. One might consider it to be an “Unsafe RL” method: to achieve safety after training, we actually encourage unsafe behaviour during training. Comparing against methods that enforce safety at all times would be unfair both for us and for them; we simply do not operate under the same assumptions. In the same vein, our method is actually completely compatible with “Safe RL” methods. Since “Safe RL” methods offer safety guarantees, they should gracefully handle the increased risk coming from environments that utilize our method, and the underlying RL agent should also benefit from the goal distribution that our method offers.

Here, we present three large categories of “Safe RL” methods: methods utilizing “recovery agents” in Section 2.2.2.1, methods utilizing reward shaping in Section 2.2.2.2, and methods enforcing optimization bounds in Section 2.2.2.3.

2.2.2.1. **Recovery**.

There exists a category of “Safe RL” methods that is particularly well-adapted to robotic problem settings. These methods utilize *recovery agents*. In such methods, the DRL agent optimizes for performance. A second agent (the recovery agent), which is either learned or hand-coded, is tasked with detecting and recovering from unsafe states. Some variations of this idea include (1) Learning from data collected prior to training, and then optimizing a mixture of two different policies (one targets performance, the other targets safety) [**59**], (2) Learning a safe DRL policy that recovers from the unsafe states visited by a DRL policy that guides a low-level MPC controller [**65**], or (3) Leveraging a library of recovery experts to select from and learning a curriculum of recovery strategies for the DRL policy [**61**].

There is a related imitation learning technique (learning a policy from expert demonstrations [**25**]) that also leverages the mixing of a learner agent and an expert policy. In Dataset Aggregation (DAGGER) [**50**], a classical policy (the *expert*, controlled using MPC, PID, or others) acts in an environment. Sometimes, the imitation learning policy acts in its stead (or random noise is applied to the expert’s actions), forcing the expert away from the safe behaviour it normally acts in. The expert then shows the policy how to recover to its normal behaviour. This enables the imitation learning agent to learn how to act safely in states it would normally have a difficult time navigating.

2.2.2.2. **Reward Shaping for Safety**.

Some works seek to encode the safety considerations directly in the reward function. One example of this category of works is “Saute RL” [**56**], where an encoding of the safety constraints are concatenated to the observations sent to the RL agent and the reward objective is reshaped to reflect safety violations. This enables the policy to operate under different “safety budgets”, a measure of how many safety violations it can make. The converged policy *acts safely with probability one*.

Of particular interest to this work is “Intrinsic Fear” [**32**]. A *fear network* is learned to predict if a given state s_t is within k transitions[2] from a *catastrophe state*, a state that violates safety conditions. This intrinsically-motivated fear network is then used to shape the reward signal given to the policy such that it is incentivized to avoid states within the *fear radius* k of states around the catastrophe states.

However, in general, encoding the safety considerations directly in the reward function is considered to be mostly intractable [**45**][**1**][**12**]. In fact, such reward shaping is often detrimental to the policy's optimization in non-trivial problems because of *early termination* issues. Giving the policy negative rewards for acting unsafely during its initial exploration steps can produce overly-cautious policies that fail to find good behaviours [**51**][**29**]. In specific task settings however, it is possible to fully describe safety through rewards. Our method requires this to be true, as explained in Chapter 3.1.

2.2.2.3. **Encoding Safety in the Optimization Process**.

This class of methods seeks to impose safety through constraints applied to the optimization of the DRL agent. Perhaps the most famous are methods using the conditional value at risk (CVaR) formulation. In this setting, satisfying the maximal CVaR is encoded in the optimality criterion for RL [**13**]. By satisfying the criterion, the agent is able to maximize performance conditioned on the risk of bad outcomes in mind. In other words, the agent achieves the optimal value function while avoiding the worst-case outcomes.

2.2.3. Goal Selection

The task of selecting goals such that an agent learns with more efficacy, or such that an agent reaches some ultimate goal through a series of sub-goals, is called “goal selection”. We refer the reader to the excellent survey on RL in GMDPs by *M. Liu et al.* [**33**] for a more in-depth investigation into such methods.

We present two important examples. In the first, a GAN [**20**] is learned to generate goals of intermediate difficulty for a learning DRL agent [**23**]. This helps guide the agent's

2. With k being a hyperparameter.

training, improving convergence speed and policy generalization. In the second, a second DRL agent called the “teacher” is introduced [**9**]. This agent generates goals for the student, and is trained using a reward signal based on how long the learner agent (the “student”) takes to reach the goal. This again helps improve convergence speed, and also enables the policy to solve harder tasks than when trained using baseline methods.

We compare against a state-of-the-art method of this category as a baseline. We present this method in Section 2.2.3.1.

We note that many works of this category are tuned for task settings with sparse goals (achieving goals requires visiting rare states), with sparse reward functions (the reward signal is zero for most transitions). Some strategies that deal with this issue include incentivizing visitation of past achieved goals [**47**] or learning an intrinsically-motivated uniform distribution for image-based goals [**48**]. In our case however, we experience the opposite problem: we assume a dense reward reward function where the (continuous) goals are easily achievable. Instead, the safety violation cases are sparse and **must be made even sparser** by improving the policy.

2.2.3.1. **Value Disagreement Sampling**.

In this training-goal selection method, an ensemble of DRL critics are trained in place of the single critic traditionally used by DRL. Whenever a goal must be selected, a large number of goals are sampled and are then adequately concatenated to the state s_t. The ensemble then scores them. Because each critic has different initial weights, and there is some stochasticity in their training, the values produced by the ensemble have some variance, i.e. *disagreement*. Different notions of disagreement exist, but as the paper shows, simply basing the goal selection strategy off of the standard deviation of the scores leads to very good results [**67**]. Value disagreement sampling (VDS) greatly outperforms baseline methods and is considered a state-of-the-art training-goal selection method (or curriculum method for goal selection).

Chapter 3

Dread Enforced Interactions for More Optimal Sampling

In this work, we investigate the influence of goal selection during training on the evaluation safety and performance of deep reinforcement learning policies over a subset of GMDPs that models common robotic control problems.

3.1. Problem setting

As defined in Chapter 2.1.2.1, we consider GMDPs defined by tuples of the shape $(\mathcal{S}, \mathcal{G}, \mathcal{A}, P, R, \rho_0, \rho_g, \mathcal{T})$. We consider $\mathcal{G}$ to be some continuous vector space.

Note that here, the GMDP formulation outlined in Chapter 2.1.2.1 is augmented by the "goal sampling rate" or "resampling time" $\mathcal{T}$. During both training and deployment episodes, a new goal $g \sim \rho_g$ is sampled every $[\mathcal{T} > 1]$ GMDP timesteps, regardless of the current state of the GMDP. We call the goal-change events "interactions" and they are described by tuples (s_t, g_t, g_{t+1}), with t the current GMDP timestep, s_t and g_t the current state and goal, and g_{t+1} the goal at timestep $t+1$.

We consider infinite-horizon episodes. We assume that $\mathcal{R}$ is a dense[1] reward function (the vast majority of transitions have a non-zero associated reward) that is majorly based on matching g_t. Because DRL for robotics is negatively affected by early termination problems [**30**][**51**], we assume positive reward signals, i.e. $\mathcal{R}(s_t, a_t, g_t) \geq 0$ on all inputs.

A subset $\mathcal{S}_c$ of $\mathcal{S}$ are undesirable terminal states called crash states. Avoiding the crash states is the only safety consideration of the system. Some examples of crash states include a legged robot flipping on its back, a manipulator arm crossing some undesirable force

1. Some might argue that assuming dense rewards for robotics problems is unrealistic. However, as mentioned in Chapter 1, our method seeks to achieve better **evaluation** safety by incentivizing unsafe behaviour during training. We argue that because of this, our method is only tractable for training in simulation. Thus we can safely assume access to privileged information that describes the state of the world. Hence, dense reward functions are very tractably definable.

threshold, or a wheeled robot exiting its designated operational area. Again because of early termination problems, these states do not have any associated negative reward terms.

This GMDP formulation models the behaviour of a low-level DRL policy responding to robotic *commands* $\in \mathcal{G}$ set by some higher-level policy. One example would be a remote-controlled car tasked with following linear and angular velocities requested by some joystick controller: the high-level policy (the joystick) may issue new high-level goals (a desired velocity and a steering angle) at any time that the low-level policy (the controller onboard the car regulating power to the wheel motors) must immediately match. In this toy-car scenario, crash-states would be collisions with surrounding obstacles. **We desire low-level policies that match requested goals to the best of their ability while still being aware of safety considerations.**

3.1.1. Optimization objectives

To study the influence of goal selection during training, we require a more expressive goal distribution. We endow ρ_g with the ability to select future goals according to the current state and goal.

In this setting, the DRL policy's performance (J) and safety (J_s) objectives are as such:

$$\max J(\pi) = \mathbb{E}_{\substack{g_t \sim \mathbb{p}_g \\ a_t \sim \pi(s_t, g_t) \\ s_{t+1} \sim P(s_t, a_t)}} \left[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, g_t) \right] \tag{3.1.1}$$

$$\min J_s(\pi) = \mathbb{E}_{\substack{g_t \sim \mathbb{p}_g \\ a_t \sim \pi(s_t, g_t) \\ s_{t+1} \sim P(s_t, a_t)}} \left[\sum_{t=0}^{\infty} \mathbb{1} \left\{ s_{t+1} \in S_c \right\} \right] \tag{3.1.2}$$

$$\text{where } \mathbb{p}_g = \begin{cases} \rho_g(s_t, g_{t-1}) & \text{if } [t \mod \mathcal{T} = 0] \\ g_{t-1} & \text{otherwise} \end{cases}$$

Notice that because we consider infinite-horizon problems with dense non-negative rewards, the safety objective (Eq 3.1.2) is already being optimized for as a byproduct of Eq 3.1.1. Visiting a crash-state ends the episode early, directly resulting in lower potential cumulative rewards. Given an adequate discount factor γ, an optimal policy would avoid such states. And so we pose a strong but principled assumption for the problems for which our method is applicable: avoiding crash-states is the optimal strategy for the decision policy.

3.2. Method intuition

We posit that an optimal distribution of training goals exists such that policies π_θ trained under it visit less crash-states at evaluation time than when trained under a sub-optimal goal distribution.

An oracle goal selector ρ_g would presumably select goals that are neither too easy (already solvable) nor too hard (intractable), yet still suitably handle *catastrophic forgetting* [**28**] (perhaps by occasionally sampling easier goals) and exploration (perhaps by occasionally sampling harder goals). The mechanics of learner agents being incredibly complex, we pose three major hypotheses to render this problem tractable: (1) The probability of π_θ crashing after receiving a goal is a useful signal for goal selection, (2) An oracle would presumably select goals with intermediate crash probabilities often, but also select easy and hard goals occasionally to ensure the policy can handle the entire goal space, and (3) As the policy learns to act upon them, goals that lead to crash-states implicitly become rarer and rarer; making them more common would give the policy more experience related to crash-states to learn from.

Named after the policy's own intrinsic fear of crashing, we call the probability of crashing after receiving a goal the "fear score", given by a "fear function" (following Intrinsic Fear's naming scheme [**32**], see Chapter 2.2.2.2). Inspired by the hypothetical oracle outlined above, we introduce a new agent to the policy's training, the "Director" ρ_g^d, who uses the fear scores to inform the selection of training goals in a way that is conducive to the policy's training. Because π_θ is non-stationary, the likelihood of a given goal causing a crash is itself non-stationary, and thus the fear function must be learned online, alongside π_θ. This naturally implies a game between the policy and the Director, where π_θ must handle goals issued by ρ_g^d, and ρ_g^d learns about the goal space $\mathcal{G}$ and its relationship to π_θ's crashes online, as π_θ improves.

The idea of selecting sub-goals that force the policy to reckon with unsafe states is summarily similar to DAGGER's intuition [**50**]. As mentioned in Chapter 2.2.2.1, dataset aggregation (DAGGER) utilizes two agents, one being an expert, the other being an agent that learns from the expert's behaviour. By perturbing the expert's behaviour, the learning agent forces the expert to gather data from states the expert would not have otherwise visited. In other words, by experiencing more states that are **not** part of an optimal policy's trajectory, DAGGER agents are still safer, more performant, and more generalizable after training than baseline methods [**50**]. This hinges on the desirable exploration behaviour induced by the perturbations applied to the expert's behaviour. Similarly, we aim to select goals that force the DRL policy to visit crash-states more often. By forcing the policy into crash states, we fill the policy's rollout buffer with more states that are near crash-states (states that are few transitions away from crash states). By showing the policy more examples of the states near crash-states, we posit (and empirically verify, see Chapter 4.5) that the policy will learn to avoid such states, owing to the fact that crash-states terminate the infinite-horizon episodes and are therefore undesirable, as mentioned in Chapter 3.1.1.

For the sake of simplicity, we initially formulate this as a zero-sum game. This follows the formulation of numerous other works on adversarial training and goal selection for DRL

[**46**][**58**][**66**]. However, because fully adversarial adversaries are often not the most conducive to training DRL agents [**14**], we impose a sub-optimal sampling strategy upon the Director. Using a hyperparameter, we control *how adversarial* the Director's selected goals are (see Chapter 3.4). By preventing the Director from acting in a fully adversarial manner, we prevent the Director from acting in a way that might hinder the policy's exploration behaviour. Or rather, we shape the Director's strategy so that the Director acts in a way that helps the policy's training; a *cooperatively adversarial* way.

3.3. Method: the Director's strategy

For π_θ, the task is to select an optimal sequence of actions in a GMDP to survive $\mathcal{T}$ timesteps for a given a goal set during an interaction with the Director while also acting upon it optimally. For ρ_g^d, the game is about selecting a single goal g_{t+1} every $\mathcal{T}$ GMDP timesteps that will cause the policy to crash, conditioned on the information that the policy is in given state-goal pair (s_t, g_t), limited by the sub-optimal goal sampling strategy we impose.

In this game, both agents optimize over a simple binary win condition $W_\pi^d(s_t, g_t, g_{t+1}) = \mathbb{1}\{\pi(\cdot|g_{t+1}) \text{ crashes within } \mathcal{T} \text{ steps}\}$. As mentioned in Chapter 3.1, the policy already implicitly minimizes W_π^d by default as part of its RL objective (Eq 3.1.1). The Director seeks to maximize W_π^d (i.e. maximize Eq 3.1.2).

The description of the Director's task corresponds to a Bernoulli bandit defined by $\left(C = \mathcal{S} \times \mathcal{G}, A = \mathcal{G}, R = W_\pi^d\right)_\pi$, with the context being partial interactions of the shape (s_t,g_t), the actions being the act of selecting the next goal g_{t+1}, and the reward function being the win condition W_π^d. The policy's non-stationary nature also influences the bandit's possible contexts (the policy visits different states depending on its training) and optimal actions and reward structure (the goals that crash π_θ change over training, which directly also changes W_π^d).

Because of the bandit's continuous action space and non-stationary nature, the Beta distribution used in standard Thompson sampling is ill-fitted (see Chapter 2.1.3 for a definition of Thompson sampling). Following our hypotheses detailed in Chapter 3.2, we instead learn a fear function $\mathcal{F}_\Psi : \mathcal{S} \times \mathcal{G} \times \mathcal{G} \to [0,1]$ on-policy. This neural network takes in interactions and outputs fear scores representing crash likelihoods.

For the Director, the fear function $\mathcal{F}_\Psi$ corresponds to a Q function for the reward function W_π^d. In this Q function, the partial interactions (s_t,g_t) correspond to the context and the future goals g_{t+1} correspond to the actions. The result of applying $\mathcal{F}_\Psi$ on an interaction (s_t, g_t, g_{t+1}) is the likelihood of ρ_g^d winning the interaction by selecting g_{t+1} in context (s_t, g_t). Assuming an oracle $\mathcal{F}_\Psi$, an oracle fully adversarial Director might act by finding the optimal goal $g_{t+1}^* \in \mathcal{G}$ for the context (s_t, g_t) such that $\mathcal{F}_\Psi\left(s_t, g_t, g_{t+1}^*\right) \geq \mathcal{F}_\Psi\left(s_t, g_t, g_{t+1}\right) \forall\ g_{t+1} \in \mathcal{G}$.

And so, similarly to the technique used in Thompson sampling, we effectively obtain ρ_g^d by "inverting" $\mathcal{F}_\Psi$. At every instance of the Director's decision process, the Director receives the current state of the agent and of its environment in the form of a partial interaction (s_t, g_t). A set Γ of uniformly-sampled g_{t+1} goals is then collected[2]. The set is then used to create many possible $(s_t, g_t, g_{t+1} \in \Gamma)$ interactions from the partial interaction. Finally, the constructed interactions are scored using the fear network. Given enough sampled goals, we recover through the fear scores an approximation of the optimal adversarial goal (subject to the limitations we impose on ρ_g^d, see Chapter 3.4).

Because our novel goal selection strategy uses the fear network at its core with the goal of producing better interactions to gather data from, we call it **D**read **E**nforced **I**nteractons for **M**ore **O**ptimal **S**ampling (DEIMOS). See Figure 3.1 for a visualization of our method.

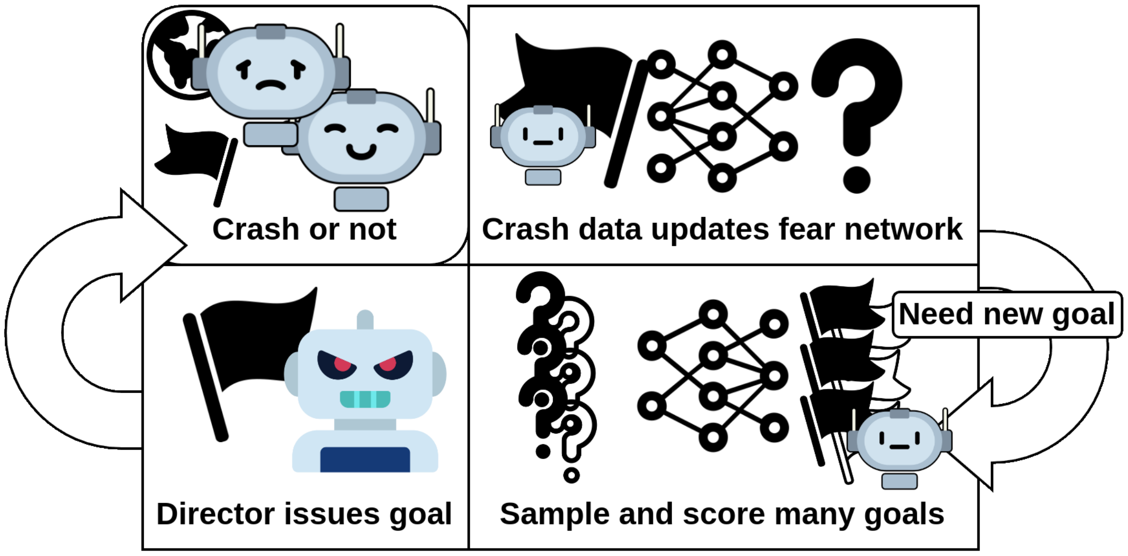


Figure 3.1 – Visualization of our method's main feedback loop. The Director emits goals sampled from the fear network. Upon receiving and acting upon the goals, the agent crashes or survives the goals. We then update the Director's belief about the value of goals by changing the fear network's parameters. Then, the loop begins anew.

3.4. Disempowering the Director

It is natural to expect a fully adversarial and powerful Director to be too good at crashing the policy, leading to poor goal sampling behaviour, or to catastrophic forgetting. But a fully cooperative Director would not fare any better, as it would offer poor goal-space exploration

2. Note that we assume access to a uniform distribution over the goal space. This is tractable in our problem setting, where the goal space is a small continuous vector spaces. In settings with more complex goals (e.g., large vector spaces, images, or text goals), a more principled way of sampling goals would be required.

behaviour by always giving goals that are easy for the policy. We verify this empirically in Chapter 4.5.

To address this, we "disempower" the Director, reining in its desirable adversarial predictions while avoiding their undesirable effects. This amounts to defining how the Director selects the next goal g_{t+1} from the sample Γ of N possible goals[3], informed by the fear scores $F_s = \mathcal{F}_\Psi(((s_t,g_t)) \times \Gamma)$ (with "$\times$" the cartesian product).

We control *how adversarially* the Director acts by separating Γ into two "buckets" according to a fear threshold. Being an adversarial agent, the Director always samples from the bucket containing the highest fear scores. By changing the threshold that defines the two buckets, we control the how cruelly the Director acts.

We define the threshold as a percentile p. Certain states are inherently riskier than others, and so the fear networks outputs a higher baseline fear value for all possible goals. For example, the same sampled goals for a quadrupedal robot would receive higher fear values if the robot is presently in a stairway than if the robot is on flat terrain. By using a percentile value, the threshold naturally adapts to the baseline fear for a given state.

The Director samples goals above the p-th percentile of F_s. With $p = 10$, the Director acts mostly cooperatively; it discards all goals in the 10th lowest percentile and then uniformly samples g_{t+1} from the rest of Γ. With $p = 90$, the Director acts mostly adversarially; it discards all goals outside the 10th highest percentile and then uniformly samples g_{t+1} from the remaining goals.

3. N is an hyperparameter that controls how many suggested future goals are sampled and then scored by the fear network. In all of our experiments, $N = 100$.

Chapter 4

Experiments

In this chapter, we present an empirical analysis of DEIMOS. We introduce the experimental training and evaluation environments in Chapter 4.1. We detail a series of ablations upon DEIMOS in Chapter 4.2. We explain our selection of baseline methods in Chapter 4.3. Finally, we present the results of our experiments in Chapter 4.5.

4.1. Environments

4.1.1. Toy problem: *Particle Gym*

In this section, we define a toy problem of our own design tailored to evaluate DEIMOS and the baselines in study.

In *Particle Gym*, a unit-mass particle spawns at the origin (point (0,0,0)) in a three dimensional cube. This cube is 2000 length units a side, and the particle crashes if it exits it. From the point of view of the particle, should any component of the (x,y,z) values describing its current position go above (below) 1000 (-1000), it crashes.

The physics of the environment are simple:

(1) At every timestep, the environment receives the policy's actions. The actions are three-dimensional vectors that are interpreted as forces upon the particle.

(2) The forces are halved and are then directly added to the particle's velocity.

(3) The absolute value of the components of the resulting (x,y,z) velocities can never exceed some maximal velocity. This maximal velocity is set so that many timesteps are required before the particle comes close to the edges of the cube. These edges correspond to the only states where crashes are possible. DRL has trouble reasoning about long-horizon problems, so needing to experience many timesteps before it is possible to crash makes the environment harder.

(4) The particle's position is then updated by adding the resulting velocities to it.

(5) The state is sent to the policy so that it can act upon it. The state given to the policy is a vector containing the particle's (x,y,z) velocities and positions, the last action output by the policy, and the current goal.

Three versions of the environment exist:

(1) **Absolute**: the goals are absolute positions in space (which **can** fall outside the cube). The reward is based on the distance from the particle to the desired position.

(2) **Directional**: the goals describe requested velocities. The reward is based on the difference between the particle's (x,y,z) velocities and those described in the current goal.

(3) **Relative**: the goals are relative positions in space. When selecting a goal, an absolute position in space is first selected. Every timestep thereafter, the current distance from the policy to the absolute position is what is sent to the policy as g_t. The goal's components thus become closer and closer to (0,0,0) if the particle gets closer to its target, or increase if the particle gets further away. The reward is based on the norm of the goal, i.e. the current distance from the particle to the target position.

In all three versions, the policy must learn to achieve the goals to the best of its ability while keeping the particle inside the cube. See Figure 4.1 for a visualization. Because the reward signals are all strictly positive (never ≤ 0), and because we consider infinite-horizon episodes, staying alive is always preferable to terminating the episode early.

Particle Gym is a massively parallel environment where 2000 environments are simulated at the same time. Every simulation timestep, 2000 particles receive forces, 2000 observations are sent to the policy, and the policy produces 2000 actions in response. See Figure 4.2 for a visualization of what a *massively parallel environment* entails.

All results for *Particle Gym* come from training 10 DRL seeds using a PPO policy.

4.1.2. *Legged Gym*

We implement DEIMOS in *Legged Gym* [**51**], a state-of-the-art simulator for legged robots based on Isaac Gym [**36**]. The simulator includes noisy terrain surfaces and steep stairs and slopes, along with banal flat terrain. Similar to *Particle Gym*, this is a massively parallel simulator where 4096 robots are simulated at the same time (without robot-to-robot collisions). See Figure 4.2 for a visualization of this parallelization. As the policy performs better (worse) according to the reward function, the robots are placed in areas of increasing (decreasing) difficulty. This increasing difficulty is reflected by the increased steepness of the slopes and stairs and the level of noise applied to the uniformly-noised terrain. The state given to the robot is comprised of the current robotic state, a height map of the surrounding terrain, and the current goal. During training, the friction coefficient of the terrain is randomized, and the robots are subjected to random force pushes. This

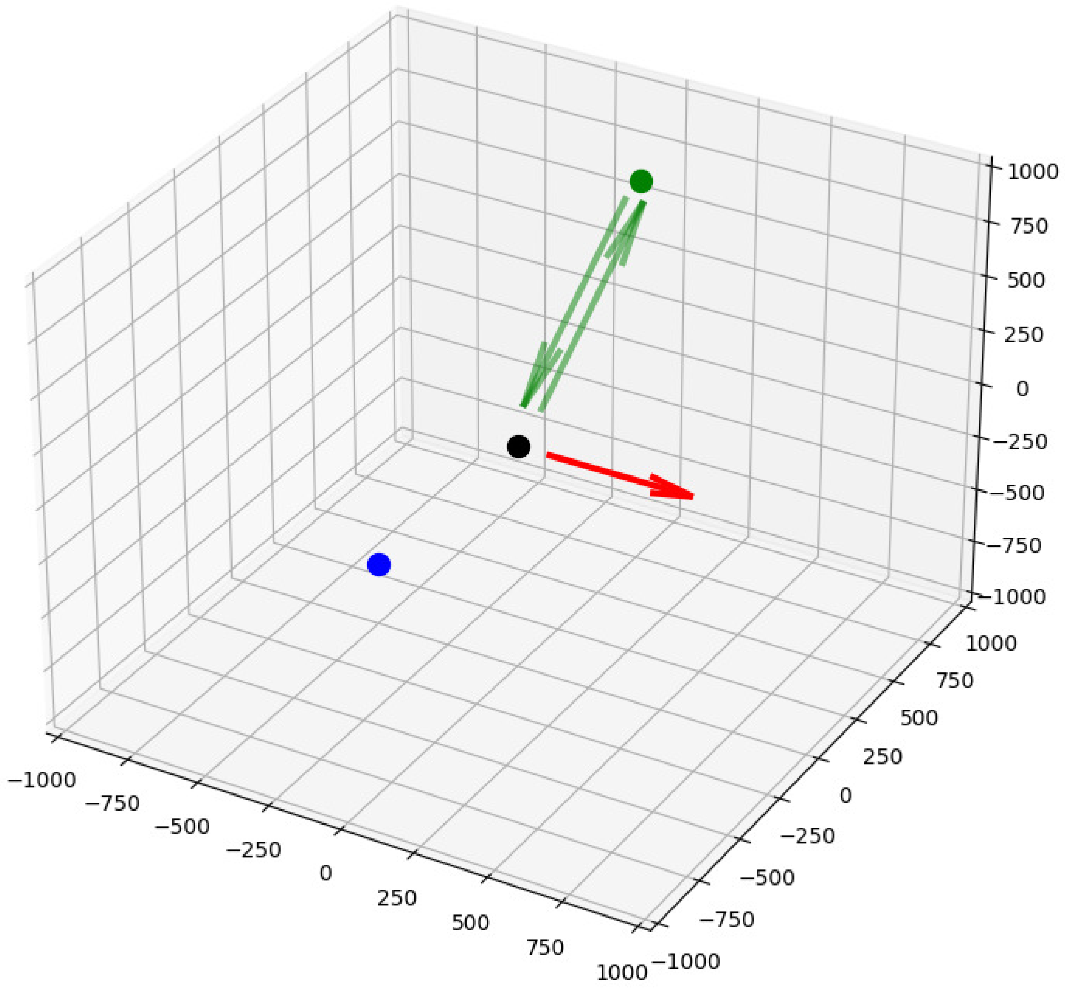


Figure 4.1 – In *Particle Gym*, the particle starts every episode at (0,0,0) (in black). Three variations exist. In the **directional** case, goals are desired velocities (visualization in red). In the **relative** case, goals are the distance from the particle's current position to a set point in space (visualization in green). In the **absolute** case, goals are an absolute position in space that the particle must reach (visualization in blue). In all variations, should the particle exit the $[-1000,1000]^3$ cube, the episode terminates.

is a standard practice for simulation environments for legged robots that encourages better generalization behaviour.

We study two different robotic morphologies (shown in Figure 4.3).

(1) **Quad**: in this version, a quadrupedal robot modelled after the *Go1* robot must follow (x,y,angular) velocity goals. The goals are in $[-1,1]^3$.

(2) **Quad+arm**: in this version, a quadrupedal robot with an arm on board (modelled after a *Go1* robot with a mounted *ReactorX* arm) must follow the same velocity goals

as Quad, with the addition of relative positional (x,y,z) goals that the arm's end effector must match. The goals are in $[-1,1]^6$. This is a very unstable morphology [**17**]. The shifting weight of the arm makes traversing steep stairs at fast speeds while also matching desired positional arm goals incredibly difficult.

In both variants, the reward function is primarily based on the goal-matching performance but also includes terms to make learning walking gaits easier [**51**]; we only report a weighted aggregate of the goal-matching rewards. No penalty for crashing is present. Crashes are registered when any part of the robot's base touches any part of the terrain.

We train using $[\mathcal{T}_{10} = 10 \text{ simulated seconds} = 500 \text{ timesteps}]$, but to showcase agile behaviour and generalization to faster goal changes, we also evaluate using $[\mathcal{T}_2 = 2 \text{ simulated seconds} = 100 \text{ timesteps}]$. The $\mathcal{T}_2$ timing in particular requires rapidly and responsively handling changes in momentum. For the Quad+arm case, the quickly shifting center of weight of the morphology caused by the movements of the arm complicates things further.

All results for *Legged Gym* come from training 30 DRL seeds using a PPO policy. While we do not conduct experiments on the real hardware because it would be infeasible to evaluate enough seeds to obtain accurate results, we do evaluate the seeds in a different evaluation environment than the one they are trained in. While the general terrain itself is the same, we remove all friction coefficient randomization and the random force pushes that the robots are subjected to during training.

4.1.3. Evaluation and goal selection

In both environments, we evaluate using two different goal distributions ρ_g.

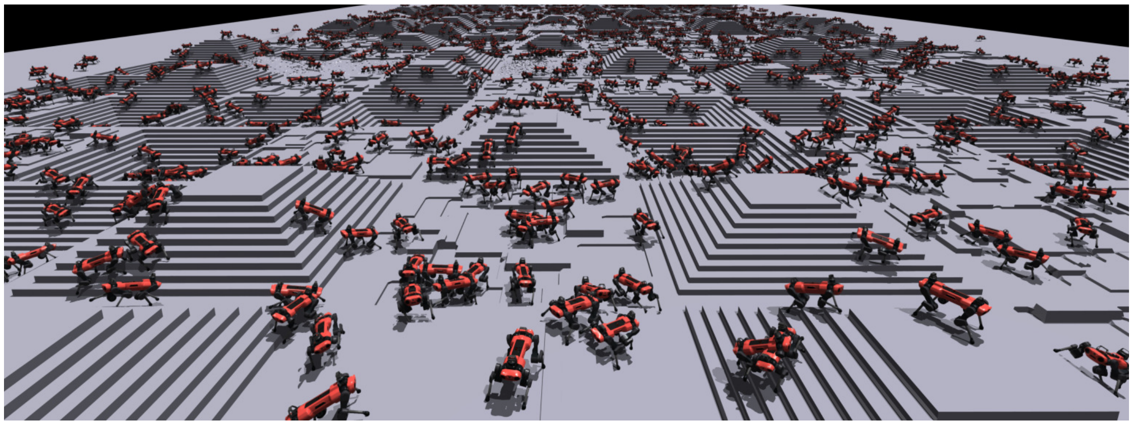


Figure 4.2 – Visualization of the *Legged Gym* environment. Many robots are simulated in parallel, all controlled by a single policy. The policy must learn to issues commands such that the robots walk on flat terrain, noised terrain, sloped terrain, and stairs without crashing. *Picture credit: Legged Gym* [**51**].

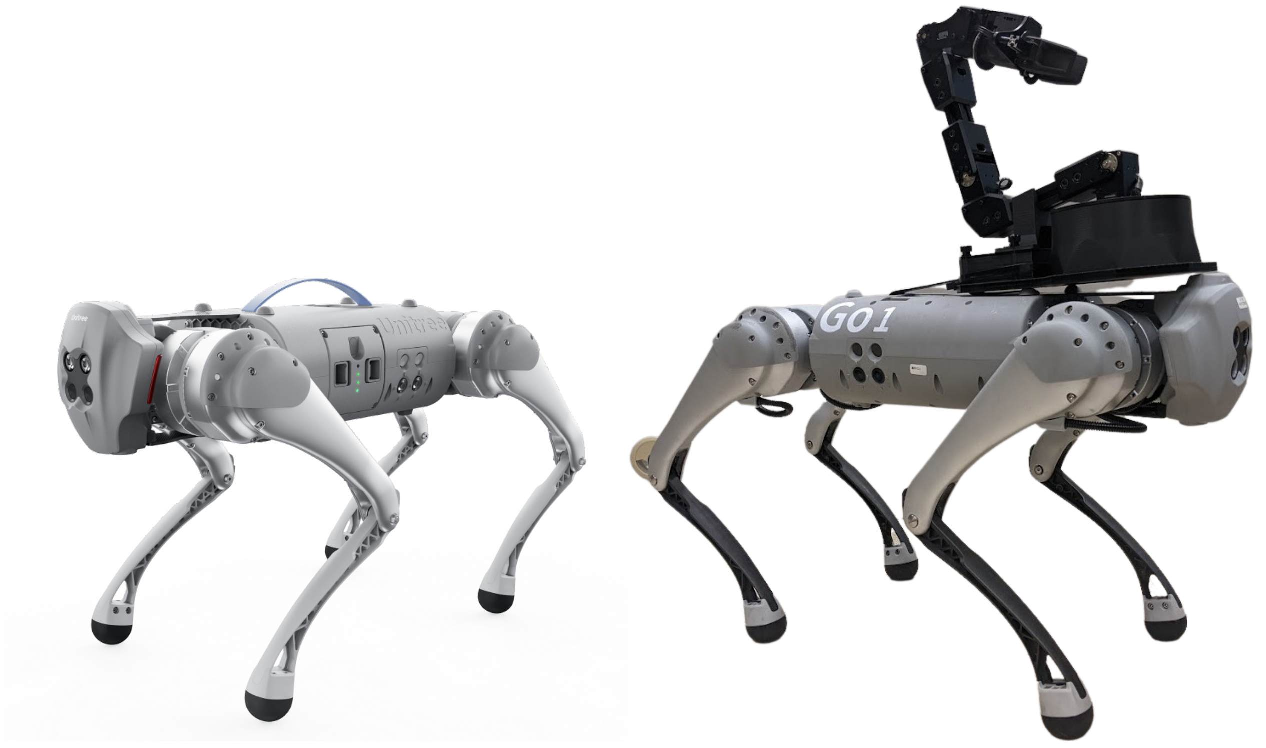


Figure 4.3 – Pictures of Quad and Quad+arm, modelled after the *Unitree Go1* robot. Quad+arm additionally has a *Interbotix ReactorX* arm onboard.

— **Naive**: the “Naive” ρ_g is a uniform distribution over $\mathcal{G}$. This covers all goals in all states indiscriminately.
— **Adversary**: the “Adversary” ρ_g is a fully adversarial Director that leverages a fear network trained for 1500 timesteps after the training of π_θ has concluded. The fear network used by the Adversary is trained independently from the fear network used by the Director to suggest training goals when using DEIMOS. We use the Adversary to evaluate the policy’s robustness to adversarial attacks upon its goals, where an attacker would selects goals that are most likely to crash the robot.

All environment randomization (friction, random pushes, random terrain) is disabled during evaluation.

4.2. Implementing DEIMOS

The fear network used by DEIMOS’s Director is trained using a binary cross-entropy loss on interactions $\left[(s_t, g_t, g_{t+1}) : W^d_\pi(s_t, g_t, g_{t+1})\right]$ sampled from π_θ’s on-policy rollout buffer. The binary cross-entropy loss is weighted so that the crash and survival observations in the batches sent to $\mathcal{F}_\Psi$ are of equal contribution to the loss term. As output, $\mathcal{F}_\Psi$ produces a value from 0 to 1.

It is important to note that the fear network is not calibrated. Its only predictive power is a ranking of different g_{t+1} goals for a given partial interaction (s_t,g_t). In other words, should a state s_t be particularly perilous, all possible fear scores will be relatively high. For example, in *Particle Gym*, essentially all possible g_{t+1} goals for point $A = (-999,0,0)$ would receive higher fear scores than the goals for point $B = (0,0,0)$ by virtue of point A being closer to edge of the $[-1000,1000]^3$ cube. This does not hinder DEIMOS, as it only requires a relative ranking of future goals for a given (s_t, g_t) to inform its selection process.

It is also important to note that the first goal issued during an episode is selected randomly. Indeed, immediately after resetting the environment, we seek to select g_0. However, to use the Director, we require a previous goal. But no g_{-1} or s_{-1} is available to form the partial interaction (s_{-1}, g_{-1}) that the Director requires to select g_0. In all environments, g_0 is instead sampled from a uniform distribution over $\mathcal{G}$.

4.2.1. Ablations

To accurately characterize how the cruelty of the Director impacts the policy's training, we cover the [0,1] range of possible p values. We present results for p = [10,25,50,75,90], respectively identified as "p10" through "p90".

We also present results for a fully cooperative Director that selects goals using the decision rule $\left[\text{argmin}_{g_{t+1}}\right]$ over the fear scores and a fully adversarial Director that selects using the decision rule $\left[\text{argmax}_{g_{t+1}}\right]$. They are respectively identified by "Coop" and by "Antagonist".

4.3. Baseline methods

— **Hand-made curriculum**: For both variants of *Legged Gym*, we compare against a hand-made curriculum. We refer to this hand-made curriculum as "CL". In CL, we slowly increase the possible range of goals from which we uniformly sample. Empirically, this leads to a training distribution that is conducive to learning. This curriculum is based on the curriculum used in the original *Legged Gym* work [**51**].

— **Uniform Sampling**: In contrast to the CL baseline, the "Uniform" baseline uses no goal selection strategy whatsover. It simply samples goals uniformly from the full goal space $\mathcal{G}$.

— **Random Network Distillation**: Traditionally, RND is used to generate rewards for the environment's transitions (see Chapter 2.2.1.1). In this work, we instead apply RND to the space of goals. We sample many different g_{t+1} goals and have both the frozen and trained networks score them for a single (s_t, g_t). We then select the goals with the highest surprise, effectively using the surprise of the policy to explore the goal space.

— **Value Disagreement Sampling**: We apply VDS to our environments [**67**]. See Chapter 2.2.3.1 for an outline of VDS. VDS already tackles a very similar problem to the one that we tackle. Where we desire to select goals in a way that trained policies are safer while still optimizing for reward performance, VDS only optimizes for reward performance. Of course, since we assume that a truly optimal policy would never crash, VDS also implicitly aims at producing safer policies.
All VDS results in this work utilize an ensemble of 10 critics.

— **VDS and DEIMOS**: For *Legged Gym*, we find empirically that VDS seems to be the best baseline in study. VDS offers great initial goal-space exploration by using the policy's own uncertainty about the difficulty of the goals to select the future goals. However, we find that about halfway through training, VDS-trained policies start experiencing very few crashes. See the figures in Appendix A for the related training curves.
As per our initial intuition, experiencing more crashes during training should help the VDS-trained policies learn not to crash. For *Legged Gym* only, we showcase a simple training schedule utilizing both VDS and DEIMOS. Goals are fully sampled from VDS until halfway through training (1500 training steps), and then they are fully sampled using DEIMOS until the end of training. We use what we deem to be among the best performing version of DEIMOS for *Legged Gym*: $p = 75$. In our results, this method is identified by "VDS+p75".

4.4. Terminology

Here, we present a short recapitulation of the terms we use to identify key parts of our experimental results.

(1) A **timestep** refers to a transition (s_t, a_t, r_t, s_{t+1}) in the environment.

(2) The **Director** is the agent used in **DEIMOS** to select goals during training. It selects goals above some percentile threshold p. In our results, the different p values of DEIMOS are identified by "p" and the p value. For example, DEIMOS with $p = 50$ is identified by "p50".

(3) The **Naive** is a uniform distribution over the goal space $\mathcal{G}$ from which goals are sampled during evaluation.

(4) The **Uniform** baseline is a uniform distribution over the goal space $\mathcal{G}$ from which goals are sampled during training. Its goal selection strategy is identical to the **Naive**'s, but for readability reasons we refer to them by different names.

(5) The **Adversary** is a fully adversarial Director from which goals are sampled during evaluation.

(6) The **Antagonist** goal distribution is an ablation of DEIMOS where goals are selected using the *argmax* of the fear scores. Its goal selection strategy is identical to the **Adversary**'s, but for readability reasons we refer to them by different names.

(7) The **Coop** goal distribution is an ablation of DEIMOS where goals are selected using the *argmin* of the fear scores.

4.5. Results

Accurately evaluating robotic RL is difficult. In our problem setting, reducing crashes is obviously desirable, but so is increasing rewards. Often, doing one comes at the cost of the other. Lowering the population standard errors for both metrics is also important.

Because of the very large amount of data to present, we present a succinct visualization of our evaluation results in the form of box-and-whisker plots. This captures both the spread of the populations of trained agents and their expected performance and safety. While these plots often present overlapping boxes for some environments, we deem our results significant; there exist consistent trends across environments.

We caution the reader against only looking at the performance (reward) results. Not crashing often means recognizing which states are unsafe, and not listening to commands that drive the robot towards such states. As such, the cost of avoiding a crash is often paid in rewards, at least myopically. Less crashes with slightly less timestep-wise rewards is a desirable outcome. However, in some cases, the policies achieve a very low amount of crashes, but also a very low amount of rewards. This implies that the policies are useless, and learned not to crash by completely ignoring the goals. The reader should focus on the safety (crash) results; the performance results should only be used to ensure that the policy actually learned the task.

As for the training curves for both environments, because we are evaluating different goal-selecting methods, it follows that the reward and crash behaviours during training are also intrinsically tied to the training methods in study. The reward and crash data during training are idiosyncratic to their given goal distribution and cannot be cross-compared with results obtained from another distribution. These curves are especially unclear for *Legged Gym*, and so we omit them from the main body of this work. They are presented in Appendix A. Thanks to the simplicity of *Particle Gym*, its training curves are clearer and are shown in Chapter 4.5.1.

We present results for 4M timesteps for *Particle Gym* and $\approx$ 8.2M timesteps for *Legged Gym*. For the evaluation results, we present the average reward and the total number of crashes.

4.5.1. Training results for *Particle Gym*

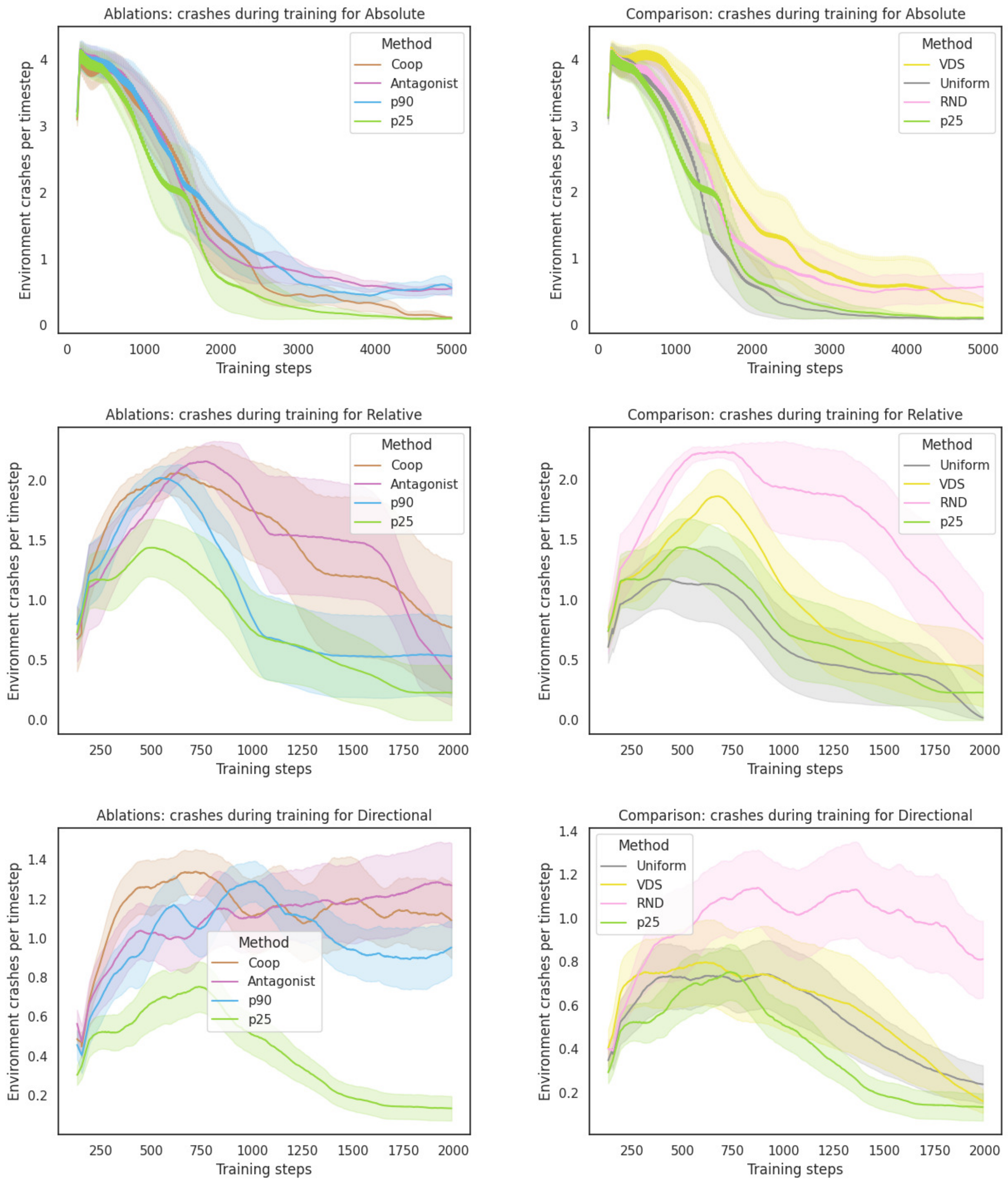


Figure 4.4 – Crashes during training for10 PPO random seeds for all three variants of *Particle Gym*. Most p values were omitted for readability. On the left are the results obtained from the different ablations applied to DEIMOS. On the right, we compare one representative p value for DEIMOS to the results obtained from the baseline methods.

In Figure 4.4 we find the results for training 10 PPO random seeds in the three variants of *Particle Gym*. The Absolute variant was trained for 5000 steps, while the Directional and Relative variants were trained for 2000. On the left, we find the results for the different versions of DEIMOS. On the right, we find the results comparing DEIMOS to the baseline methods. The results shown in Figure 4.4 were obtained from applying a rolling window average to the very noisy data obtained from training;

The reader might notice that the results shown in Figure 4.4 show more than one crash per timestep for some timesteps. Recall that *Particle Gym* is a massively parallel environment where 2000 environments are simulated at once. Each training step shown here thus corresponds to 2000 parallel timesteps. This is why more than one crash can happen on a single timestep. In truth, early in training, some timesteps had 2000 crashes at once.

Notice that in some variants, some methods seem not to have reached convergence yet. This is normal. When subjected to a particularly sub-optimal goal distribution, the policies take much more time to converge. We desire DRL that converges in a reasonable amount of time, thus still we only train for a limited number of timesteps.

As shown in Figure 4.4, we find that RND does not lend itself well to generating training goal distributions for *Particle Gym*. The two other baselines, Uniform and VDS, seem to perform fairly well in all three cases, but these training curves are not necessarily representative of their true safety and performance after training. The goal distribution during training greatly influences the behaviour of the policies: the policies are tasked with achieving the goals issued from these distributions. So the curves presented in Figure 4.4 cannot truly be compared to one another due to their idiosyncratic goal distributions.

Another trend is that the Antagonist and Coop Directors are not good at reducing crashes. One interesting trend is that $p = 90$ seems to achieve remarkably similar behavior to the Coop and the Antagonist versions of DEIMOS, most probably because of a lack of goal diversity: the Director only samples in the top 10th percentile of the fear scores.

4.5.2. Evaluation results for Absolute

Figure 4.5 presents the evaluation results for the Absolute environment. On the left, we find results for the policies when they respond to goals selected by the Naive goal distribution. On the right, we find results for the policies when they respond to goals selected by the Adversary.

Against the Naive ρ_g, the value of p for DEIMOS seems to have little influence over the safety behaviour after training. Most versions of DEIMOS produce similar policies. The baselines hold similarly little influence. RND-trained policies experience the most crashes (and variance over crashes) out of all the baselines. VDS and Uniform produce similar policies in terms of crashes, but Uniform policies receive markedly higher rewards. This implies that

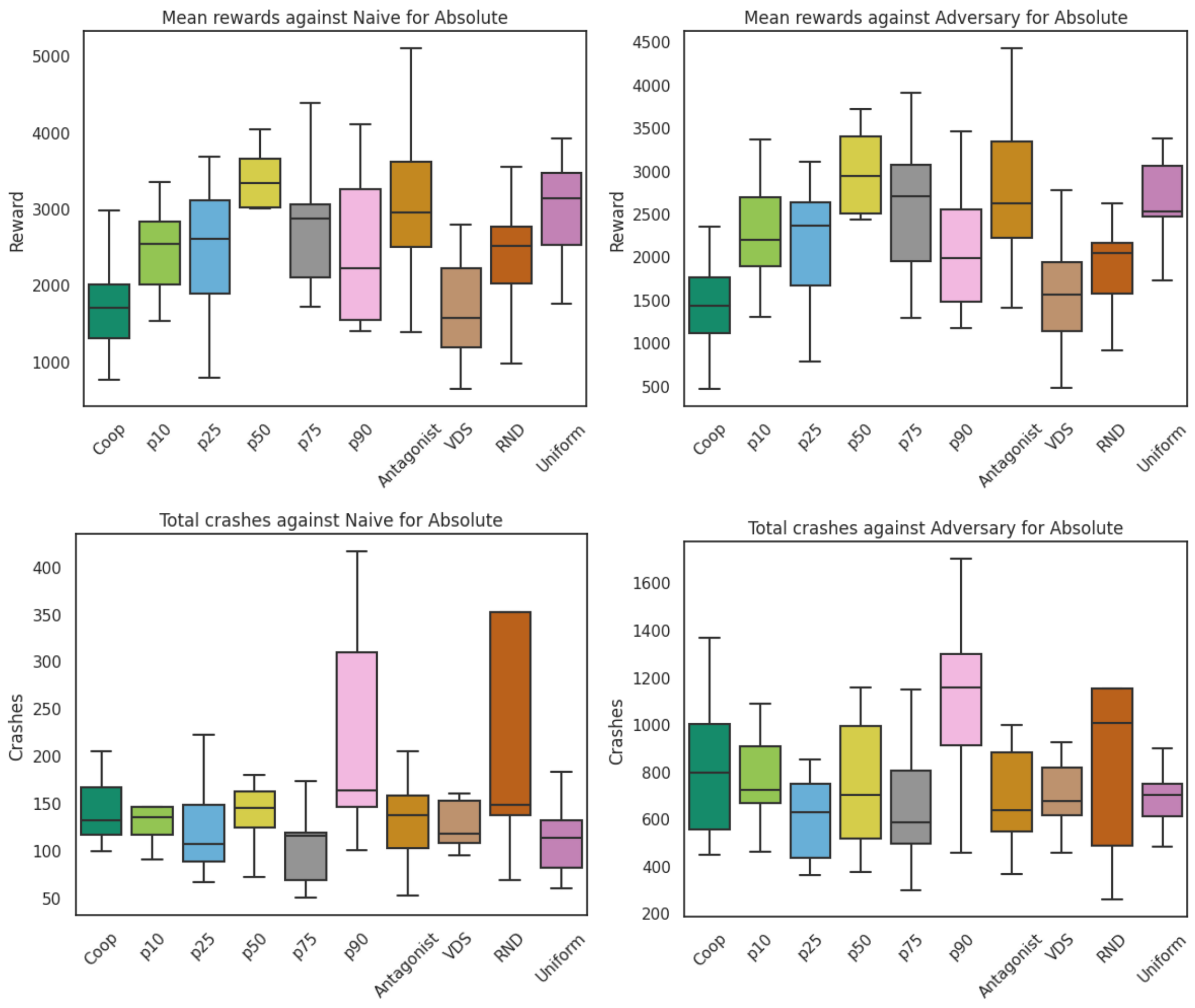


Figure 4.5 – Evaluation results for 10 PPO seeds in the *Particle Gym* Absolute environment.

VDS-trained policies did not successfully learn the task. With $p = 25$ and $p = 75$, DEIMOS matches VDS and Uniform. With $p = 50$, we obtain high rewards but slightly higher median crashes. The single highest possible rewards are obtained by the Antagonist, as evidenced by the top whisker for that method.

Against the Adversary ρ_g, the goal distribution during training holds more importance. Every method experiences significantly more crashes. Again, Uniform's policies obtains the same amount of crashes as VDS's but markedly better rewards. DEIMOS with $p = 25$ matches Uniform and VDS. Again, the single highest possible rewards are obtained by the Antagonist, as evidenced by the top whisker for that method.

Overall, we identify Uniform and DEIMOS with $p = 25$ as the best methods for Absolute.

4.5.3. Evaluation results for Relative

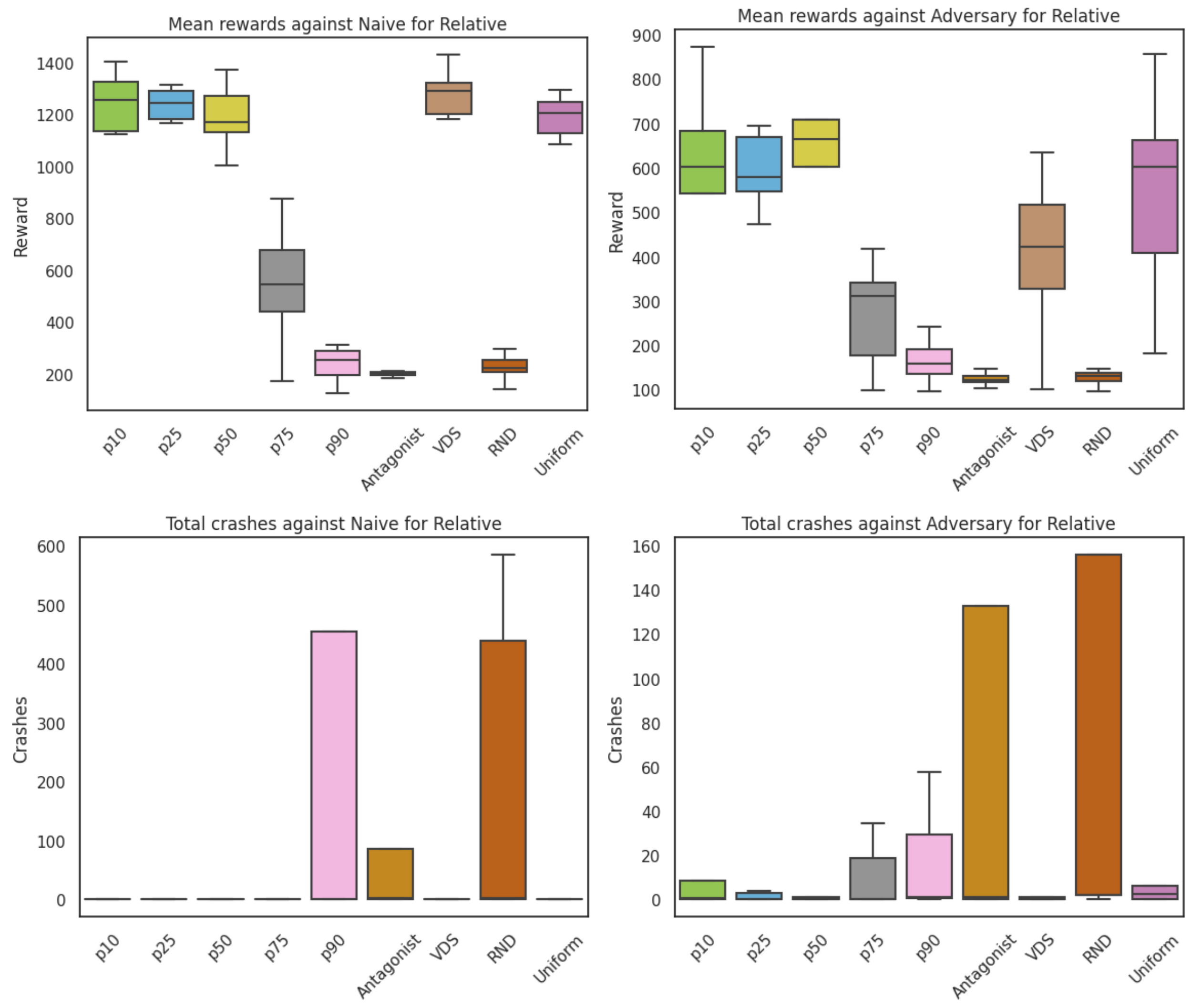


Figure 4.6 – Evaluation results for 10 PPO seeds in the *Particle Gym* Relative environment. Coop omitted because of scaling issues. Very small boxes for crashes are not mistakes or visual artefacts; some methods achieved ≈ 0 crashes.

Figure 4.5 presents the evaluation results for the Relative environment. On the left, we find results for the policies when they respond to goals selected by the Naive goal distribution. On the right, we find results for the policies when they respond to goals selected by the Adversary.

Against the Naive ρ_g, the value of p for DEIMOS seems to have little influence over the safety behaviour after training. Most versions of DEIMOS produce similar policies, except for $p = 90$.

The Antagonist completely hindered its policies' training. Its policies obtained very poor rewards, which is related to why they appear to act safely: the policies learned not to move the particle for fear of crashing. The Coop Director was omitted because of scaling issues, but its results are similar to the Antagonist's.

RND-trained policies experienced the most crashes (and variance over crashes) out of all the baselines, yet very low rewards. This implies that RND policies learn to move in random directions, ignoring their goals. VDS and Uniform produce similar policies in terms of both crashes and rewards. With $p = 10$, $p = 25$, and $p = 50$, DEIMOS matches VDS and Uniform. The single highest rewards are obtained by the Antagonist, at the cost of very large variance.

Against the Adversary ρ_g, the goal distribution during training holds more importance. Interestingly, we observe an increase in crashes for the policies that experienced few crashes against the Naive ρ_g, yet a decrease for the policies that were less safe. This might imply that those policies learned to be resistant against adversarial goals at the cost of poorer behaviour against random goals.

Here, Uniform's policies obtain more rewards than VDS's, but also more crashes. As such, we deem VDS to be the best baseline in study, obtaining the lowest amount of crashes for the baseline methods and adequate rewards. DEIMOS with $p = 50$ obtains great rewards with low variance, and very few crashes.

Overall, we identify DEIMOS with $p = 50$ as the best method for Relative. VDS, Uniform, and DEIMOS with $p = 25$ come close.

4.5.4. Evaluation results for Directional

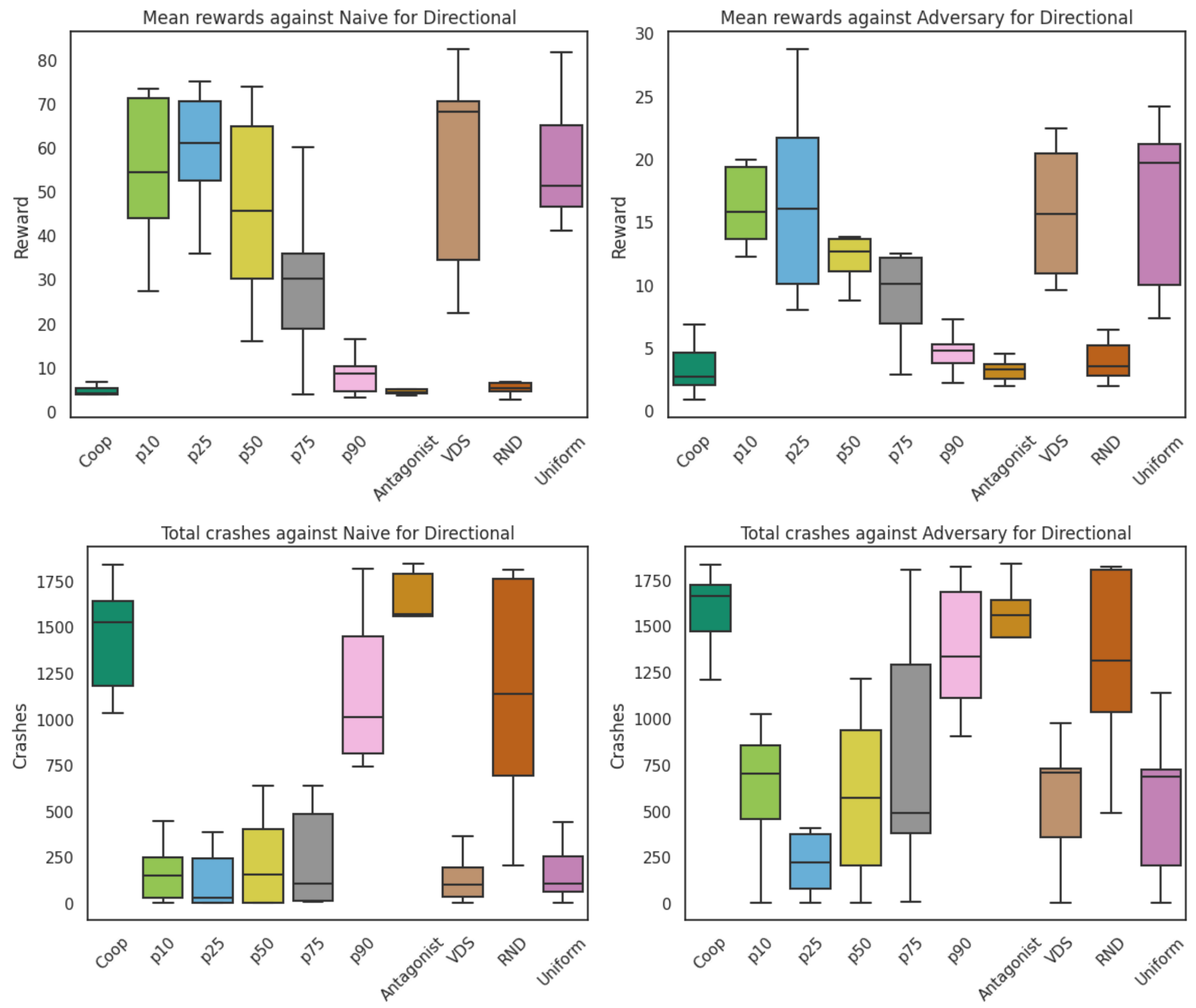


Figure 4.7 – Evaluation results for 10 PPO seeds in the *Particle Gym* Directional environment.

Figure 4.7 presents the evaluation results for the Directional environment. On the left, we find results for the policies when they respond to goals selected by the Naive goal distribution. On the right, we find results for the policies when they respond to goals selected by the Adversary.

Against the Naive ρ_g, the value of p for DEIMOS holds more importance than for the Absolute and Relative cases.

Both the Coop agent and the Antagonist completely hindered their policies' training. They obtained both the poorest rewards and the most crashes. Again, this implies that the policies learned to not move at all.

RND-trained policies also experienced the worst rewards and the most crashes (and variance over crashes). VDS and Uniform again produce similar policies in terms of both crashes and rewards. With $p = 10$, $p = 25$, and $p = 50$, DEIMOS matches VDS and Uniform.

Against the Adversary ρ_g, for the baseline methods, we observe the same trends as against the Naive ρ_g. We find that in particular, DEIMOS with $p = 25$ obtains the highest rewards (equal to Uniform and VDS) and the least crashes. The variance for both metrics is also great.

Overall, we identify DEIMOS with $p = 25$ as the best method for Directional.

4.5.5. Evaluation results for Quad

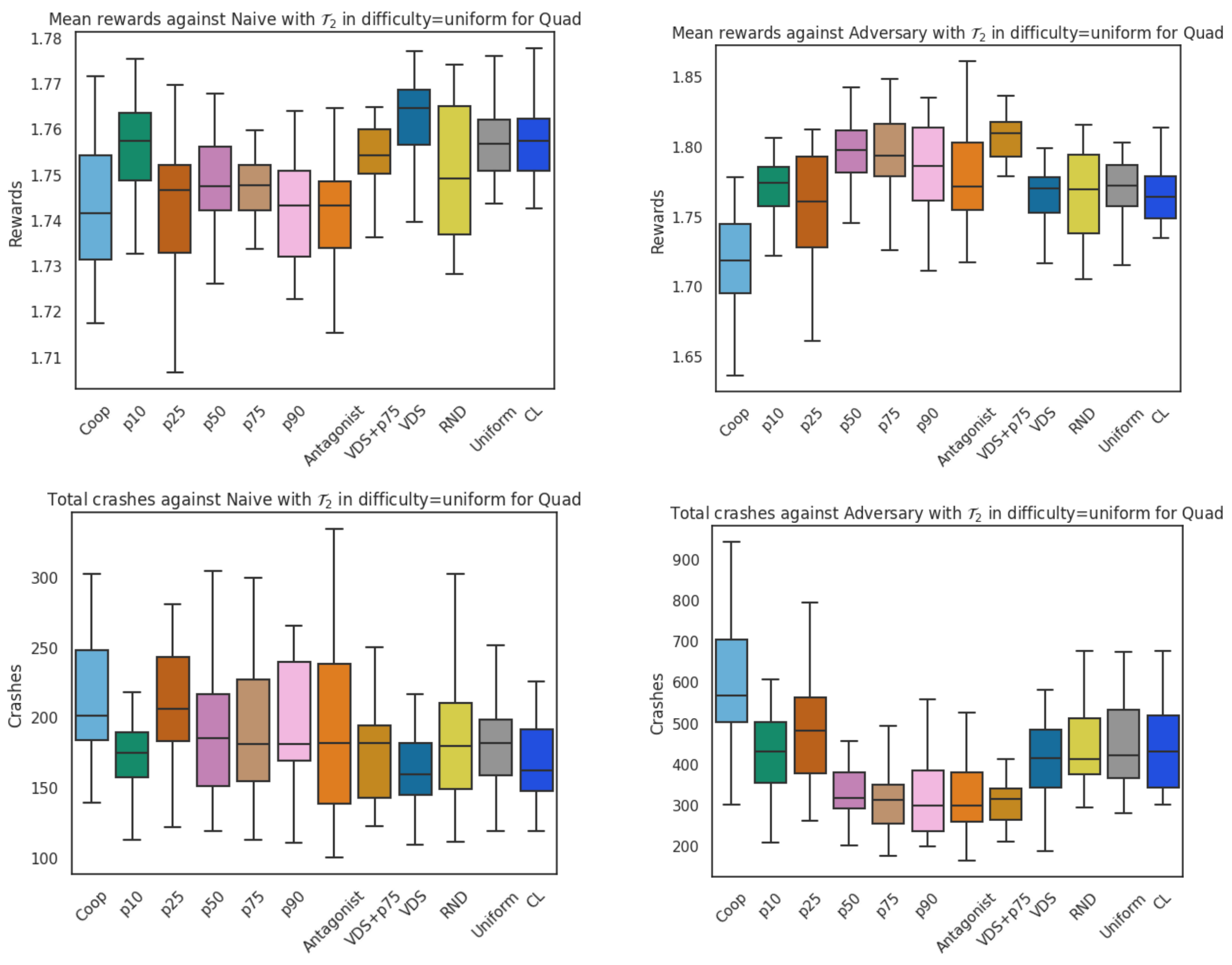


Figure 4.8 – Results for 30 PPO seeds in the Quad environment with the $\mathcal{T}_2$ goal resampling timing.

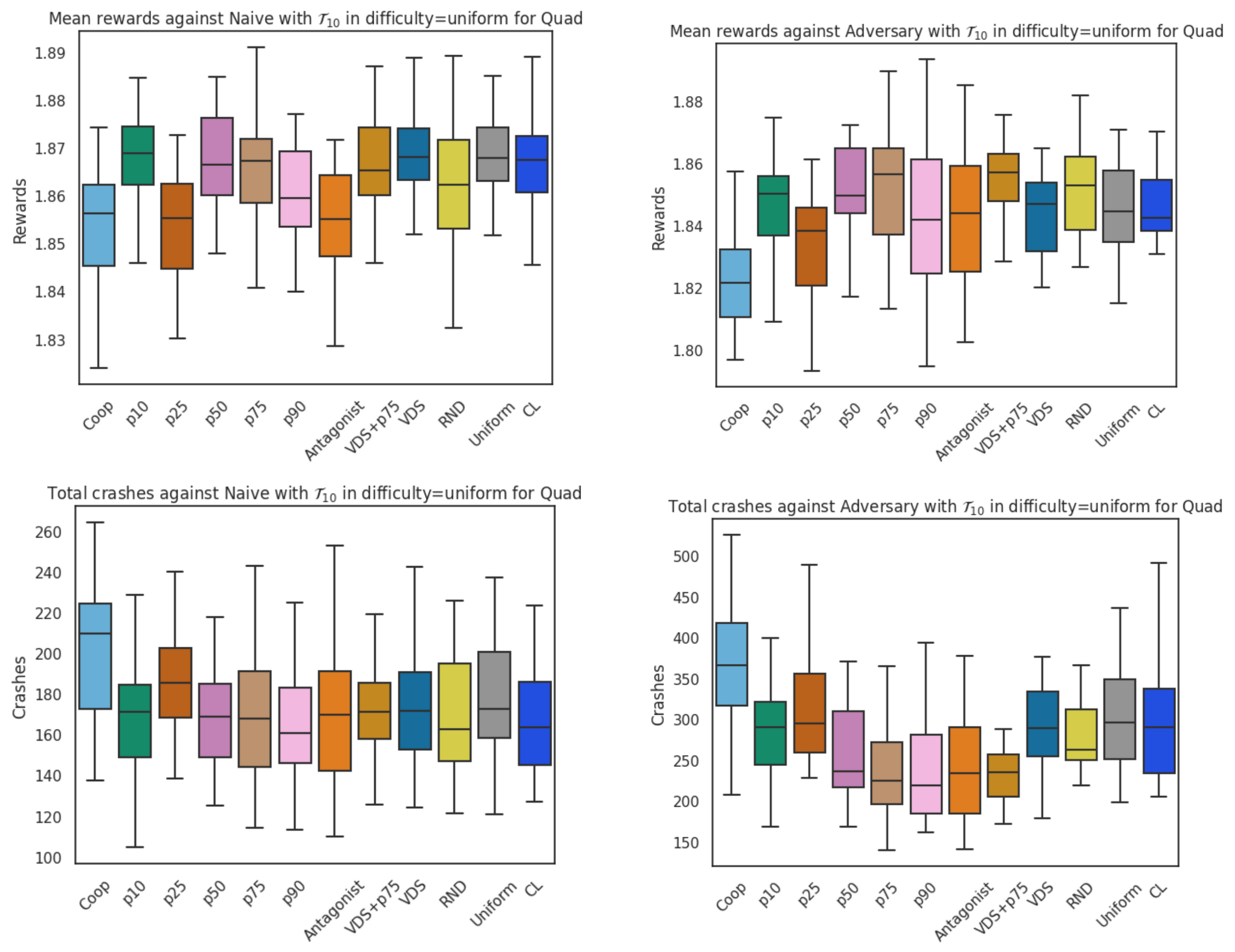


Figure 4.9 – Results for 30 PPO seeds in the Quad environment with the $\mathcal{T}_{10}$ goal resampling timing.

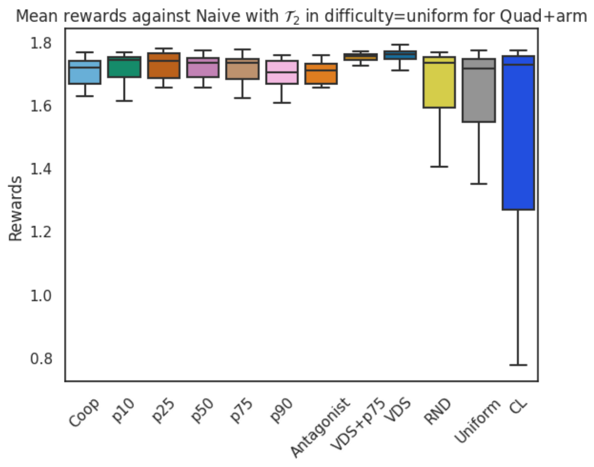
Figures 4.8 and 4.9 present the evaluation results for the Quad environment with goal resampling timing $\mathcal{T}_2$ and $\mathcal{T}_{10}$, respectively. As a reminder, this means that the goals respectively change every 2 seconds (100 timesteps) and 10 seconds (500 timesteps). For both figures, on the left, we find results for the policies when they respond to goals selected by the Naive goal distribution. On the right, we find results for the policies when they respond to goals selected by the Adversary.

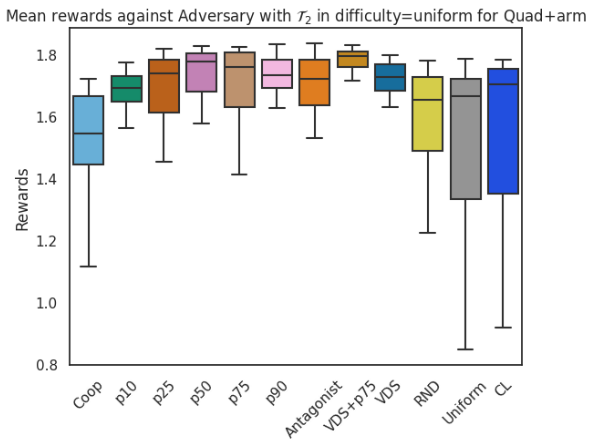
Against the Naive ρ_g, for both timings, the value of p for DEIMOS holds little importance. In fact, almost all the methods in study obtain similar amounts of crashes. As shown in Figure 4.8, we find that DEIMOS experiences high variance for almost all p values compared to the baseline methods, yet similar median policies. Overall, VDS slightly outperforms the baselines, obtaining higher rewards for similar safety behaviour. We find that VDS+p75 obtains policies that are similar to VDS alone. We find that RND does much better here than in the *Particle Gym* environments. Both the Coop Director and the Antagonist perform poorly in terms of rewards (and slightly worse than other DEIMOS versions in terms of crashes).

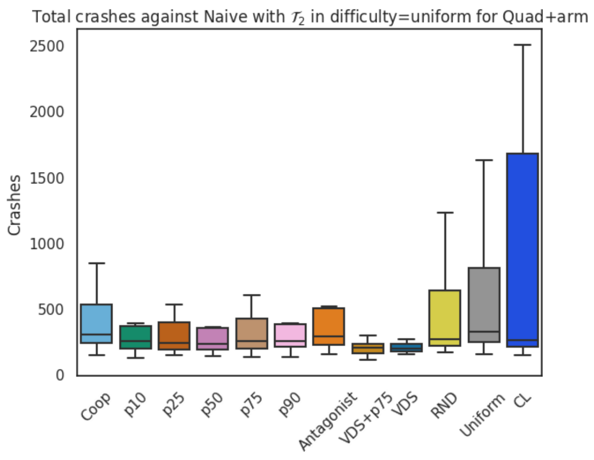
Against the Adversary ρ_g, for the baseline methods, the goal distribution during training holds more importance on evaluation safety. Coop produces the worst policies both in terms of rewards and of crashes. All baseline methods produce similar policies. For DEIMOS, the best p values ($p = 75$ and $p = 90$) produce policies with fairly safe and performant behaviour. As shown in Figure 4.8, against the $\mathcal{T}_2$ timing, the variance is adequately low, but against the $\mathcal{T}_{10}$ timing shown in Figure 4.9, variance is quite high, which is not desirable. We observe that VDS+p75 obtains the best policies for both timings, with quite low variance, the lowest amount of crashes, and very high rewards.

Overall, we identify VDS+p75 as the best method for Quad for both timings. We believe that the much improved behaviour against the Adversary compared to the other methods is worth the very slightly worse behaviour against the Naive ρ_g compared to the second-best method, VDS.

4.5.6. Evaluation results for Quad+arm







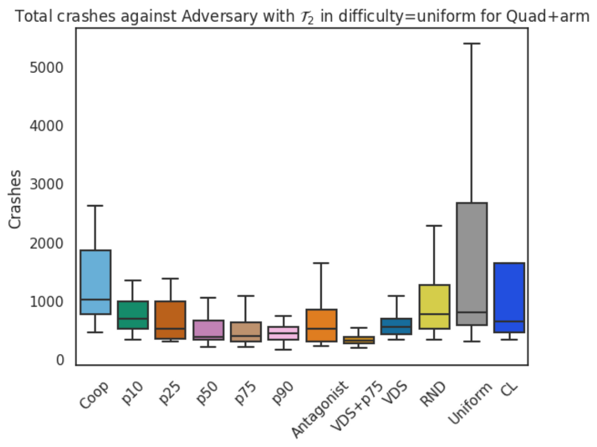


Figure 4.10 – Results for T_2 for Quad+arm. All DEIMOS versions produce better policy populations that the baselines, except for VDS. The VDS+p75 schedule does best overall, with almost identical Naive results as VDS and better Adversary results.

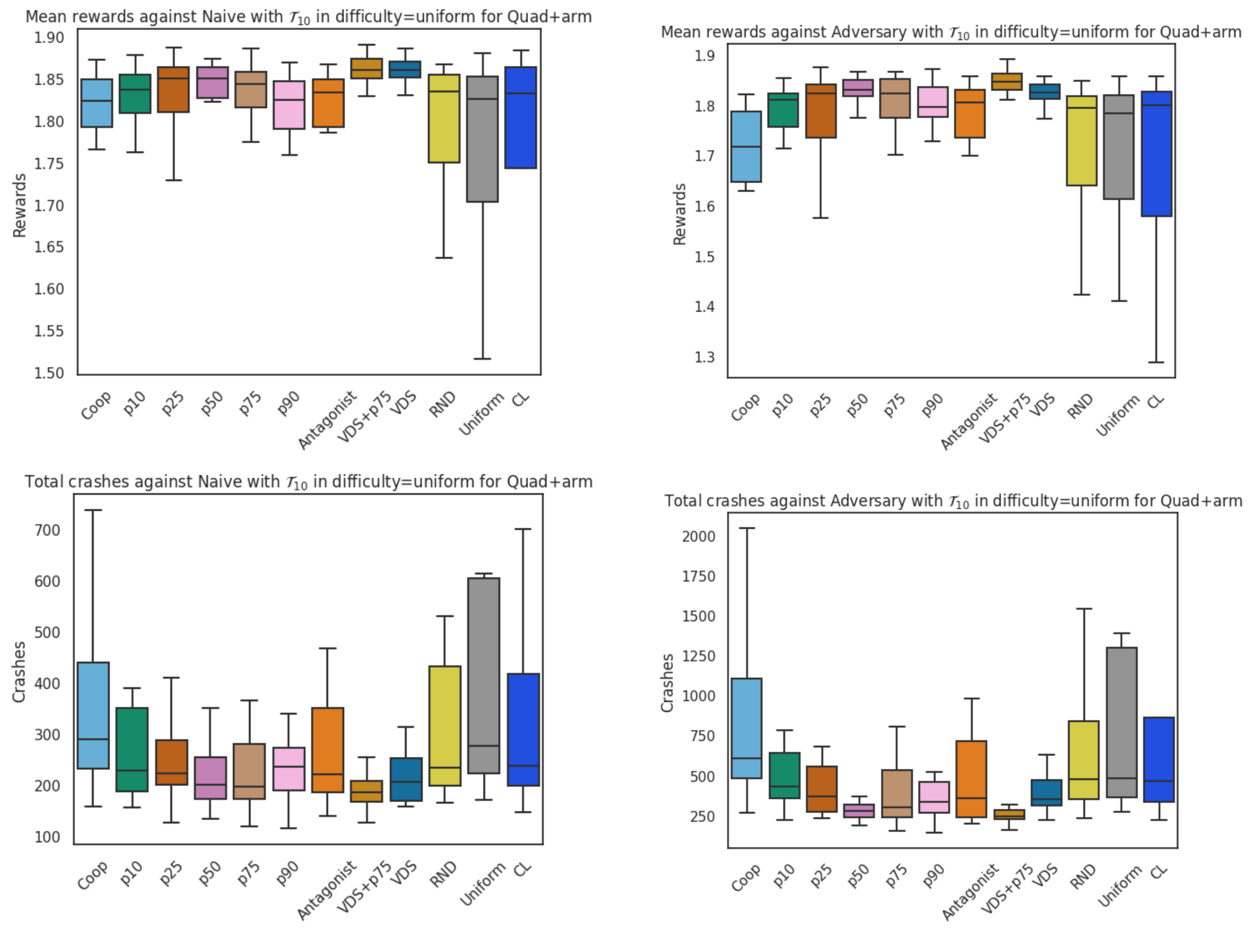


Figure 4.11 – Results for T_{10} for Quad+arm. Again, almost all DEIMOS versions produce better policy populations that the baselines, except for VDS. The VDS+p75 schedule does best overall, with almost identical Naive results as VDS and better Adversary results.

Figures 4.10 and 4.11 present the evaluation results for the Quad+arm environment with goal resampling timings $\mathcal{T}_2$ and $\mathcal{T}_{10}$, respectively. For both figures, on the left, we find results for the policies when they respond to goals selected by the Naive goal distribution. On the right, we find results for the policies when they respond to goals selected by the Adversary.

Against the Naive ρ_g, for both timings, we identify DEIMOS with a p value of around 50 as the best version of DEIMOS. Here, all baseline methods produce poor policies with very high variance, with the exception of VDS. In fact, VDS and VDS+p75 best all of the other methods, both in terms of median and of variance.

Against the Adversary ρ_g, we observe the same trends as against the Naive ρ_g for the p values of DEIMOS. We identify $p = 50$ as the best p value. Contrarily to the results obtained in the Quad environment, the variance for the $\mathcal{T}_2$ timing is higher than the variance for the

$\mathcal{T}_{10}$ timing (shown in Figures 4.10 and 4.11, respectively). We find that all of the baseline methods perform poorly with the exception of VDS. We identify VDS and VDS+p75 as the best methods, with VDS+p75 obtaining more rewards, less crashes, and less variance than VDS.

Overall, we identify VDS+p75 as the best method for Quad+arm for both timings.

4.5.7. Discussion

Against the Naive ρ_g, in general, the goal distribution during training holds little influence on safety after training. For most environments, the value of p for DEIMOS also seems to have little influence over the reward and safety behaviour after training. Regarding the baselines, we identify VDS as the best baseline method. The Uniform goal distribution also obtains similar results for *Particle Gym*. **In *Particle Gym***, DEIMOS either matches or bests the best baseline method. **In *Legged Gym***, VDS+DEIMOS either matches or bests the best baseline method, VDS.

Against the Adversary ρ_g, the goal distribution during training holds more importance. **In *Particle Gym***, DEIMOS with an adequate p value produces better policy populations than the baseline methods. **In *Legged Gym***, VDS+DEIMOS with $p = 75$ seems to be the best method, both in terms of variance reduction and of median safety and rewards. Otherwise, VDS has a noticeable edge over the other baselines, and seems to perform slightly better against the Naive ρ_g overall. The other baselines do poorly, and especially poorly in Quad+arm.

Overall, we identify VDS+p75 as the best overall method for *Legged Gym*, and DEIMOS with $p = 25$ as the best method for *Particle Gym*.

In all environments, by looking at the median policy safety and rewards (and their variance), we notice an interesting trend: the different p values for DEIMOS seem to produce a gaussian curve (this curve is most evident against the Adversary, especially in Figures 4.7, 4.10 and 4.11). Indeed, notice that in all environments, Coop fares poorly. By increasing the p hyperparameter, the safety and reward metrics (including the population variance) improve until the best p values (around $p = 25$ for *Particle Gym* and around $p = 50$ for *Legged Gym*), and then the metrics worsen until reaching the Antagonist DEIMOS. Interestingly, Coop and Antagonist produce policies with similar metrics.

The similar behaviour of a fully adversarial and of a fully cooperative training environment distribution was also observed in *Generalization Games for Reinforcement Learning* [**15**], where a DRL agent called "nature" generated *Minigrid* [**11**] environments for a second DRL agent to solve. *Particle Gym* and *Legged Gym* are continuous environments with dense reward functions; *Minigrid* is a discrete environment with a sparse reward function. While in no way conclusive, we observe some evidence that for reinforcement learning, fully adversarial and fully cooperative agents that act as part of the training environment (like DEIMOS's Director or the aforementioned "nature") produce similar populations of policies. The cooperative Director is too kind and produces policies that are ill-equipped to deal with the rigors of the evaluation environment. The antagonist Director has the potential to produce great policies; however, its cruel nature can also hinder the policy's training, leading to relatively bad median policies.

4.6. Informing goal selection using a single network

Most methods that tackle the task of goal selection during training utilize a GAN to select the future goals [**23**]. These training setups mostly require introducing at least two additional neural networks (the discriminator and the generator) to the DRL agent's training. Moreover, discriminator-generator setups are famously difficult to train [**20**], and this is rendered even harder by the presence of the DRL agent. This goal selection regime becomes a complex training setup with three agents influencing one another. We eschew the traditional three-agent setup of generator, discriminator, and DRL agent; we instead rely on a single additional network.

Other similar works leverage a training setup of online adversarial DRL agents [**14**][**9**]. Online adversarial DRL agents are very slow to train. They require many additional neural networks [1]. These setups often require intricate hyperparameter and/or environment tuning to prevent one agent from overpowering the other.

To our knowledge, the only other state-of-the-art goal-selection method that directly utilizes data sampled from the policy's rollouts like DEIMOS is VDS [**67**]. VDS is an *ensemble method* and thus requires at least three additional networks. In contrast to the two other types of methods outlined above, VDS is actually very stable to train. However, the requirement of an ensemble of networks implies that utilizing VDS leads to slower training because updating neural parameters is a very computationally costly process.

In DEIMOS, a single additional network is required. The simplicity of our method comes at the cost of goal expressivity. Like VDS, our method would not handle generating image goals or goals over large vector spaces well.

As shown in Table 4.1, this simplicity results in fast simulation. While DEIMOS is slower than the simpler baseline methods, it still obtains quite high simulation frames per second (FPS). The slowest method in study is VDS. Note that resetting the environments after crashes is costly, so methods that produce unsafe policies are slightly penalized in terms of FPS.

1. Because different DRL methods utilize different numbers of neural networks, the amount of additional neural networks required by such methods varies. In any case, it is some multiplicative factor of the base DRL policy's number of neural networks.

Simulation frames per second during training

Method	Absolute	Directional	Relative
DEIMOS	276000 ± 5000	274000 ± 2000	268000 ± 3000
RND	284000 ± 5000	270000 ± 4000	272000 ± 3000
Uniform	331000 ± 7000	329000 ± 5000	332000 ± 3000
VDS	125000 ± 2000	119000 ± 1000	118000 ± 1000

Method	Quad	Quad+arm
DEIMOS	87000 ± 1000	75000 ± 4000
RND	94000 ± 1000	70000 ± 4000
Uniform	101000 ± 1000	78000 ± 4000
VDS	70000 ± 1000	63000 ± 1000
CL	101000 ± 1000	75000 ± 4000

Table 4.1 – Frames per second across all parallel environments during training, rounded to the nearest thousand, ± STD Error.

Chapter 5

Conclusion

We proposed DEIMOS, a novel training-goal selection method for deep reinforcement learning in GMDP problems. In this method, a neural network called the “fear network” learns to predict the learner agent’s crashes after receiving a new goal. This fear network parameterizes a new agent that we introduce, the “Director”. The Director utilizes the fear network to inform its selection of the goal to issue to the policy. By using an appropriate sampling strategy, the Director enables more reliable training of DRL policies for a subset of GMDP problems compared to baseline methods.

We conduct experiments in challenging quadruped locomotion and manipulation tasks and variants of a generalist toy problem. We show that our method improves trained policy population variance while also either improving or conserving the median and mean crashes and rewards compared to the best baselines for all evaluation environments. We show that fully cooperative and fully antagonist Directors seem to produce similar policies for the training and evaluation environments that we employ.

In the future, we plan to compare the real-world safety of the different methods in study on real hardware. We believe that DEIMOS will enable easier transfer of policies from simulation to the real world. We also plan to extend this work to more training environments. We would also like to investigate the possibility of training a fear network to work on the real hardware. We would then be able to use the fear scores to ignore unsafe commands or implement an “emergency stop” when the fear scores become too high. This would lead to smarter and safer higher-level controllers for embodied DRL.

One weakness of DEIMOS is that it requires access to a uniform distribution over the possible goals, and it also requires that the goals that are possible during training are the same as those that are possible during deployment/evaluation. To obtain an even stronger method, we would like to parameterize the process that samples the goals during goal generation. Instead of assuming access to a uniform distribution, we could learn a neural network to noisily generate likely goals in the goal space. Viewed from an RL perspective, this is similar to treating the fear network used in DEIMOS as a critic, and learning an accompanying actor to select actions for it. This would obviously complicate the training of the Director, but it would also enable us to generate image, text, or large vector-space goals.

In general, we believe that such “particle-based” methods are under-utilized in the current machine learning field. The idea of generating data by leveraging the capability of neural networks to handle massive input batches is quite simple, and could be applied to many different fields. We plan on conducting more research involving this idea in the future.

References

[1] Anonymous : The reward hypothesis is false. *In Submitted to The Eleventh International Conference on Learning Representations*, 2023. under review.

[2] Arthur Aubret, Laëtitia Matignon et Salima Hassas : A survey on intrinsic motivation in reinforcement learning. *CoRR*, abs/1908.06976, 2019.

[3] Richard Bellman : *Dynamic Programming.* Dover Publications, 1957.

[4] Yoshua Bengio, Jérôme Louradour, Ronan Collobert et Jason Weston : Curriculum learning. *In Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA, 2009. Association for Computing Machinery.

[5] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski et Susan Zhang : Dota 2 with large scale deep reinforcement learning. *CoRR*, abs/1912.06680, 2019.

[6] Glen Berseth, Daniel Geng, Coline Devin, Chelsea Finn, Dinesh Jayaraman et Sergey Levine : Smirl: Surprise minimizing RL in dynamic environments. *CoRR*, abs/1912.05510, 2019.

[7] Léon Bottou : Online algorithms and stochastic approximations. *In* David Saad, éditeur : *Online Learning and Neural Networks.* Cambridge University Press, Cambridge, UK, 1998. revised, oct 2012.

[8] Yuri Burda, Harrison Edwards, Amos J. Storkey et Oleg Klimov : Exploration by random network distillation. *CoRR*, abs/1810.12894, 2018.

[9] Andres Campero, Roberta Raileanu, Heinrich Küttler, Joshua B. Tenenbaum, Tim Rocktäschel et Edward Grefenstette : Learning with amigo: Adversarially motivated intrinsic goals. *CoRR*, abs/2006.12122, 2020.

[10] Olivier Chapelle et Lihong Li : An empirical evaluation of thompson sampling. *In* J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira et K.Q. Weinberger, éditeurs : *Advances in Neural Information Processing Systems*, volume 24. Curran Associates, Inc., 2011.

[11] Maxime Chevalier-Boisvert, Lucas Willems et Suman Pal : Minimalistic gridworld environment for gymnasium, 2018.

[12] Y. Chow : *Risk-Sensitive and Data-Driven Sequential Decision Making.* Thèse de doctorat, Stanford University, Dept. of Aeronautics and Astronautics, Stanford, California, mars 2017.

[13] Yinlam Chow, Aviv Tamar, Shie Mannor et Marco Pavone : Risk-sensitive and robust decision-making: a cvar optimization approach. *CoRR*, abs/1506.02188, 2015.

[14] Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre M. Bayen, Stuart Russell, Andrew Critch et Sergey Levine : Emergent complexity and zero-shot transfer via unsupervised environment design. *CoRR*, abs/2012.02096, 2020.

[15] Manfred Diaz, Charlie Gauthier, Glen Berseth et Liam Paull : Generalization games for reinforcement learning. *In ICLR 2022 Workshop on Gamification and Multiagent Solutions*, 2022.

[16] Benjamin Eysenbach, Abhishek Gupta, Julian Ibarz et Sergey Levine : Diversity is all you need: Learning skills without a reward function. *CoRR*, abs/1802.06070, 2018.

[17] Zipeng Fu, Xuxin Cheng et Deepak Pathak : Deep whole-body control: Learning a unified policy for manipulation and locomotion. *In Conference on Robot Learning (CoRL)*, 2022.

[18] Carlos E. García, David M. Prett et Manfred Morari : Model predictive control: Theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

[19] Ian Goodfellow, Yoshua Bengio et Aaron Courville : *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[20] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville et Yoshua Bengio : Generative adversarial networks, 2014.

[21] Shixiang Gu, Ethan Holly, Timothy P. Lillicrap et Sergey Levine : Deep reinforcement learning for robotic manipulation. *CoRR*, abs/1610.00633, 2016.

[22] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel et Sergey Levine : Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *CoRR*, abs/1801.01290, 2018.

[23] David Held, Xinyang Geng, Carlos Florensa et Pieter Abbeel : Automatic goal generation for reinforcement learning agents. *CoRR*, abs/1705.06366, 2017.

[24] Shengyi Huang, Rousslan Fernand Julien Dossa, Antonin Raffin, Anssi Kanervisto et Weixun Wang : The 37 implementation details of proximal policy optimization. *In ICLR Blog Track*, 2022. https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/.

[25] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan et Chrisina Jayne : Imitation learning: A survey of learning methods. *ACM Comput. Surv.*, 50(2), apr 2017.

[26] Prateek Jain et Purushottam Kar : Non-convex optimization for machine learning. *Found. Trends Mach. Learn.*, 10:142–336, 2017.

[27] Nils P. Johnson : The brachistochrone problem. *The College Mathematics Journal*, 35:192 – 197, 2004.

[28] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran et Raia Hadsell : Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, 2017.

[29] Nikki Lijing Kuang et Clement H. C. Leung : Performance dynamics and termination errors in reinforcement learning: A unifying perspective. *CoRR*, abs/1902.04179, 2019.

[30] Nikki Lijing Kuang et Clement H. C. Leung : Performance dynamics and termination errors in reinforcement learning: A unifying perspective. *CoRR*, abs/1902.04179, 2019.

[31] Yann LeCun, Y. Bengio et Geoffrey Hinton : Deep learning. *Nature*, 521:436–44, 05 2015.

[32] Zachary C. Lipton, Jianfeng Gao, Lihong Li, Jianshu Chen et Li Deng : Combating reinforcement learning's sisyphean curse with intrinsic fear. *CoRR*, abs/1611.01211, 2016.

[33] Minghuan LIU, Menghui ZHU et Weinan ZHANG : Goal-conditioned reinforcement learning: Problems and solutions. *CoRR*, abs/2201.08299, 2022.

[34] Zuxin LIU, Zhepeng CEN, Vladislav ISENBAEV, Wei LIU, Zhiwei Steven WU, Bo LI et Ding ZHAO : Constrained variational policy optimization for safe reinforcement learning. *CoRR*, abs/2201.11927, 2022.

[35] Viktor MAKOVIYCHUK, Lukasz WAWRZYNIAK, Yunrong GUO, Michelle LU, Kier STOREY, Miles MACKLIN, David HOELLER, Nikita RUDIN, Arthur ALLSHIRE, Ankur HANDA et Gavriel STATE : Isaac gym: High performance gpu-based physics simulation for robot learning, 2021.

[36] Viktor MAKOVIYCHUK, Lukasz WAWRZYNIAK, Yunrong GUO, Michelle LU, Kier STOREY, Miles MACKLIN, David HOELLER, Nikita RUDIN, Arthur ALLSHIRE, Ankur HANDA et Gavriel STATE : Isaac gym: High performance gpu-based physics simulation for robot learning. *CoRR*, abs/2108.10470, 2021.

[37] Kathryn MANNIE : Chess-playing robot breaks 7-year-old's finger during russian tournament - national, Jul 2022.

[38] James Clerk MAXWELL : *On Governors*, volume 2 de *Cambridge Library Collection - Physical Sciences*, page 105–120. Cambridge University Press, 2011.

[39] N. MINORSKY. : Directional stability of automatically steered bodies. *Journal of the American Society for Naval Engineers*, 34(2):280–309, 1922.

[40] Mehdi MIRZA et Simon OSINDERO : Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.

[41] Volodymyr MNIH, Koray KAVUKCUOGLU, David SILVER, Alex GRAVES, Ioannis ANTONOGLOU, Daan WIERSTRA et Martin A. RIEDMILLER : Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[42] Volodymyr MNIH, Koray KAVUKCUOGLU, David SILVER, Andrei A. RUSU, Joel VENESS, Marc G. BELLEMARE, Alex GRAVES, Martin RIEDMILLER, Andreas K. FIDJELAND, Georg OSTROVSKI, Stig PETERSEN, Charles BEATTIE, Amir SADIK, Ioannis ANTONOGLOU, Helen KING, Dharshan KUMARAN, Daan WIERSTRA, Shane LEGG et Demis HASSABIS : Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb 2015.

[43] OPENAI, Marcin ANDRYCHOWICZ, Bowen BAKER, Maciek CHOCIEJ, Rafal JÓZEFOWICZ, Bob MCGREW, Jakub PACHOCKI, Arthur PETRON, Matthias PLAPPERT, Glenn POWELL, Alex RAY, Jonas SCHNEIDER, Szymon SIDOR, Josh TOBIN, Peter WELINDER, Lilian WENG et Wojciech ZAREMBA : Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177, 2018.

[44] Deepak PATHAK, Pulkit AGRAWAL, Alexei A. EFROS et Trevor DARRELL : Curiosity-driven exploration by self-supervised prediction. *In International Conference on Machine Learning (ICML)*, 2017.

[45] Marek PETRIK, Mohammad GHAVAMZADEH et Yinlam CHOW : Safe policy improvement by minimizing robust baseline regret. *In Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 2306–2314, Red Hook, NY, USA, 2016. Curran Associates Inc.

[46] Lerrel PINTO, James DAVIDSON, Rahul SUKTHANKAR et Abhinav GUPTA : Robust adversarial reinforcement learning. *CoRR*, abs/1703.02702, 2017.

[47] Silviu PITIS, Harris CHAN, Stephen ZHAO, Bradly C. STADIE et Jimmy BA : Maximum entropy gain exploration for long horizon multi-goal reinforcement learning. *CoRR*, abs/2007.02832, 2020.

[48] Vitchyr H. PONG, Murtaza DALAL, Steven LIN, Ashvin NAIR, Shikhar BAHL et Sergey LEVINE : Skew-fit: State-covering self-supervised reinforcement learning. *CoRR*, abs/1903.03698, 2019.

[49] Steve ROBERTS : Babyrobot. https://github.com/WhatIThinkAbout/BabyRobot, 2020.

[50] Stéphane ROSS, Geoffrey J. GORDON et J. Andrew BAGNELL : No-regret reductions for imitation learning and structured prediction. *CoRR*, abs/1011.0686, 2010.

[51] Nikita RUDIN, David HOELLER, Philipp REIST et Marco HUTTER : Learning to walk in minutes using massively parallel deep reinforcement learning. *CoRR*, abs/2109.11978, 2021.

[52] David E. RUMELHART, Geoffrey E. HINTON et Ronald J. WILLIAMS : Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[53] Daniel RUSSO, Benjamin Van ROY, Abbas KAZEROUNI et Ian OSBAND : A tutorial on thompson sampling. *CoRR*, abs/1707.02038, 2017.

[54] Julian SCHRITTWIESER, Ioannis ANTONOGLOU, Thomas HUBERT, Karen SIMONYAN, Laurent SIFRE, Simon SCHMITT, Arthur GUEZ, Edward LOCKHART, Demis HASSABIS, Thore GRAEPEL, Timothy LILLICRAP et David SILVER : Mastering atari, go, chess and shogi by planning with a learned model. *Nature*, 588(7839):604–609, Dec 2020.

[55] John SCHULMAN, Filip WOLSKI, Prafulla DHARIWAL, Alec RADFORD et Oleg KLIMOV : Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

[56] Aivar SOOTLA, Alexander I COWEN-RIVERS, Taher JAFFERJEE, Ziyan WANG, David H MGUNI, Jun WANG et Haitham AMMAR : Saute RL: Almost surely safe reinforcement learning using state augmentation. *In* Kamalika CHAUDHURI, Stefanie JEGELKA, Le SONG, Csaba SZEPESVARI, Gang NIU et Sivan SABATO, éditeurs : *Proceedings of the 39th International Conference on Machine Learning*, volume 162 de *Proceedings of Machine Learning Research*, pages 20423–20443. PMLR, 17–23 Jul 2022.

[57] Richard S SUTTON et Andrew G BARTO : *Reinforcement learning: An introduction.* MIT press, 2018.

[58] Chen TESSLER, Yonathan EFRONI et Shie MANNOR : Action robust reinforcement learning and applications in continuous control. *In* Kamalika CHAUDHURI et Ruslan SALAKHUTDINOV, éditeurs : *Proceedings of the 36th International Conference on Machine Learning*, volume 97 de *Proceedings of Machine Learning Research*, pages 6215–6224. PMLR, 09–15 Jun 2019.

[59] Brijen THANANJEYAN, Ashwin BALAKRISHNA, Suraj NAIR, Michael LUO, Krishnan SRINIVASAN, Minho HWANG, Joseph E. GONZALEZ, Julian IBARZ, Chelsea FINN et Ken GOLDBERG : Recovery RL: safe reinforcement learning with learned recovery zones. *CoRR*, abs/2010.15920, 2020.

[60] WILLIAM R THOMPSON : ON THE LIKELIHOOD THAT ONE UNKNOWN PROBABILITY EXCEEDS ANOTHER IN VIEW OF THE EVIDENCE OF TWO SAMPLES. *Biometrika*, 25(3-4):285–294, 12 1933.

[61] Matteo TURCHETTA, Andrey KOLOBOV, S. SHAH, Andreas KRAUSE et Alekh AGARWAL : Safe reinforcement learning via curriculum induction. *ArXiv*, abs/2006.12136, 2020.

[62] Christopher J. C. H. WATKINS et Peter DAYAN : Q-learning. *Machine Learning*, 8(3):279–292, May 1992.

[63] WIKIPEDIA : Brachistochrone curve — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Brachistochrone%20curve&oldid=1123571483, 2022. [Online; accessed 29-November-2022].

[64] Zheng XU, Michael J. WILBER, Chen FANG, Aaron HERTZMANN et Hailin JIN : Beyond textures: Learning from multi-domain artistic images for arbitrary style transfer. *CoRR*, abs/1805.09987, 2018.

[65] Tsung-Yen Yang, Tingnan Zhang, Linda Luu, Sehoon Ha, Jie Tan et Wenhao Yu : Safe reinforcement learning for legged locomotion, 2022.

[66] Peng Zhai, Jie Luo, Zhiyan Dong, Lihua Zhang, Shunli Wang et Dingkang Yang : Robust adversarial reinforcement learning with dissipation inequation constraint. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(5):5431–5439, Jun. 2022.

[67] Yunzhi Zhang, Pieter Abbeel et Lerrel Pinto : Automatic curriculum learning through value disagreement. *CoRR*, abs/2006.09641, 2020.

Appendix A

Additional results for *Legged Gym*

Here, we present additional results for *Legged Gym*.

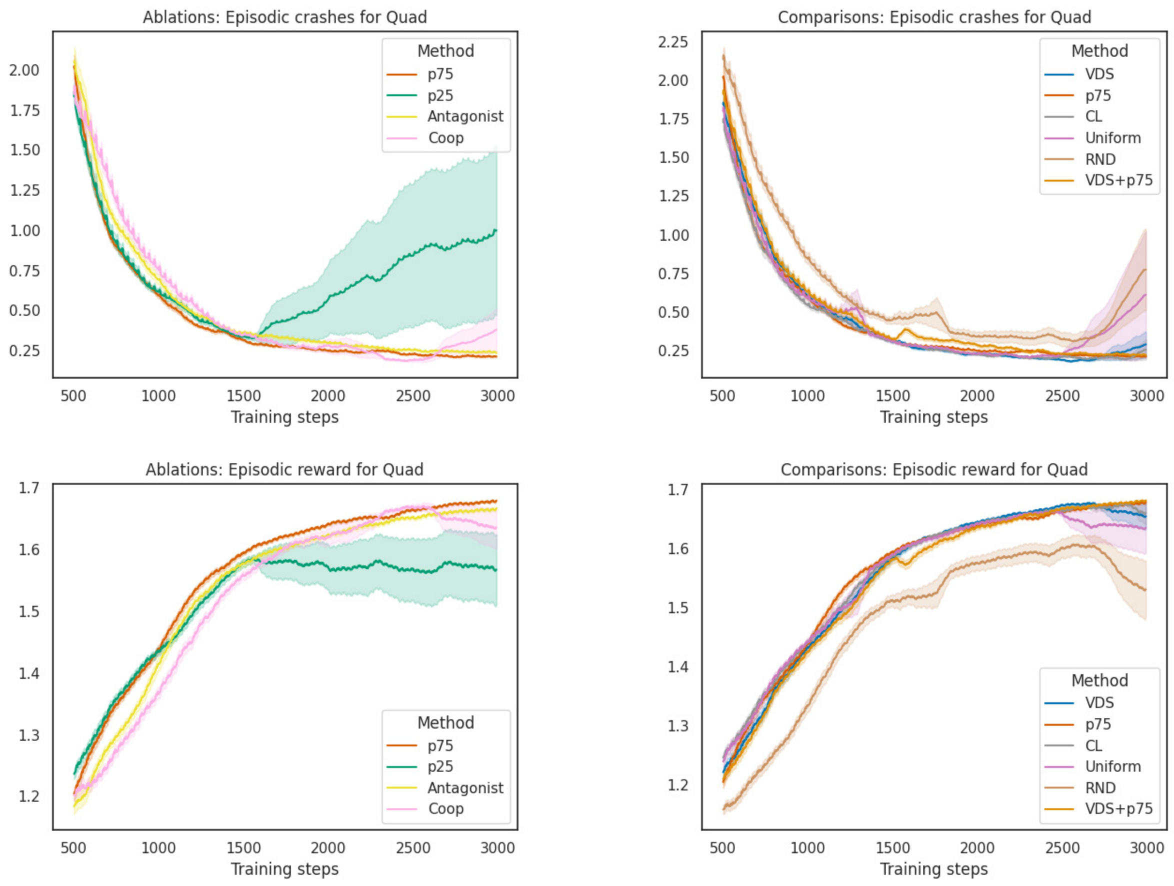


Figure A.1 – Reward and safety training curves for Quad.

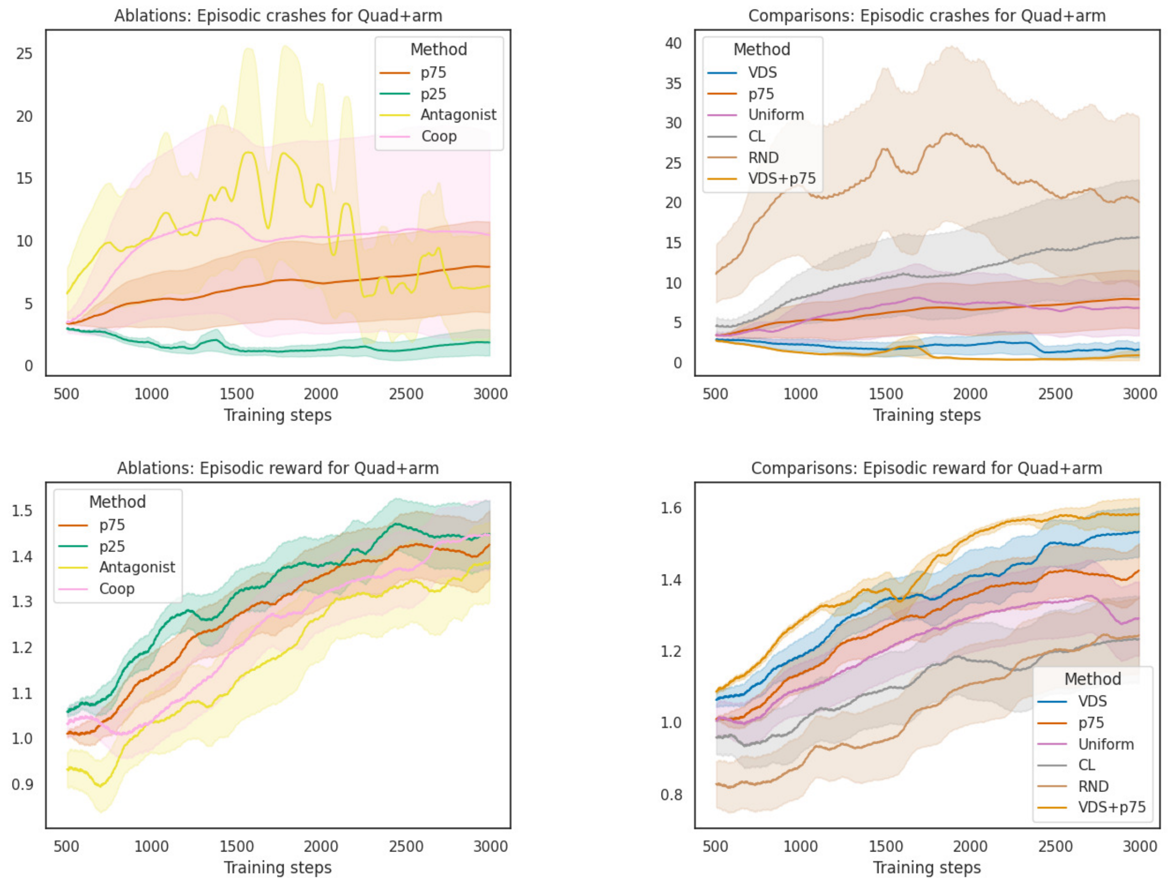


Figure A.2 – Reward and safety training curves for Quad+arm.

The curves in Figures A.1 and A.2 reflect the policy's goal-matching and goal-crashing behaviours. As such, they cannot truly be compared against one another: each curve is idiosyncratic to their goal-selection method. Pitting the policies against another goal-selecting method during evaluation shows behaviour that often belies the curves presented here.

Still, we can infer further supporting evidence of the findings presented in 4.5. Both for Quad and Quad+arm, we observe that Antagonist produces policy populations with large standard errors. In Quad+arm, we also clearly observe that Coop similarly produces high-variance populations.

We also verify that VDS and VDS+p75 indeed produce policies in the way that we had intuited in Chapter 4.3. Both in Quad and Quad+arm, the two methods' curves are obviously very similar (since both are simply sampling using VDS). After DEIMOS takes over after 1500 timesteps, we observe a change in the trajectory of the curve. In Quad, the policies experience an heightened number of crashes until the end of training. This makes sense, as DEIMOS indeed samples adversarial goals. In Quad+Arm, the policies initially suffer the same fate, crashing more often than when trained with VDS. But then the policies quickly adapt, leading to even less goals than before DEIMOS took over.