INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2023IPPAXXXX

Thèse de doctorat

ÉCOLE POLYTECHNIQUE

IP PARIS

Simulating Crowds with Reinforcement Learning

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à l'École Polytechnique

École doctorale n°626 École doctorale de l'Institut Polytechnique de Paris (EDIPP)
Spécialité de doctorat: Informatique

Thèse présentée et soutenue à Palaiseau, le 30 novembre 2023, par

Ariel Kwiatkowski

Composition du Jury :

Sylvie Gibet Université Bretagne Sud	Président
Ludovic Denoyer Sorbonne Université	Rapporteur
Nicolas Courty Universite Bretagne Sud	Rapporteur
Ioannis Karamouzas University of California, Riverside	Examinateur
Jesse Read École Polytechnique	Examinateur
Marie-Paule Cani École Polytechnique	Directrice de thèse
Julien Pettré INRIA Rennes	Co-directeur de thèse
Vicky Kalogeiton École Polytechnique	Co-directrice de thèse (Invité)

Abstract

Simulating crowd behavior is an important aspect of creating immersive digital environments, be it for video games or other virtual experiences. Traditional methods lead to satisfactory results but are often limited in their capacity to accurately emulate the complexity of human behavior. Recently, Reinforcement Learning (RL) has emerged as a new approach to tackle this problem. However, there are many details of RL-driven crowd simulation that may seem irrelevant, but turn out to be rather impactful. This includes the underlying physics simulation, models of observations and dynamics, and details of the RL algorithm optimizing the crowd's behavior.

This thesis aims to shed light on these critical details and their effects on virtual crowds trained with RL. Our overarching objective is to establish an understanding of relevant design choices, enabling the creation of more realistic crowd simulations.

In the first part of the thesis, we focus on evaluating how various design choices of the foundational crowd simulation impact both the learning performance and the overall quality of the resulting behavior. We present a classification of observation methods and dynamics, and evaluated their impact with DRL experiments. This shows that nonholonomic controls with a variant of egocentric observations produce better results compared to other, simpler alternatives.

Following this, we investigate the details of reward function design for simulating human-like crowds. We explore different reward functions, providing theoretical insights on their properties, and evaluate them empirically in different scenarios. Our experiments show that directly minimizing energy usage, when paired with a properly scaled guiding potential, are effective in producing more efficient crowd behaviors.

In the final part of the thesis, we explore the discounting mechanism in RL. We present the Universal Generalized Advantage Estimation (UGAE) algorithm, a novel solution that enables using modern RL algorithms with arbitrary discounting. We also introduce Beta-weighted discounting to parameterize non-exponential discounting methods. We demonstrate that UGAE outperforms the Monte Carlo baseline using both standard RL benchmarks and crowd simulation scenarios. This paves the way to future crowd simulation methods using non-exponential discounting, which may help overcome some of the challenges identified in our previous work.

This work, combined, provides critical insights into the dynamics of reinforcement learned crowds, and contributes significantly to the development of new and improved techniques for crowd simulation.

Résumé

Simuler le comportement des foules constitue une composante clé de la création d'espaces numériques immersifs. Les méthodes traditionnelles, bien qu'efficaces, sont souvent limitées dans leur capacité à reproduire fidèlement la complexité du comportement humain. Récemment, l'apprentissage par renforcement (RL) a émergé comme une nouvelle approche pour surmonter ce défi. Cependant, de nombreux détails de la simulation des foules par RL peuvent sembler négligeables, mais s'avèrent avoir un impact majeur, incluant la simulation physique sous-jacente, les modèles d'observations et de dynamiques, et les détails de l'algorithme RL lui-même.

Cette thèse vise à mettre en lumière ces détails cruciaux et leurs effets sur les foules virtuelles formées par RL. Notre objectif est d'établir une compréhension des choix de conception pertinents qui permettraient la création de simulations de foules plus réalistes.

Dans la première partie, nous nous concentrons sur l'évaluation de l'impact des divers choix de conception sur la performance d'apprentissage et la qualité du comportement résultant. Nos expériences avec le Deep RL montrent que les contrôles non holonomiques avec une variante d'observations égocentriques produisent de meilleurs résultats par rapport aux autres alternatives plus simples.

Ensuite, nous examinons les détails de la conception de la fonction de récompense pour simuler des foules semblables aux humains. Nos expériences montrent qu'une minimisation directe de l'utilisation d'énergie, lorsqu'elle est couplée à un potentiel de guidage correctement calibré, permet de générer des comportements de foule plus efficaces.

Enfin, nous explorons le mécanisme d'escompte dans le RL. Nous présentons l'algorithme UGAE, une nouvelle solution qui permet l'utilisation d'algorithmes RL modernes avec un escompte arbitraire. Nous démontrons que UGAE surpasse la base de référence de Monte Carlo en utilisant à la fois des critères de référence RL standard et des scénarios de simulation de foule. Ceci ouvre la voie à de futures méthodes de simulation de foule utilisant un escompte non exponentiel.

Dans l'ensemble, cette recherche apporte des éclairages essentiels sur la dynamique des foules formées par RL, et contribue significativement au développement de nouvelles techniques et à l'amélioration des techniques existantes pour la simulation de foule.

Contents

1 Introduction **9**

1.1 Objective . . . 12
1.2 Methodology . . . 12
1.3 Thesis structure . . . 13
1.4 Publications . . . 13

2 State of the Art **15**

2.1 Introduction . . . 15
2.1.1 Problems in Character Animation . . . 16
2.2 Definitions and Preliminaries . . . 16
2.2.1 Reinforcement Learning Formalisms . . . 17
2.2.2 Fundamentals of RL Algorithms . . . 21
2.2.3 Reward Hypothesis, Discounting, Advantage . . . 23
2.3 Classification of RL Algorithms . . . 25
2.3.1 Policy-based or Value-based . . . 25
2.3.2 Actor-Critic . . . 25
2.3.3 On-policy or Off-policy . . . 25
2.3.4 Model-free or Model-based . . . 26
2.3.5 Single-agent or Multiagent . . . 26
2.3.6 Summary . . . 26
2.4 Single-agent RL Algorithms . . . 27
2.4.1 DQN . . . 27
2.4.2 Rainbow . . . 28
2.4.3 REINFORCE . . . 29
2.4.4 TRPO . . . 29
2.4.5 PPO . . . 30
2.4.6 A3C, A2C . . . 31
2.4.7 GAE . . . 31
2.4.8 DDPG . . . 32
2.4.9 TD3 . . . 33
2.4.10 SAC . . . 33
2.4.11 Learning from Data . . . 34
2.4.12 Summary . . . 35
2.5 Multiagent RL Algorithms . . . 35
2.5.1 Independent Learning . . . 35

2.5.2 MADDPG . . . 35
2.5.3 MAPPO . . . 36
2.5.4 QMIX . . . 36
2.5.5 Summary . . . 37
2.6 Skeletal Animation . . . 37
2.6.1 RL for Kinematic Motion Synthesis . . . 37
2.6.2 The Many Challenges Beyond the Choice of Algorithm . . . 37
2.6.3 RL for Individual Character Skills . . . 39
2.7 Crowd Animation . . . 43
2.7.1 Challenges of Crowds . . . 43
2.7.2 Applications . . . 43
2.8 Human Interaction . . . 45
2.9 Frameworks . . . 46
2.9.1 Neural Networks . . . 46
2.9.2 Environments . . . 47
2.9.3 Algorithm implementations . . . 49
2.9.4 Summary . . . 50
2.10 Conclusions . . . 51

3 Reinforcement Learning for Crowd Simulation 55
3.1 Introduction . . . 55
3.2 Environment Design Choices . . . 56
3.2.1 Problem Formulation . . . 56
3.2.2 Observation Space . . . 57
3.2.3 Action Space and Dynamics . . . 58
3.3 Reward Function Design . . . 60
3.3.1 Energy and Metrics . . . 61
3.3.2 Reward and Preferred Velocity . . . 61
3.3.3 Energy as reward . . . 63
3.4 Experimental setup . . . 64
3.4.1 Policy Optimization . . . 65
3.4.2 Network Architecture . . . 65
3.5 Experiments . . . 67
3.5.1 Dynamics and Observations performance . . . 67
3.5.2 All Scenarios . . . 67
3.5.3 Velocity Reward Exponent . . . 70
3.5.4 Importance of collision penalty . . . 70
3.5.5 Common Failure Modes . . . 72
3.6 Discussion . . . 73
3.6.1 Limitations and Future Work . . . 74
3.6.2 Conclusions . . . 74

4 Reward function design 75
4.1 Introduction . . . 75
4.2 Energy Usage Model . . . 76

4.2.1 Acceleration correction . . . 77
4.3 Navigation reward design . . . 78
4.3.1 Energy as reward . . . 78
4.3.2 Energy-based potential . . . 80
4.3.3 Discounting invariance . . . 81
4.3.4 Non-finishing penalty . . . 82
4.3.5 Alternative approaches . . . 82
4.4 Reward evaluation . . . 83
4.4.1 Experimental setup . . . 83
4.4.2 Reward function structure . . . 84
4.5 Results . . . 85
4.5.1 Is potential necessary? . . . 87
4.5.2 Impact of acceleration . . . 88
4.6 Conclusions . . . 88

5 Non-exponential reward discounting **91**
5.1 Introduction . . . 91
5.2 UGAE – Universal Generalized Advantage Estimation . . . 92
5.2.1 UGAE . . . 93
5.2.2 Added estimation bias . . . 94
5.2.3 Non-exponential discounting . . . 95
5.3 Beta-weighted discounting . . . 95
5.3.1 Beta-weighted discounting . . . 95
5.3.2 Beta distribution properties . . . 96
5.4 Analysis of non-exponential discounting methods . . . 97
5.4.1 Properties of discounting . . . 97
5.4.2 Experimental Analysis . . . 98
5.4.3 Discussion . . . 99
5.4.4 Why non-exponential discounting? . . . 100
5.5 DRL Experiments . . . 100
5.5.1 Results . . . 102
5.5.2 Computation time . . . 102
5.5.3 Discussion . . . 102
5.6 Conclusions . . . 103

6 Conclusion **105**

A Non-exponential reward discounting **111**
A.1 Proofs . . . 111
A.2 Beta-weighted Discounting Properties . . . 117
A.3 Pathworld experiments . . . 118
A.3.1 Setup . . . 118
A.3.2 Pathworld results . . . 119

B Reward function design **121**
B.1 Discounting-invariant reward 121
B.2 Algorithmic details 122
B.3 Reward implementation details 122

List of Figures

1.1 A crowd of real people walking through a crossing in Tokyo. 9

1.2 A crowd of virtual people in virtual Paris, as seen in *Assassin's Creed Unity*. . . . 10

1.3 A crowd of virtual people in a simple virtual Reinforcement Learning environment. 11

2.1 A visual depiction of the basic Reinforcement Learning loop corresponding to the POMDP formalism. The agent and the environment exchange information between each other. The agent perceives the environment state and executes an action. The environment then updates its state, and communicates it to the agent via an observation function, together with the reward for the last action. 17

2.2 A diagram showing a taxonomy of the Reinforcement Learning algorithms described in this work. We focus on two divisions: single agent or multiagent, and policy-based or value-based. The colors of nodes correspond to whether the algorithm is on-policy (red), off-policy (blue), or in between (green). Algorithms marked with an asterisk (∗) can only be used with discrete action spaces. 24

2.3 Imitation-based Learning. Proposed methods as in [126] allow to successfully synthesize animations from motion capture data. In other works, as in [129], they combine such techniques with the possibility of adding low-level behaviours to control the production of high-complexity animations. 40

2.4 There is a wide variety of methods that also address the synthesis of animations for quadrupedal or arbitrary morphology [108] [126] [190]. While the limited amount of motion capture data introduces an additional challenge, such methods try to overcome this constraint by covering a wide range of techniques, from imitation-based approaches to pure objective RL. 42

2.5 The relative popularity of PyTorch (red) and TensorFlow (blue) in terms of search volume on Google according to Google Trends, worldwide, between 01/01/2016 and 27/07/2021. 47

2.6 A visualization of different legged models of varying complexity. The agents' objective is moving each of the joints so that the overall center of mass moves forward or is balanced, while minimizing the energy expenditure. From left to right: Hopper, Walker2d, Humanoid (MuJoCo) and Humanoid (PyBullet). 48

3.1 A schematic representation of the available action spaces. In each case, we take a bird's-eye view of an agent moving in the positive Y direction at an intermediate speed, represented by the blue arrow. The blue circle represents the space of all physically possible velocities (i.e. below the maximum speed). The green area represents the velocities that the agent is able to have in the following timestep under the specific action space. 59

3.2 **(a)** Rewards and energy for an agent moving at a constant speed, in the simplified model described in Section 3.3.2. All curves are normalized to be in the [0, 1] range in order to enable direct comparison. We consider energy values with the opposite sign, because energy is supposed to be minimized, while the reward is maximized. **(b)** MSE between the reward and the energy, as a function of the velocity reward exponent. **(c)** Optimal velocity as a function of the velocity reward coefficient c_v, varied by the exponent c_e. **(d)** Discounted negative energy expenditure as a function of velocity. 62

3.3 Agent's initial positions and goals in four scenarios: **(a)** Circle with 30 agents. **(b)** Corridor with 72 agents. **(c)** Crossing with 32 agents. **(d)** Random with 15 agents. 64

3.4 The neural architecture used as the policy. Green blocks represent inputs, blue blocks represent feed-forward neural networks, yellow blocks represent vector operations, red blocks represent outputs. Depending on the observation model used, certain elements of the architecture are disabled. 66

3.5 Comparison of training results after a hyperparameter search in the Circle 12 scenario. **(a)** Mean episodic reward **(b)** Mean energy expenditure. Black bars represent the standard error of the mean. 68

3.6 Comparison of various design choices in a given environment. The last bar corresponds to the best-performing agent across all design choices. All values are averaged across 8 independent training runs with different random seeds and otherwise identical parameters. Lower is better. AP stands for Agent Perception as defined in Section 3.2.2. 69

3.7 Comparison of energy usage in agents trained with a different exponent in the velocity term of the reward function. Lower is better. 71

3.8 Comparison of energy usage and success rate in agents trained in Circle 12 scenario, with a varied collision penalty in the reward function. 71

3.9 Comparison of energy usage, collision count and success rate in agents trained in Crossway 50 scenario, with a varied collision penalty in the reward function. . . . 72

4.1 Agent's initial positions and goals in five scenarios: **(a)** Circle with 40 agents. **(b)** Corridor with 50 agents. **(c)** Crossing with 50 agents. **(d)** Choke with 20 agents. **(e)** Car with 20 agents. In each scenario, agents must reach the goal with the same color as them. In the circle scenario, initial starting positions are randomly perturbed during each episode. In the car scenario, the obstacle at the bottom of the scene moves upwards. 75

4.2 Energy used in a single timestep when moving at a velocity of v, after having the velocity of 1.3 m/s in the previous timestep, with $\Delta t = 0.01$ s. 77

4.3 Normalized discounted reward, with energy optimization as the direct objective. Depending on the distance d and the discount factor γ, the global optimum is different, and in some cases, the optimal behavior is standing still with $v = 0$. . . . 79
4.4 Success rates of agents trained with certain reward functions in the Circle scenario. 86
4.5 Energy+ metric as a function of training progress with various reward functions. To maintain the performance from the first stage of the training, it is necessary to either use a potential term, or set the discount factor to $\gamma = 1$. Agents without a potential or a final heuristic converge to standing still, while other variants' performance significantly degrades. 87
4.6 Histogram of accelerations in the Circle scenario, trained with and without the acceleration-based term in the reward function. 88

5.1 Different properties of a discounting, as a function of η, with given (μ, T_{max}) parameters listed in the legend. (a) Importance of the near future (b) Variance measure (c) Effective time horizon (d) Total discounting sum 99
5.2 Visualizations of the two crowd simulation scenarios used in the experiments. In both cases, each agent needs to reach the opposite end of their respective route, and is then removed from the simulation. 100
5.3 Training curves in DRL experiments using non-exponential discounting. All curves are averaged across 8 independent training runs. Shading indicates the standard error of the mean. In all experiments, using λ values that were tuned for optimality with exponential discounting, significantly outperform the MC baseline ($\lambda = 1$). This indicates that UGAE enables translating the benefits of GAE to non-exponential discounting. 101
5.4 Time needed to compute GAE (orange) and UGAE (blue) with a single consumer CPU, on log-log scale. The green line is a reference duration of 10 seconds representing a typical training iteration. While UGAE is more expensive, with typical training step durations, the time to compute its values is negligible. 102

A.1 Pathworld environment results under different discounting schemes. Hyperbolic [36] and exponential (various curves) discountings fail to approximate the empirical (dashed) value. Instead, the proposed Beta-weighted discounting approximates it much better, despite its different functional form. 118

List of Tables

2.1 A comparison of different formalisms used to define an RL problem. Legend: × – the property cannot be modelled in this formalism, ∼ – the property can be modelled in this formalism, but is not the intended use or requires extra effort, ✓ – the property can be modelled in this formalism, ★ – this formalism is particularly suitable for this property. Multiagent Cooperative and Competitive refers to the rewards being either shared or zero-sum, respectively. Multiagent Mixed is neither fully cooperative nor competitive. Simultaneous and Turn-based refers to whether all agents take their actions at the same time, or only one agent does. 20

2.2 A summary of the DRL algorithms, simulation engines, and neural network frameworks in the described papers, where applicable and stated in the paper or the provided source code. [1] Value Iteration, [2] Open Dynamics Engine, [3] Temporal Difference learning, [4] Maximum A Posteriori Policy Optimization. 52

2.3 A comparison of algorithm support between various frameworks. Legend: ✓ – algorithm supported by the framework, × – algorithm not supported by the framework. Multiagent refers to the capability of training in multiagent environments, with or without parameter sharing. Note that this is not a complete list of algorithms implemented by each framework, as some of them include many other, less relevant algorithms. 53

2.4 Continuation of Table 2.3. 53

4.1 Mean value of the Energy+ metric after training in a given scenario, using a given reward function. Each value is based on 8 independent training runs. Lower is better. 86

A.1 The values of different metrics for a chosen set of discounting method and their parameters. 117

A.2 Values of the Mean Square Error for different discounting methods on the Pathworld environment, summed across the first 14 paths $i \in \overline{1, 14}$. Lower is better. 118

Chapter 1

Introduction

Figure 1.1: A crowd of real people walking through a crossing in Tokyo.

Virtual worlds have become increasingly important in today's entertainment. From movies, through video games, and all the way to immersive virtual reality experiences, people often want to be in a different world – whether as part of it or shaping it to their own preferences. For this to be successful, it is important to create a world that feels *alive*, whether or not it is realistic.

Beyond just entertainment, crowd simulation has practical applications in fields like urban design and transport planning. By simulating realistic human behavior, city planners can optimize public spaces, transit routes, and building layouts. These simulations help make real-world infrastructures more efficient, directly impacting daily life, showing that it is also worthwhile to pursue *realistic* simulations.

The key component in building such lively worlds is the people (or, in some cases, other creatures) – and since the worlds are virtual, so are they. Media with multiple users have the "simplest" solution, with each character being controlled by a real-life human being, ensuring that they are as human-like as possible. Or are they? There is a subtle but important distinction between the

Figure 1.2: A crowd of virtual people in virtual Paris, as seen in *Assassin's Creed Unity*.

way humans move in real life, and the way they move when controlling a virtual character.

Whether we want to simulate the motion of real-world humans, or the motion of *characters* controlled by real-world humans, this task can be roughly split into two constituent parts – animating each individual character, often at the level of skeletal or muscle control; and simulating the entire crowd, often at the level of trajectories. The two components can then be composed for a fully animated crowd, filled with human-like avatars.

In this thesis, we focus on the latter component – simulating the trajectories of crowd participants, abstracting away the details of their movement. The central question is "Given a certain arrangement of humans, where should they go to reach their goals, and at what pace?" The solution to this can then be used to guide individual animation models for a full simulation of humans in a virtual world.

Focusing on the level of trajectories offers a "bird's eye" perspective on the crowd, where each individual's movement is represented as a curve on a manifold, exhibiting a network of paths across the environment. This can also be explored from an egocentric point of view, considering the relative motions and interactions of the agents from the perspective of each individual. The movements captured at this level might include position, speed, and their changes due to obstacle avoidance or group interactions. These components serve as a basis for a realistic crowd simulation.

When the crowd becomes increasingly dense and complex, the efficiency provided by such a trajectory-level scale becomes even more important. Simulating the motion of the body of each character can prove computationally heavy and inefficient when applied to large crowds. Conversely, the abstraction provided by trajectory simulation enables the fast and efficient processing of crowd dynamics, focusing on the high-level paths of the individuals. Such a method allows for rapid simulations even for huge crowds, with the finer details of individual behavior added selectively as required, optimizing computational resources without compromising realism.

Current techniques for crowd simulation often rely on simple models such as fluid dynamics, force-based algorithms or heuristic velocity-based approaches. This results in behaviors that appear

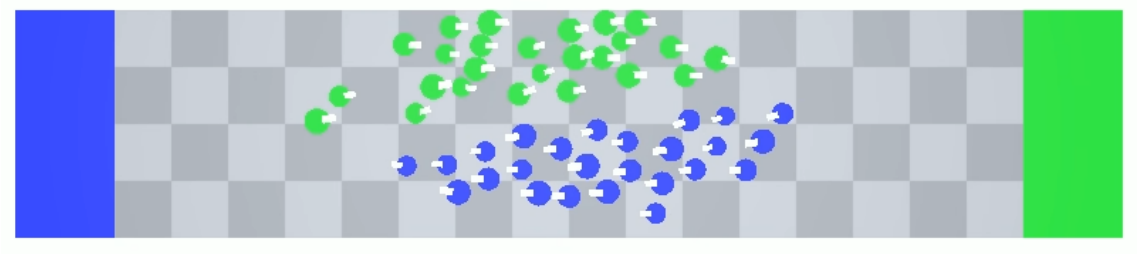


Figure 1.3: A crowd of virtual people in a simple virtual Reinforcement Learning environment.

artificial or robotic, lacking the subtlety and spontaneity of human motion. Moreover, adapting these methods to new scenarios often requires tedious, manual tuning of many parameters defining the simulation.

Reinforcement Learning (RL), an area of machine learning, emerges as a promising tool to address these issues. Its capacity to learn from experience, adjust to new situations, and make optimized decisions aligns well with the challenges of crowd simulation. Through the RL framework, agents can be taught to emulate more human-like behavior, thus enhancing the realism of the crowd simulation.

At its core, RL is a method of learning sequential decision-making through trial and error, where the agent learns how to act in an environment to maximize some notion of cumulative reward. Agents learn to predict the impact of their actions over time, considering both immediate and future outcomes, and then act in such a way that on average, the outcomes are as good as possible. This framework mirrors the process of human decision-making in crowds, making it a suitable choice for trajectory simulation.

Further complexities arise when multiple agents act in a shared environment, leading to the domain of Multiagent Reinforcement Learning (MARL). In MARL, the environment's dynamics become non-stationary from an individual agent's perspective, as other agents constantly modify their policies over the course of the training. Dealing with this non-stationarity is challenging, yet it is essential for simulating dynamic crowd behaviors.

The concept of using RL for crowd simulation is somewhat recent. In prior work, these algorithms have been used to model both individual and crowd behaviors. Each person in the crowd is generally treated as an individual RL agent. They navigate the environment, observing their current state, taking actions, and receiving rewards based on their success in achieving their goals, such as reaching a destination or avoiding collision with others. These agents continuously learn and adjust their policies over time, aiming to optimize their actions for future rewards. When multiple agents are involved, as is the case in crowd simulation, we transition into the realm of MARL, which makes the optimization procedure more complex.

We build upon these ideas, focusing on the fundamental aspects of crowd simulation that often go overlooked. As mentioned earlier, we can view the crowd from a bird's eye view – but should we? Maybe it is beneficial for each agent to perceive its surroundings from its own reference frame? Similarly, should the agents follow dynamics similar to video games, or perhaps more inspired by real-world mechanics? Finally, how should we structure the reward function? Is it really enough to find something that seems to work well enough, and just go with it?

In this thesis, we aim to shed some light on these questions. We believe that before building ambitious, impressive animations, it is important that the RL crowd paradigm stands on a solid

foundation. We hope that this work enables faster and more reproducible work on simulating crowds with RL in the future.

1.1 Objective

In many domains, there is a clear question to be answered. Take for instance object detection in computer vision – it is fairly simple to define what success is, both intuitively and numerically. Even general-purpose RL is fairly well-specified– we assume that an environment is given as part of the problem statement, so obtaining higher average rewards than previous algorithms is a clear example of progress.

However, when exploring RL for crowd simulation, we do not have this luxury, and must instead take a step back to ask "What is the question?" Before building the simulation, we need to choose the right level of abstraction. On one end of the spectrum, we could have a fully physically-simulated world, where each agent acts by contracting its accurate virtual muscles. Would that work? Absolutely. Would that work efficiently, with our current level of knowledge and technology? Absolutely not. On the other side of the spectrum, we could represent the crowd as a liquid, focusing on its statistical properties – an approach that is not uncommon in the wider domain of simulating huge crowds.

After choosing the right abstraction level, we still have to make some decisions on how to setup the simulation. Even with the full physical simulation, while we used an example of muscle contractions, perhaps neuronal activations would be more realistic? Fortunately, this is beyond the scope of this work, but it demonstrates the overall challenge – nothing is as simple or as obvious as it seems, and there are always choices to be made. In this thesis, we seek to make those choices explicit rather than implicit.

1.2 Methodology

This thesis focuses on the intersection of crowd simulation and RL, and as such uses ideas from both of these, largely independent, fields. Throughout the work included here, we use a combination of mathematical, numerical and empirical methods to validate our ideas and findings for how to simulate human-like crowds.

We focus on simulating the time-dependent trajectories of virtual agents. This means that anything below that level – footsteps, body orientation, limb positions etc. are abstracted away and considered outside of the scope of this work. Instead, we train our agents to efficiently and dynamically choose their trajectories (including the pace) as they navigate through their world. The details of effectively using this level of abstraction are the main focus of this thesis.

On the RL side, we use a consistent implementation of the Proximal Policy Optimization (PPO) algorithm, written specifically for this thesis. The algorithm is applied independently to each agent in the multiagent crowd setting, with each agent serving as an unpredictable obstacle for its neighbors. The simulation itself is built in Unity to enable seamless integration with standard animation tooling.

1.3 Thesis structure

The following chapters correspond to papers written during the preparation of this thesis.

Chapter 2 is based on "A Survey on Reinforcement Learning Methods in Character Animation" [84] and provides an overview of applications of RL to character animation as a whole, with special emphasis on crowd simulation.

Chapter 3 is based on "Understanding reinforcement learned crowds" [87] and contains an initial exploration of the basic design choices necessary in any RL-based crowd simulation system, focusing on observation and action spaces.

Chapter 4 is based on "Reward Function Design for Crowd Simulation via Reinforcement Learning" [85] and delves into the details of designing a reward function for navigation, which can be used as an objective for the RL training.

Chapter 5 is based on "UGAE: A Novel Approach to Non-exponential Discounting" [86] and describes theoretical properties of non-exponential reward discounting, along with a practical algorithm to use non-exponential discounting with modern RL algorithms.

Finally, Chapter 6 summarizes the work and concludes the thesis.

1.4 Publications

International Conferences

- A. Kwiatkowski, V. Kalogeiton, J. Pettré, M-P. Cani. **Reward Function Design for Crowd Simulation via Reinforcement Learning** *MIG 2023* [85]

International Journals

- A. Kwiatkowski, V. Kalogeiton, J. Pettré, M-P. Cani. **Understanding reinforcement learned crowds** *Computers & Graphics, Volume 110, 2022* [87]
  (this paper was selected and extended from a conference paper presented at MIG2022)
- A. Kwiatkowski, E. Alvarado, V. Kalogeiton, C. Karen Liu, J. Pettré, M. Van de Panne, M-P. Cani. **A Survey on Reinforcement Learning Methods in Character Animation.** *Eurographics STAR, Computer Graphics Forum 2022* [84]

Workshops and Dissemination

- AAAI-23 Workshop on Multi-Agent Path Finding
- AAAI-22 Doctoral Consortium

Preprints

- A. Kwiatkowski, V. Kalogeiton, J. Pettré, M-P. Cani. **UGAE: A Novel Approach to Non-exponential Discounting** [86]

Software

- **Coltra-RL** A flexible and modular implementation of multiagent PPO [82].
  Available at `github.com/redtachyon/coltra-rl`
- **CrowdAI** Unity-based configurable crowd simulation designed for RL training [83].
  Available at `github.com/redtachyon/CrowdAI`

Chapter 2

State of the Art

This thesis is centered around crowd simulation, primarily seen through the lens of character animation, which in turn is a part of computer graphics. With this perspective in mind, it is worthwhile to be aware of the overall field of character animation in graphics. This is the goal of the following chapter, based on the paper "A survey on reinforcement learning methods in character animation" by Kwiatkowski et al. [84]. We cover the state of the art in using RL for character animation, serving as a foundation for my work.

We begin by providing a general context of the field (Sec. 2.1), followed by some of the most recent challenges in character animation (Sec. 2.1.1). Then, we present the key principles and notations in RL (Sec. 2.2), and continue with a general classification of the most common approaches (Sec. 2.3). Subsequently, we divide the addressed RL solutions into two groups: single-agent (Sec. 2.4) and multi-agent (Sec. 2.5) problems. Then, we describe how these methods are used to solve computer animation problems, for skeletal motion control (Sec. 2.6) and navigation problems (Sec. 2.7), as well as some works concerning interactions between virtual agents and humans (Sec. 2.8). Finally, we present a description of current, available frameworks to apply RL-based solutions (Sec. 2.9), before concluding with a summary of the most relevant algorithms used for a particular problem.

2.1 Introduction

Modern Machine Learning is commonly divided into three categories: Supervised Learning (SL), Unsupervised Learning (UL), and Reinforcement Learning (RL). Supervised Learning refers to learning using data with labels, Unsupervised Learning, including Self-Supervised Learning makes use of raw data without labels, and Reinforcement Learning does not use data in the usual sense. Instead, the learning stage in RL consists of an agent taking a sequence of actions in one or more environments, and trying to maximize a reward function dependent on the states it visits. During this process, the agent progressively trains its own controller module, which in the case of Deep Reinforcement Learning (DRL) is represented by a deep neural network. Once learned, the network can be used in a new, and possibly evolving environment, to make the agent take actions in a successful way towards its goals.

RL stands out as a promising approach for character animation because it provides a versatile framework to learn motor skills without the need of labelled data. RL is particularly useful when the dynamic equations of the environment are unknown or non-differentiable, to which conventional

gradient-based optimal control algorithms do not apply.

Compared to traditional methods in AI, the designer does not need to specify what the character should do in each case – a time-consuming and non scalable method. In contrast, the agent will discover the appropriate actions during the learning stage, given the targeted task or goals expressed in the form of a reward function.

This survey reviews the most common modern **DRL** algorithms, and how they can be used to tackle the main challenges in **character animation**. We consider two main categories of tasks – individual motion skills, and motion planning tasks. Individual scenarios typically involve skeletal motion control of a physically-based character, while motion planning often involves multiple characters interacting in a shared environment. In particular, we focus also on the problem of **crowd simulation**, which focuses on determining the trajectories of multiple agents in a shared environment, often abstracting away their internal structure.

Our work is largely complementary to a recent survey on deep learning for skeleton-based human animation [117], which we also recommend to readers. In particular, we provide a detailed review of current RL methods (both single agent and multiagent) and their mathematical foundations, a full review of RL methods for character navigation methods, and a complementary classification of physics-based character RL methods.

2.1.1 Problems in Character Animation

In the most general sense, the field of **Character Animation** concerns everything related to animating virtual characters. In this work specifically, we focus on the aspects of **behavior** of said agents, on their skeletal motion control, as well as on their **interactions** with a possible human user. Topics related to modeling and animating the character's face, skin, muscles, hair and clothes, or rendering it are out of scope of this report.

When dealing with a single animated character (which may also encompass situations with several independent characters), there are two main levels that need to be considered:

- Skeletal Animation
- Character Motion Planning

Skeletal Animation deals with internal motions of an agent – how the individual limbs move, while the position in the global frame may be of secondary concern. Character Motion is the opposite – it abstracts away the details of the character's shape, instead focusing on its displacements through the scene.

When considering Character Motion for multiple interacting characters, the problem turns into that of Crowd Simulation. Typically, in those problems, each agent has a destination it wants to reach, while avoiding collisions with the environment and with other agents. Van Toll and Pettré [167] wrote an overview of the modern approaches from the last decade.

2.2 Definitions and Preliminaries

In this section, we introduce the basic formal background of Reinforcement Learning. First, we describe and compare different ways of formalizing the RL task to specify **what** we want to solve. Then, we describe the fundamental theorems supporting modern RL methods to show **how** we can solve those tasks.

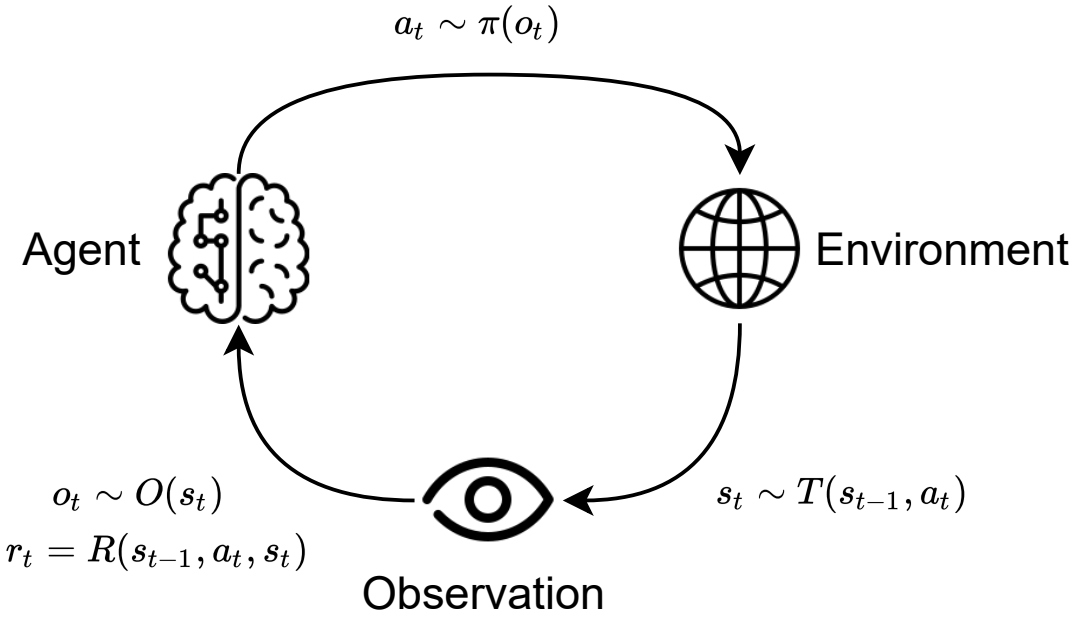


Figure 2.1: A visual depiction of the basic Reinforcement Learning loop corresponding to the POMDP formalism. The agent and the environment exchange information between each other. The agent perceives the environment state and executes an action. The environment then updates its state, and communicates it to the agent via an observation function, together with the reward for the last action.

2.2.1 Reinforcement Learning Formalisms

While there exist several frameworks that are used to formalize the Reinforcement Learning problem, they are based on the Markov Decision Process [14, 159, 160] (MDP), with variations adapting it to the specific task at hand. In this section, we describe the variants relevant to character animation, both for individual agents, as well as multiagent scenarios.

In essence, a Reinforcement Learning problem consists of two parts – an **environment**, and an **agent** acting within that environment in order to achieve some goals. The agent observes the environment, receiving its **state** or **observation**, and based on that executes an **action**. The state of the environment then changes, and the agent receives a reward signal indicating how good that action was. The agent's objective is maximizing the total reward collected during an episode. An episode starts from an initial state, and lasts until the agent reaches a terminal state, or the environment terminates otherwise (e.g. due to a time limit). A schematic representation of this loop is in Figure 2.1.

Single Agent

A general **Markov Decision Process** (**MDP**) is defined by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, R, \mu)$, optionally with a sixth component γ (which can also appear in all other formalisms, and hence will be omitted from their descriptions), where:

- $\mathcal{S}$ is a set of states of the environment.
- $\mathcal{A}$ is a set of actions available to the agent.
- $T\colon \mathcal{S} \times \mathcal{A} \to \Delta\mathcal{S}$ is the environment transition function, representing its dynamics.

- R is the reward function, defined either as $\mathcal{S} \to \mathbb{R}$, $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$, or $\mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, which defines the agent's task.
- $R\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function which is used to define the agent's task.
- $\mu \in \Delta\mathcal{S}$ is the initial state distribution.
- $\gamma \in [0, 1]$ is the discount factor.

Note that we use the notation ΔX to represent for the set of all probability distributions over the set X.

During an episode, an initial state $s_0 \in \mathcal{S}$ is sampled from μ. The state is typically represented by a continuous vector in $\mathbb{R}^n$, or in simple cases, a discrete value. After which the agent repeatedly selects an action a_t from $\mathcal{A}$, observes a new state $s_{t+1} \sim T(s_t, a_t)$ and receives a reward $r_t = R(s_t, a_t, s_{t+1})$. The actions, similarly to observations, are typically continuous vectors or discrete values, although more complex nested structures are also used. This can repeat infinitely, or until some termination condition, defined either by a terminal state in $\mathcal{S}$, or a time limit. The agent's objective is maximizing the **total discounted reward** $\sum_t \gamma^t r_t$, or simply non-discounted **total reward** $\sum_t r_t$ when $\gamma = 1$.

The solution to an MDP is defined as an **optimal policy**, typically denoted as $\pi^*\colon \mathcal{S} \to \Delta\mathcal{A}$. It is the policy that, when executed, leads to the highest expected total discounted reward. While a policy may be stochastic or deterministic, depending on the properties of the action distributions it outputs, note that the optimal policy is generally stochastic, i.e. it returns a distribution over actions rather than a specific action. For consistency, the notation we use in this work is that the action distribution of a policy π in a given state s is $\pi(s)$, whether that policy is stochastic or not. The action is then sampled from the policy $a \sim \pi(s)$. An alternative notation uses the notion of a conditional probability of the action given the current state $\pi(a|s)$, and is equivalent to ours.

A key property of MDPs is their full observability - agents have complete information of the current environment state. This is rarely the case in real applications, and thus a **Partially Observable Markov Decision Process** [74] (**POMDP**) is often used instead.

A POMDP is defined by a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, T, R, \Omega, O, \mu)$, where $\mathcal{S}, \mathcal{A}, T, R, \mu$ are defined as in MDPs. Ω is a set of possible observations, and $O\colon S \to \Delta\Omega$ is the observation function mapping states to observations. This time, the agent does not perceive the real state s_t of the environment, but rather the observation $o_t \sim O(s_t)$ which may not contain the full information, hence the partial observability.

Multiagent

While the MDP and POMDP formalisms are sufficient for problems with a single agent, the generalization to multiple agents can be done in different ways depending on the extent of flexibility required for a given application. The most general case is a **Partially Observable Stochastic Game** [50] (**POSG**) which is defined as a tuple $\mathcal{M} = (\mathcal{I}, \mathcal{S}, \{\mathcal{A}_i\}, \{\Omega_i\}, \{O_i\}, T, \{R_i\}, \mu)$, where:

- $\mathcal{I}$ is the finite set of agents, indexed $1, \ldots, n$
- $\mathcal{S}$ is a set of states of the shared environment.
- $\mathcal{A}^i$ is a set of actions available to agent i, and $\mathcal{A} = \times_{i \in \mathcal{I}} \mathcal{A}^i$ is the joint action set.
- Ω^i is the set of observations available to agent i.

- $O^i \colon \mathcal{S} \to \Omega_i$ is the observation function for agent i.
- $T \colon \mathcal{S} \times \mathcal{A} \to \Delta\mathcal{S}$ is the environment transition function, representing its dynamics.
- $R^i \colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function of agent i, which defines the agent's task.
- $\mu \in \Delta\mathcal{S}$ is the initial state distribution.

Similarly to the single-agent scenario, the environment is initialized with a state s_0 sampled from μ. Each agent then receives an observation $o_t^i = O_i(s_t)$ and based on that, chooses an action a_t^i. The environment changes according to the joint action of all agents $a_t = (a_t^1, a_t^2, \ldots, a_t^n)$ generating the new state s_{t+1}, and each of them then receives their rewards $r_t^i = R_i(s_t, a_t, s_{t+1})$ and observations o_{t+1}^i. This repeats until the episode ends. Since each agent receives its own reward, the objective of an agent i is maximizing its total reward $\sum_t r_t^i$.

A special case of POSG is a **Decentralized Markov Decision Process** [17] (**DecPOMDP**) in which all agents work together to optimize a shared reward function $\forall_i R_i = R$. This formalism is suitable for fully cooperative tasks. It is worth noting that any POSG can be converted into a POMDP by setting the reward to be equal to the sum of of individual rewards, but it will not make sense in all POSGs (consider for example any zero-sum game).

An alternative, but equivalent to POSG formulation, is the **Agent Environment Cycle Game** [165] (**AEC**) formalism. As opposed to the previous options, it is more adapted to dealing with environments in which agents do not act simultaneously. Formally, an AEC is defined by a tuple $\mathcal{M} = (\mathcal{I}, \mathcal{S}, \{\mathcal{A}_i\}, \{\Omega_i\}, \{O_i\}, P, \{T_i\}, \{R_i\}, \nu, \mu)$, where $T_i \colon \mathcal{S} \times \mathcal{A}_i \to \mathcal{S}$ is a deterministic agent transition function, $P \colon \mathcal{S} \to \Delta\mathcal{S}$ is the environment transition function, $\nu \colon \mathcal{S} \times \mathcal{I} \times \mathcal{A} \to \Delta\mathcal{I}$ is the *next agent* function which determines which agent will be taking the action next. The other symbols are defined as before, with the exception of $\mathcal{I}$ which now also includes environment itself considered as a separate agent, represented by the symbol 0. Furthermore, $\mathcal{A}$ is now an union of all individual action spaces. All agents, including the environment, take turns taking their actions and modifying the shared state, which enables a greater flexibility compared to the POSG formalism. AEC environments are primarily used in the Petting Zoo framework [164] (see Section 2.9).

In some cases, a more game theory-based approach is useful. The **Extensive Form Game** [88] (**EFG**) formalism is notably used in the OpenSpiel framework. It contains implementations of many board games, which is the context that it excels in. However, it is not very applicable to character animation, and thus we refer the reader to the associated paper for further details on this formalism.

Note: Many details of the described formalisms vary between sources in the ordering of their elements, the size of the tuple, and the signatures of the functions. This does not change the underlying behavior, and we will therefore omit discussing the different descriptions of the same formalism.

Environment design

A crucial element when applying RL to new problems is designing an appropriate environment. This often involves building a simulation that implements the common API of Gym (see Section 2.9.2), since a purely mathematical formulation would quickly become very convoluted in a complex scenario. Note that we omit the transition function from this description, as this is typically part of the underlying simulation, and can therefore be implemented in any way.

The first consideration is the observation space. This is commonly represented as a fixed-size vector space $\mathbb{R}^n$, which can be directly used with regular feed-forward neural networks. More complex nested structures as well as images are also possible, but they require an adaptation in the structure of the policy being learned.

Second comes the action space. Depending on the environment, a common choice is either a vector space $\mathbb{R}^n$, or simply a finite set of actions $|\mathcal{A}| = n < \infty$. While from the point of view of the implementation it is important that the action space remains constant between different states, one can employ invalid action masking to restrict the available actions to a specific subset. Similarly to observations, it is also possible to use nested structures as long as the policy is adapted correspondingly.

Finally, the reward function defines the actual task and guides the agent's behavior. This is often the most critical component to develop, as a misspecified reward function can lead to unexpected and undesirable behaviors. The simplest reward function can be obtained by choosing a goal state, and giving the agent a reward of 1 if it reaches that state, or 0 otherwise. However, this sparse reward tends to make it very difficult for the agent to learn, as it needs to reach it with random exploration to receive any training signal. A common method is then using reward shaping [119] by adding a smaller, dense reward that guides the agent towards the goal. In other cases, there might be a natural dense reward that can be used instead of the sparse one, such as the distance from the goal in environments with relatively simple dynamics.

Summary

Table 2.1: A comparison of different formalisms used to define an RL problem. Legend: × – the property cannot be modelled in this formalism, ∼ – the property can be modelled in this formalism, but is not the intended use or requires extra effort, ✓ – the property can be modelled in this formalism, ★ – this formalism is particularly suitable for this property. Multiagent Cooperative and Competitive refers to the rewards being either shared or zero-sum, respectively. Multiagent Mixed is neither fully cooperative nor competitive. Simultaneous and Turn-based refers to whether all agents take their actions at the same time, or only one agent does.

Property	MDP	POMDP	POSG	DecPOMDP	AEC	EFG
Single Agent	★	★	✓	✓	✓	✓
Multiagent Cooperative	×	×	✓	★	✓	✓
Multiagent Competitive	×	×	✓	×	✓	✓
Multiagent Mixed	×	×	★	×	★	★
Multiagent Simultaneous	×	×	★	★	∼	∼
Multiagent Turn-based	×	×	∼	∼	★	★
Partial Observability	×	★	✓	✓	✓	✓
Full Observability	★	✓	✓	✓	✓	✓

We presented the most commonly used formalisms underlying the RL problem, which serve as a basis for finding ways to solve these tasks. The similarities and differences between them are in Table 2.1. Typically, either MDP or POMDP can be used with a single agent. POMDP offers stronger theoretical justification if the agent does not observe the full environment state, but this high rigor is not always necessary. Instead, MDP is often used due to its simplicity. With multiple agents, POSG is a versatile choice that can work with any scenario. If one needs to put an emphasis on some aspect of the environment, other options are also available. It is worth noting that those formalisms have dynamic programming solutions associated with them for cases with

discrete action and state spaces. This however is impractical in complex scenarios that emerge in character animation, requiring more sophisticated algorithms.

2.2.2 Fundamentals of RL Algorithms

In this section we describe the mathematical theorems underlying the most important RL algorithms used today. Specifically, we show how the **Policy Gradient Theorem** enables directly optimizing a behavior policy function, and the **Bellman Equation** enables learning the expected utilities of actions that the agent can take in a certain state. These will serve as a basis for many modern algorithms, which often combine the two aspects. We use the notation of MDPs described in Section 2.2.1 because they provide sufficient generality. Under partial observability, states are replaced with observations, and multiagent extensions of relevant algorithms are discussed in Section 2.5.

In both cases, modern algorithms use Neural Networks as approximators for the relevant functions. Because a detailed explanation of training neural networks is out of the scope of this work, we refer the readers to e.g. the Deep Learning Book [40] for more information on that topic.

Policy Gradients

The Policy Gradient Theorem is a basis for all **Policy Gradient** (**PG**) algorithms, starting with the seminal REINFORCE algorithm [161]. In the context of deep reinforcement learning, the policy $\pi\colon \mathcal{S} \to \Delta\mathcal{A}$ is represented as a neural network, and its free parameters, e.g., the weights, are optimized using gradient ascent on the total expected reward. In order to do that, we need to find the gradient with respect to the network's weights using a batch of collected experiences. Here we show a proof of the theorem based on that published in OpenAI Spinning Up [2], although other approaches for proving the same result exist [180, 72].

Consider a trajectory in the environment, defined as a sequence of consecutive states and actions taken by the agent, and rewards $\tau = (s_0, a_0, r_0, s_1, a_1, r_1 \dots)$. Given the parametrized policy π_θ, we know that the probability of a trajectory is

$$P(\tau) = \mu(s_0) \prod_t P(s_{t+1}|s_t, a_t)\pi_\theta(a_t|s_t) \tag{2.1}$$

$$\log P(\tau) = \log \mu(s_0) + \sum_t \left(\log P(s_{t+1}|s_t, a_t) + \log \pi_\theta(a_t|s_t)\right) \tag{2.2}$$

and the total reward obtained in the trajectory is $R(\tau) = \sum_t r_t$

Consider now the expectation across all trajectories τ. With the optimization target defined as $J(\theta) = \mathbb{E}_{\tau\sim\pi_\theta} R(\tau)$, using a few calculus transformations, we can express the policy gradient as:

$$\nabla_\theta J(\theta) = \nabla_\theta \mathop{\mathbb{E}}_{\tau\sim\pi_\theta} R(\tau) \tag{2.3}$$

$$= \nabla_\theta \int_\tau P(\tau|\theta)R(\tau) \tag{2.4}$$

$$= \int_\tau \nabla_\theta P(\tau|\theta)R(\tau) \tag{2.5}$$

$$= \int_\tau P(\tau|\theta)\nabla_\theta \log P(\tau|\theta)R(\tau) \tag{2.6}$$

$$= \mathbb{E}\left[\nabla_\theta \log P(\tau|\theta)R(\tau)\right] \tag{2.7}$$

$$= \mathop{\mathbb{E}}_{\tau\sim\pi_\theta}\left[\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t)R(\tau)\right] \tag{2.8}$$

With this, given a batch of trajectories $\mathcal{D}$ collected using the policy we are optimizing, we can finally compute the gradient estimate:

$$\hat{g} = \frac{1}{|\mathcal{D}|}\sum_{\tau\in\mathcal{D}}\sum_t \nabla_\theta \log \pi_\theta(a_t|s_t)R(\tau) \tag{2.9}$$

Note that this is merely the base form of the theorem, and various modifications are possible, most notably in the form of **importance sampling** [140], or adding a baseline to the reward $R(\tau)$. Some of these are discussed in the context of specific algorithms that use them in Section 2.4.

Bellman Equation

The Bellman Equation [15] is a basis for all value-based algorithms. Unlike the Policy Gradient method, here we do not learn a policy directly. Instead, we try to approximate a state value function $V(s)$ or a state-action value function $Q(s,a)$. The former estimates the expected reward that the agent will collect in the future, given that it is present in a given state s. The latter estimates the same quantity, but given that the agent will take the specific action a in the state s. Then, we use these functions to generate a policy by choosing the best action in a given state. With a state value function V, this requires access to the environment transition function, which is not necessary with a state-action value, where the policy is simply given by $a = \arg\max_{a'} Q(s,a')$.

The value function Q^π (or analogously V^π) associated with a policy π represents the expected total reward if the agents is in a given state s, takes a certain action a ($a \sim \pi(s)$ for the state value function), and then proceeds by following the policy π for the rest of the episode.

$$V^\pi(s) = \mathop{\mathbb{E}}_{a_t\sim\pi}\left[\sum_t \gamma^t r_t | s_0 = s\right] \tag{2.10}$$

$$Q^\pi(s,a) = \mathop{\mathbb{E}}_{a_t\sim\pi}\left[\sum_t \gamma^t r_t | s_0 = s, a_0 = a\right] \tag{2.11}$$

The Q (or V) values of different state-action pairs (states) are obviously not independent – they are in fact related via the transition function, which determines what state comes after them. This is formalized by the Bellman Equation, which defines the consistency criterion of a Q (or V) function (Equations 2.12, 2.14), and the optimal function Q^* (or V^*) (Equations 2.13, 2.15):

$$V^{\pi}(s) = \mathop{\mathbb{E}}_{\substack{a\sim\pi \\ s'\sim T}} [R(s,a) + \gamma V^{\pi}(s')] \tag{2.12}$$

$$V^{*}(s) = \max_{a} \mathop{\mathbb{E}}_{s'\sim T} [R(s,a) + \gamma V^{*}(s')] \tag{2.13}$$

$$Q^{\pi}(s,a) = \mathop{\mathbb{E}}_{s'\sim T} \left[R(s,a) + \gamma \mathop{\mathbb{E}}_{a'\sim\pi} Q^{\pi}(s',a') \right] \tag{2.14}$$

$$Q^{*}(s,a) = \mathop{\mathbb{E}}_{s'\sim T} \left[R(s,a) + \gamma \max_{a'} Q^{*}(s',a') \right] \tag{2.15}$$

The intuition behind these equations is that the value of a state is equal to the instant reward obtained at that state, and the discounted expected value of the following state – which also includes the value of the state after that (due to the recursive nature of the equation), and so on until a terminal state. The value of a terminal state is typically considered to be 0, however a different convention may be used in certain cases, e.g. if the episode timed out. It also induces a dynamic programming solution of MDPs through the Value Iteration algorithm [160]. It is however inapplicable or impractical for many modern problems with complex state and action spaces, and instead, the Bellman Equation is used as the source of a differentiable loss function for value-based algorithms, as we describe in detail in Section 2.4.

It is worth noting that by using a Q function estimator $\hat{Q}^{\pi}$, we can obtain an alternative formulation of the Policy Gradient Theorem. Indeed, as shown by Sutton et al. [160], we get the following expression for the policy gradient:

$$\hat{g} = \sum_{s} d^{\pi}(s) \sum_{a} \nabla\pi(a|s) \hat{Q}^{\pi}(s,a) \tag{2.16}$$

where $d^{\pi}(s)$ is the marginal state distribution under the policy π. This formulation does not use individual transitions, but instead relies on statistics of the policy's performance, and can thus be used as an alternative algorithm to estimate the policy gradient.

2.2.3 Reward Hypothesis, Discounting, Advantage

It is worth taking a closer look at the assumption underlying all Reinforcement Learning research, sometimes called the **Reward Hypothesis**. It is formulated by Richard Sutton as "That all of what we mean by goals and purposes can be well thought of as maximization of the expected value of the cumulative sum of a received scalar signal (reward)" [160]. This is reflected in the described formalisms and equations by the inclusion of a reward function R, with the goal of agents being maximization of the total reward obtained over their lifetime. Some argue that just the reward signal is sufficient to represent any goals that intelligent agents might have [150], while others point out that certain objectives cannot be represented with a single scalar reward [5]. That being said, as we focus specifically on Reinforcement Learning in this work, we do not consider alternative formulations – but it is possible that they will become more relevant in the coming years as the field continues to develop.

An important element related to the reward function is the **discount factor** mentioned in the description of an MDP in Section 2.2.1. It can be considered either as a property of the environment, or the learning agent, and while the two views are mostly equivalent from the optimization

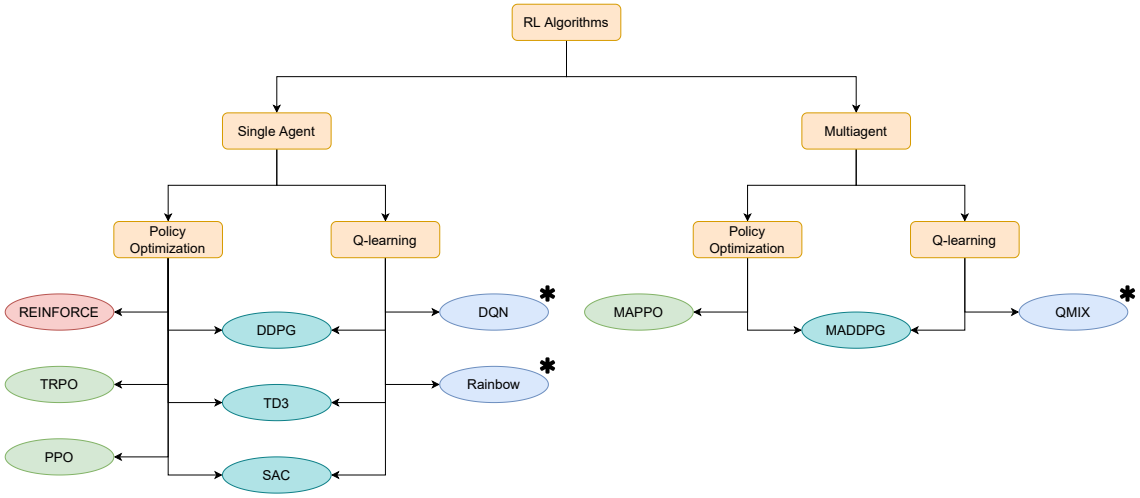


Figure 2.2: A diagram showing a taxonomy of the Reinforcement Learning algorithms described in this work. We focus on two divisions: single agent or multiagent, and policy-based or value-based. The colors of nodes correspond to whether the algorithm is on-policy (red), off-policy (blue), or in between (green). Algorithms marked with an asterisk (∗) can only be used with discrete action spaces.

point of view, they have potential implications relating to the Value Alignment problem [152]. If we consider the discount factor to be a property of the MDP, this is the real reward we want the agent to optimize, whereas otherwise, we really want to optimize the total reward, and discounting the rewards helps improve the training in some way, e.g. as a form of regularization [8]. It can also impact the range of methods that we can use – when considered as a part of the learning agent, any arbitrary method of reward discounting can be used, including non-exponential methods such as hyperbolic [36] or truncated [89] discounting.

One issue with using the raw rewards/utility for training is that it is an absolute metric, with no a priori point of reference. If the agent only perceives a single timestep where a certain action a_0 leads to a reward of -1, this action's probability will be decreased as the value is negative. However, it could still be the optimal action if the counterfactual rewards due to taking other actions are even lower. Asymptotically, this is all balanced out due to the fact that decreasing the probabilities of other actions will necessarily increase the probability of a_0. To decrease the variance of gradient estimation, some algorithms use the notion of **Advantage** instead. This often results in more stable and efficient training. Intuitively, advantage measures how much a certain action is better (or worse) than expected. Given both the Q and V function approximations, we define the advantage as:

$$A(s,a) = Q(s,a) - V(s) \tag{2.17}$$

In practice, algorithms that use advantage often compute $Q(s,a)$ from collected experience, i.e. , look at the trajectory and compute the total reward, while $V(s)$ is approximated with a separate neural network. Examples of this are included in Section 2.4.

2.3 Classification of RL Algorithms

In this section we describe the main categories of modern RL algorithms. While the division is not clear-cut and many algorithms at least draw on ideas from other types, we nevertheless consider this classification to be useful for building an intuition of the RL algorithm landscape. A diagram classifying the algorithms described in this work is in Figure 2.2. The details of these algorithms are provided in Sections 2.4 and 2.5.

2.3.1 Policy-based or Value-based

The first axis of division is whether the algorithm is **policy-based (PB)** or **value-based (VB)**. Although the state-of-the-art algorithm often use both components via Actor-Critic architectures, oftentimes they still have one part that is dominant in the overall picture. The difference between PB and VB algorithms is in what the model is actually trained to predict. In pure PB algorithms such as REINFORCE [180, 161], the neural network is trained to directly output the **action** that will maximize the expected future reward. On the other hand, pure VB algorithms like Deep Q Learn ing (DQN) [113] train the network to instead output the **value** of each action in a given state, that is the expected future reward. This works in environments with a discrete action space, because a policy can then be generated by taking the action with the maximum expected value.

2.3.2 Actor-Critic

Very commonly, RL algorithms use the so-called Actor-Critic architecture, which involves training two networks. One, the **Actor**, also called the **policy**, is responsible for predicting the action that the agent should take, as in PB algorithms. The other, the **Critic**, is responsible for predicting the **value** of an action in a given state, as in VB algorithms. The outputs of the two networks, while not always in agreement with each other, can be used to improve the training process in ways that depend on the exact algorithm – for example, by using the value prediction as a baseline for advantage estimation as in PPO [144], or by training the Actor to find the action with the highest value predicted by the Critic in order to use value-based methods in continuous action spaces as in DDPG [99].

2.3.3 On-policy or Off-policy

Another point of difference between RL algorithms is the data used for the optimization process, which does not necessarily have to be obtained with the same policy that is being learned. We normally refer to the **target policy** as the policy that is being optimized and will be used for evaluation, and the **behaviour policy** as the policy used by the agent to select actions and explore the environment. In **on-policy** algorithms like REINFORCE, the neural network can only be trained using data collected with the policy that is being optimized, meaning that the behavior policy matches the target policy. This implies that after performing a single gradient update, the data (in theory) has to be discarded. On the other hand, in **off-policy** algorithms like DQN, any data (trajectories) can be used, regardless of how it was generated (target and behaviour policies can be different). Some algorithms like PPO toe the line between being on-policy and off-policy, by allowing a relatively small number of gradient updates before the data has to be discarded by using tricks like importance sampling and clipping the loss function. Nevertheless, these algorithms are

typically considered to be on-policy, as they cannot use data collected by an arbitrary behavior policy.

Typically, on-policy algorithms use a **rollout buffer** which stores the environment transitions collected with the current policy, and is emptied after performing the gradient update. Off-policy algorithms instead use an **experience replay buffer**, which stores older transitions, replacing the oldest ones once it reaches maximum capacity.

2.3.4 Model-free or Model-based

This division relies on whether or not the learning agent has access to a model of the environment $T(s, a)$. In **Model-free approaches** like DQN, PPO or DDPG, the agent learns in a true trial-and-error fashion – it has no way of "knowing" the consequences of an action until it tries it, and observes the outcome. On the other hand, **Model-based approaches** additionally learn a model of the environment, allowing the algorithm to do something akin to traditional planning algorithms by considering potential future states and actions, without actually having to execute them in the environment. This is famously present in the AlphaZero [149] algorithm that achieved superhuman performance in the game of Go, where one of the components is the Model-based Monte Carlo Tree Search (MCTS) [30]. While Model-based approaches can provide an advantage in planning terms, the effectiveness of the agent will be limited by the quality of the learned model, which can be negatively affected if the environment is very complex, which is often the case in character animation. This is not the case with Model-free approaches, which do not require an accurate characterisation of the environment to be effective, although they lack the ability to explicitly foresee future states and actions. In this work, we focus on model-free algorithms due to their relevance to character animation.

2.3.5 Single-agent or Multiagent

Finally, an algorithm can be designed to work with either one agent, or multiple agents sharing the same environment. While most of RL development focuses on single-agent algorithms, those can be extended to become multiagent algorithms through Independent Learning (see Section 2.5.1). In competitive multiagent scenarios, algorithms typically use the concept of self-play, training against (possibly old) copies of themselves so that they can be robust when matched with a wide range of opponents. In cooperative scenarios, a common trend is introducing some type of centralization of information so that the agents can coordinate more effectively.

2.3.6 Summary

Looking at modern RL algorithms, it is difficult to cleanly separate them into different categories. Many of the most successful approaches combine different concepts, resulting in an algorithm that is, technically speaking, actor-critic and off-policy. That being said, if we are content with the definitions being fuzzy, we can still gain useful insights about the differences between them.

Typically, value-based algorithms are also off-policy, and enjoy higher sample efficiency compared to policy-based ones. This is because any environment transition, once generated, can be used in perpetuity in multiple gradient updates. Conversely, policy-based algorithms like PPO make it possible to perform fewer gradient updates, because they involve optimizing the objective function directly through gradient ascent. This indicates that value-based methods can be better

when it is difficult to obtain additional data, whereas policy-based methods can often be trained with smaller hardware needs, as they require fewer network updates.

2.4 Single-agent RL Algorithms

In this section we provide descriptions of the most important modern RL algorithms. Due to the large quantity of different methods that appeared in the recent years, this is not meant to be a comprehensive list of all algorithms that could be applied in character animation, but rather the ones with the most relevance, either to this application in specific, or for the field in general. We also provide a sufficient amount of detail for the reader to grasp the main ideas of the algorithms, but refer them to the source papers for the remaining information. Specifically, we do not aim to include sufficient information that would make it possible to reimplement the algorithms without referring to the main paper or existing implementation, as that tends to be a very complex process, with many details being important.

2.4.1 DQN

The first algorithm we discuss is **Deep Q Network (DQN)** [113], which gained prominence when it was used to master a suite of Atari games, achieving superhuman performance in some of them, drawing significant attention to the field. It is a prime example of a Value-based, Off-policy algorithm, and is remarkably simple in its basic form, allowing for a plethora of modifications which we discuss further in this section. DQN is a modern version of the older Q-Learning algorithm [176] which relies on the same principles, but only works on tabular domains (i.e. with a finite number of states and actions).

In DQN, the agent is defined by a state-action value function $Q(s, a)$, represented with a neural network, which is then trained to approximate the real optimal Q function of the environment. This is achieved by performing gradient descent on a Bellman loss function, defined as

$$\mathcal{L}_i(\theta_i) = \mathop{\mathbb{E}}_{s,a\sim\rho(\cdot)} \left[(y_i - Q(s, a; \theta_i))^2\right] \tag{2.18}$$

$$y_i = \mathop{\mathbb{E}}_{\substack{s,a\sim\rho(\cdot)\\ s'\sim T(s,a)}} \left[R(s, a) + \gamma \max_{a'} Q(s', a'; \theta_{i-1})\right] \tag{2.19}$$

where i is the current training iteration, θ_i are the weights of the neural network, and ρ is the probability distribution over state-action sequences according to the behavior policy. Intuitively, the network is trained in a way similar to supervised learning, with the target being the empirical Q value of a given state-action pair, obtained by executing the policy and estimating the future utility using the same current Q function estimate. Typically, an automatic differentiation software is used to find the gradient of the loss function with respect to the network weights θ, leading to the actual parameter update proportional to $\nabla_\theta \mathcal{L}(\theta)$

In order to ensure sufficient exploration, DQN uses ϵ-greedy sampling. This means that given a value of $\epsilon \in [0, 1]$, then while collecting data for optimization, the agent will choose a random action with a probability of ϵ, and the optimal action (according to the current Q function estimate) with a probability of $1 - \epsilon$. Commonly, ϵ is treated as a constant during a single training iteration, and progressively reduced to 0 as the training proceeds.

DQN also uses a replay buffer – the collected data is stored and reused throughout the training, which is possible because DQN is an Off-policy algorithm. So the general flow of the algorithm is as follows. First, collect a batch of data using the current behavior policy (defined by the weights θ_i and some value of ϵ), and add that data to the persistent replay buffer. Then, sample some data from the replay buffer, and perform gradient updates according to Equation 2.18. Repeat this process, updating the weights and decreasing ϵ until convergence.

A crucial limitation of the DQN algorithm lies in the max operator of Equation 2.19. With a discrete action space, finding the optimal action is easy – simply evaluate the function for each action, and then choose the best one. However, when dealing with continuous action spaces, this turns into a potentially non-trivial and nonlinear optimization problem, which in unfeasible to solve each time the agent needs to choose an action, which means that effectively, DQN is limited only to discrete action spaces. This can be avoided by changing the action space through discretization, or changing the algorithm (see Section 2.4.8).

2.4.2 Rainbow

Over the last few years, many modifications of the core DQN algorithm have been developed, aiming at various improvements to its performance. Six of them were combined in the **Rainbow** [57] algorithm:

1. Double Q-Learning [171]
2. Prioritized Experience Replay [142]
3. Dueling Networks [175]
4. Multi-step Learning [158]
5. Distributional RL [13]
6. Noisy Nets [37]

The main ideas of them are as follows. **Double Q-Learning** trains two neural networks, decoupling the action selection from evaluation, in order to mitigate the problem of the learned Q networks overestimating the utilities. **Prioritized Experience Replay** changes the way in which old experience is sampled to optimize the Q network, so that more informative samples (i.e. ones with large updates) occur more frequently. **Dueling Networks** have two computation streams, one for the value, and one for advantage, with some of the weights shared between them. **Multi-step Learning** involves a different way of bootstrapping the future rewards, by looking a few steps ahead (as opposed to just one). **Distributional RL** has the algorithm learn to predict the distribution of rewards, as opposed to just the mean reward itself. Finally, **Noisy Nets** improve exploration by using partially stochastic linear layers. For further details on each of these modifications, we refer the reader to the relevant papers.

Overall, Rainbow agents generally train faster and reach a higher performance than the baseline DQN agents. This comes at the cost of implementation complexity, with only some of the standard frameworks supporting it (see Section 2.9), whereas DQN is very common, and relatively easy to implement in its basic form even for beginners.

2.4.3 REINFORCE

Similarly to how DQN is the simplest Value-based algorithm, **REINFORCE** [180, 161] is the original Policy-based method that is used with neural networks as function approximators. In its simplest form, it is a direct implementation of the Policy Gradient Theorem (see Section 2.2.2). It involves training a neural network to directly approximate the optimal stochastic policy $\pi\colon \mathcal{S} \to \Delta\mathcal{A}$, so that the expected total reward is maximized. This process is performed in an On-policy manner, with a fundamentally simple basic training loop:

1. Execute the policy and collect a batch of experience.
2. Perform a single gradient update of the policy and discard the data.
3. Repeat (1) and (2) until convergence.

REINFORCE can employ some improvements to a naive implementation of the Policy Gradient Theorem. Recall the general policy gradient estimate:

$$\hat{g} = \frac{1}{|\mathcal{D}|} \sum_{\tau \in \mathcal{D}} \sum_{t} \nabla_\theta \log \pi_\theta(a_t|s_t) R(\tau) \tag{2.20}$$

While the reward $R(\tau)$ is computed for the entire trajectory, it is reasonable that when considering the action at a step $t_0 > 0$, we disregard the rewards obtained before, i.e. for $t < t_0$, since the action at t_0 could not have affected them. Furthermore, subtracting a state-dependent baseline from the reward does not change its expectation, which means we can use the **advantage** instead, as defined in Section 2.2.3. This is useful as it decreases the variance of the gradient estimation, leading to a faster and more stable training procedure.

The policy trained by REINFORCE is stochastic, which means that it outputs a distribution over actions $\Delta\mathcal{A}$ rather than a single action. During training, the agent samples an action from the distribution in accordance with the policy gradient theorem. During deployment, it might be desirable to use the deterministic optimal action (i.e. $\max_{a \in \mathcal{A}} \pi_\theta(\cdot|s)$) for improved stability and predictability of the agent. Typically, a stochastic policy with continuous actions is modeled by a Normal (or Multivariate Normal for multidimensional action spaces) distribution. The neural network then outputs the mean action μ, and the variance σ^2 under the assumption that the individual components of the action vector are uncorrelated. Alternatively, a global, state-independent variance can be maintained in the model, and adjusted during the training. In the case of discrete actions, the policy uses a Categorical distribution, with the neural network outputs corresponding to their logits. Mixed action spaces are also possible, and can be modeled as joint distributions.

REINFORCE, as well as the algorithms based on it, can be trained as Actor-Critic algorithms. The Actor is the policy network π_θ which is responsible for the actual decision making, while the Critic V_θ is trained using regular supervised learning techniques, and is responsible for the value estimation in computing the advantage.

2.4.4 TRPO

Trust Region Policy Optimization (TRPO) [143] is based on REINFORCE combined with the notion of a Natural Policy Gradient [75]. It aims to improve the amount of utility that the agent can obtain from a single batch of data. Recall that REINFORCE can only perform a single gradient update with a batch of data, usually with a constant or decaying learning rate. If the

learning rate is too large, a small change in the policy weights can have a large impact on the behavior of the agent, making it difficult to tune while still maintaining good training efficiency.

In TRPO, there are several approximations that deviate from the theoretically-justified REINFORCE algorithm, but instead enable better practical performance. The key idea is the **trust region**, which corresponds to a constraint on the allowed KL divergence between policies in consecutive training steps. The general (theoretical) TRPO update in the training step $k+1$ is:

$$\theta_{k+1} = \arg\max_{\theta} \mathcal{L}(\theta_k, \theta) \tag{2.21}$$

$$\text{s.t. } \bar{D}_{KL}(\theta||\theta_k) < \delta \tag{2.22}$$

where δ is a hyperparameter defining the size of the trust region, and $\mathcal{L}$ is the surrogate advantage:

$$\mathcal{L}(\theta_k, \theta) = \mathop{\mathbb{E}}_{s,a\sim\pi_{\theta_k}} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_k}(a|s)} A(s,a) \right] \tag{2.23}$$

which measures how the new policy performs compared to the old one. The most important feature of this approach is that theoretically, the KL divergence constraint ensures monotonic improvements with a sufficiently small δ, while still being more sample-efficient than REINFORCE.

Due to the arg max operator in Equation 2.21, each step is a constrained optimization problem, which is infeasible to solve hundreds or thousands of times throughout the training. For this reason, the actual algorithm uses additional approximations, resulting in an efficient, but complex Policy Gradient method. Due to this complexity, as well as the fact that other methods can be applied on minibatches of data and are more efficient (see: PPO, Section 2.4.5), TRPO is rarely used in practice.

2.4.5 PPO

Proximal Policy Optimization (PPO) [144] is the successor to TRPO, which through additional simplifications and approximations achieves comparable performance, but with a significantly simpler implementation. It is the de facto standard Policy Gradient algorithm at the moment, and is supported by all major libraries.

Its core idea is to take several gradient update steps with an importance sampling term, without making the policy deviate too far from the original behavior policy. There are two main variants of PPO: PPO-Clip and PPO-Penalty. The former introduces a clipping term to the relative action probabilities in order to disincentivize large policy changes, as measured by KL divergence. The latter adds a penalty term to the loss function for the same effect. In practice, the PPO-Clip variant is more commonly used. Their respective loss functions are as follows:

$$L^{CLIP}(\theta) = \mathbb{E}\left[\min(r_t(\theta)A_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)A_t)\right] \tag{2.24}$$

$$L^{KLPEN}(\theta) = \mathbb{E}\left[r_t(\theta)A_t - \beta \mathrm{KL}[\pi_{\theta_{old}}(\cdot|s_t), \pi_\theta(\cdot|s_t)]\right] \tag{2.25}$$

where ϵ is a hyperparameter, $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio of the action, and β is a coefficient which is adaptively adjusted during the training (if using the Penalty variant).

PPO typically uses an entropy bonus to improve exploration. This means that there is an

additional term in the loss function proportional to the entropy of the policy π_θ, resulting in the policy maintaining some randomness, even at the cost of efficiency.

PPO is an Actor-Critic algorithm, with the Critic being responsible for value estimation that is then used to compute the advantages. The Critic network is typically trained by performing gradient descent on a Mean Square Error loss function between its outputs, and the empirical returns observed in the collected data.

While PPO is typically considered as an On-Policy algorithm, that is not entirely accurate. A single PPO update typically involves several gradient updates, often performed on minibatches of experience, after which the data is discarded as is the case in REINFORCE. This means that while the data can be reused, it can only be done in a very limited way, unlike typical Off-Policy algorithms.

It is worth noting that policy-gradient algorithms (REINFORCE, TRPO, PPO), tend to be sensitive to the implementation details which we omit from this survey. This phenomenon is analyzed in three large-scale studies, to which we refer interested readers: [55, 35, 9].

2.4.6 A3C, A2C

The **Asynchronous Advantage Actor Critic (A3C)** [114] algorithm, and its synchronous equivalent **Advantage Actor Critic (A2C)** [183] are largely of historical value now. The key idea of A3C is using multiple parallel copies of the environment, from which the data can be collected asynchronously, without needing to synchronize them between episodes, or between individual steps. This is meant to improve training efficiency by eliminating the time when an individual worker has to wait for the main process to collect their experience and perform a gradient update.

When researchers continued working with A3C, they discovered that the asynchrony was not a necessary component, but rather an implementation detail, so they developed a simplified, synchronous version named A2C. This algorithm, in its essence, is very similar to REINFORCE with specific details such as using multiple parallel copies of the environment, and using a learned baseline for advantage estimation (which is not the original intent of REINFORCE, but is nevertheless an option in it).

2.4.7 GAE

While it is not a Reinforcement Learning algorithm in the same sense that DQN and PPO are, **Generalized Advantage Estimation (GAE)** [145] is a method that can be applied to any algorithms which use the notion of advantage. It is heavily based on the concept of TD-lambda [158], and can be seen as its extension using Advantages. In the simplest sense, given a trajectory with rewards r_t and a value estimation at each step V_t, we define the **Monte Carlo** advantage as:

$$A_t = \sum_i \gamma^i r_{t+i} - V_t \tag{2.26}$$

which is to say that we compute the expected total reward obtained by the agent, and subtract its estimated value. To use this expression directly, we need a full episode, which in certain environments might be infeasible or inefficient. Furthermore, as the sum of rewards depends on many decisions that the agent has yet to take in the future, the variance of this advantage estimation tends to be very large.

An alternative way is using **Temporal Difference (TD)** estimation by bootstrapping the expected returns, using the value function itself. Like before, given the rewards r_t and value estimations V_t, we define the TD advantage, or one-step advantage, as:

$$A_t^{(1)} = r_t + \gamma V_{t+1} - V_t \tag{2.27}$$

With an unbiased value estimator, the expected value of this expression is the same as Equation 2.26. At the same time, the variance can be significantly lower due to the lack of direct dependence on future rewards. With a biased value estimate, this becomes an example of the classic bias-variance trade-off, prevalent in Machine Learning.

Notice that intermediate n-step advantages can be defined by simply delaying the bootstrapping:

$$A_t^{(2)} = r_t + \gamma r_{t+1} + \gamma^2 V_{t+2} - V_t \tag{2.28}$$

$$A_t^{(n)} = \sum_{i=0}^{n-1} \left[\gamma^i r_{t+i}\right] + \gamma^n V_{t+n} - V_t \tag{2.29}$$

which introduces a wide range of possible advantage estimation methods. What GAE proposes is using all n-step advantage estimates, weighted exponentially with a factor of $\lambda \in [0, 1]$:

$$A_t^{GAE} = (1 - \lambda)(A_t^1 + \lambda A_t^2 + \lambda^2 A_t^3 + \dots) \tag{2.30}$$

This turns out to have a simple analytic expression that can be computed with a single pass algorithm. Empirically, GAE often noticeably improves the performance of RL algorithms, and is the de facto standard for advantage estimation in Actor-Critic algorithms.

2.4.8 DDPG

An algorithm on the boundary between Value-based and Policy-based methods is the **Deep Deterministic Policy Gradient (DDPG)** [99]. It is based on the notion of a Deterministic Policy Gradient [148], which is the gradient of a state-action value function with respect to the action. It can also be seen as an adaptation of the DQN algorithm to continuous action spaces.

In DDPG, we train two separate networks – a state-action value network $Q_\phi \colon \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, and a (deterministic) policy network $\pi_\theta \colon \mathcal{S} \to \mathcal{A}$. The value network is trained in a way similar to DQN, with some tricks such as using a replay buffer and a target network to stabilize the training. The key difference lies in the max operator of Equation 2.19, which is not trivial with a continuous action space. This is where we use the second, policy network, trained to predict the optimal action according to the reward function. The Q network is optimized to minimize the following loss functions:

$$L(\phi) = \mathbb{E}\left[\left(Q_\phi(s, a) - y(r, s', d)\right)^2\right] \tag{2.31}$$

$$y(r, s', d) = \left(r + \gamma(1 - d) \max_{a'} Q_\phi(s', a')\right) \tag{2.32}$$

where (s, a, r, s', d) are the transitions in the replay buffer, with s, s' being the current and next

state, a the action that was taken, r the reward, and d is equal to 1 if the state was terminal, and 0 otherwise. Then, the policy is optimized using gradient ascent to maximize the following objective:

$$L(\theta) = \mathbb{E}\left[Q_\phi(s, \mu_\theta(s))\right] \tag{2.33}$$

This can then be differentiated using the chain rule, giving the policy gradient of:

$$\nabla_\theta L(\theta) = (\nabla_a Q_\phi(s, a)) \cdot (\nabla_\theta \mu_\theta(s)) \tag{2.34}$$

Overall, DDPG can be seen as the simplest way of adapting DQN to continuous action spaces, without having to discretize the action space. Because it is off-policy, it can be more sample-efficient than competing on-policy algorithms, making it suitable for environments in which it is difficult to collect large amounts of data. However, its asymptotic performance is often worse than that of competing on-policy algorithms like PPO, which leads to its limited practical use in character animation.

2.4.9 TD3

Twin Delayed DDPG (TD3) [38] is to DDPG what Rainbow is to DQN – it introduces a series of tricks that significantly improve the algorithm's performance. The main changes are as follows:

1. Clipped Double Q-Learning
2. Delayed Policy Updates
3. Target Policy Smoothing

Clipped Double Q-Learning works similarly to Double Q-Learning described in Rainbow (Section 2.4.2, using the smaller value of the two networks' outputs to prevent value overestimation. **Delayed Policy Updates** involves performing policy updates less frequently than Q function updates. Finally, with **Target Policy Smoothing**, noise is added to the target action, so that it is more difficult for the policy network to exploit errors in the Q function.

2.4.10 SAC

Soft Actor-Critic (SAC) [47] is in many ways similar to TD3, in that it is a modification of DDPG with certain changes introduced in order to improve its performance. Primarily, it uses entropy regularization by adding a term proportional to the policy's entropy to its optimization objective. This encourages the policy to remain stochastic, increasing exploration. Similarly to TD3, it uses Clipped Double Q-Learning, minimizing the Bellman loss of DDPG. However, there is no explicit policy smoothing, as SAC trains a stochastic policy instead of a deterministic one. As a result, the additional regularization is unnecessary, since actions are sampled from a nontrivial distribution.

Specifically, SAC learns three functions in parallel: the policy π_θ, and two Q functions Q_{ϕ_1}, Q_{ϕ_2}, with the usual double Q-learning approach. Since they are trained on an entropy-regularized objective, the Q function optimization objective takes the following form:

$$L(\phi) = \mathbb{E}\left[\left(Q_{\phi_i}(s, a) - y(r, s', d)\right)^2\right] \tag{2.35}$$

$$y(r, s', d) = r + \gamma(1 - d) \left(\min_{j=1,2} Q_{\phi_j}(s', \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s') \right) \tag{2.36}$$

where $\tilde{a}' \sim \pi_\theta(s')$, and $\alpha > 0$ is the entropy regularization coefficient. Notice the similarity to Equations 2.31 and 2.32 of DDPG, with the key difference being that the objective now has a term proportional to the entropy of the action distribution $\alpha \log \pi_\theta(\tilde{a}'|s')$, and the action used for computing the Q value of the following step is taken directly from the behavior policy.

When it comes to policy learning, as SAC learns a stochastic policy, it must output a distribution over the action space. The optimization takes the following form:

$$L(\theta) = \mathbb{E} \left[\min_{j=1,2} Q_{\phi_j}(s, \tilde{a}') - \alpha \log \pi_\theta(\tilde{a}'|s) \right] \tag{2.37}$$

where $\tilde{a}' \sim \pi_\theta(s')$. Notice again the similarity to Equation 2.33, which confirms that SAC is, in its essence, an updated and improved version of DDPG.

It is important to keep in mind that while this is a general outline of the algorithm, there are many details that can significantly affect its performance. For more information on this, we refer the reader to the original paper, as well as the existing open-source implementations (Section 2.9.3).

2.4.11 Learning from Data

As a general rule, Reinforcement Learning does not need expert data to train agents, instead using an environment that the agent can interact with. In some cases, however, it may be beneficial to use expert data to augment the learning process, or even eliminate the use of a simulation whatsoever. This is often referred to as **Imitation Learning**, because the agent learns to imitate the actions of an expert whose experience is shown to it.

Behavior Cloning (BC) [12, 139, 32] is the simplest way to perform Imitation Learning. Its core idea is to treat Imitation Learning as a supervised learning problem, which given a dataset consisting of observations and actions, learns to map the former to the latter by training a classifier or a regressor.

By including a model training phase in which the agent can interact with the environment, we can remove the requirement that the dataset contains the actions [168]. This significantly simplifies the data required to perform imitation learning, and enables learning by simply observing someone, much like humans do in the real world. However, the quality of the resulting behaviors is typically lower due to the fact that the dynamics model is only trained with on-policy data, which means that out-of-distribution errors are likely to occur. For this reason, if the data about actions is available, it is better to use it instead of relying only on observations.

The **Generative Adversarial Imitation Learning (GAIL)** [60] algorithm represents the main alternative to Behavior Cloning. It relies on the concept of **Inverse Reinforcement Learning (IRL)** [196], which means learning the reward function from demonstrations (as opposed to regular RL, where the agent learns a policy, or generates demonstrations, given the reward function). This, combined with the notion of adversarial learning known from **Generative Adversarial Networks (GAN)** [41], and a PG-based update rule (originally TRPO) produces an efficient algorithm for Imitation Learning.

A common practice is using Imitation Learning methods in conjunction with standard, reward-based RL algorithms [39]. This can be done by including a term derived from Imitation Learning

either in the reward function, indirectly encouraging the agent to act similarly to the data, or by including it directly in the optimization objective.

2.4.12 Summary

We described the most noteworthy RL algorithms used in single-agent environments. From a practical point of view, we recommend either using the on-policy **PPO** with GAE for advantage estimation, or the off-policy **SAC**, which are the most popular algorithms of their respective categories. If the training data is difficult to obtain, SAC is typically better as it can reuse the data enabling higher sample efficiency. On the other hand, PPO often offers faster training in terms of the wall time by using parallelism in data collection and larger performance improvements per gradient update. If working with discrete actions, Rainbow or another version of DQN is also a viable choice. Finally, if one wants to incorporate real data in the training process, both BC and GAIL are strong options and can be integrated with other algorithms.

2.5 Multiagent RL Algorithms

Here, we describe the algorithms that are adapted specifically for multiagent environments. Those are typically based on existing single-agent algorithms, with modifications that improve the training process by abusing the specific multiagent structure of the problem.

2.5.1 Independent Learning

Any single-agent algorithm can be used in a multiagent scenario by using **Independent Learning**, with the resulting algorithms typically called **IPPO**, **IDDPG** etc. This entails treating the other agents as parts of the environment, possibly including information about them in the observation, and then simply training as if it were a single-agent task. A simple way to accelerate this training process when all agents are identical is treating them as **homogeneous**, also called **Parameter Sharing** [163]. With this approach, every agent receives their own observation and takes their own action, but they share the underlying neural network, and their experience can be combined for the training. Otherwise, if each agent has its own separately trained neural network, it is referred to as **heterogeneous**. It is possible to introduce some degree of heterogeneity by including an agent indicator in the agent's observations, as shown by Gupta et al. [44].

2.5.2 MADDPG

MultiAgent DDPG (MADDPG) [107] is an extension of the DDPG algorithm to explicitly use the structure of multiagent environments in the training procedure. It relies on the idea of **Centralized Training, Decentralized Execution (CDTE)**, which means that the algorithm can use global or hidden information, as long as the resulting agent only needs access to its own observations.

In multiagent environments, there are two main pieces of information that is not available during execution – other agents' observations (or the global state), and the actions they take. However, when training in a simulation that we have total control over, these quantities are readily available, and so can be used in an Actor-Critic paradigm to optimize the Critic. Then, in the execution phase, only the agent's local observation is necessary for the Actor network to choose the action.

2.5.3 MAPPO

MultiAgent PPO (MAPPO) [189] is the result of extending the PPO algorithm analogously to the difference between DDPG and MADDPG. Because PPO is an Actor-Critic algorithm, the Critic similarly use centralized information such as other agents' observations and actions, while only the Actor is actually involved in the decision making during evaluation.

Since the concept and the name of MAPPO is generic, there are other works that introduce a similar extension [43, 101, 64]. The details are different between those papers, but in the most robustly evaluated version of it uses the following five tricks:

1. Value normalization through a running mean, for robustness with respect to the reward scale
2. Value function input includes both global and agent-specific features, pruned to reduce the input dimensionality
3. Data is not split into minibatches, and the algorithm uses relatively few training epochs
4. The clipping factor is tuned as a trade-off between training stability and fast convergence
5. Using death masking (inputs for dead or deactivated agents) through zero states with agent ID

The resulting algorithm delivers results comparable with more sophisticated off-policy algorithms, while being viable to train using a single machine with one GPU.

2.5.4 QMIX

QMIX [136] and its derivatives are a family of algorithms that adapt Q-learning in cooperative scenarios, so that it can use centralized training, while maintaining the option to perform decentralized execution. The core idea is that the joint state-action value is a monotonic function of the state-action values of each individual agent.

Consider the two extremes in terms of centralizing Q value estimation. On one hand, we have fully independent Q learning, where each agent optimizes their own reward, which can be a viable option as described in Section 2.5.1. On the other hand, we can also consider fully centralized Q learning, with a single network processing all agents' observations, and outputting their joint action. A simple middle ground can be found in Value Decomposition Networks (VDN) [157], where a joint Q function is expressed as a simple sum of the agent's individual Q functions:

$$Q_{tot}(s, a) = \sum_i Q_i(s^i, a^i) \tag{2.38}$$

QMIX introduces additional flexibility to this approach. It replaces the summation operator with an arbitrary function of the individual values, with the only restriction being that it is monotonic with respect to all its inputs:

$$\frac{\partial Q_{tot}}{\partial Q_i} \geq 0, \forall i \in \mathcal{I} \tag{2.39}$$

This is obtained by using a **mixing network** to represent Q_{tot}. The weights of the mixing network are the outputs of a set of hypernetwork [46] conditioned on the environment state. This whole setup can be trained with significant information sharing between the cooperating agents, while in the execution phase, each agent only requires its own Q function Q_i.

Due to the popularity and effectiveness of QMIX, researchers have developed various modifications aimed at improving its performance even further [195, 173, 187, 137, 153]. However, recent work suggests that using regular QMIX with appropriate implementation details is enough to achieve results comparable or even superior to the more complicated algorithms [62].

2.5.5 Summary

When working with multiagent problems (e.g. crowd simulation), we typically recommend using one of the single-agent algorithms and applying it with an **Independent Learning** approach as a starting point, with either **IPPO** or **ISAC** following the notation from Section 2.5.1, as well as **Parameter Sharing**. This is significantly simpler in implementation than using algorithms that introduce centralized communication, and can often yield competitive results. While adding some additional communication or centralization may be beneficial, MADDPG tends to be difficult to train in new environments.

2.6 Skeletal Animation

Individual characters can be animated using kinematic or physics-based methods. For the former case, the action space directly consists of kinematic poses or existing motion clips, and are defined based on motion capture data. In contrast, physics-based methods have action spaces that directly or indirectly produce joint torques that drive the motion. In this section, we first provide an abridged overview of RL as applied to kinematic methods. We then shift our focus to physics-based methods. This begins with a general summary of the many nuances involved when using RL to control physics-based character movement, given that the default motions produced by RL algorithms for humanoid characters in the RL literature are usually of low quality as compared to what is needed for computer animation applications. We then categorize and review many of the recent methods and results for RL-based physics-driven character animation.

2.6.1 RL for Kinematic Motion Synthesis

RL has a long-standing history of being used to learn kinematic controllers from motion capture data. Here we provide a brief overview of work in this direction. Motion generation can be framed as an RL problem where actions correspond to the choice of motion clips, as first applied to automatically-constructed graphs [11, 80, 91] and then in ways that were better tailored to locomotion tasks, e.g., [90, 170]. Lee et al. [96] later introduced the concept of continuous motion fields in support of a data-driven state-dynamics model. Optimal actions on this model are then learned via a table-based representations for the policy and value function. Modern motion matching methods can be seen as a short-horizon version of motion-fields. Ling et al. [100] learn a latent action space using autoregressive variational autoencoders to define character controllers and thereby enabling optimal goal-based animations.

2.6.2 The Many Challenges Beyond the Choice of Algorithm

A considerable amount of thought is typically required to define a character movement task, articularly in a physics-based setting. This begins with the design of the character, which involves making decisions related to joint torque limits, contact friction, mass distribution, joint limits,

joint damping, simulation and control time steps, and more. The choice of action space can also have implications for the learned results. Available options include joint torques, joint PD-target angles, joint PD-target angle offsets from an available reference motion, muscle-based activations, or more abstract actions for hierarchical control approaches. It is also sometimes possible to learn simplified actions spaces that avoid redundancies or that sample from a reduced-dimensionality action manifold, which can possibly be learned as well. The definition of the state of a character that is provided to the policy can also have a significant impact. The pose can be represented as Cartesian joint locations, or in a more traditional form consisting of a root position and orientation, followed by a set of internal joint angles. Contact information can also be an important part of the state.

Next, the task rewards need to be designed, which may need to balance generic and possibly temporally-sparse rewards related to the goals, rewards that encourage energy-efficient behavior, and shaping rewards that help guide the solution in what can otherwise be an exceedingly-large search space. Rewards also tend to work better when mapped to a fixed range, as commonly done using a negative exponential. Episode termination criteria are also important, as they effectively constrain the search space and, by virtue of providing no further rewards, also provide an implicit negative task reward. Reward terms can be combined, using a weighted addition, e.g., [125] or in a multiplicative fashion, e.g., [120], and these choices can strongly impact the final learned policies.

The optimization criteria to define a natural human or animal motion are difficult to determine, and thus a natural alternative is to instead seek to imitate motion capture data, either as individual motion sequences, or as distributions using adversarial approaches. The choice of initial states for a task is important, as it can affect the task difficulty [138], and can also simplify the learning, as in the case of a motion imitation task where the initial states can be drawn from the given reference trajectory [126]. Curriculum-driven learning can enable an easy-to-difficult learning order for a task [184]. Policies can be "warm-started" from existing solutions. Prior knowledge should be used where possible to set the relevant variances and exploration rewards. External forces can also be allowed early on in the optimization [190], and then slowly withdrawn. Hierarchical learning can also be leveraged, by first learning low-level control that operates at a fine time scale, followed by higher-level control that allows for long-horizon tasks [125].

The algorithms themselves are challenging to work with, with a typical improve-and-test debugging iteration requiring between hours and days, depending on the task difficulty and the availability of compute. In many cases, wall-clock time is a more important consideration than sample-complexity, and algorithms whose common implementations support a high-degree of parallelization, e.g., PPO, are then sometimes preferred over that are more difficult to parallelize, e.g., SAC. Tuning the algorithm hyper-parameters plays an important role in the learning efficiency and success, and may require grid search or other hyper-parameter optimizations. The results of model-based trajectory optimization can be used to guide policies towards suitable solutions for difficult tasks. Debugging RL tasks is also an important skill, and points to initially working with simplified or more-constrained systems, visualizing reward terms, understanding the limitations of physics engines, and much more. More specific algorithmic features to consider include the use of a Huber loss instead of the conventional quadratic loss for Q-learning, considering various forms of conservative Q-learning, choice of temporal-difference horizon, and more.

Many simulated robotic control environments are standard benchmarks for RL algorithms. MuJoCo [166] and PyBullet [31], two of the most commonly used physics simulation engines in RL, provide several robot models with a Gym [19] interface. These robots range from abstract

ones like Hopper or Reacher, through animal-like Ant and HalfCheetah, to more human-like ones like Humanoid and Walker2d. While not realistic, they share many of the principles of skeletal character animation.

We next review work that uses reinforcement learning to develop a variety of full-body motion skills for physics-based characters. These leverage many of the insights described above.

2.6.3 RL for Individual Character Skills

For the remainder of this section, we further categorize methods into: (a) those which use motion capture data, typically as a key part of the imitation objective, and (b) methods that use a more general "pure" learning objective. In both cases, there exists a variety of prior art that is entirely model-based or uses other optimization methods. However, for our purposes here, we restrict ourselves to methods that use reinforcement learning for motion imitation.

Motion imitation RL methods

One of the first RL methods to be able to successfully imitate motion capture data, including highly dynamic motions such as flips, uses data from a stochastic planning method, first developed as an open-loop trajectory optimization method [104]. Building on this type of method, the work of [105] proposed to use data from multiple runs of the stochastic trajectory optimizer to then learn a state-conditioned feedback policy. The desired motion sequence is divided into a sequence of 0.1 s duration control fragments, and for each such fragment computes a multivariate linear regression of the actions with respect to the state. This yields a simple linear policy for actions as a function of the state, for the duration of the control fragment. This model is then able to robustly imitate walking, running, spin-kicks, and flips, as well as transitions. Further work has then shown how learned control fragments can be treated as abstract actions, which can be resequenced using deep Q-learning [102], and can further be adapted to learn basketball playing skills [103].

The use of policy gradient RL methods to imitate human motion capture clips was first explored by Peng et al. [125] for a variety of walking gaits. This also introduced a hierarchical reinforcement learning approach, with a low-level policy first being trained to reach target stepping locations while also striving to imitate the reference motion. A high-level policy then operates once for every walking step, generating step targets in support of tasks, including control of a ball with the feet, navigating paths, and avoiding dynamic obstacles. Peng et al. [126] further develop the imitation learning approach to train controller for a diverse set of motions, including highly-dynamic spin kicks and flips for humanoids, sequencing such motions, and using the same imitation approach for quadruped controllers. Imitation-based learning of a wider variety of quadruped gaits, including sharp turns, is demonstrated in [128], along with successful transfer to quadruped robots. Peng et al. [129] use ideas from adversarial imitation learning by combining a reward function to control the high-level behaviors, with low-level controls specified with an unstructured dataset of motion clips. This method can be used on both humanoid and non-humanoid models. It produces high-quality animations that match tracking-based methods, but the training process can still be prone to mode collapse, as is common in GAN-like algorithms. Some of these examples in Imitation Learning are shown in Figure 2.3. The choice of action space is also shown to have an impact the speed and quality of imitation-based learning [122].

Computer vision based pose tracking can also be used as a source of motions to imitate, allowing robust control policies to be learned from video clips [127]. Isogawa et al. [69] construct an end-

to-end pipeline that converts Non-Line-Of-Sight measurements to 3D human pose estimation by employing a diverse set of techniques, including an RL-based humanoid control policy. Yuan et al. [192] introduce the SimPoE framework, which trains an RL agent to control a physics-based character to estimate plausible human motion, while conditioning it on a monocular video.

The majority of the works described above develop control policies that only reproduce single clips, or a specific set of motion clips. The motion to imitate plays a role via the reward, but is not provided to the policy as an input. The policies are conditioned on a time or motion phase. An important next step has been to reproduce a richer variety of motions by conditioning the policy on a short time window of the future motion to imitate. This can also be seen as a generalized form of learned inverse dynamics, with a longer anticipatory window as needed to make motion corrections for more difficult motions. Chentanez et al. [24] first develop this type of conditioning and apply it to large motion datasets. Significant further developments follow from improvements that target scalability, motion transitions, motion quality, generalization, and learning efficiency [120, 16, 174, 182]. These methods are further extended to work with muscle-based actuations [94], a large diversity of body shapes [181], and producing large motion variations even from a single motion clip [95]. Other work shows how to allow for more flexible forms of imitation [109], and that leverage residual external forces to enable learning more challenging motions [191]. Imitation-based controllers can also be used to learn a latent human-like action space via distillation ("neural probabilistic motor primitives"), which can then be used as an abstract action space for new tasks [112]. Similarly, Luo et al. [108] learn a natural action distributions from reference motions for quadrupeds, while a GAN-based controller reproduces suitable actions based on user-input. This is followed by high-level DRL fine-tuning.

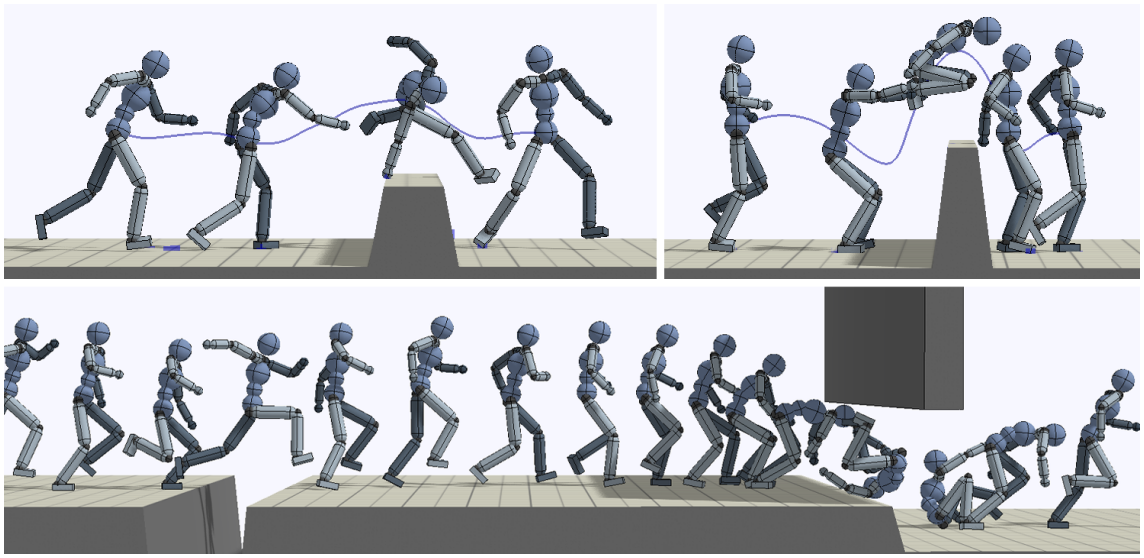


Figure 2.3: Imitation-based Learning. Proposed methods as in [126] allow to successfully synthesize animations from motion capture data. In other works, as in [129], they combine such techniques with the possibility of adding low-level behaviours to control the production of high-complexity animations.

Pure objective RL methods

Reinforcement learning has also been successfully used for full-body character animation without an imitation objective. Here, the objective can be framed in terms of rewards that include energy,

progress towards a goal, stylistic hints, and regularization terms.

Model-predictive control (MPC) methods, which iteratively re-plan and then execute the first action, have been successfully employed for humanoid animation and are a form of model-based RL. The work of Tassa et al. [162] demonstrated the online use of iLQG (Iterative Linear Quadratic Gaussian) trajectory optimization for online control of humanoid characters for a variety of tasks, including getting up, using a 0.5 s planning time horizon. Sampling-based methods can also be used to achieve trajectory optimization over a finite planning horizon, and have been explored in detail by Hämäläinen et al. [48, 49]. Online trajectory optimization and policy learning can also be used in a mutually supportive fashion [135], with the policy serving to accelerate the trajectory optimization, and the trajectory optimization helping to bootstrap the policy learning. In addition, trajectory optimization can benefit from more complex search spaces, for instance by including contact points [115] to improve simultaneously both, trajectory and policy learning.

Actor-critic methods for RL can more easily tackle motion tasks, such as locomotion, by being provided with task-specific action abstractions. For example, the action space can consist of a discrete set of existing controllers, with a high-level actor-critic controller being trained to make a discrete selection among the set of available controllers at each step of the gait. This setup is used with a kNN-based value function approximator to achieve high-level objectives by Coros et al. [29]. An abstracted, tailored action space is used by Peng et al. [123] to include a continuous action space as defined by a conveniently-parameterized finite-state machine controller. A kNN-based actor-critic pair is then used to train dog-like and bipedal models to traverse variable terrain. Later, Peng et al. [124] develop a mixture-of-experts based Actor-Critic algorithm named MACE for improved performance on a similar dynamic locomotion task, this time using deep neural networks for the actors and critics, and thereby eliminating some of the feature engineering required by the previous approach.

Can policy-gradient RL algorithms be used with pure learning objectives to generate natural human movement, as opposed to the unrealistic frenetic motions commonly seen resulting from popular RL benchmarks? Yu et al. [190] encourage symmetric and low-energy motions by appropriately modifying the loss function of the algorithm, by adding the so called mirror-symmetry Loss to the usual surrogate loss of PPO. This allows for high-quality motions without using any imitation of motion examples. This is particularly important for non-humanoid characters for which there is no motion data available. Example non-humanoid models that can be trained this way are in Figure 2.4. Abdolhosseini et al. [1] further investigate multiple methods of incorporating symmetry constraints for skeletal animation tasks, inspired by the observation that human and animal gaits found in nature are typically symmetrical. They use the PPO algorithm with four options of enforcing symmetry on the learned policy. They show that this can in fact be harmful to the training process, but in the end can produce higher quality motions. Xie et al. [184] explore a curriculum-based learning solution to train characters to walk and run over a wide range of stepping stones, with varying step heights, lengths, yaw angles, and step pitches, in the absence of an imitation objective. PPO is used in conjunction with a parameterized generator of individual steps, and the learning curricula advance the step difficulty in several different ways. PPO is used to train the physical legged model.

To further improve on the realism, more biomechanically accurate models can be considered. The NeurIPS conference hosted three challenges using the osim-rl platform (see Section 2.9.2): "Learning to Run" (2017), "AI for Prosthetics" (2018) and "Learn to Move - Walk Around" (2019) [78], all of which dealt with different aspects of controlling a human body model. The lead-

ing solutions used learned models of the environment, and off-policy algorithms such as DDPG. Jiang et al. [70] continue this research direction by using another human body model based on the OpenSim [146] platform. With muscles outnumbering joints, the larger action space of biomechanical models can be significantly more expensive to train. This paper addresses this by allowing the optimization to nevertheless operate in joint actuation space, as afforded by two neural networks that model the state-dependent torque limits and the metabolic energy. The overall approach is agnostic to the choice of RL algorithm.

Non-locomotion tasks are also important for full-body character animation. Kumar et al. [81] use an algorithm based on MACE for the task of teaching a virtual humanoid model to safely fall by minimizing the maximal impulse experienced by its body. They train a mixture of actor-critic networks associated with all possible contacting body parts, and further use a form of hierarchical reinforcement learning, with an abstract policy deciding the high-level behavior, and a joint policy responsible for actually executing the action. Clegg et al. [28] consider the problem of simulating the movement of a human dressing themselves using a combination of physics simulation and RL. They use a virtual human-like model and is tasked with putting on one of three different pieces of clothing. The process of getting dressed is divided into several subtasks, each of which is treated as a Reinforcement Learning problem with an appropriate reward function. Subtasks are trained using TRPO, and then sequenced together to produce a full motion. Yin et al. [188] explore the problem of learning diverse jumping motions, including high jumps. For a given takeoff state, a curriculum is used to learn a policy for increasing bar heights. The space of takeoff states is then explored using Bayesian diversity search, to synthesize a diverse set of jumping styles, including jump well-known techniques such as the Fosbury flop.

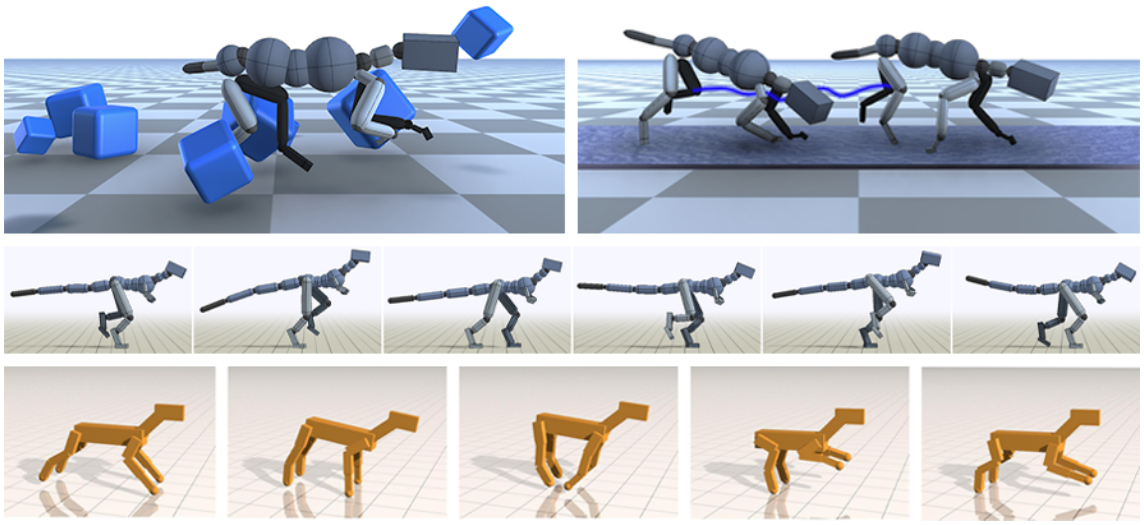


Figure 2.4: There is a wide variety of methods that also address the synthesis of animations for quadrupedal or arbitrary morphology [108] [126] [190]. While the limited amount of motion capture data introduces an additional challenge, such methods try to overcome this constraint by covering a wide range of techniques, from imitation-based approaches to pure objective RL.

2.7 Crowd Animation

In this section we discuss the specific work that used RL algorithms in Crowd Animation scenarios, as well as the challenges that make this task distinct from single-agent cases. Unlike Skeletal Animation, Crowd Animation typically uses multiagent algorithms so that each individual agent has access to its own information, but not necessarily to the global state. While the task can often be seen as fully cooperative (e.g. making a realistic simulation), this enables more realistic behaviors, and not using a centralized controller enables easier scaling to different numbers of agents. We focus on applications pertaining to the challenge of multiple agents navigating in a shared environment, and omit the discussion of more advanced topics around coordination and division of tasks, considering them to be out of scope of this work.

2.7.1 Challenges of Crowds

The key factor distinguishing cooperative multiagent learning from single agent RL is the phenomenon of nonstationarity. Typically, RL algorithms assume that the environment is stationary, which means that the environment dynamics (represented by the transition function) remain the same throughout the training. That is not true in multiagent training as observed by a single agent – as other agents learn, their policies change, which affects the perceived environment dynamics.

In the case of Crowd Simulation specifically, there is also the question on how exactly to represent the physics of the problem. While some works use holonomic cartesian controls in which each agent can move in any direction, this is not entirely realistic, and instead, polar controls may be used, where an agent decides its linear motion and turning left or right. Furthermore, the agents may either control their velocities directly, or apply accelerations to their motion. While these approaches can be seen as nearly equivalent, it is still necessary to choose one, which may impact the final performance in nontrivial ways.

In certain cases, competitive and general-sum scenarios may be relevant, which carries additional complications. Most notably, evaluation of trained systems is challenging in the absence of an expert model or an external performance measure. This is because the typical training paradigm relies on self-play, and a winrate against a copy of itself cannot be reliably translated to objective performance. Furthermore, the details of the self-play procedure can also impact the training. Finally, as training progresses, agents might learn to specialize to play against specific strategies, forgetting about their older versions, and underperforming when matched up against them.

Finally, it is worth mentioning that many of the challenges in Skeletal Animation, still apply here. Depending on the physical model of the agent behaviors and interactions, the exact choice of the actuator may be important for the learned policies. Similarly, designing the reward function is crucial for good performance – a reward that is too sparse may be prohibitively difficult, whereas one that is adapted to be more dense, may lead to unexpected behaviors. Finally, it is often desirable to have agents exhibit human-like behavior, but this task in itself is not well-defined, and it may be helpful to use real-world data in order to generate a specific reward function.

2.7.2 Applications

Long et al. [106] apply an RL-based approach to the task of collision avoidance. While this is not exactly the same thing as character crowd animation, collision avoidance is nevertheless a

significant component of crowd simulation systems. They train a policy which receives as input a depth map, the goal's relative position and the current velocity, on the task of reaching the goal and avoiding collisions with other agents in the shared environment. Because they also deploy them on real robots, there is additional emphasis on avoiding collisions, with contact between two agents leading to removing both of them from operation with a large negative reward. They train the policy using the PPO algorithm, and show that this method results in higher success rates and more efficient policies compared to ORCA. With a similar approach, Lee et al. [93] apply the DDPG algorithm to the task of basic crowd simulation – a number of agents in a shared environment move through it, and attempt to reach their respective destination. They use polar dynamics, with the RL agent setting a linear velocity and a rotation at each timestep. The agents receive a positive reward for getting closer to their goals, a negative reward when they collide, and there is also a regularizing reward term that encourages smooth movement. They show that this setup is sufficient to obtain agents that reach their goals and avoid collisions, although the results are still imperfect. There is also no regard for human-like behavior.

To explicitly combine ORCA with RL methods on the same standard crowd simulation task, Xu et al. [185] introduce the ORCA-DRL algorithm. They use PPO to predict a preferred velocity at any given step, which is then used as an input to ORCA. This is then responsible for actually avoiding collisions. They show that this approach leads to successful collision avoidance, which is not surprising given its reliance on a classical collision avoidance algorithm. It also does not consider whether the behavior is human-like. Sun et al. [156] use a different approach – they train four "leader" agents responsible for guiding parts of the crowd, while the remaining agents follow their respective leaders. They use PPO with a recurrent LSTM [61] policy and combine it with a classical collision avoidance algorithm RVO. They train the agents to act in an unknown environment with dynamic obstacles. The resulting behaviors manage to achieve their goals, but do not take into consideration whether the behavior is human-like. This method also still relies on classical collision avoidance algorithms.

Haworth et al. [52] introduces a method on the borderline between single character and crowd animation. By employing a method based on the ideas of Hierarchical Reinforcement Learning, they train two policies interacting with one another. The high-level policy is responsible for navigation and reaching global goals. It sets objectives for the low-level policy which directly controls the joints of a humanoid model. They use this on multiple characters in a shared environment, with the low-level policies shared between them. Both policies are trained using PPO, with the low-level policy learning to match motions from a database of stepping actions using a PD controller.

Hüttenrauch et al. [68] introduce a method that, while not directly applied in crowd simulation, is nevertheless very relevant. Their work focuses on swarm systems, which are inherently similar to crowd scenarios – they both focus on a large number of agents acting in a shared environment, often with a shared goal, with the individual agents typically being indistinguishable. They introduce a method called Mean Feature Embedding, similar to existing Relation Networks [141, 194]. This approach uses a modified neural network architecture that ensures permutational invariability between identities of different neighboring agents that are perceived by another agent. This inductive bias can accelerate the training process, and improves scaling to different numbers of agents in the environment. Alonso et al. [7] explore the applicability of RL methods for the task of crowd navigation in AAA games. They use a large and complex 3D environment built on Unity, modelled after real games that typically use a Navigaton Mesh (NavMesh) [151] approach. They use a recurrent LSTM network to give their agents memory, and use the SAC algorithm for policy

optimization. As inputs, the agents receive their absolute positions, relative goal position, their speed and acceleration, as well as 3D occupancy maps obtained via box casts, and depth maps from ray casts. They show that this approach has a high level of success, and can enable more flexible map designs, without requiring the designers to specify each possible link.

Zou et al. [197] consider the problem of understanding and predicting crowd behavior specifically from the perspective of Imitation Learning. They introduce a new framework named Social-Aware Generative Adversarial Imitation Learning (SA-GAIL) which is trained to replicate behavior recorded in demonstrations, while disentangling the different factors of decision-making in pedestrian movement. This allows them to obtain a human-understandable interpretation for the model's predictions, as well as for the real data. They use the TRPO algorithm for policy optimization and show that this approach can produce high-quality, interpretable behaviors. A different approach for using data to obtain more human-like behaviors is used by Xu and Karamouzas [186]. They use the concept of Knowledge Distillation introduced earlier by Hinton et al. [59]. Along with the standard PPO algorithm for policy optimization, they train a neural network in a simple supervised way, mapping observations to actions based on data from a crowd motion dataset. Then, the outputs of this network are used as an additional source of reward for the policy learning, encouraging the agent to act similarly to what the supervised network predicts. This way, they obtained more human-like behaviors on typical crowd simulation scenarios, as compared to a regular RL baseline, without a detriment to their performance.

We summarize the algorithms from the papers listed in Sections 2.6 and 2.7 in Table 2.2. While there is a decent diversity of physics engines, as well as a split between TensorFlow in PyTorch for the neural network optimization, the RL algorithm of choice is predominantly PPO.

2.8 Human Interaction

While not directly a part of Character Animation, interaction between humans and learning-based agents are highly relevant to its applications, notably for Virtual Reality games. For this reason, in this section we describe some of the work towards interactive RL agents. For agents to be interactive, it means that they must be capable of acting in a shared environment with a human-controlled agent, in such a way that they retain their performance on their original goals, while simultaneously reacting to the human's actions appropriately. This is far from trivial, especially when the human behavior significantly differs from the behavior of the trained agents.

An important concept to discuss is the **Theory of Mind (ToM)** [131]. Stemming from developmental psychology, ToM refers to the ability that humans and some other animals possess, of reasoning about the internal state of someone else – their goals and beliefs. As Rabinowitz et al. [132] show, it is also possible to train RL agents so that they can learn ToM of other agents in their environment by observing their actions.

Chodhury et al. [26] consider whether it is worthwhile for an agent to learn a full environment model, to learn a ToM model of the human, as opposed to using a model-free approach, in order to cooperate with a human agent. They use an autonomous driving task and show that general black-box model-based methods can work as well as ToM learning, and both of them outperformed the model-free approach.

Carroll et al. [22] analyze this problem in the general case of cooperative multiagent reinforcement learning, using an environment based on the game Overcook, which requires a high level of cooperation. They find that agents trained in the usual ways, such as with self-play or population-

based training, perform significantly when paired with a human played, as compared to their original group performance. They introduce a method based on training a Behavior Cloning agent on data collected from human gameplay. The policy of this agent is then frozen, and it is used as part of the environment dynamics for the actual agent we want to train. Despite the BC agent's low quality, this turns out to be sufficient to improve the performance of the actual agent when evaluated together with a human player.

Christiano et al. [27] include humans in the loop in the training phase, as opposed to enabling cooperation with them. They introduce a method with which it is not necessary to specify a reward function for an agent to optimize. Instead, the algorithm produces demonstrations which are then judged by the human, allowing it to assign reward values from which it learns. This way, RL agents can learn complex behaviors which are not trivial to define mathematically, instead relying on human preferences, while only requiring human input on about 1% of the actual frames used during training.

2.9 Frameworks

In this section, we discuss the most relevant libraries and frameworks used for training RL agents, which are then applied to character animation. Because the field of Reinforcement Learning relies on neural networks, we begin by describing main frameworks used in Deep Learning. Those are responsible for efficiently performing algebraic operations on tensors (here understood as n-dimensional arrays), using parallelism when possible. They also take care of computing the gradients of functions with the backpropagation algorithm, enabling gradient-based optimization. They do that either on CPU or GPU, sometimes also supporting TPU (Tensor Processing Unit). Then we move on to libraries that function as backbones for RL tasks, by providing standard implementations, offering a common API, or even enabling development of new environments. Finally, we describe the tools used specifically for Reinforcement Learning, either by providing full algorithms, components of them, or other auxiliary functionalities.

All the listed frameworks are written for the Python programming language, although they frequently use other languages for efficient computation that the end user does not need to know. This is the de facto standard in Machine Learning and Reinforcement Learning specifically. Despite its relatively slow performance, through the use of the aforementioned libraries, all the heavy computation is off-loaded to a more efficient language that the end used does not need to use directly.

2.9.1 Neural Networks

While many open-source tensor computation libraries exist today, three in particular stand out. The first is **TensorFlow (TF)** [111] developed by Google – originally released in 2015, it underwent major changes in 2019 when the version 2.0 was released with a new philosophy. For this reason, TF1 and TF2 are mutually incompatible and can be seen as two distinct frameworks. The main difference between them is the default handling of computation graphs. In TF1, we have to explicitly define a graph, and then run it within a session. This means that any intermediate values are hidden from the user, leading to a difficult debugging process. On the other hand, TF2 uses eager evaluation, building an implicit computation graph. This way, the developer can access the values of any tensors at any time, also in interactive mode e.g. in a Jupyter Notebook [79], without

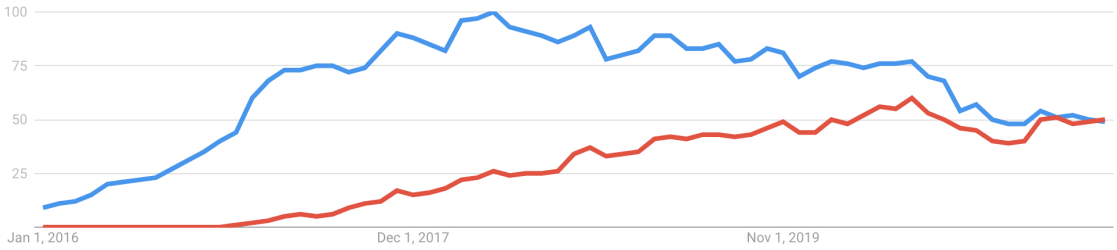


Figure 2.5: The relative popularity of PyTorch (red) and TensorFlow (blue) in terms of search volume on Google according to Google Trends, worldwide, between 01/01/2016 and 27/07/2021.

needing to modify the computation graph for this purpose. TensorFlow can run computations on CPU, GPU (via CUDA and ROCm), and TPU.

It is important to mention **Keras** [25], originally developed as an independent library, is now integrated as part of TF2. It exposes a higher-level API that allows faster development at the cost of fine-tuned control over the computation graph – this, however, can be regained by including lower-level TF2 code as custom layers. While it is primarily designed for Supervised Learning, it is possible to use it with Reinforcement Learning, particularly when using only some of its features in conjunction with TF2 code.

PyTorch [121], developed by Facebook, was released in 2016 as a Python version of the existing library Torch, which used the Lua language. Its design is very similar to NumPy [51] and inspired TF2 in that it uses eagerly-executed tensors that can be used in dynamic computation graphs. PyTorch can run computation on CPU and GPU (via CUDA), and with some extra effort, TPU. Its simplicity of use, combined with performance that matches TensorFlow, led to its widespread usage, particularly in research context [53].

A relatively new framework that is worth mentioning is **Jax** [18]. It offers simple acceleration and parallelization of code with a simple interface that can nearly be used as a drop-in replacement for NumPy. It is heavily inspired by the functional programming paradigm, focuses on composable transformations of functions, notably including differentiating arbitrary functions. By itself, Jax contains efficient numerical operations on tensors, and does not explicitly include neural networks. However, libraries in its ecosystem fill that gap, notably Flax [54] and Haiku [56].

In the words of Andrej Karpathy, "I've been using PyTorch a few months now and I've never felt better. I have more energy. My skin is clearer. My eye sight has improved." [76] This sentiment is also visible in the data. According to the 2020 Stack Overflow Developer Survey [154], while TensorFlow is more commonly used than PyTorch (11.5% vs 4.6%), it has a lower Loved score (65.2% vs 70.5%), and higher Dreaded score (34.8% vs 29.5%). According to Google Trends, despite its later release date, PyTorch matches the popularity of TensorFlow is terms of search volume, as we show on Figure 2.5. Overall, the common perception is that TensorFlow is more suited for deployment and industry applications, while PyTorch can be more effective for research and development.

2.9.2 Environments

The main framework underlying nearly all modern RL research is **Gym** [19], recently succeeded by **Gymnasium** [169]. It contains a set of commonly used environments that serve as benchmarks for RL algorithms, together with a unified Environment API and a way to implement new environments

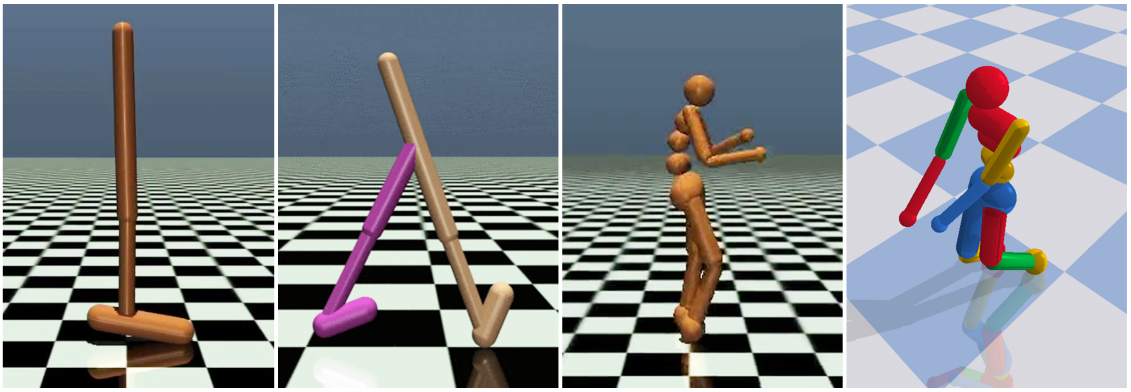


Figure 2.6: A visualization of different legged models of varying complexity. The agents' objective is moving each of the joints so that the overall center of mass moves forward or is balanced, while minimizing the energy expenditure. From left to right: Hopper, Walker2d, Humanoid (MuJoCo) and Humanoid (PyBullet).

as Python classes. This way, a single implementation of the environment can be used with multiple algorithms for easier comparison and benchmarking.

The main parts of the Gym API are the following methods:

- reset() $\rightarrow$ observation
- step(action) $\rightarrow$ (observation, reward, done, info)

where the observation and action can be in any format agreed between the environment and the agent. The observations are usually tensors, and actions are either vectors or discrete values, but more complex, tree-like structures are also sometimes used. The reward is a single scalar value, and done is a binary flag indicating whether an episode has ended. Finally, info is a dictionary containing arbitrary additional information. Other properties like observation_space and action_space exist to specify the structure of the information exchanged with the agent, but the extent to which they are required depends on the specific algorithm implementation.

Canonically, Gym only supports partially observable single-agent scenarios, corresponding to POMDPs. Supporting multiagent environments is possible to an extent without changing the abstractions. Specifically, a DecPOMDP can be represented by taking the state and action spaces to be the product spaces of all agents. The reward can then remain as a single scalar value, shared between the agents, maintaining full compatibility with the Gym API.

The situation is more complicated when dealing with more general multiagent problems like POSGs. In this case, each agent may receive its own reward independently of others, which cannot be represented with a single scalar value. What is more, some agents might not be active throughout the episode. For that reason, frameworks like **RLlib** [97] use a modified version of a gym Environment where everything is based on Python dictionaries – observations, actions, rewards and 'done' values (i.e. boolean episode termination flags) are Python dictionaries, where each agent's respective values are indexed by the name of that agent.

Other general multiagent formalisms also have their corresponding libraries. The **Petting Zoo** [164] library implements the AEC formalism, and is well-suited for general multi-agent problems in which agents do not act simultaneously, but also supports simultaneous actions. Similarly to Gym, it also contains a number of standard benchmark multiagent environments. The EFG

formalism is implemented in the OpenSpiel [88] library, which also contains several ready board and card games.

MuJoCo [166], which stands for Multi-Joint dynamics with Contact, is a physics simulation engine that is widely used in RL research. It can be used to design various robots whose control is then learned with RL algorithms. Certain robots are included in Gym and serve as a common benchmark for new algorithms, notably the Humanoid [162] which is relevant to the topic of character animation. It is worth noting that MuJoCo used to require a paid license to use, but has now become open-source. **PyBullet** [31] and **DART** [92] are common open-source alternatives, filling the same role as a physics simulation engine, and containing many of the same environments ready to use. **NVIDIA Isaac Gym** [98] fulfills a similar role, enabling very fast parallel simulations accelerated with GPUs. Different legged models from MuJoCo and PyBullet can be seen in Figure 2.6.

ML-Agents [73] is a plugin to the Unity game engine which exposes an API through which Python-based agents can interact with games developed in Unity. This can greatly accelerate the development of new environments, as many features from game development are available out-of-the-box, and are relevant for RL tasks. ML-Agents also contains implementations of PPO and SAC algorithms in PyTorch, with the possibility to record and train with user-made demonstrations, using BC or GAIL.

Osim [77], based on the OpenSim framework [146], was used in the 2017 NeurIPS "Learning to Run" challenge is a simulator with a physiologically accurate model of the human body. Its main goal is bridging the gap between biomechanics, neuroscience and computer science communities by providing a common ground for research. It contains a physics simulator, an RL environment, as well as a competition platform to compare different solutions.

Alternatively to gradient-free physical engines, further implementations such as **Nimble** [178] introduce novel ways of fast and complete differentiable rigid bodies simulations, which can be used for hard optimization problems dealing with complex contact geometries or elastic collisions. Differentiable physics [110] and the possible combination with stochastic gradient-free methods show promising directions for research into more efficient physics engines for learning and optimization.

2.9.3 Algorithm implementations

In recent years, many frameworks with implementations of RL algorithms have appeared, many of them including the same algorithms, but with differences in the tricks included, or in the implementation philosophy. A common issue is that a framework is too rigid, and therefore it is difficult to customize it for arbitrary research purposes. Here we describe the most relevant frameworks, with a comparison of the algorithms they feature in Table 2.3

One of the earliest libraries with high-quality implementations of modern RL algorithms is **OpenAI Baselines** [34]. Released in 2017, it contains the implementations of the most important algorithms existing at the time, including DQN, DDPG and PPO, using TensorFlow. However, at the time of writing it is in maintenance mode, which means that it is not updated with the new developments in the field. Furthermore, the implementations are considered to not be very readable and are challenging to modify or update.

For this reason, the **Stable Baselines (SB2)** [58] library was developed, with re-implementations of the same algorithms with an addition of SAC and TD3, with a series of improvements to their usability, including: better documentation, tests, using custom policies, and a shared interface

between all algorithms. A full comparison is included in the official GitHub repository. Stable Baselines uses TensorFlow for its algorithms, but is now in maintenance mode. The more up-to-date version is **Stable Baselines 3 (SB3)** [134] which has the same purpose as SB2, but uses PyTorch instead, and is actively updated with new features. Both SB2 and SB3 have very limited support for multiagent training.

RLlib [97] is built on Ray [116], a platform for parallel computing, and because of that it can achieve high performance through efficiently parallelizing data collection. It has a large selection of both single-agent and multiagent algorithms in both PyTorch and TensorFlow (depending on the algorithm), with a wide variety of options to adjust for those algorithms. The implementations are very efficient, but the code-base is complex, and therefore difficult to understand and modify for most users.

CleanRL [65] is a library with a different design philosophy – all implementations are contained entirely in a single file. This allows for simple customization, and can serve as a reference when re-implementing those algorithms. CleanRL uses PyTorch for its algorithm implementations. It also includes the Open RL Benchmark, which contains reproducible experiments that use the implemented algorithms on a wide range of standard RL environments.

Dopamine [23] is a research framework developed by Google, with the design principles of *easy experimentation*, *flexible development*, *compactness and reliability*, and *reproducibility*. It focuses on variants of DQN, including Rainbow and several other modifications. The framework primarily uses TensorFlow, but it also contains Jax implementations of the algorithms.

TF-Agents [42] is a part of the TensorFlow ecosystem dedicated to Bandits and Reinforcement Learning algorithms. It contains high-quality implementations of modern RL algorithms written in TF2. The implementations are modular and include tests, benchmarks and tutorials on how to use the library.

Tianshou [177] maintained by researchers from Tsinghua University is a modular RL framework based on PyTorch. While focusing on single-agent model-free algorithms, it also supports multiagent environments, model-based algorithms and imitation learning. The implementations are flexible so that it is possible to modify them for research purposes, and very efficient due to parallelization.

Rlax [21], developed by Deepmind as part of the Jax ecosystem, takes a different approach than all the aforementioned frameworks. Instead of full algorithms, it contains building blocks that can then be used in implementing the algorithms. This includes exploration strategies, policies, update strategies and more.

2.9.4 Summary

As we show, there are numerous libraries that can be used for creating environments and training RL algorithms, and the best choice will necessarily depend on the application. Regarding environment creation, highly specialized problems might require custom simulators, but for generic character and crowd animation problems, we recommend using the Unity engine with ML-Agents to build a gym or gym-like interface. For training, if customization of the algorithm is unnecessary, we recommend using either RLlib or Stable Baselines 3 for single-agent (skeletal animation) scenarios. If some degree of customization is necessary, Stable Baselines 3 or Tianshou are worthwhile options, with RLlib typically being too rigid. Finally, if it is necessary to introduce major changes to the typical reinforcement learning loop, it might be necessary to use a custom-written algorithm

- components from Tianshou, CleanRL and Rlax can then prove to be helpful. Naturally, all of this is conditioned on the availability of the desired algorithm in a specific framework (see Table 2.3).

2.10 Conclusions

Reinforcement Learning is a rapidly growing field of Artificial Intelligence and Machine Learning, concerned with authoring intelligent behaviors by specifying their tasks, instead of describing the specific behaviors. This method is of high utility for applications in Character Animation, both for individual characters, as well as entire crowds.

In fact, many works already use RL algorithms to create more believable or higher-quality animations. In many cases, with an appropriate simulator, it is sufficient to specify the desired task in terms of a reward function (e.g. agents in a crowd heading towards a certain goal, while avoiding collisions with one another), and then train one of the state-of-the-art algorithms to obtain interesting behaviors. However, Deep Reinforcement Learning is still a relatively young field, and thus its use is not common in the industry. This is likely to change in the upcoming years.

As for the implementation, there are many resources available to significantly accelerate the development of new applications of RL. While different frameworks excel in different aspects, there exist options for diverse use cases and degrees of complexity, so that in-depth expertise in the inner workings of RL algorithms is not absolutely necessary to be able to apply them.

We anticipate significant progress on the intersection of Character Animation and Reinforcement Learning in the upcoming years. The algorithms become more and more efficient, seeing success after success in classic challenges like the games of Go and Starcraft. This, combined with the widespread usage of GPUs, will make it possible to seamlessly integrate them into typical Computer Graphics workflows.

Table 2.2: A summary of the DRL algorithms, simulation engines, and neural network frameworks in the described papers, where applicable and stated in the paper or the provided source code. [1] Value Iteration, [2] Open Dynamics Engine, [3] Temporal Difference learning, [4] Maximum A Posteriori Policy Optimization.

Citation	Year	Algorithm	Physics Simulation engine	NN Framework
[123]	2015	VI[1]	Box2D	–
[124]	2016	MACE	Bullet	Caffe
[102]	2017	DQN	ODE[2]	Theano
[125]	2017	TD[3]	–	–
[122]	2017	TD	–	–
[81]	2017	MACE	DART	Caffe
[103]	2018	DDPG	ODE	Theano
[126]	2018	PPO	Bullet	TensorFlow
[127]	2018	PPO	Bullet	TensorFlow
[24]	2018	PPO	MuJoCo	TensorFlow
[190]	2018	PPO	DART	TensorFlow
[28]	2018	TRPO	DART	PyTorch
[106]	2018	IPPO	Stage	TensorFlow
[93]	2018	IDDPG	–	TensorFlow
[197]	2018	TRPO + GAIL	–	TensorFlow
[120]	2019	PPO	DART	TensorFlow
[16]	2019	PPO	Bullet	TensorFlow
[94]	2019	PPO	DART	PyTorch
[181]	2019	PPO	DART	TensorFlow
[1]	2019	PPO	Bullet, MuJoCo	PyTorch
[156]	2019	IPPO + RVO	Unity3D	–
[68]	2019	ITRPO	–	TensorFlow
[128]	2020	PPO	Bullet	TensorFlow
[174]	2020	PPO	Flex	-
[182]	2020	PPO	Bullet	TensorFlow
[191]	2020	PPO	MuJoCo	PyTorch
[112]	2020	V-MPO[4]	MuJoCo	NumPy
[108]	2020	PPO	Bullet	TensorFlow
[100]	2020	PPO	Bullet	PyTorch
[69]	2020	PPO	MuJoCo	TensorFlow
[184]	2020	PPO	Bullet	PyTorch
[185]	2020	IPPO + ORCA	–	PyTorch
[52]	2020	IPPO, MADDPG	Bullet	Caffe
[7]	2020	SAC	Unity3D	–
[129]	2021	PPO + GAIL	Bullet	TensorFlow
[95]	2021	PPO	DART	TensorFlow
[109]	2021	PPO	Bullet	PyTorch
[192]	2021	PPO	MuJoCo	PyTorch
[188]	2021	PPO	Bullet	PyTorch
[186]	2021	IPPO	-	PyTorch

Table 2.3: A comparison of algorithm support between various frameworks. Legend: ✓ – algorithm supported by the framework, × – algorithm not supported by the framework. Multiagent refers to the capability of training in multiagent environments, with or without parameter sharing. Note that this is not a complete list of algorithms implemented by each framework, as some of them include many other, less relevant algorithms.

Algorithm	OpenAI Baselines	Stable Baselines	Stable Baselines 3	RLLib	Dopamine
DQN	✓	✓	✓	✓	✓
Rainbow	×	×	×	✓	✓
DDPG	✓	✓	✓	✓	×
TD3	×	✓	✓	✓	×
SAC	×	✓	✓	✓	×
TRPO	✓	✓	×	×	×
PPO	✓	✓	✓	✓	×
QMIX	×	×	×	✓	×
BC/GAIL	✓	✓	×	✓	×
Multiagent	×	×	×	✓	×

Table 2.4: Continuation of Table 2.3.

Algorithm	CleanRL	TF-Agents	Tianshou	ML-Agents
DQN	✓	✓	✓	×
Rainbow	×	×	×	×
DDPG	✓	✓	✓	×
TD3	✓	✓	✓	×
SAC	✓	✓	✓	✓
TRPO	×	×	✓	×
PPO	✓	✓	✓	✓
QMIX	×	×	×	×
BC/GAIL	×	×	✓	✓
Multiagent	×	×	✓	✓

Chapter 3

Reinforcement Learning for Crowd Simulation

When setting up a crowd simulation, or in fact any RL task, it is important to properly design the underlying decision process. A typical abstraction used for RL is a Markov Decision Process [14, 159, 160], although having multiple agents and a partially observable structure of the crowd simulation scenario requires additional nuance. We must choose each element of this process carefully, or else the choices will be made for us – by accident.

Throughout this thesis, we work in the paradigm of microscopic crowd simulation – that is, each agent is simulated separately, with its individual observations, actions and objectives. However, each of these three elements carries additional considerations that must be thought through in order to have a solid foundation. This is exactly the topic of this chapter. How to design observations and actions? Is there even a difference between various options? And if so, what are the best choices?

3.1 Introduction

Simulating virtual human crowds is a common task when creating lively, populated scenes for graphics applications. This includes a large variety of scenarios – ranging from small, artificially structured scenes used in research, to large scale simulations with thousands of virtual agents. They can be used either for real-life applications (e.g. testing evacuation scenarios in airports or sport stadiums) or targeting automatic content creation in films and games, featuring a diversity of background characters, each with their own goals and motivations.

While the approaches to simulating such crowds typically use hand-crafted or data-driven algorithms, in recent years there has been an increasing research interest in using Deep Reinforcement Learning (DRL) methods. These algorithms have a promise of a flexible, problem-agnostic training process that can autonomously produce diverse behaviors. However, they also bring many new challenges, which must be explored independently of the previous knowledge of crowd simulation algorithms.

It is worth noting that just "crowd simulation" is somewhat of an underdefined problem. Although we have an intuition as to what a crowd is and how it behaves, it is not clear how to formalize it, which is exemplified by the variety of descriptions in prior work. At the same time,

as Reinforcement Learning (RL) is designed to optimize a given scalar reward function, it excels when the objective is clearly stated. For this reason, we investigate the various ways to specify the crowd simulation problem.

There are three main components of an RL system – observations, actions, and rewards. For each of them, it is not necessarily obvious what is the appropriate level of abstraction, or even the implementation, to properly simulate human crowds. Take actions, for example – it is infeasible to perform a fully accurate biomechanical simulation of each muscle movement, so we must use a simplified model. Similarly for observations, performing a full rendering of the scene to emulate human vision would not only be expensive to perform, it would also difficult to train.

Designing the reward function is arguably even more complex. While navigating in our daily life, we balance several, often mutually contradictory objectives, such as getting to the destination efficiently, moving at a comfortable speed or avoiding bumping into others. Even though some of them are simple to formalize as a single scalar reward, it is not obvious how they should be balanced. Is it acceptable to reach the destination two seconds earlier, but increase the risk of bumping into someone by 5%? Can we take a shorter path which leads through a group of people, or should we go around, reducing risk of collision, but increasing the distance? The answer to each of this question will vary from person to person, and many external factors. If going a bit slower would cause us to barely miss a train, we are likely to accept a higher risk of colliding than if we are just taking a walk around the park.

In this work, we intend to bring clarity to the intersection of crowd simulation and DRL, by exploring in detail the impact of these design choices on the generated virtual crowds. We perform a theoretical analysis of controlling the agents' velocities, and an empirical investigation of various observation and action spaces. We evaluate them in terms of optimizing the reward function, but also consider the energy expenditure, and various quantitative properties of the movement. Our contributions are:

1. Empirical evaluation of raycasting versus direct agent perception in RL crowd simulation
2. Empirical comparison of various implementations of observations and dynamics in RL crowd simulation
3. Theoretical and empirical analysis of the properties of reward functions for efficient navigation with RL

3.2 Environment Design Choices

We identify three design choices which can impact the properties of virtual crowds trained with a standard DRL algorithm – observations, actions, and the reward function. In this section, we set the problem in standard multiagent RL formalism, and describe the variants of observations and action spaces explored in this work.

3.2.1 Problem Formulation

We model the problem of crowd simulation as a Partially Observable Stochastic Game (POSG) [50]. A POSG is defined as a tuple $(\mathcal{I}, \mathcal{S}, \{\mathcal{A}_i\}, \{\Omega_i\}, \{O_i\}, T, \{R_i\}, \mu)$, where $\mathcal{I}$ is the set of agents, $\mathcal{S}$ is a set of states of the environment, $\mathcal{A}^i$ is a set of actions for agent i ($\mathcal{A} = \times_{i \in \mathcal{I}} \mathcal{A}^i$ is the joint action set), Ω^i is the set of observations, $O^i \colon \mathcal{S} \to \Omega_i$ is the observation function, $O^i \colon \mathcal{S} \to \Omega_i$ is the

observation function, $T\colon \mathcal{S} \times \mathcal{A} \to \Delta\mathcal{S}$ is the environment transition function, $R^i\colon \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function, and $\mu \in \Delta\mathcal{S}$ is the initial state distribution.

In a POSG, all agents simultaneously make decisions based on their own private observations. Then, the environment is updated according to the joint action of all agents, and each agent receives its own reward that it tries to maximize. The reward is computed the same way for each agent, but based on its individual situation (i.e. no reward sharing). We additionally specify a time limit $T_{max} \in \mathbb{N}$ which is the maximum number of steps the environment is allowed to take before resetting.

3.2.2 Observation Space

In order to navigate through the environment, each agent must perceive its environment in some way. However, it is not obvious in what form agents should receive this information, or in fact, what the information should be.

The simplest human-inspired design is to give the agent information in its own frame of reference, and to have it perceive the environment through raycasting – a simple approximation of human vision. The intention is that if humans can effectively navigate using this type of information, then it should also suffice for virtual agents, which should then act more human-like. However, it is not necessarily the case that an anthropomorphic structure is indeed optimal for virtual agents, especially with it being only a rough approximation. More realistic rendering of the agent's vision is an option, but it would result in very large observation sizes, subject to the curse of dimensionality. Thus, it is worthwhile to explore other possibilities.

An agent must receive information about its surroundings (other nearby agents and obstacles, i.e. **Environment Perception**), but also about its own internal state and knowledge (e.g. its current velocity, or its current destination, i.e. **Proprioception**). In both of these cases, it is also relevant what is the **reference frame** in which they are observed. For the Environment Perception, we consider two types of perception: Raycasting and Direct Agent Perception (AP) (Section 3.2.2). For the reference frame, we have three representations: Absolute, Relative, and Egocentric (Section 3.2.2).

Environment Perception

Raycasting refers to a method where several rays are cast from the center of the agent, in a plane parallel to the ground. Each of those rays then provides information on whether or not it collides with any object within a predefined distance, and what is the distance to the collision location.

Direct Agent Perception, similar to the method used by Xu and Karamouzas [186] is an alternative approach, where the agent directly receives the positions of other agents within a certain range, along with other relevant parameters. The reference frame in which the information is passed follows the chosen proprioception model (i.e. Absolute, Relative or Egocentric). This has the possible benefit of directly giving access to relevant information, but it introduces two important complications. Firstly, this method cannot canonically represent obstacles. While small, human-sized obstacles can be treated as stationary agents, large obstacles like walls need a different approach. Secondly, the number of neighboring agents is variable, and can grow very large in high density scenarios. The standard Multi-Layer Perceptron architecture cannot handle variable-sized inputs, so this variability has to be accounted for in some way. Furthermore, the order in which

neighboring agents are observed is irrelevant, so a permutation-invariant architecture is necessary to decrease the effective size of the observation space.

In this work, we also test a multimodal approach which combines both Raycasting and Agent Perception, described in more detail in Section 3.4.2. With this hybrid method, the raycasting is only used to perceive static obstacles such as walls, ignoring other agents, whose positions are instead observed directly. We hypothesize that this might enable gaining the benefits of both methods – the agent has accurate knowledge of others, as well as a general idea of the surrounding layout, sufficient for navigation.

Reference Frames

Absolute observations [156] use a bird's-eye view on the global scene. The agent observes a vector consisting of its position $\mathbf{p}$ and the position of its goal $\mathbf{p}_g$ in the global coordinate frame, its current orientation ϕ (which is relevant for some choices in Action Spaces), and its current velocity $\mathbf{v}$. Using the Agent Perception approach, the agent observes the positions $\mathbf{p}_i$ and velocities $\mathbf{v}_i$ of nearby agents in the global frame.

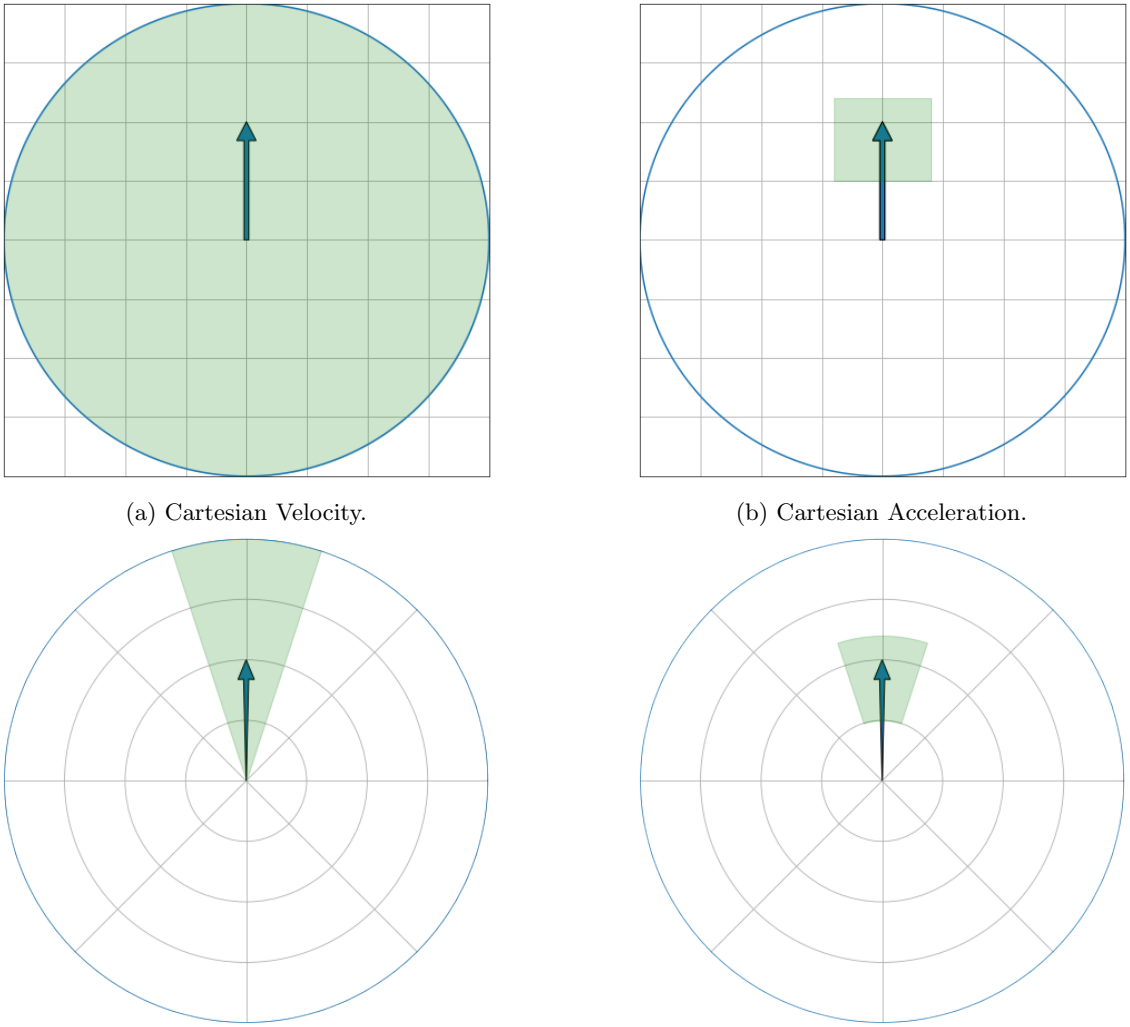
Relative observations [186, 106, 185], like Absolute, use the global frame, however it is translated so that it is centered on the agent. It again receives its absolute position $\mathbf{p}$ and velocity $\mathbf{v}$ in order to retain the information about the large context, but the goal position is given relatively to the agent's own position, as $\mathbf{p}_g - \mathbf{p}$. Similarly, using Agent Perception, other agents' positions are given as $\mathbf{p}_i - \mathbf{p}$, and velocities are given in the absolute form $\mathbf{v}_i$.

Egocentric observations [63, 93] use the agent's local frame according to its orientation. The agent observes its position $\mathbf{p}$ and orientation angle ϕ in the global frame. We write R_ϕ to be the rotation matrix associated with the agent's current orientation. The agent observes its goal position as $R_\phi(\mathbf{p}_g - \mathbf{p})$. Using Agent Perception, the positions of other agents are also represented as $R_\phi(\mathbf{p}_i - \mathbf{p})$, and their observed velocities are $R_\phi \mathbf{v}$.

While each of these has a theoretical justification (moving from absolute information about the scene, towards a more human-like first-person view), it is not immediately obvious which one is the best. On the one hand, absolute observations can give a high-level overall view of the scene, potentially aiding coordination. On the other hand, a relative or egocentric view allows agents to better reuse experiences between different positions and situations. If the agent is heading towards its goal, it is not that important whether it is to the left or to the right, looking from a bird's eye view. Since the Egocentric observations lose this information, navigation might be expected to be learned more efficiently. Note that the choice of the reference frame also affects the structure of Agent Perception.

3.2.3 Action Space and Dynamics

Human motion is highly complex, and a biomechanically accurate simulation of human motion is a challenging research problem in of itself, so for the purposes of creating virtual crowds, we use a simplified model. The simplest choice is holonomic locomotion [67], where at each step, the agent can choose its velocity constrained only by its magnitude. However, this approach does not correspond well to the motion constraints of real humans. Arechavaleta et al. [10] propose a nonholonomic model, in which an agent can move in the direction of its current orientation, and incrementally change its orientation for the next timestep.



(a) Cartesian Velocity.

(b) Cartesian Acceleration.

(c) Polar Velocity.

(d) Polar Acceleration.

Figure 3.1: A schematic representation of the available action spaces. In each case, we take a bird's-eye view of an agent moving in the positive Y direction at an intermediate speed, represented by the blue arrow. The blue circle represents the space of all physically possible velocities (i.e. below the maximum speed). The green area represents the velocities that the agent is able to have in the following timestep under the specific action space.

Allowing the agent to freely choose its velocity at every timestep gives it much more flexibility in choosing its behavior. However, from the perspective of Newtonian mechanics, it is more physically justified for the agent to directly choose its acceleration. This would mean that the velocity change at each timestep is incremental.

Similarly, there is a choice in how the actions should be represented. We can take the bird's eye view, where the agents choose their actions according to an absolute reference frame, moving up or down, left or right. Alternatively, we can take a more individual perspective, where the agents operate in a polar frame, choosing their linear movement and the direction of that movement.

For this reason, we consider four different dynamics models of the environment: Cartesian Velocity, Cartesian Acceleration, Polar Velocity, and Polar Acceleration.

Cartesian control (used by [186, 63]) implies that the agent separately chooses the x and y components of its motion - either of its velocity or acceleration.

Polar control (used by [93, 106, 156, 185]), implies that the agent separately updates its orientation angle, and its linear speed. The linear speed is again updated either by choosing an arbitrary value below a certain magnitude (velocity controls), or by incrementally updating it according to the acceleration chosen by the agent.

Acceleration controls (used by [63, 186]), are modeled in our implementation using a linear damping model. This means that given an acceleration a_t chosen by the agent as its action, the updated velocity will be

$$v_t = v_{t-1} + (a_t - \lambda v_{t-1})\Delta t$$

where λ is chosen so that we obtain the same maximum speed as in velocity controls, which is equal to $2\frac{m}{s}$.

With velocity controls (used by [93, 106, 156, 185]), the agent can choose an arbitrary speed of a magnitude lower than $2\frac{m}{s}$.

In Cartesian controls, the agent's orientation is defined to be parallel to its current velocity. In Polar controls, the orientation is directly controlled, and the velocity is parallel to the orientation.

3.3 Reward Function Design

Designing the reward function is arguably the most impactful, and the most difficult part of creating an RL-driven crowd simulation. The two components which are present in all existing work are a positive term correlated with reaching the goal, and a negative term correlated with collisions. Beyond that, various elements may be included to promote certain behavior characteristics, or to improve training performance through reward shaping. In this work, we consider the following reward components based on prior work:

1. Reward for reaching the goal $R_g = c_g$ (once) [106, 156, 185]
2. Reward for approaching the goal $R_p = c_p(d_t - d_{t-1})$ (every timestep) [186, 63, 93, 106, 156, 185]
3. Reward for maintaining a comfortable speed $R_v = -c_v|v - v_0|^{c_e}$ (every timestep) [186]
4. Penalty for collisions $R_c = -c_c$ (every collision, every timestep) [186, 63, 93, 106, 156, 185]
5. Reward for urgency $R_t = -c_t$ (every timestep) [156]

where c_g, c_p, c_v, c_e, c_c, c_t are arbitrary (typically positive) coefficients. Their roles are as follows: R_g is the main (sparse) reward representing the agent's destination. R_p provides a dense reward for the navigation objective to enable faster training. R_v incentivizes moving at a comfortable speed v_0. We propose raising the absolute difference of speeds to an arbitrary power c_e in order to further shape the behavior. R_c makes agents avoid colliding with obstacles and with one another. R_t is a commonly used reward component in goal-based RL environments, as it incentivizes the agents to reach their goal sooner rather than later.

3.3.1 Energy and Metrics

When considering different reward functions, it is important to have metrics that are independent of the specific reward formulation in order to have a meaningful comparison. For this reason, we consider two types of metrics.

Firstly, for each component of the reward function, we compute an unnormalized value measuring its performance, which can then be compared between different training runs. For example, we consider the total number of collision, regardless of the size of the collision penalty in the reward function.

Secondly, we compute the mean energy expenditure of all agents in the scene. We use the following formula [179, 45]:

$$E_t = e_s + e_w v^2$$

where E_t is energy spent per second, per kilogram body mass (J/kg·s), e_s and e_w are constants, and v is the current velocity (m/s). We use the values of $e_s = 2.23$ and $e_w = 1.26$ of a typical human. This induces an optimal walking speed of $v^* = \sqrt{\frac{e_s}{e_w}} = 1.33$ m/s which we use as the preferred walking speed. Work by Bruneau et al. [20] suggests that when navigating to avoid collisions, people tend to choose a path that minimizes the energy usage, so we consider it to be a valuable metric describing the efficiency of the generated trajectory.

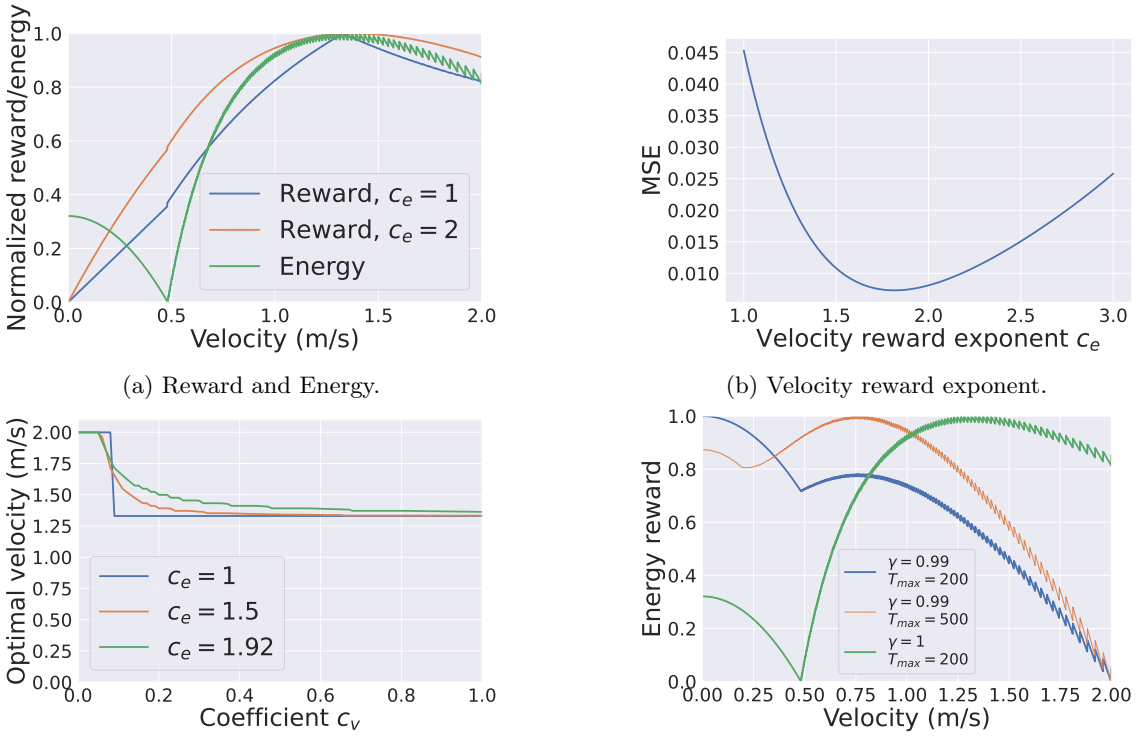
3.3.2 Reward and Preferred Velocity

As we see from the energy optimization mechanism, humans tend to move at a certain speed which is below their maximum possible speed. This must be reflected in the reward function that the RL agent optimizes. However, it goes against the typical RL incentives to obtain rewards sooner due to the discounted utility model [160].

The interaction between the values of c_g, c_p, c_v, c_e, c_t, γ, and T_{max}, make the effective preferred velocity nontrivial to predict, and as a consequence – to design. Consider a simplified environment model in which the agent must travel a distance d towards its goal, with no other agents or obstacles. In this case, the only decision it makes is the velocity throughout its motion, under the assumption that it will travel to its goal in a straight line at a constant velocity. In this model, we can express the total obtained reward as:

$$R(v) = \gamma^T c_g + \sum_{i=0}^{T} \gamma^i \left(c_p v \Delta t - c_v |v - v_0|^{c_e} - c_t\right) \tag{3.1}$$

where $T = \min(T_{max}, \lceil \frac{d}{v\Delta t} \rceil)$ Due to the discounted sum whose bounds are dependent on the velocity itself, it is difficult to investigate this expression analytically. Nevertheless, we can gain some insight through numerically analyzing this model. Note, however, that this model does not



(a) Reward and Energy.

(b) Velocity reward exponent.

(c) Velocity reward coefficient.

(d) Energy-based reward.

Figure 3.2: **(a)** Rewards and energy for an agent moving at a constant speed, in the simplified model described in Section 3.3.2. All curves are normalized to be in the [0, 1] range in order to enable direct comparison. We consider energy values with the opposite sign, because energy is supposed to be minimized, while the reward is maximized. **(b)** MSE between the reward and the energy, as a function of the velocity reward exponent. **(c)** Optimal velocity as a function of the velocity reward coefficient c_v, varied by the exponent c_e. **(d)** Discounted negative energy expenditure as a function of velocity.

capture the full complexity of RL optimization, and only serves to build general intuition.

Consider the following set of parameters values – based on prior work and then manually adjusted to produce reasonable behaviors – as a starting point of our analysis: $c_g = 10$, $c_p = 1$, $c_v = 0.75$, $c_e = 1$, $c_t = 0.005$ $v_0 = 1.33$, $\gamma = 0.99$, $T_{max} = 200$, $d = 8$, $\Delta t = \frac{1}{12}$.

Let us investigate the full reward as a function of velocity, alongside the negative energy expenditure as defined in Section 3.3.1. This relation is shown in Figure 3.2a. We can see that while there is a correlation between reward and energy, there are two main discrepancies. Firstly, the energy has a local optimum at $v = 0$, which is caused by the velocities that cause agents to not reach their destination within the allotted time. Secondly, due to the absolute difference term in Equation 3.1, there is a sharp decrease in the reward (blue curve) which does not occur in the energy.

To improve this, we evaluate the impact of the exponent c_e from the reward function. In Figure 3.2a, we also show $c_e = 2$ (orange). There, the curve is smoother and closer to the corresponding energy values when the energy is near the optimum. To quantify this, we vary the parameter c_e and compute the mean square error between the two normalized curves in the range $1\frac{m}{s} < v < 2\frac{m}{s}$, as we consider lower velocities to be less relevant due to their low efficiency. As we show in Figure 3.2b, the optimal value under this simple model is $c_e = 1.92$. We further validate this in Section 3.5, where we use different values of c_e for training actual RL agents.

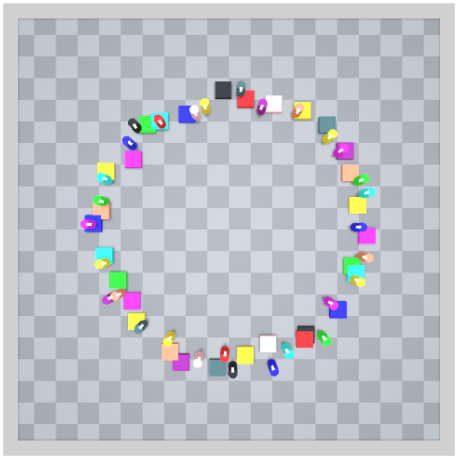
It is worth mentioning that with $\gamma = 0.99$, using $c_e = 2$ increases the effective optimal velocity to $v^* = 1.39\frac{m}{s}$. This can be further adjusted using other parameters. While modifying γ does not affect the optimal velocity when $c_e = 1$, any higher values of c_e make it so that decreasing γ, increases v^*.

The other reward coefficients also have an impact on the optimal velocity. When adjusting c_v with $c_e = 1$, there is a threshold around $c_v = 0.09$ below which the optimal velocity is the maximum value of $2\frac{m}{s}$. Above that threshold, the optimal velocity is the preferred value of $v_0 = 1.33\frac{m}{s}$. However, with $c_e > 1$, the transition between these two realms becomes more gradual, as we show in Figure 3.2c.

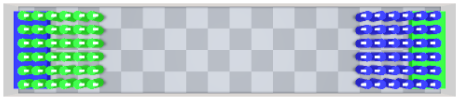
3.3.3 Energy as reward

Finally, it is worth considering using energy directly as a reward function for training RL agents. At a glance, it seems like it would incentivize efficient motion at the optimal velocity. However, there are two apparent problems which arise in this paradigm. Firstly, as we already see in Figure 3.2a, even in our simplified model there is an attractive local optimum at $v = 0$. This is likely to be even more impactful in practical scenarios, because moving at the right speed would lead to a very low reward if the direction of the movement is wrong. The reason for this is the time limit present in the environment – moving at a non-zero speed only pays off in terms of energy if the agent eventually reaches the goal. Otherwise, it will expend energy until the end of the episode, and by reducing its velocity, it can reduce the energy expenditure. Secondly, the commonly used method of reward discounting has a significant impact on the optimal policy. Using a discount factor of $\gamma = 0.99$ leads to a situation where the global optimum of the discounted energy-based reward function is standing still with $v = 0$. Potential solutions to these problems include using a discount factor $\gamma = 1$ or a nonexponential discounting mechanism, increasing the time limit, and using a curriculum-based approach.

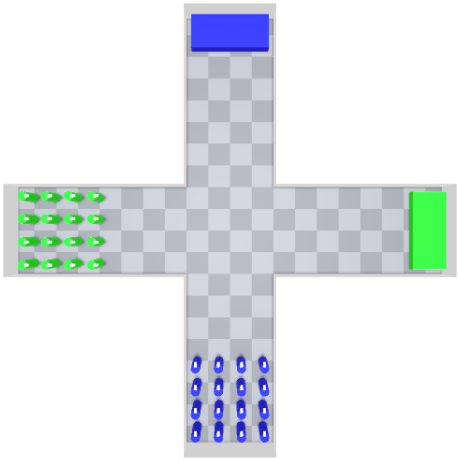
Conclusion. Even in the absence of collision avoidance and other, more complex tasks, one



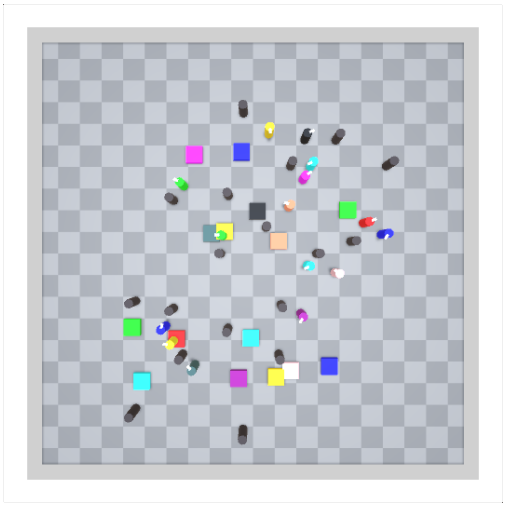
(a) Circle scenario.



(b) Corridor scenario.



(c) Crossing scenario.



(d) Random scenario.

Figure 3.3: Agent's initial positions and goals in four scenarios: **(a)** Circle with 30 agents. **(b)** Corridor with 72 agents. **(c)** Crossing with 32 agents. **(d)** Random with 15 agents.

must pay attention to the parameters defining the reward function. Notably, using an exponent in the velocity reward term causes other parameters to nontrivially affect the effective optimal velocity. For this reason, when designing the reward function, it is worthwhile to validate its parameters using a simpler model to ensure it has desired properties, whether that is closeness to the energy expenditure, or a specific value of the preferred velocity.

3.4 Experimental setup

In order to evaluate the impact and quality of the various design choices, we apply them on four commonly used crowd scenarios, in order to provide a wide range of interactions between agents: Circle, Corridor, Crossing, Random (see Figure 3.3). In the Circle scenario, agents start on the perimeter of a circle, with a random noise applied independently in both Cartesian directions. Their goals are placed on the antipodal points of the circle, with an independent noise of the same magnitude applied. In Corridor, agents start at two ends of a straight corridor whose width is

4 meters and length is 20 meters. They start either in a regular grid or in a random formation, and their goal is to reach the opposite side of the corridor. In Crossing, the agents start at the ends of two corridors intersecting at a right angle, with the same size as in Corridor. Similarly, they spawn either in a regular grid or a random formation, and must reach the other end of their respective corridors. In Random, the agents' starting positions and goals are generated according to a uniform distribution with a given maximum size. In each of these scenarios, the area available to the agents is a square of 20x20 meters. In both Circle and Random, there are optional small obstacles placed randomly in the scene, represented as immovable agents.

Implementation The code[1] used in this work is available online. We use identical agents represented as circles of radius 0.2 m. Their collisions are treated as rigid body collisions, processed by the PhysX engine default in Unity 2021.3, used via the Unity Ml-Agents framework [73]. We use a decision timestep $\Delta t = \frac{1}{12}$ s, similar to the values used in prior work. To obtain a more accurate simulation, the physics of the scene are updated 10 times after each decision, for an effective simulation timestep of $\frac{1}{120}$ s. The agent's action is repeated during each of these updates. With agent perception observations, each agent can see the 10 nearest agents. While these implementation details (i.e. agent size, collision handling, physics engine, timestep) can also affect the resulting simulations and the training performance, we do not explore their impact in this work, because we expect it to be lower compared to the other choices listed in this paper. It is nevertheless important to be aware of these choices for reproducibility purposes. A single training takes between 1 and 5 hours on GPU, depending on the number of agents and the difficulty of the scene, while running 8 independent training runs in parallel.

3.4.1 Policy Optimization

In this work, we train RL agents using PPO with Generalized Advantage Estimation (GAE) [145] to estimate the advantages. The agents are trained in an independent paradigm with parameter sharing, that is they share the same policy network, but each agent takes its own action based on its private observations. The neural network outputs the mean of a Gaussian distribution, and the standard deviation is kept as a trainable parameter of the network.

3.4.2 Network Architecture

In order to appropriately process the Agent Perception observations, we use a neural architecture depicted in Figure 3.4. It is inspired by prior work such as Deep Sets [193] and Mean Embedding [68], and extends the architecture used by Xu and Karamouzas [186].

The main desirable property of our architecture is permutation invariance – given multiple identical nearby agents, it should not matter in what order their representations are input into the network, as this order is completely arbitrary. Without this property, the agent would need to learn this invariance itself, which quickly becomes expensive as the number of observed agents grows. Furthermore, the architecture should be able to accept a variable number of observed agents, as this quantity will vary throughout the episode. For this reason, we use the following model as an

[1]The code for the environment is available at `https://github.com/RedTachyon/CrowdAI`, and the training code is available at `https://github.com/RedTachyon/coltra-rl`.

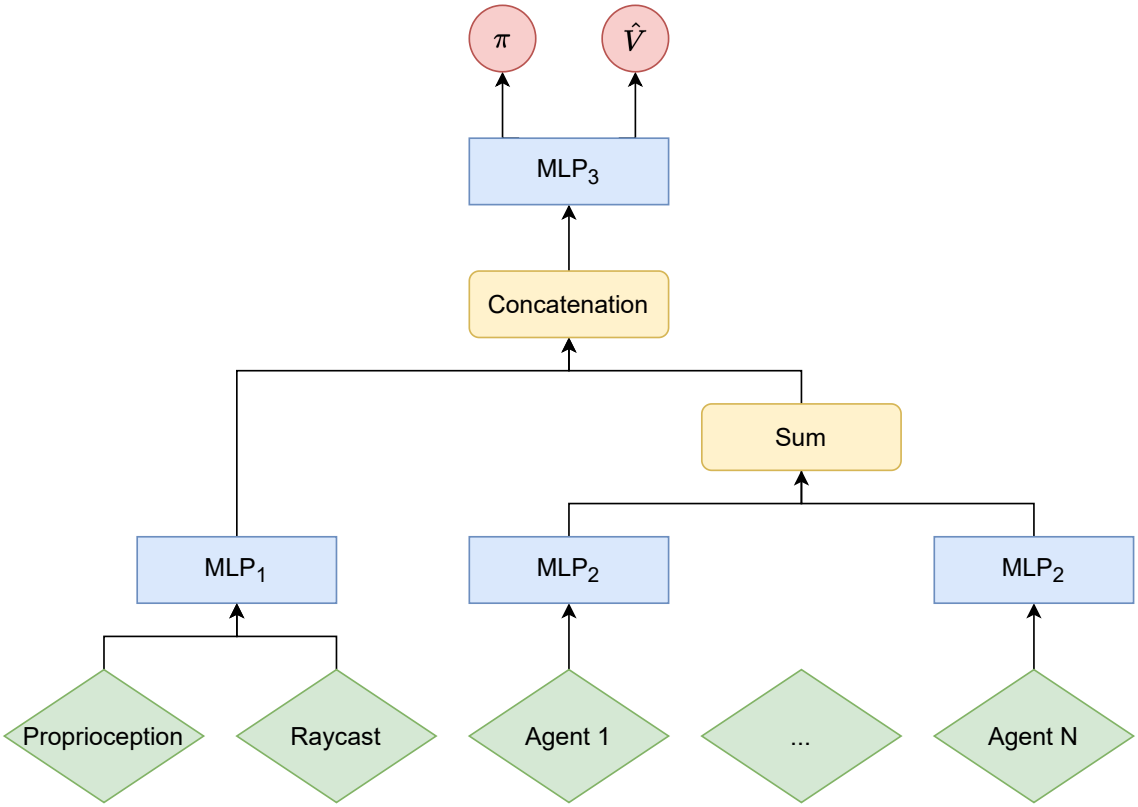


Figure 3.4: The neural architecture used as the policy. Green blocks represent inputs, blue blocks represent feed-forward neural networks, yellow blocks represent vector operations, red blocks represent outputs. Depending on the observation model used, certain elements of the architecture are disabled.

embedding of nearby agents:

$$\phi\left(\sum_i \psi(x_i)\right)$$

where ϕ and ψ are regular MLP neural networks, and x_i is the observed information about an agent i. The summation is performed over all agents visible to the agent observing the scene. Because of the summation operator, this architecture fulfills both previously stated desiderata, as the ordering information is lost, and any number of agents can be processed into a fixed-size embedding. This embedding is then concatenated with the main stream of the neural network, which processes the proprioceptive observations, as well as optionally the raycasting.

3.5 Experiments

In this section we describe the specific experiments we performed, along with their results and interpretation. Experiments are primarily evaluated in terms of their obtained rewards and energy usage, but also other behavior characteristics when necessary.

3.5.1 Dynamics and Observations performance

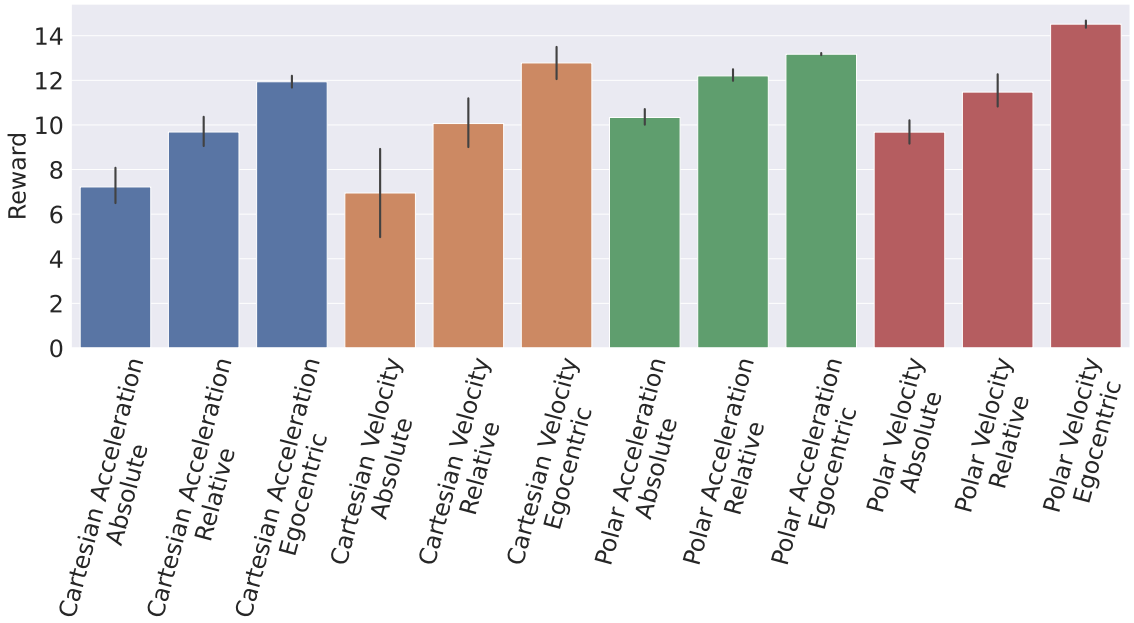
In order to robustly evaluate the performance of various dynamics and observation models, we run a hyperparameter search with each action space, with each observation reference frame, on a Circle scenario with 12 agents and a radius of 4 meters. Then, we use the best-performing hyperparameters of each model for further experiments used in this section. We sample 150 sets of hyperparameters for each (dynamics, observations) pair for a total of 1800 training runs. We use the default sampler available in Optuna [4]. These runs use a fixed reward function, with parameters $c_g = 10$, $c_p = 1$, $c_v = 0.75$, $c_e = 1$, $c_c = 0.05$, $c_t = 0.005$.

We show the results of the hyperparameter search in Figure 3.5. We consider the 5 best-performing hyperparameter sets for each model, and report their aggregate performance. There is a clear trend where agents with Egocentric observations perform better than Absolute and Relative versions. Similarly, Polar controls perform better than Cartesian controls, with Polar Velocity controls with Egocentric observations performing the best out of all investigated variants. Interestingly, in the case of Cartesian Velocity controls, while the reward follows the same trend, the energy usage is in fact the lowest with Absolute observations. This highlights the discrepancy between the reward function and the energy metric, showcasing the need for careful evaluation of emergent behaviors.

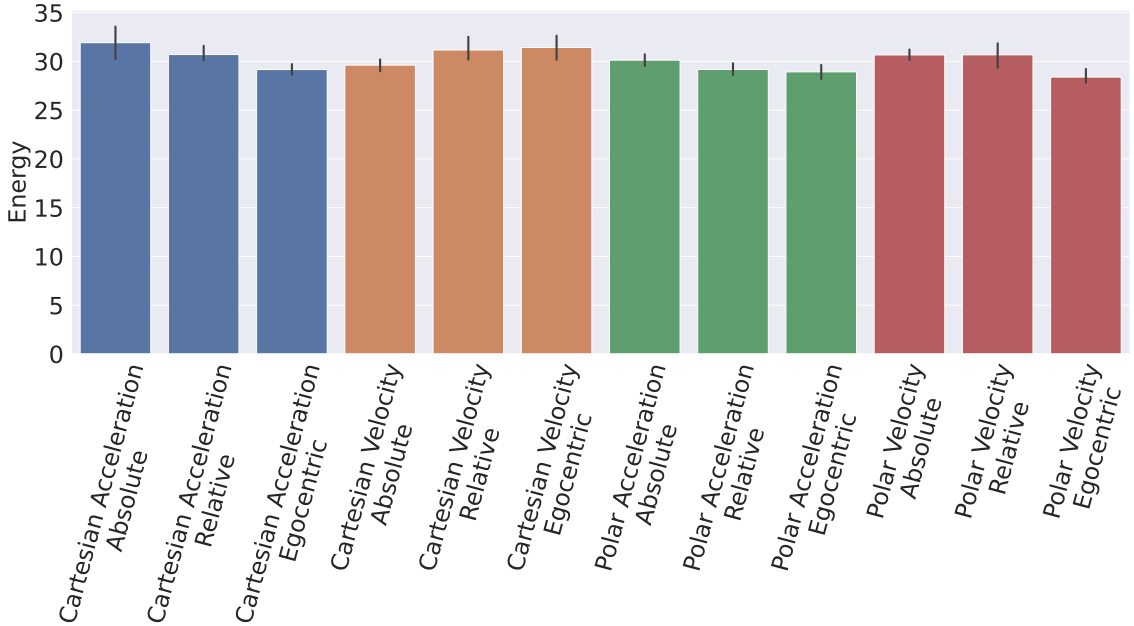
3.5.2 All Scenarios

Using the hyperparameters obtained in the experiment described in Section 3.5.1, we evaluate the various design choices on several environments. Specifically, we train agents in a Circle with 12 agents and a radius of 6 meters (circle12), a Corridor with 50 agents (corridor50), a Crossing with 50 agents (crossway50), and a Random scenario with 20 agents and random obstacles (random20).

Due to the large number of possible combinations (3 architectures $\times$ 4 dynamics $\times$ 3 observations $\times$ 4 scenarios), we consider the performance of each individual choice, averaging the remaining ones, in each environment separately. The results are in Figure 3.6.

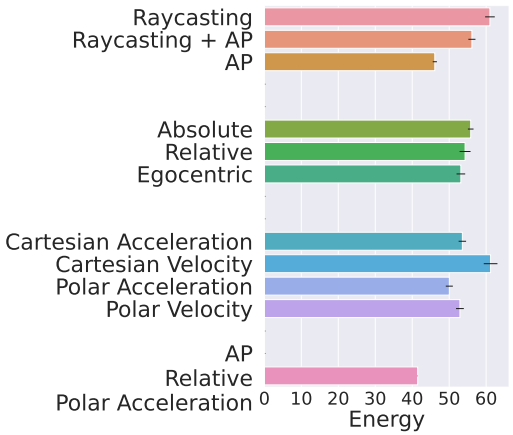


(a) Reward function (higher is better).

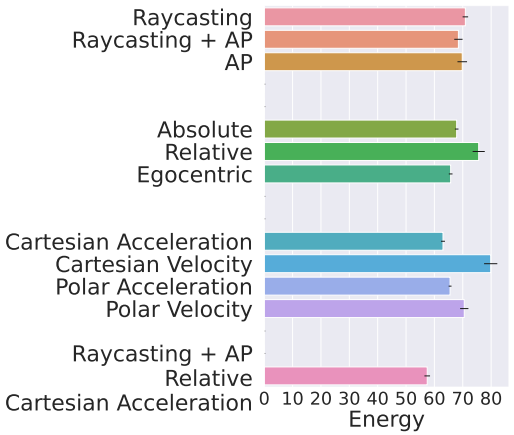


(b) Energy expenditure (lower is better).

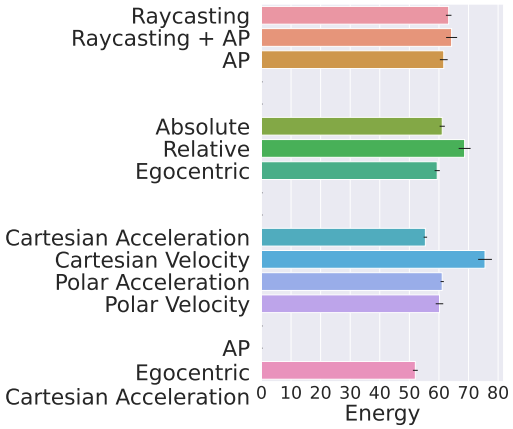
Figure 3.5: Comparison of training results after a hyperparameter search in the Circle 12 scenario. **(a)** Mean episodic reward **(b)** Mean energy expenditure. Black bars represent the standard error of the mean.



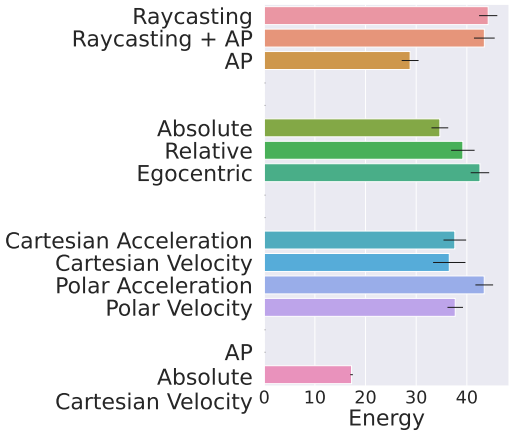
(a) Scenario circle12



(b) Scenario crossway50



(c) Scenario corridor50



(d) Scenario random20

Figure 3.6: Comparison of various design choices in a given environment. The last bar corresponds to the best-performing agent across all design choices. All values are averaged across 8 independent training runs with different random seeds and otherwise identical parameters. Lower is better. AP stands for Agent Perception as defined in Section 3.2.2.

We run each training for 1000 PPO iterations to ensure sufficient time for convergence. Because the neural networks are relatively small (2-6 layers of 32-128 units, depending on the architecture and the model, found via hyperparameter optimization), the main bottleneck is the CPU power required to run many copies of the environment as opposed to the GPU memory.

While the exact results vary between scenarios, there are some regularities. Most notably, Agent Perception consistently outperforms raycasting-based approaches with otherwise well-performing settings. In the case of the crossway50 scenario, it is beneficial to include raycasting information around the surrounding walls, however overall it does not seem to provide a large benefit as compared to the pure AP approach. This is likely due to the static nature of the evaluated environments. Given the current position in the global frame, the agent can determine its proximity from obstacles. Unsurprisingly, including raycasting to perceive walls, deteriorates the performance in scenarios without a significant presence of walls, as this information effectively becomes an additional source of noise.

In most cases, it is best to use Egocentric or Relative observations, with some Relative runs sometimes performing the best. A notable exception is the random20 scenario, where a combination of Absolute observation with Cartesian Velocity dynamics outperforms any other option. The difference is the fact that in all other scenarios, the agents need to predominantly move in a general "forward" direction, whereas in the random scenario, the goal can be in an arbitrary position relative to the agent's orientation.

An interesting observation is the common "failure mode" of Raycasting models, particularly in Circle scenarios. They generally perform worse than Agent Perception models, but their qualitative behavior may be more desirable due to its asymmetry. Because they do not manage to reach the perfectly symmetrical trajectories achieved by Agent Perception, there is a higher variability in the individual trajectories, making them look more realistic. This indicates that simply training a strong RL algorithm on any objective which does not explicitly reward human-likeness is likely to lead to overly perfect, unrealistic trajectories.

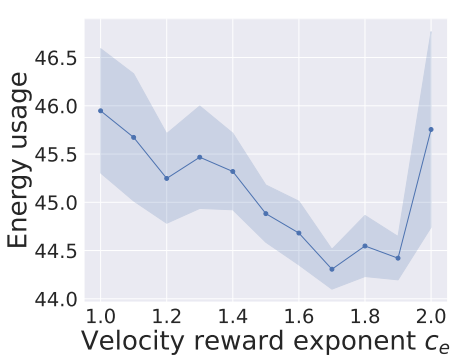
Conclusion. We recommend using AP as opposed to the more commonly used raycasting for providing agents with the information about their surrounding. In scenarios where walls are a prominent feature, it may be beneficial to add raycasting which only perceives the distance to walls, and ignores other agents. In scenarios where agents may need to make sharp turns to reach their destination, Cartesian Velocity controls with Absolute observations are favorable. Otherwise, nonholonomic controls combined with egocentric or relative observations typically perform better.

3.5.3 Velocity Reward Exponent

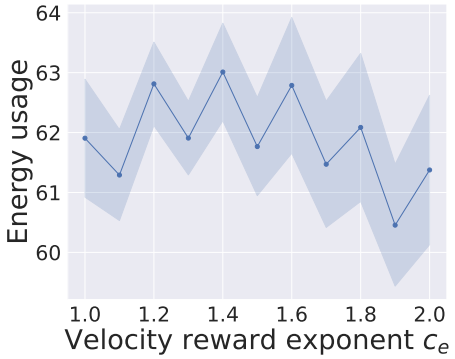
In Section 3.3.2, we show that using an exponent in the velocity reward term makes it match more closely to the energy consumption. To validate this, we train Egocentric Polar Velocity agents on a Circle 30 scenario, and Egocentric Polar Acceleration agents on a Crossway 50 scenario. In both cases, an exponent $c_e > 1$ can lead to a higher efficiency in the trained agents, as compared to the simple $c_e = 1$ (see Figure 3.7). The exact optimal value of c_e depends on the scenario and must be determined on a case-by-case basis.

3.5.4 Importance of collision penalty

In a pursuit of simplicity in the design of the reward function, one might be tempted to eliminate the collision penalty altogether. After all, if collisions result in unfavorable physical results

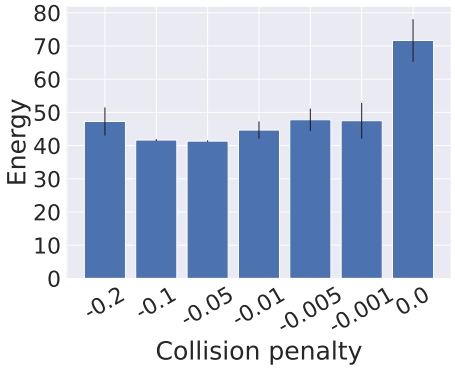


(a) Circle 30 scenario

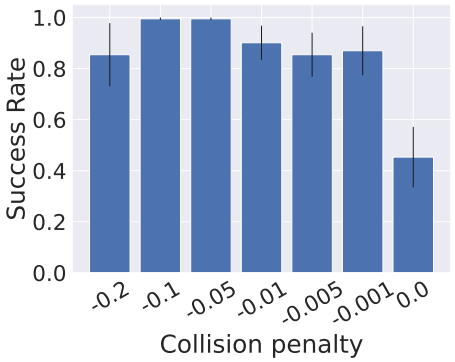


(b) Crossway 50 scenario

Figure 3.7: Comparison of energy usage in agents trained with a different exponent in the velocity term of the reward function. Lower is better.

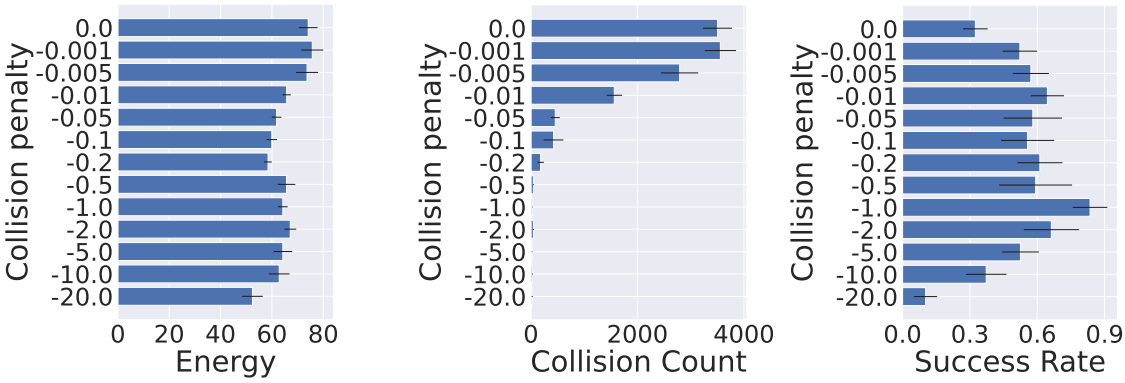


(a) Energy usage.



(b) Success rate.

Figure 3.8: Comparison of energy usage and success rate in agents trained in Circle 12 scenario, with a varied collision penalty in the reward function.



(a) Energy usage. (b) Collision count. (c) Success rate.

Figure 3.9: Comparison of energy usage, collision count and success rate in agents trained in Crossway 50 scenario, with a varied collision penalty in the reward function.

(i.e. unexpected change of velocity), sufficiently intelligent agents should learn to avoid them by themselves, at least to the extent that is necessary for effective navigation.

In Figure 3.8 we show how the collision penalty affects the energy usage, and how often the agents reach their destination. There is an optimum around -0.05, where the agents reliably reach their goals. Using a penalty that is too high or too low leads to a deterioration of the agents' performance in terms of the navigation task.

3.5.5 Common Failure Modes

Due to the stochasticity inherent to RL training, the trained agents often exhibit various types of suboptimal behaviors. This can be identified via tracking the performance (in terms of the reward and energy), but also by observing emergent behaviors that the agents learn to execute. Here, we describe some of the common ways in which the RL-trained crowds are suboptimal.

Instability When training an RL agent using PPO, the trained policy is stochastic. This is required both for training, to ensure that the agent takes sufficiently diverse actions; but it is also at the core of the resulting policy. In order to deploy or evaluate the agent, we must choose a method of sampling actions from the output of the policy. The two natural options are taking the mean for the "optimal" action, or simply sampling from the distribution. The former does not fully correspond to the optimization objective, and in a crowd scenario, it can get stuck on an obstacle or another agent. The latter necessarily causes the policy to act randomly, which leads to potential unpredictable mistakes due to small erratic movements.

Value alignment failure As we describe in Section 3.3, the reward function used in this and prior work is a weighted sum of several components with largely arbitrary coefficients. This implies that there is no guarantee it will correspond well to the actual objective we intend the agents to achieve. Furthermore, while the energy usage can be a useful, non-arbitrary metric, it is also sensitive to some details of the practical RL setup, as we show next.

Consider the experiment described in Section 3.5.4, investigating the importance of the collision penalty. We perform the same analysis on the Crossway 50 scenario with Polar Velocity dynamics and Egocentric observations, extending the range of evaluated collision penalties to -20. We

present these results in Figure 3.9.

When considering the energy usage and collisions, the result seems to follow the intuition with a collision penalty up to -1. When the collision penalty reaches -20, we expect the performance to deteriorate, since the penalty is so steep that the agents will never put themselves at any risk of collision. We confirm this by investigating the success rate, which decreases with large collision penalties. However, at the same time, the energy usage also decreases – in fact, using a very large collision penalty leads to the lowest energy usage among the evaluated options. So while energy has its value as a metric, it clearly breaks down in extreme cases, where most agents remain stationary.

This phenomenon is caused by a relatively short time limit. In general, energy minimization induces a simple optimal velocity (see Section 3.3.1), in RL we must set a time limit after which the episode is terminated. The energy penalty for not finishing the episode is relatively low when compared to the energy cost of actually navigating to the destination, and an undesired emergent behavior may turn out to be optimal according to the metrics.

Depending on the design choices made, this same problem can manifest itself differently. When training on the same setup, but using different design choices, a commonly occurring emergent behavior is that one group of agents efficiently navigates to their destination, while the other one remains stationary. Similarly to the previous situation, the energy usage metric is very low in that situation, but the success rate plateaus at 50%.

3.6 Discussion

In this work, we present an analysis of various design choices made by designers of RL-trained crowd simulation systems. We show that many of these choices, typically ignored by researchers, can in fact significantly impact the resulting simulation, in particular when evaluated in terms of the energy efficiency. Here we summarize the main findings of this paper.

We show that the commonly used raycasting underperforms when compared to a method we call Agent Perception where the information about neighboring agents is directly available. This is likely a consequence of a much simpler and accessible representation of that information. Even when raycasting uses frame stacking which enables movement perception, the reasoning needed to infer the positions and velocities of nearby agents is rather complex.

Designing the right reward function is also important for obtaining desired properties of the motion. Navigation, speed control, and collision avoidance rewards, all have to balanced in just the right way that each of them contributes to the agent's decisions. Crucially, quantitative comparisons of crowds are non-trivial, because even ignoring the question of believability, the arbitrary reward function and energy usage are flawed in certain scenarios.

Qualitative properties of the obtained motion in the Circle scenario indicate that a naive approach of reward maximization for any reward that does not explicitly incentivize human-like behavior is likely to create trajectories that look artificial, or even too carefully choreographed to pass for natural human behavior. While decreasing the model's capacity and performance might lead to more believable behavior in the short term, we believe a more deliberate approach is necessary to truly approach human-like behavior.

In summary, our main findings are as follows:

1. Direct agent perception outperforms simple raycasting

2. Egocentric controls tend to outperform absolute ones

3. The reward design is important and nontrivial

4. Many failure modes may still occur in RL trained crowds

5. Simple reward is not sufficient for human-like behavior

3.6.1 Limitations and Future Work

All experiments in this work are performed on relatively small, static scenarios with a single destination. The described design choices mostly affect local navigation and more complex scenarios can be expressed as a sequence of partial objectives or checkpoints. That being said, naively implementing this would likely cause issues near the transition points where agents switch their destinations. Therefore, a more complex training scenario would be beneficial so that the agent is exposed to these situations.

Furthermore, each agent is only trained on a single scenario. Prior work suggests that using various scenarios in the training process enables generalization, which was considered out of the scope of this work. We also limit our analysis to the efficiency of the resulting trajectories, ignoring realism or believability.

Similar to prior work, we train agents using an arbitrarily designed reward function. While using the energy usage as a reward has certain problems (see Section 3.3.2), it might be a viable option to use a curriculum-based approach where the reward function changes as the training progresses; and a different discounting mechanism that improves the global optimization properties. Furthermore, by using recent work on evaluating the realism of generated trajectories [33], a promising direction is using a realism metric as a reward. This would allow going beyond efficiency, and creating crowds which behave in a believable way.

It is also possible to improve the dynamics available to the agent. In this work, we use relatively simple, 2-DoF models, but the RL paradigm makes it viable to implement arbitrary nonholonomic constraints like sidesteps or walking backwards, without having to change the learning logic. Thus, a promising option is introducing a more complex, human-like range of motion actions available to the agent, with the goal of improving the believability of motion.

3.6.2 Conclusions

Crowd simulation with RL is a complex problem, and despite recent advances, many challenges remain. Observations, actions, underlying physics, and especially the reward function, all have a significant impact on the results, and a lack of attention to these design choices makes it impossible to compare various approaches. We bring these issues to attention, and introduce a basic methodology for comparison between various approaches by comparing the energy expenditure under a specified time limit. In the absence of a standard benchmark, we call researchers to be more explicit and precise about their choices, and encourage them to explore different options in their work, to ensure the robustness of their approach.

Continuing this line of work, we extend our investigation into the reward function started in Section 3.3. There, we identified the main problems with a naively designed reward function, and proposed a simple heuristic approach that worked "well enough". Next, we aim to approach this problem more methodically and design a reward function that can be successfully applied to learning crowd navigation, using as few arbitrary components as possible.

Chapter 4

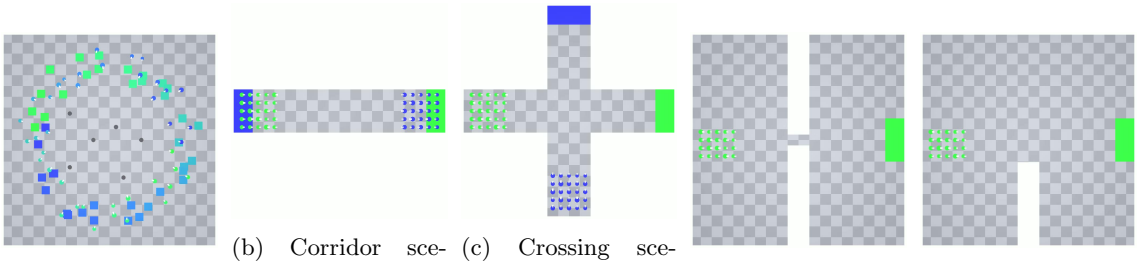
Reward function design

In Chapter 3, we provided a basic analysis of the reward function design, along with the problems with simply optimizing the energy usage as the reward. In this chapter, we dive deeper into this topic by formally introducing two arguments preventing the usage of direct energy optimization – the **local optimum** and the **global optimum** problems. To address these issues, we propose adding a potential term, whose scaling factor we derive analytically based on the desired properties. We also explore alternative solutions, such as curriculum learning for the local optimum problem, and setting the discount factor $\gamma = 1$ for the global optimum problem.

4.1 Introduction

Reinforcement Learning (RL) holds a unique potential for simulation of human crowds, offering flexibility and power that traditional control or planning algorithms often lack. However, successfully using RL for this purpose brings about new challenges, primarily rooted in the need to design an effective reward function.

The design of the reward function is crucial for the success of RL algorithms in real-world applications. The balance between sparsity and density of rewards has major implications for the performance of these algorithms. Sparse rewards may lead to the standard algorithms not converging in reasonable time. Conversely, overly dense reward could potentially impact the optimal



(a) Circle scenario. (b) Corridor scenario. (c) Crossing scenario. (d) Choke scenario. (e) Car scenario.

Figure 4.1: Agent's initial positions and goals in five scenarios: **(a)** Circle with 40 agents. **(b)** Corridor with 50 agents. **(c)** Crossing with 50 agents. **(d)** Choke with 20 agents. **(e)** Car with 20 agents. In each scenario, agents must reach the goal with the same color as them. In the circle scenario, initial starting positions are randomly perturbed during each episode. In the car scenario, the obstacle at the bottom of the scene moves upwards.

policy and the relative performances of various suboptimal policies. This issue is particularly relevant in the context of simulating human crowds where, apart from clear objectives like navigation and collision avoidance, the goal of reproducing human-like behavior remains somewhat vague.

During locomotion, humans tend to move at a certain comfortable speed that is specific to the individual, usually around 1.3 m/s [179]. Following Guy et al. [45], this is as a result of minimizing the energy expended when moving between two points. In principle, this measure could be used as a reward function for an RL agent to optimize. In practice, however, this tends to be ineffective due to the unique structure of energy minimization, where agents must take short-term negative rewards to obtain long-term positive rewards. The typical solution is designing an artificial reward function, lacking an explicit connection to the energy minimization aspect, but focusing on rewarding movement towards the goal at the right speed.

We propose the development of a more principled reward function that takes into consideration energy efficiency of motion, serving as a proxy for human-likeness. This choice stems from the lack of metrics that specifically quantify human-likeness in existing literature. It is important to note that energy efficiency does not fully describe human behavior, ignoring aspects like long-term goals and subtle inter-personal interactions. Nonetheless, this approach lays the groundwork for more advanced future methods.

We validate our approach both theoretically and empirically. First, we analyze the properties of various reward functions under the discounted utility paradigm. Second, we train RL agents using these reward functions, and compare their performance using the metric of energy usage.

Our contributions are:

1. Physically-based extension of the energy usage model that accounts for acceleration.

2. Evaluation of various reward functions as proxies for energy minimization.

4.2 Energy Usage Model

In this work, we follow the hypothesis of the Principle of Minimum Energy (PME) as stated by Guy et al. [45], according to which humans tend to choose their trajectories based on minimizing the energy usage. Therefore, we use the energy efficiency as the main benchmark for the quality of a given trajectory. While it does not fully describe human-likeness, it is well-defined and easy to estimate with a simple model.

As a starting point, we consider a model of energy usage based on biomechanical research [179], and used as a metric in a number of works concerning crowd simulation [45, 20, 186, 63, 87]. We estimate the energy used in a discrete timestep Δt as:

$$E = (e_s + e_w v^2)\Delta t \tag{4.1}$$

where e_s and e_w are parameters specific to a given person, with typical values of $e_s = 2.23$ and $e_w = 1.26$ in SI units, computed per unit mass [179].

It is important to keep in mind that this model does not account for acceleration or turning, and instead only applies to linear motion. In this case, the optimal velocity (i.e. one that minimizes the energy usage on a fixed straight trajectory) is $v^* = \sqrt{e_s/e_w}$. This value emerges from integrating the energy usage across the entire path – moving too quickly uses too much energy, and moving too slowly extends the duration of the trajectory, also increasing the energy usage.

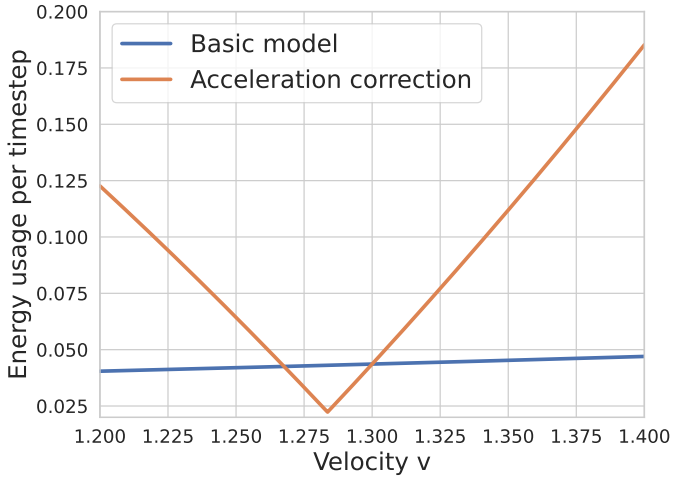


Figure 4.2: Energy used in a single timestep when moving at a velocity of v, after having the velocity of 1.3 m/s in the previous timestep, with $\Delta t = 0.01$ s.

4.2.1 Acceleration correction

In order to improve the energy estimation, we expand the model in Equation 4.1 so that it also considers the acceleration of agents throughout their trajectories. We start by deriving its basic form. Consider a body moving at a constant velocity v, subject to a force opposite to the direction of movement $F_d = -\lambda v$. In Newtonian mechanics, we know that the amount of energy used during displacement is $E = Fs$, where F is the applied force, and s is the distance. To adapt this to our discrete model, we factor out the timestep, obtaining $E = Fv\ \Delta t$. Substituting the force of drag F_d, and setting $\lambda = e_w$ we get:

$$E = -\lambda v^2 \Delta t = -e_w v^2 \Delta t \tag{4.2}$$

This is the energy lost due to drag in each timestep. To counteract it, the agent needs to use energy equal to the absolute value of this quantity. Combining it with with a constant basal energy usage of $e_s dt$, we get $e = e_s dt + e_w v^2 dt$, recovering Equation 4.1.

To extend this reasoning, consider an agent that moves at velocities $\boldsymbol{v}_0$ and $\boldsymbol{v}$ in two consecutive timesteps, that is with an acceleration $\boldsymbol{a} = \frac{\boldsymbol{v}-\boldsymbol{v}_0}{dt}$. Assume that the agent is applying a certain force $\boldsymbol{F}_a$ in an arbitrary direction in order to modify its velocity. Using simple Euler integration, we have:

$$\boldsymbol{v} = \boldsymbol{v}_0 + \boldsymbol{F}\Delta t - e_w \boldsymbol{v}_0 \Delta t = (1 - e_w \Delta t)\boldsymbol{v}_0 + \boldsymbol{F}\Delta t \tag{4.3}$$

Transforming this to obtain the force, we get:

$$\boldsymbol{F} = \frac{1}{\Delta t}(\boldsymbol{v} - (1 - e_w \Delta t)\boldsymbol{v}_0) = \frac{1}{\Delta t}(\boldsymbol{v} - \boldsymbol{v}_0 + e_w \boldsymbol{v}_0 \Delta t) \tag{4.4}$$

From this we can compute the energy usage as follows:

$$\begin{aligned} E &= \boldsymbol{F} \cdot \boldsymbol{v} \Delta t \\ &= \boldsymbol{v} \cdot \boldsymbol{v} - \boldsymbol{v} \cdot \boldsymbol{v}_0 + e_w \boldsymbol{v}_0 \cdot \boldsymbol{v} \Delta t \\ &= \boldsymbol{v} \cdot (\frac{\boldsymbol{v} - \boldsymbol{v}_0}{\Delta t})\Delta t + e_w \boldsymbol{v}_0 \cdot \boldsymbol{v} \Delta t \\ &= (\boldsymbol{v} \cdot \boldsymbol{a} + e_w \boldsymbol{v}_0 \cdot \boldsymbol{v})\Delta t \end{aligned} \tag{4.5}$$

Again taking the absolute value and adding a basal energy usage, we obtain our proposed model for energy usage:

$$E = (e_s + |\boldsymbol{v} \cdot \boldsymbol{a} + e_w \boldsymbol{v}_0 \cdot \boldsymbol{v}|)\,\Delta t \tag{4.6}$$

To better understand Equation 4.6, consider an agent moving with linear acceleration a in the following four cases:

1. Constant motion $a = 0$
2. Acceleration $a > 0 \iff v > v_0$
3. Passive deceleration $0 > a > -e_w v_0 \iff v_0 > v > (1 - e_w \Delta t) v_0$
4. Active deceleration $a < -e_w v_0 \iff v < (1 - e_w \Delta t) v_0$

In the first case $a = 0$, the agent moves at a constant speed $v = ||\boldsymbol{v}_0|| = ||\boldsymbol{v}_1||$. The energy usage is then:

$$E = e_s \Delta t + |0 + e_w v^2| \Delta t = (e_s + e_w v^2) \Delta t \tag{4.7}$$

which agrees with Equation 4.1.

If $a > 0$, the agent increases its movement speed. The energy usage then simplifies to:

$$\begin{aligned} E &= (e_s + av + e_w v_0 v)\Delta t \\ &= (e_s + e_w(v - a\Delta t)v + av)\Delta t \\ &= (e_s + e_w v^2 + (1 - e_w \Delta t)av)\Delta t \\ &\approx e_s \Delta t + e_w v^2 \Delta t + av \Delta t \end{aligned} \tag{4.8}$$

where the term $av\Delta t$ corresponds to the additional kinetic energy needed to move at a velocity v.

If $a < 0$, the agent decelerates. Note, however, that there are two distinct possibilities. If the agent simply stops putting in effort, it will automatically slow down by a factor of $(1 - e_w \Delta t)$. We call any deceleration below this threshold **passive deceleration**, which decreases the energy usage. In contrast, if the agent wants to slow down to a speed lower than $(1 - e_w \Delta t) v_0$, this is **active deceleration**, which requires using additional energy.

We depict this relationship in Figure 4.2. When the velocity remains constant at $v = v_0 = 1.3\,\mathrm{m/s}$, the energy usage is the same in both models. The lowest energy usage (i.e. only from the basal metabolic rate) occurs at $v = (1 - e_w \Delta t) v_0 = 1.28\,\mathrm{m/s}$, when the agent decelerates naturally.

4.3 Navigation reward design

Our main goal in this work is designing a reward function which, when optimized, leads to a policy that minimizes the energy usage, as estimated using the model from Section 4.2. In this section, we discuss a few issues in designing such a reward function.

4.3.1 Energy as reward

A natural starting point is simply using a reward equal to the negative energy usage:

$$R = -e_s \Delta t - e_w v^2 \Delta t \tag{4.9}$$

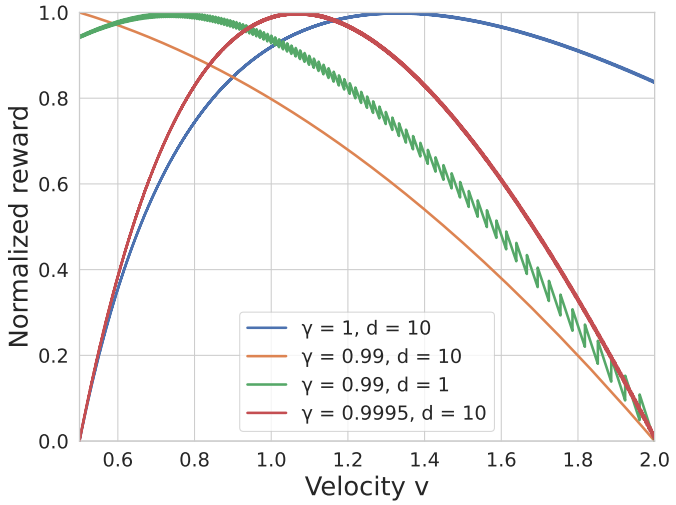


Figure 4.3: Normalized discounted reward, with energy optimization as the direct objective. Depending on the distance d and the discount factor γ, the global optimum is different, and in some cases, the optimal behavior is standing still with $v = 0$.

or

$$R = -e_s \Delta t - |\boldsymbol{v} \cdot \boldsymbol{a} + e_w \boldsymbol{v}_0 \cdot \boldsymbol{v}| \Delta t \tag{4.10}$$

This formulation has two critical issues, which make it unfit for being used as a reward function directly. To see this, consider the base reward of Equation 4.9 for simplicity.

Local optimum

In an RL training procedure, each agent begins by taking random actions. In the case of microscopic crowd simulation, that corresponds to choosing a direction, and setting either the velocity or the acceleration in that direction. If an action leads to a higher reward, its probability increases, and if it leads to a lower reward, its probability decreases.

Consider an agent with a simple objective of moving to a specific location, maximizing the reward from Equation 4.9. The reward is accumulated from the beginning of the episode, until the agent reaches the goal, or until a predefined time limit. Note that with this structure, the real penalty for not reaching the goal is delivered by the agent having to accumulate the negative reward until the time limit. If the time limit is sufficiently high, it is better for the agent to spend some energy in order to reach its goal and not use any energy afterwards, as compared to spending a long time at rest, even without using energy for movement.

However, during training, the agent is more likely to try to move, but fail reaching the goal. It then gets the full time-based penalty, but also a penalty for using additional energy for movement. The agent does not know how to reduce the time-based penalty, but it can decrease its energy usage by slowing down. Eventually, it will settle into a local optimum of standing still, which is a failure case.

Global optimum

The second problem is related to the fact that modern RL algorithms predominantly use the discounted utility paradigm, weighing future rewards with an exponentially decaying discount

factor. Similarly to not reaching the goal, the penalty for moving too slowly is that the agent will have to spend energy in many more timesteps towards the end of the episode. When making a decision at the beginning of the episode, those rewards are heavily discounted, and thus less important.

Consider now the following experiment: the agent travels in a straight line, and has to reach $x = d$ while moving at a constant velocity v. The reward is discounted exponentially with a discount factor γ. In Figure 4.3, we show the discounted reward for some values of d and γ. The global optimum for a typical discount factor around 0.99 is $v = 0$, which corresponds to the agent not moving at all. Depending on the exact values, the optimal value may be anywhere between 0 and $\sqrt{\frac{e_s}{e_w}}$, which is a significant problem if our goal is training an agent whose optimal velocity is exactly $\sqrt{\frac{e_s}{e_w}}$.

The fact that discounting changes the optimal policy is not necessarily unexpected. Naik et al. [118] show that using discounted rewards when training RL agents may change the optimal policy. In many practical problems, this is not a big concern, and the discount factor is treated as yet another hyperparameter. In this case, however, the discount factor directly impacts the properties of the environment.

Possible solutions

There are various ways to tackle the problems described above, but it is important to note that both of them have to be solved together. In order to avoid the local optimum of standing still, we could employ a curriculum-based approach, where the agents initially learn to navigate a short distance without any obstacle. As the training progresses, the distance and the number of agents can be increased, with the hope that the agents will not stop moving.

To fix the issue with the global optimum, the obvious solution is not using any reward discounting. In practice, however, this turns out to be much more unstable and difficult to train. Alternatively, a different non-exponential discounting method could be employed, so that the variance of the gradient estimation is low enough for efficient training, but the optimal velocity remains correct.

Both of these solutions add a non-negligible amount of complexity to the learning algorithm. While in certain situations that might be acceptable, note that all these issues stem from the simple scenario of a single agent navigating to a goal in an energy-efficient manner. With more complicated applications, the complexity is likely to become even higher, e.g. via a curriculum designed for a different objective.

To avoid the compounding complexity, we instead propose changing the reward function. Ideally, it should remain similar to the energy usage so that the emergent behavior is still energy-efficient. It should also tackle both of the aforementioned issues – that is, the reward for moving towards the goal should be higher than for standing still, and the optimal velocity should be invariant under temporal discounting.

4.3.2 Energy-based potential

Adding a guiding potential to the reward function is a common technique of making sparse rewards more dense. Ng et al. [119] show that adding a reward of the form $R(s, a, s') = \gamma\Phi(s') - \Phi(s)$ does not change the optimal policy for the γ-discounted rewards. Note that this assumes that the discounted reward is the true objective of the RL task. This is not true in the case of navigation, as

we generally want the global energy usage to be optimal. Nevertheless, it can serve as inspiration for designing an analogous guiding term.

In the context of human navigation, there is a simple heuristic that we can use as a guiding potential – the distance from the goal. Consider the following reward function:

$$r(\boldsymbol{v}) = -e_s \Delta t - e_w v^2 \Delta t + \tilde{c}_p \boldsymbol{v} \cdot \hat{\boldsymbol{g}} \tag{4.11}$$

where $\hat{\boldsymbol{g}}$ is a unit vector pointing from the agent to the goal. Note that the potential term $\boldsymbol{v} \cdot \hat{\boldsymbol{g}}$ is equal to the change in the distance between the agent and its goal in two consecutive timesteps.

This induces a total discounted reward of:

$$R^\gamma = \int_0^T dt(e^{t \ln \gamma}(-e_s \Delta t - e_w v^2 \Delta t + \tilde{c}_p \boldsymbol{v} \cdot \hat{\boldsymbol{g}})) \tag{4.12}$$

To obtain a bound on the value of c_p, we set the condition that when moving directly towards the goal, $R(v^*) > R(0)$, i.e. it is better to move towards the goal than stand still. This implies that $\tilde{c}_p > \sqrt{e_s e_w}$. For simplicity of further analysis, we define $c_p = \frac{\tilde{c}_p}{\sqrt{e_s e_w}}$.

4.3.3 Discounting invariance

With a simple simulation, it is clear that there is a nontrivial interaction between the values of the discount factor γ, the coefficient c_p, and the optimal velocity v^γ.

Consider the discounted sum of rewards defined in Equation 4.11, with a simple policy of moving towards the goal with a speed v. With a continuous model of the problem, we can define the discounted sum of rewards as:

$$\begin{aligned} R^\gamma &= \int_0^T e^{t \ln \gamma} \left(-e_s - e_w v^2 + c_p \sqrt{e_s e_w} v\right) dt \\ &= \frac{1 - \gamma^{\frac{d}{v}}}{-\ln \gamma} \left(-e_w v^2 + c_p \sqrt{e_s e_w} - e_s\right) \end{aligned} \tag{4.13}$$

We differentiate this expression w.r.t. v to obtain an expression for the optimal velocity, and interpret it as an implicit function whose roots correspond to the optimal velocity with a given discount factor γ:

$$F(v, \gamma) = \frac{(-2e_w v + \tilde{c}_p)\left(1 - \gamma^{\frac{d}{v}}\right)}{-\ln \gamma} - \frac{\left(-e_w v^2 + \tilde{c}_p v - e_s\right) \gamma^{\frac{d}{v}} d}{v^2} = 0 \tag{4.14}$$

Solving this analytically for v is difficult. Instead, we consider the implicit derivative:

$$\frac{dv}{d\gamma} = -\frac{dF}{d\gamma} \Big/ \frac{dF}{dv} \tag{4.15}$$

While the resulting expression is highly complex, it is solvable for c_p analytically, yielding the result:

$$\frac{dv}{d\gamma} = 0 \iff c_p = 2 \tag{4.16}$$

This means that using the reward from Equation 4.11 with $\tilde{c}_p = 2\sqrt{e_s e_w}$, the optimal velocity is independent of the discount factor. Note that if we consider non-exponential discounting as a

weighted sum of exponential discountings, this conclusion extends to other discounting methods, enabling the application of methods like hyperbolic discounting [36] or arbitrary non-exponential discounting [86].

4.3.4 Non-finishing penalty

When measuring the energy usage as a reward function, or even as a metric, there is another consideration that stems from the RL setting – the time limit. While theoretically an agent could infinitely explore until they reach the goal, this is impractical. Instead, RL algorithms typically set a maximum number of timesteps allowed in an episode. After this limit passes, the episode terminates, regardless of the state that the agent is in.

In principle, the value of the time limit should not matter as long as it is sufficient to reach the goal. However, the structure of the energy-based reward (Equations 4.9 and 4.10) makes it potentially impactful. Let T be the time limit in seconds, d the total distance from the goal. Moving in a straight line at the optimal velocity v^*, the time needed to reach the goal is $T^* = \frac{d}{v^*} = \sqrt{\frac{e_w}{e_s}}d$, and the energy used in this process is $2\sqrt{e_s e_w}d$. If this energy is greater than that of standing still until the end of the episode $e_s T$, then the optimal policy according to the metric may indeed be simply standing still.

To prevent this, one option is simply setting the time limit so that $T > 2\frac{d}{v^*}$, in which case moving at the optimal velocity will result in a lower energy usage than standing still until the end of the episode. This corresponds to an episode length more than twice as long as it would take the agent to reach the goal moving at optimal velocity. A significant drawback of this approach is its inefficiency, as the duration of each episode is significantly extended, which increases the amount of time necessary to collect experience for training. Furthermore, complex scenarios with many agents may extend the optimal trajectories in ways that are difficult to predict before training the agents.

Instead we propose two variants of a heuristic that is added as an additional penalty at the end of the episode if a given agent has not reached its goal. In the first variant, we use the **optimal** heuristic – if the agent is at a distance d from its goal, it incurs a penalty of $2\sqrt{e_s e_w}d$, which corresponds to the energy cost it would take to reach the goal moving at the optimal speed in a straight line. In the second variant, instead of using the optimal speed, we use the **average** speed towards the goal across the agent's trajectory to estimate the remaining energy cost.

Both of these variants have their flaws. Using the optimal heuristic, in certain cases it may be beneficial for agents to only move part of the way, and then stop when they encounter a more dense situation, which requires more energy to navigate. While the average heuristic avoids this issue by directly tying the final penalty to the agent's past performance, the estimated velocity has to be capped at a minimum value (in our experiments: 0.1 m/s). This avoids issues where the agent has made very little progress towards the goal (which leads to very high penalties, destabilizing the training), or even made negative progress by moving farther from the goal, leading to a negative energy cost and a positive final reward.

4.3.5 Alternative approaches

In existing literature, most approaches to crowd simulation via RL disregard the problems of energy efficiency, and of encouraging agents to prefer an intermediate velocity throughout their motion. The most common approach to obtain motion with an given velocity v^* is simply setting v^* as

the maximum in the environment dynamics [106, 156, 185, 63]. This is then combined with a guiding potential and a one-time reward for reaching the goal, and due to the incentive structure of the discounted utility paradigm common in RL, this leads to the agents mostly moving at the "optimal" (i.e. maximum) speed. The downside of this approach is that agents are unable to move faster than that predefined limit, in contrast with humans, who tend to easily walk, when needed, a little slower or faster than their optimal comfortable speed.

Other works [93, 186, 87] include a velocity-dependent reward term that incentivizes moving at a specific speed which is below the highest allowed speed. Here we analyze and compare each of those approaches.

Lee et al. [93] use a function they call FLOOD, defined as follows:

$$\begin{aligned} &FLOOD(v, v_{min}, v_{mav}) = \\ =&|\min(v - v_{min}, 0)| + |\max(v - v_{max}, 0)| \end{aligned} \tag{4.17}$$

where v_{min}, v_{max} define the range of comfortable speed. When applied to the linear velocity, this term disincentivizes velocities outside of the preferred range. While this structure is not directly connected with energy optimization, it serves a similar purpose of controlling the movement speed.

A similar structure was used by Xu and Karamouzas [186]. In their reward function, they use a velocity regularization term:

$$r(\boldsymbol{v}) = \exp\left(\sigma_v ||\boldsymbol{v} - \boldsymbol{v}^*||\right) \tag{4.18}$$

where σ_v is a parameter, and $\boldsymbol{v}^*$ is a vector pointing towards the goal, whose magnitude is equal to the optimal velocity. In our energy optimization framework, it is $\boldsymbol{v}^* = v^*\hat{\boldsymbol{g}} = \sqrt{e_s/e_w}\hat{\boldsymbol{g}}$.

Consider the term within the exponent $||\boldsymbol{v} - \boldsymbol{v}^*|| = ||\boldsymbol{v} - v^*\hat{\boldsymbol{g}}||$, and take its square. Interpreting this as a scalar product, we have $(\boldsymbol{v} - v^*\hat{\boldsymbol{g}}) \cdot (\boldsymbol{v} - v^*\hat{\boldsymbol{g}})$. When we multiply the terms, substitute $v^* = \sqrt{e_s/e_w}$ and use the fact that $||\hat{\boldsymbol{g}}|| = 1$, we get $v^2 - 2\sqrt{e_s/e_w}\boldsymbol{v} \cdot \hat{\boldsymbol{g}} + \frac{e_s}{e_w} = \frac{1}{e_w}\left(e_s + e_w v^2 - 2\sqrt{e_s e_w}\boldsymbol{v} \cdot \hat{\boldsymbol{g}}\right)$. This happens to be proportional to the discounting-invariant energy usage with potential. Note, however, that the final reward used by Xu and Karamouzas [186] applies additional operations to this value (square root and exponent).

Kwiatkowski et al. [87] use an explicit potential term, and a speed similarity term $c_v|v - v^*|^{c_e}$ which does not take into account the direction of the movement. With the exponent $c_e = 2$, this expands to $\tilde{c}_p\boldsymbol{v} \cdot \hat{\boldsymbol{g}} - v^2 + 2\sqrt{\frac{e_s}{e_w}}v - \frac{e_s}{e_w}$, which is equal to the energy usage with potential, but with an additional positive term proportional to the agent's speed. This results in a bicycle-like behavior where an agent prefers to artificially extend its trajectory while maintaining its optimal speed, instead of simply slowing down.

4.4 Reward evaluation

In this section, we empirically evaluate our proposed reward structure, and compare it to previously proposed formulations. We also perform an ablation on various parts of the reward function to investigate their importance and impact on the final results.

4.4.1 Experimental setup

We performed the experimental evaluation on five crowd scenarios:

1. Circle – agents start at the perimeter of a circle, and must reach the antipodal point of the circle. We apply noise to both the start and goal positions, and add stationary obstacles in the middle of the circle.
2. Corridor – agents start at two ends of a corridor and must reach the opposite end.
3. Crossing – agents start at southern and western ends of perpendicularly crossed corridors, and must reach the northern and eastern ends, respectively.
4. Choke – agents must pass from west to east through a narrow opening in a wall.
5. Car – agents must wait for a moving obstacle to open a passage to the goal.

In each scenario, all agents are given a time limit of 200 time-steps, each lasting 0.1 s. Each agent is removed from the simulation once it touches its goal. Following the classification by Kwiatkowski et al. [87], we use Egocentric observations with Polar Acceleration dynamics. Each agent has randomly sampled parameters of e_s, e_w as defined in Section 4.2. These values are included in the observation, and used to compute the individual reward of each agent.

The main metric we use for evaluation is Energy+, defined as energy usage with the acceleration correction (Equation 4.8), plus the non-finishing penalty using the average heuristic (Section 4.3.4). The penalty is meant to additionally penalize agents which do not reach their goals in time, to ensure that agents cannot hack the reward function by stopping in the middle of the trajectory.

4.4.2 Reward function structure

Throughout the various reward functions we evaluate in this work, we use the following components:

1. Basal energy usage $r_b = -e_s$
2. Velocity-based energy usage $r_v = -e_w v^2$
3. Dynamics-based energy usage $r_d = -|\boldsymbol{v} \cdot \boldsymbol{a} + e_w \boldsymbol{v}_0 \cdot \boldsymbol{v}|$
4. Guiding potential $r_p = 2\sqrt{e_s e_w}\boldsymbol{v} \cdot \hat{\boldsymbol{g}}$
5. Preferred speed matching $r_s = |v - v^*|^{c_e}$
6. Speeding penalty $r_z = \max(v - v^*, 0)^{c_e}$
7. Exponential velocity matching $r_m = \exp(\sigma_v ||\boldsymbol{v} - \boldsymbol{v}^*||)$
8. Final non-finishing penalty using the optimal speed heuristic (Section 4.3.4) r_o
9. Final non-finishing penalty using the average speed heuristic (Section 4.3.4) r_a
10. One-time goal-reaching reward r_g
11. Constant collision penalty for each frame when an agent collides with another agent or an obstacle r_c

A complete reward function is a weighted sum of a subset of these terms. For terms (1)-(4) and (8)-(9), their coefficients are equal to 1 due to their physics-based formulation. Terms (1)-(3) and (5)-(7) are also multiplied by the duration of the timestep in the simulation.

We primarily focus on evaluating the following reward functions. Note that all of these variants include components (10) and (11) (goal-reaching and collision penalty, respectively)

(a) **Base curriculum** – a curriculum which initially has components (4), (6) (with $c_e = 2$), and after 200 training steps, switches to (1), (3), (4), (9)

(b) **Base curriculum (no acceleration)** – like (a), but using component (2) instead of (3)

(c) **Base curriculum (no heuristic)** – like (a), but without component (9)

(d) **Base curriculum (optimal heuristic)** – like (a), but using component (8) instead of (9)

(e) **Energy (acceleration)** – components (1), (3), (4)

(f) **Energy (no acceleration)** – components (1), (2), (4)

(g) **Energy (no potential)** – components (1), (2)

(h) **Speed matching** – components (4), (5), based on Kwiatkowski et al. [87]

(i) **Speeding penalty** – components (4), (6), based on Lee et al. [93]

(j) **Exponential velocity matching** – component (7), based on Xu and Karamouzas [186]

We trained agents using each of these reward functions, and summarize the results in Section 4.5.

Furthermore, to investigate the importance of the potential term, we also evaluated the following reward functions:

(A) **Base curriculum** – same as reward (a), serving as a baseline

(B) **No potential** – same as (A), but without component (4) (potential)

(C) **No potential and final penalty** – same as (B), but also without component (9) (non-finishing penalty)

(D) **No potential and goal** – same as (B), but also without component (10)

(E) **No potential and goal, optimal heuristic** – same as (D), but with component (10) instead of (9)

(F) **Pure energy** – same as (C), but also without component (10) (goal). The second phase of the curriculum only uses components (1) and (3).

(G) **Pure energy, no discounting** – same as (F), but the discount factor is set to $\gamma = 1$ throughout the training

We describe the results of these experiments in Section 4.5.1.

4.5 Results

While the details differ based on the scenario, in all of them except for the Car scenario, the best-performing reward is a curriculum leading to energy optimization. In the Car scenario, the best-performing reward in terms of the Energy+ metric is directly optimizing energy from the beginning.

The benefit of the curriculum becomes apparent when we consider the progression of the training. We show the success rates in the Circle scenario as a function of the training steps in Figure 4.4.

Table 4.1: Mean value of the Energy+ metric after training in a given scenario, using a given reward function. Each value is based on 8 independent training runs. Lower is better.

	Circle	Crossing	Corridor	Car	Choke
Base curriculum	**58.2 ± 0.54**	66.38 ± 1.39	77.56 ± 6.19	110.95 ± 3.99	94.97 ± 4.03
Base curriculum (no acceleration)	61.62 ± 0.82	72.56 ± 1.14	85.26 ± 6.07	112.81 ± 2.51	112.18 ± 5.49
Base curriculum (no heuristic)	59.18 ± 0.51	**65.81 ± 1.03**	**63.29 ± 0.32**	95.63 ± 8.31	114.78 ± 12.55
Base curriculum (optimal heuristic)	59.17 ± 1.01	67.56 ± 2.12	69.34 ± 2.1	103.76 ± 6.57	**94.53 ± 7.99**
Energy (acceleration)	74.59 ± 2.48	73.55 ± 3.36	96.19 ± 9.02	**85.05 ± 9.36**	105.58 ± 9.09
Energy (no acceleration)	67.1 ± 1.97	81.32 ± 3.26	102.75 ± 5.08	108.32 ± 1.26	106.39 ± 5.43
Energy (no potential)	459.53 ± 53.16	454.01 ± 5.29	450.65 ± 5.06	463.26 ± 1.44	460.28 ± 1.0
Speed matching	60.13 ± 0.71	81.04 ± 5.66	68.35 ± 1.7	126.28 ± 2.03	276.55 ± 14.99
Speeding penalty	58.55 ± 0.87	88.47 ± 4.42	98.71 ± 6.06	119.72 ± 1.27	130.03 ± 5.66
Exponential velocity matching	63.5 ± 1.73	77.18 ± 4.33	85.33 ± 4.91	107.66 ± 0.7	129.9 ± 9.4

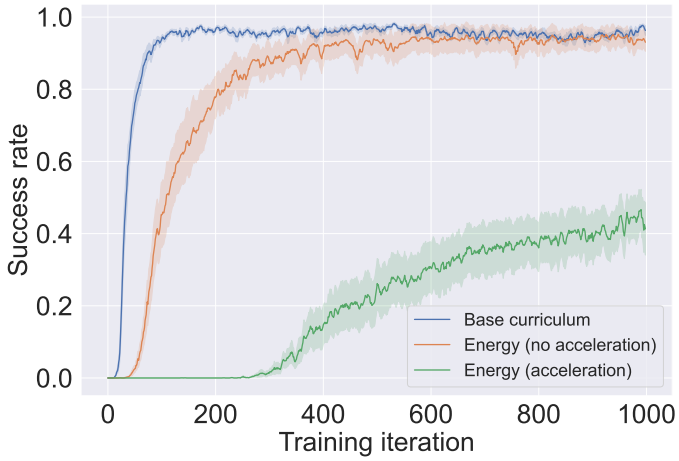


Figure 4.4: Success rates of agents trained with certain reward functions in the Circle scenario.

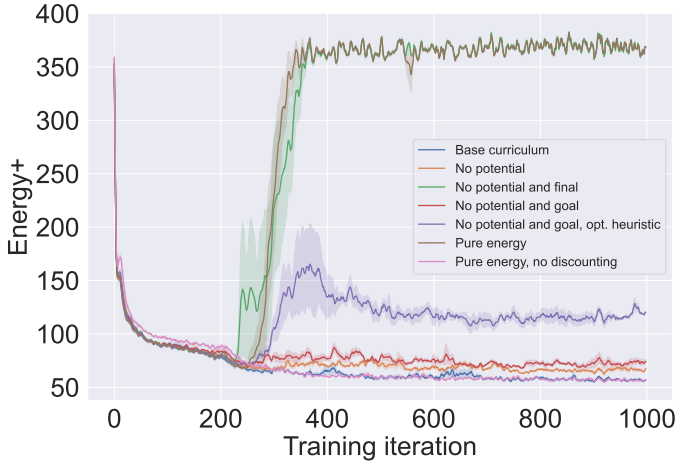


Figure 4.5: Energy+ metric as a function of training progress with various reward functions. To maintain the performance from the first stage of the training, it is necessary to either use a potential term, or set the discount factor to $\gamma = 1$. Agents without a potential or a final heuristic converge to standing still, while other variants' performance significantly degrades.

This scenario has a difficult coordination task embedded in it – when agents travel through the central part of the scene, they must avoid many other agents moving in all directions to prevent collisions. Each collision may lead to additional energy usage in order to resume movement, which effectively increases the collision penalty. Because of this, agents learn the navigation task much more slowly. Conversely, using a simple speeding penalty for the initial part of the training allows the agents to quickly reach a high success rate, which is then maintained after the reward is switched to energy optimization.

On the other hand, in the Car scenario, the best-performing variant is direct energy optimization. This is because agents trained with speeding penalty (as opposed to energy minimization) initially converge to attempting to quickly go in front of the car, passing before it hits them. In contrast, agents trained to minimize energy usage simply wait for the obstacle to pass, or start moving behind it. It is difficult to progressively switch from the former to the latter behavior, so the curriculum fails to produce efficient behavior.

4.5.1 Is potential necessary?

In Section 4.3.1, we provide theoretical justification for why simply optimizing energy is likely to fail. The data in Table 4.1 confirms at least the local optimum argument – directly optimizing the energy usage consistently leads to the worst performance, corresponding to standing still. To empirically validate our global optimum argument, we conducted additional experiments on the Circle scenario, using reward functions (A)-(G).

We show the results in terms of the Energy+ values in Figure 4.5. The **no potential** variant maintains a reasonable performance, but its energy efficiency drops compared to the baseline. Both variants without the final non-finishing penalty (with or without the goal reward – (C) and (F) respectively) rapidly deteriorate to a policy which stays still for the entire duration of the episode. The variants that retain some of their performance are (B) and (D), i.e. ones which still use the average heuristic penalty for not reaching the goal, however their success rate is significantly lower than the baseline. Using the optimal heuristic (E) instead of the average heuristic degrades

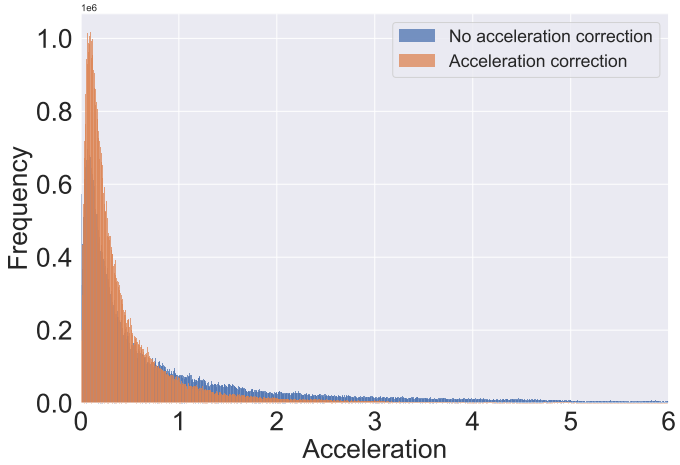


Figure 4.6: Histogram of accelerations in the Circle scenario, trained with and without the acceleration-based term in the reward function.

performance significantly, leading agents to slowly approach the goal, abusing the generous reward they receive at the end of the episode. Finally, using pure energy optimization in a curriculum without a discount factor retains the same performance as the base curriculum.

This confirms that absent of additional goals, with a discount factor of $\gamma = 0.99$, using energy as a reward without a guiding potential fails to converge to a valid policy, even when initialized with a goal-seeking policy trained with a different reward function. This may be mitigated by including a guiding potential, which in some cases enables effective end-to-end training using that reward function. Alternatively, if the training converges without discounting, i.e. with $\gamma = 1$, pure energy may also be a valid approach as a second (or later) stage of a curriculum. This is consistent with the analysis by Naik et al. [118], who describe theoretical problems with the discounted utility paradigm.

4.5.2 Impact of acceleration

In order to evaluate the impact of the acceleration correction to the energy estimation introduced in Section 4.2.1, we compare agents trained with the base curriculum, with and without the acceleration correction. We show the histogram of accelerations, collected across 8 independent training runs in the Circle scenario, in Figure 4.6.

The average magnitude of the acceleration across the trajectories is $0.339\,\mathrm{m/s^2}$ with the acceleration correction in the energy estimation, and $0.679\,\mathrm{m/s^2}$ without it. This result is statistically significant with $p < 0.01$ using the two-sample Kolmogorov-Smirnov test. This shows that including the acceleration in energy estimation successfully leads to smoother behavior. At the same time, the energy usage without the acceleration correction remains similar for both variants – 49.77 ± 1.077 and 50.47 ± 1.284 respectively, indicating that the reduced acceleration does not come at the cost of otherwise less efficient movement.

4.6 Conclusions

In this work, we introduce two contributions: a new, more accurate way to estimate energy usage in the context of crowd simulation, and a novel reward function formulation for training agents

navigating in an energy-efficient manner. We demonstrate a successful curriculum learning approach, where an initial speeding penalty is replaced by a simpler energy optimization formulation in later training stages. This method allows the agents to learn basic navigation first, and then focus on efficiency.

Our experiments on several crowd navigation scenarios show that training using an energy-based reward consistently outperforms other reward functions used in prior work. A critical component of this reward structure is the guiding potential, which ensures that agents navigate towards the goal, and do not simply stay still to minimize the energy usage. We empirically verify this conclusion through additional experiments that exclude this term from the reward function.

Interestingly, in some scenarios, such as the Car scenario, a curriculum approach does not provide any benefits, and agents perform optimally when trained directly with the energy optimization reward. This can be attributed to the specific nature of this scenario, where the initial policy learned by the agents with a speeding penalty makes them rush in front of the moving obstacle, a strategy that contrasts with the more efficient wait-and-follow behavior learned through energy optimization. This highlights the potential need for a more scenario-specific reward formulation or a flexible curriculum training approach that can adjust itself based on the scenario's complexity and nature.

Furthermore, our analysis of discount factor effects on training outcomes with a pure energy reward function aligns with the theoretical discussions raised by Naik et al. [118]. It shows that if the training is conducted without discounting, using energy as a reward without a guiding potential can converge to a valid policy when initialized with a goal-seeking policy trained with a different reward function. This discovery invites future research to explore the utility of different discounting paradigms in such energy optimization tasks and potentially other reinforcement learning applications.

While the current results are promising, several directions remain for future work. The energy estimation for motion with acceleration could be made more accurate by considering the agent's physical model more closely. Additionally, the potential function could be replaced with a more sophisticated heuristic that considers the actual shortest path to the goal, taking into account other agents and obstacles. Another possible direction could be developing an adaptive curriculum that considers the nature of the scenario or the learning progress of the agent. Finally, integrating this energy-efficient approach with social norms and considering more realistic crowd behaviors could lead to generating more realistic behaviors with RL.

Moving on to the final part of this thesis, we think back to one of the problems with using energy as a reward function – the **global optimum** problem, related to the discount factor. A potential solution would be a different, non-exponential discounting method; however, there is no general method that allows us to use non-exponential discounting with modern on-policy RL algorithms like PPO. In the following chapter, we aim to close that gap.

Chapter 5

Non-exponential reward discounting

Once a reward function is chosen, it is up to the RL algorithm to optimize it. That, however, does not mean that we are done. While a superintelligent AGI would, without a doubt, manage to design an agent perfectly optimizing, current RL algorithms are unfortunately not at that level yet (or fortunately – depending on one's views on existential risk due to AI).

To obtain good results via RL training, it is important to stay cognizant of all the finer details of the algorithms in question. In particular, RL algorithms typically follow the discounted utility paradigm, where rewards are considered less important, the further away in the future they are. As we showed in Chapter 4, the usual exponential discounting may lead to undesirable outcomes via global optima in the direct optimization objective.

As a potential way to address this issue, in this chapter, we explore non-exponential discounting mechanisms and introduce a practical way to use arbitrary discounting with modern RL algorithms. We also introduce a simple parametrization of certain non-exponential discounting to enable future theoretical and empirical work on this topic.

5.1 Introduction

Building a Reinforcement Learning (RL) algorithm for time-dependent problems requires specifying a discounting mechanism that defines how important the future is relative to the present. Typically, this is done by setting a discount rate $\gamma \in [0, 1]$ and decaying the future rewards exponentially by a factor γ^t [14]. This induces a characteristic planning horizon, which should match the properties of the environment. In practice, γ is one of the most important hyper-parameters to tune. Small deviations may massively decrease the algorithm's performance, which makes training an RL agent less robust, and more difficult for users to perform a hyper-parameter search with new algorithms and environments.

The choice of a discounting is an example of the bias-variance trade-off. If the discount factor is too low (or correspondingly, the general discounting decreases too rapidly), the value estimate is too biased, and in an extreme case, the agent cannot plan sufficiently far into the future. Conversely, with a discount factor too high, the variance of the value estimation is very large due to the high impact of the distant future, often irrelevant to the decision at hand.

In the literature, a variety of discounting mechanisms have been proposed, from the widely used exponential discounting [155] to hyperbolic discounting, first introduced by psychologists to describe human behavior [3], which we show to be two special cases of our Beta-weighted discounting. Other options include fixed-horizon discounting [89], where all rewards beyond a certain horizon are ignored, or not using any discounting [118]. Any discounting method can also be truncated by setting it to zero for all timesteps after a certain point.

The Generalized Advantage Estimation [145] (GAE) algorithm, which can be seen as an extension of the TD(λ) algorithm, is widely used in training RL agents, but it can only use exponential discounting. This limits the behaviors that we can observe in trained agents with respect to balancing rewards at multiple timescales. To enable using arbitrary discounting methods, we introduce Universal General Advantage Estimation (UGAE) – a vectorized formulation of GAE that accepts any arbitrary discount vectors. We also define a novel discounting method, named Beta-weighted discounting, which is obtained by continuously weighing all exponential discount factors according to a Beta distribution. We show that this method captures both exponential and hyperbolic discounting by properly setting its parameters.

Moreover, we offer an analysis of several exponential and non-exponential discounting methods and their properties. While these methods (except for Beta-weighted discounting) are not new, they can be used in practical RL experiments thanks to UGAE; therefore, it is worthwhile to understand their differences.

As pointed out by Pitis [130], exponential discounting with a constant discount factor fails to model all possible preferences that one may have. While our beta-weighted discounting only introduces a time-dependent discount factor and thus does not solve this problem in its entirety, it enables using more complex discounting mechanisms. Furthermore, our UGAE can serve as a step towards a practical implementation of state-action-dependent discounting.

Finally, we experimentally evaluate the performance of UGAE on a set of RL environments, and compare it to the unbiased Monte Carlo advantage estimation method. We show that UGAE can match or surpass the performance of exponential discounting, without a noticeable increase in computation time. Since it can be seamlessly used with existing codebases (usually by replacing one function), it offers a good alternative to the conventional approach, and enables a large range of future empirical research into the properties of non-exponential discounting.

While currently research into non-exponential discounting is largely limited to toy problems and simple tabular algorithms, our UGAE makes it possible to use arbitrary discounting with state-of-the-art algorithms. It can be used to solve a wide range of problems, including ones with continuous observation and action spaces, and multiagent scenarios, by combining it with algorithms like PPO [144].

In summary, our contributions are twofold: we introduce UGAE, a modification of GAE that accepts arbitrary discounting methods, offering greater flexibility in the choice of a discounting; and we introduce a novel discounting method, named Beta-weighted discounting, which is a practical way of using non-exponential discounting.

5.2 UGAE – Universal Generalized Advantage Estimation

In this section, we introduce the main contribution of this paper, UGAE, which is a way of combining the GAE algorithm [145] with non-exponential discounting methods. Then, we define several discounting methods that will be further explored in this work.

Problem Setting We formulate the RL problem as a Markov Decision Process (MDP) [14]. An MDP is defined as a tuple $\mathcal{M} = (\mathcal{S}, \mathcal{A}, P, R, \mu)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $P\colon S \times A \to \Delta S$ is the transition function, $R\colon S \times A \to \mathbb{R}$ is the reward function and $\mu \in \Delta S$ is the initial state distribution. Note that ΔX represents the set of probability distributions on a given set X. An agent is characterized by a stochastic policy $\pi\colon \mathcal{S} \to \Delta\mathcal{A}$, at each step t sampling an action $a_t \sim \pi(s_t)$, observing the environment's new state $s_{t+1} \sim P(s_t, a_t)$, and receiving a reward $r_t = R(s_t, a_t)$. Over time, the agent collects a trajectory $\tau = \langle s_0, a_0, r_0, s_1, a_1, r_1, \dots \rangle$, which may be finite (episodic tasks) or infinite (continuing tasks).

The typical goal of an RL agent is maximizing the total reward $\sum_{t=0}^{T} r_t$, where T is the duration of the episode (potentially infinite). A commonly used direct objective for the agent to optimize is the total discounted reward $\sum_{t=0}^{T} \gamma^t r_t$ under a given discount factor γ [160]. Using a discount factor can serve as a regularizer for the agent [8], and is needed for continuing tasks ($T = \infty$) to ensure that the total reward remains finite.

In this work, we consider a more general scenario that allows non-exponential discounting mechanisms defined by a function $\Gamma^{(\cdot)}\colon \mathbb{N} \to [0, 1]$. The optimization objective is then expressed as $R^\Gamma = \sum_{t=0}^{\infty} \Gamma^{(t)} r_t$.

5.2.1 UGAE

The original derivation of GAE relies on the assumption that the rewards are discounted exponentially. While the main idea remains valid, the transformations that follow and the resulting implementation cannot be used with a different discounting scheme.

Recall that GAE considers multiple k-step advantages, each defined as:

$$\hat{A}_t^{(k)} = -V(s_t) + \sum_{l=0}^{k-1} \gamma^l r_{t+l} + \gamma^k V(s_{t+k}). \tag{5.1}$$

Given a weighing parameter λ, the GAE advantage is then:

$$\begin{aligned} \hat{A}_t^{GAE(\gamma,\lambda)} &:= (1-\lambda)(\hat{A}_t^{(1)} + \lambda \hat{A}_t^{(2)} + \dots) \\ &= \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V \end{aligned} \tag{5.2}$$

where $\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$. While this formulation makes it possible to compute all the advantages in a dataset with an efficient, single-pass algorithm, it cannot be used with a general discounting method. In particular, δ_t^V cannot be used as its value depends on which timestep's advantage we are computing.

To tackle this, we propose an alternative expression using an arbitrary discount vector $\Gamma^{(t)}$. To this end, we redefine the k-step advantage using this concept, as a replacement for Equation 5.1:

$$\tilde{A}_t^{(k)} = -V(s_t) + \sum_{l=0}^{k-1} \Gamma^{(l)} r_{t+l} + \Gamma^{(k)} V(s_{t+k}) \tag{5.3}$$

We then expand it to obtain the equivalent to Equation 5.2:

$$\tilde{A}_t^{UGAE(\Gamma,\lambda)} := (1-\lambda)(\tilde{A}_t^{(1)} + \lambda\tilde{A}_t^{(2)} + \dots) \tag{5.4}$$
$$= -V(s_t) + \sum_{l=0}^{\infty} \lambda^l \Gamma^{(l)} r_{t+l}$$
$$+ (1-\lambda)\sum_{l=0}^{\infty} \Gamma^{(l+1)} \lambda^l V(s_{t+l+1})$$

Note that the second and third terms are both sums of products, and can therefore be interpreted as scalar products of appropriate vectors. By defining $\boldsymbol{r}_t = [r_{t+i}]_{i\in\mathbb{N}}$, $\boldsymbol{V}_t = [V(s_{t+i})]_{i\in\mathbb{N}}$, $\boldsymbol{\Gamma} = [\Gamma^{(i)}]_{i\in\mathbb{N}}$, $\boldsymbol{\Gamma'} = [\Gamma^{(i+1)}]_{i\in\mathbb{N}}$, $\boldsymbol{\lambda} = [\lambda^i]_{i\in\mathbb{N}}$, we rewrite Equation 5.4 in a vectorized form in Equation 5.5. We use the notation that $\boldsymbol{x}\odot\boldsymbol{y}$ represents the Hadamard (element-wise) product, and $\boldsymbol{x}\cdot\boldsymbol{y}$ – the scalar product.

Theorem 1. ***UGAE: GAE with arbitrary discounting***

Consider $\boldsymbol{r}_t, \boldsymbol{V}_t, \boldsymbol{\Gamma}, \boldsymbol{\Gamma'}, \boldsymbol{\lambda}, \lambda$ defined as above. We can compute GAE with arbitrary discounting as:

$$\tilde{A}_t^{UGAE(\Gamma,\lambda)} := \tag{5.5}$$
$$:= -V(s_t) + (\boldsymbol{\lambda}\odot\boldsymbol{\Gamma})\cdot\boldsymbol{r}_t + (1-\lambda)(\boldsymbol{\lambda}\odot\boldsymbol{\Gamma'})\cdot\boldsymbol{V}_{t+1}$$

*If $\Gamma^{(t)} = \gamma^t$, this is equivalent to the standard GAE advantage. **Proof** is in the supplemental material.*

Discussion. Theorem 1 gives a vectorized formulation of GAE. This makes it possible to use GAE with arbitrary discounting methods with little computational overhead, by leveraging optimized vector computations.

Note that while the complexity of exponential GAE computation for an entire episode is $O(T)$ where T is the episode length, the vectorized formulation increases it to $O(T^2)$ due to the need of multiplying large vectors. Fortunately, truncating the future rewards is trivial using the vectorized formulation, and that can be used through truncating the discounting horizon, by setting a maximum length L of the vectors in Theorem 1. The complexity in that case is $O(LT)$, so again linear in T as long as L stays constant. In practice, as we show in this paper, the computational cost is not of significant concern, and the truncation is not necessary, as the asymptotic complexity only becomes noticeable with unusually long episode lengths.

5.2.2 Added estimation bias

An important aspect of our method is the additional bias it introduces to the value estimation. To compute a k-step advantage, we must evaluate the tail of the reward sequence using the discounting itself (the $V(s_{t+k})$ term in Equation 5.1). This is impossible with any non-exponential discounting, as the property $\Gamma^{(k+t)} = \Gamma^{(k)}\Gamma^{(t)}$ implies $\Gamma^{(\cdot)}$ being an exponential function. Seeing as we are performing an estimation of the value of those last steps, this results in an increase in the estimation bias compared to Monte Carlo estimation.

This clearly ties into the general bias-variance trade-off when using GAE or TD-lambda estimation. In its original form, it performs interpolation between high-variance (Monte Carlo) and high-bias (TD) estimates for exponential discounting. In the case of non-exponential discounting,

using UGAE as opposed to Monte Carlo estimates has the same effect of an increase in bias, but decreasing the variance in return.

The difference between the non-exponential discounting and its tail contributes to an additional increase of bias beyond that caused by using GAE, but we show that this bias remains finite in the infinite time horizon for summable discountings (including our βGAE as well as any truncated discounting).

Theorem 2. ***UGAE added bias***

Consider an arbitrary summable discounting $\Gamma^{(t)}$. The additional bias, defined as the discrepancy between the UGAE and Monte Carlo value estimations, is finite in the infinite time horizon. ***Proof*** *is in the supplemental material.*

In practice, as we show in our experiments, the decreased variance enabled by UGAE effectively counteracts the added bias, resulting in an overall better performance over Monte Carlo estimation.

5.2.3 Non-exponential discounting

Beta-weighted Discounting is described in the following section. Exponential and hyperbolic discountings are equivalent to Beta-weighted discounting with η=0 and η=1, respectively. The former is given by $\Gamma^{(t)}=\mu^t$ with $\mu\in[0,1]$, and the latter by $\Gamma^{(t)}=\frac{1}{1+kt}=\frac{1}{1+\frac{1-\mu}{\mu}t}$ parametrized by $k\in[0,\infty)$ or $\mu\in(0,1]$.

No Discounting implies that the discount vector takes the form $\forall_{t\in\mathbb{N}}\Gamma^{(t)}=1$. This is non-summable, but it is trivial to compute any partial sums to estimate the importance of future rewards or the variance. The effective planning horizon depends on the episode length T, and is equal to $(1-\frac{1}{e})T$.

Fixed-Horizon Discounting Here, there is a single parameter T_{max} which defines how many future rewards are considered with a constant weight. The discount vector is then $\Gamma^{(t)}=\mathbf{1}_{t<T_{max}}$, and the planning horizon, according to our definition, is $(1-\frac{1}{e})T_{max}$.

Truncated Discounting All discounting methods can be truncated by adding an additional parameter T_{max}, and setting the discount vector to 0 for all timesteps $t > T_{max}$. Truncating a discounting decreases the importance of future rewards and the effective planning horizon, but also decreases the variance of the total rewards.

5.3 Beta-weighted discounting

In this section, we present our second contribution, i.e. Beta-weighted discounting. It uses the Beta distribution as weights for all values of $\gamma\in[0,1]$. We use their expected value as the effective discount factors $\Gamma^{(t)}$, which we show to be equal to the distribution's raw moments.

5.3.1 Beta-weighted discounting

As a simple illustrative example, let us use two discount factors γ_1, γ_2 with weights p and $(1-p)$ respectively, where $p\in[0,1]$. This can be treated as a multiple-reward problem [147] where the total reward is a weighted sum of individual rewards $R^\gamma=\sum_{t=0}^{\infty}\gamma^t r_t$. Therefore, we have:

$$R^{(\gamma_1,\gamma_2)} = p\sum \gamma_1^t r_t + (1-p)\sum \gamma_2^t r_t \\ = \sum r_t(p\gamma_1^t + (1-p)\gamma_2^t) \tag{5.6}$$

We extend this reasoning to any countable number of exponential discount factors with arbitrary weights, that sum up to 1. Taking this to the continuous limit, we also consider continuous distributions of discount factors $w \in \Delta([0,1])$, leading to the equation:

$$R^w = \sum r_t \int_0^1 w(\gamma)\gamma^t dt = \sum r_t \Gamma^{(t)} \tag{5.7}$$

An important observation is that as long as $\text{supp}(w) \subseteq [0,1]$, the integral is by definition equal to the t-th raw moment of the distribution w. Hence, with an appropriately chosen distribution an analytical expression is obtained for all its moments, and therefore, all the individual discount factors $\Gamma^{(t)}$.

We choose the Beta distribution due to the simple analytical formula for its moments, as well as its relation to other common discounting methods. Its probability density function is defined as $f(x;\alpha,\beta) \propto x^{\alpha-1}(1-x)^{\beta-1}$. Note that with $\beta = 1$, it is equivalent to the exponential distribution which induces a hyperbolic discounting. Its moments are known in analytical form [71], which leads to our proposed discounting mechanism in Theorem 3.

Theorem 3. ***Beta-weighted discounting***

Consider $\alpha, \beta \in [0,\infty)$. The following equations hold for the Beta-weighted discount vector parametrized by α, β. ***Proof*** *is in the supplementary material.*

$$\Gamma^{(t)} = \prod_{k=0}^{t-1} \frac{\alpha+k}{\alpha+\beta+k} \tag{5.8}$$

$$\Gamma^{(t+1)} = \frac{\alpha+t}{\alpha+\beta+t}\Gamma^{(t)} \tag{5.9}$$

5.3.2 Beta distribution properties

Here, we investigate the Beta distribution's parameter space and consider an alternative parametrization that eases its tuning. We also analyze important properties of the Beta-weighted discounting based on those parameters, and compare them to exponential and hyperbolic baselines.

Canonically, the Beta distribution is defined by parameters $\alpha, \beta \in (0,\infty)$. It is worth noting certain special cases and how this approach generalizes other discounting methods. When $\alpha, \beta \to \infty$ such that its mean $\mu := \frac{\alpha}{\alpha+\beta} = const.$, the beta distribution asymptotically approaches the Dirac delta distribution $\delta(x-\mu)$, resulting in the usual exponential discounting $\Gamma^{(t)} = \mu^t$. Alternatively, when $\beta = 1$, we get $\Gamma^{(t)} = \prod_{k=0}^{t-1} \frac{\alpha+k}{\alpha+k+1} = \frac{\alpha}{\alpha+t} = \frac{1}{1+t/\alpha}$, i.e. hyperbolic discounting.

Mean μ and dispersion η A key property is that we would like the effective discount rate to be comparable with existing exponential discount factors. To do so, we define a more intuitive parameter to directly control the distribution's mean as $\mu = \frac{\alpha}{\alpha+\beta} \in (0,1)$. μ defines the center of the distribution and should therefore be close to typically used γ values in exponential discounting.

A second intuitive parameter should control the dispersion of the distribution. Depending on the context, two choices seem natural: β itself, or its inverse $\eta = \frac{1}{\beta}$. As stated earlier, β can take any positive real value. By discarding values of $\beta < 1$ which correspond to a local maximum of

the probability density function around 0, we obtain $\eta \in (0, 1]$. That way we obtain an easy-to-interpret discounting strategy. Indeed, as we show in Lemma 4, $\eta \to 0$ and $\eta = 1$ correspond to exponential discounting, and hyperbolic discounting, respectively, which allows us to finally define the range of η as $[0, 1]$. Other values smoothly interpolate between both of these methods, similar to how GAE interpolates between Monte Carlo and Temporal Difference estimation.

Given the values of μ and η, the original distribution parameters can be recovered as $\alpha = \frac{\mu}{\eta(1-\mu)}$ and $\beta = \frac{1}{\eta}$. The raw moments parametrized by μ and η are $m_t = \prod_{k=0}^{t-1} \frac{\mu+k\eta(1-\mu)}{1+k\eta(1-\mu)}$.

Lemma 4. ***Special cases of Beta-weighted discounting***

We explore the relation of Beta-weighted to exponential and hyperbolic discountings. Consider the Beta-weighted discounting $\Gamma^{(t)}$ parametrized by $\mu \in (0, 1), \eta \in (0, 1]$. The following is true:

- *if $\eta \to 0$, then $\Gamma^{(t)} = \mu^t$, i.e. it is equal to exponential discounting*
- *if $\eta = 1$, then $\Gamma^{(t)} = \frac{\mu}{\mu+(1-\mu)t} = \frac{1}{1+t/\alpha}$, i.e. it is equal to hyperbolic discounting*

***Proof** is in the supplemental material.*

Discussion. Beta-weighted discounting is controlled by two parameters (u, η), which includes the classic exponential discounting, but also enables more flexibility in designing agent behaviors .

Lemma 5. ***Beta-weighted discounting summability***

*Given the Beta-weighted discount vector $\Gamma^{(t)} = \prod_{k=0}^{t-1} \frac{\alpha+k}{\alpha+\beta+k}$, $\alpha \in [0, \infty)$, $\beta \in [0, \infty)$, the following property holds. **Proof** is in the supplemental material.*

$$\sum_{t=0}^{\infty} \Gamma^{(t)} = \begin{cases} \frac{\alpha+\beta-1}{\beta-1} & \text{if } \beta > 1 \\ \infty & \text{otherwise} \end{cases} \tag{5.10}$$

Discussion. Lemma 5 describes the conditions under which the Beta-weighted discounting is summable depending on its parameters. While less critical for episodic tasks, summability of the discount function is crucial for continuing tasks. Otherwise, the discounted reward can grow arbitrarily high over time.

5.4 Analysis of non-exponential discounting methods

Here, our goal is to justify the usage, and enable deep understanding of different discounting methods. To this end, we first analyze some of their main properties: the importance of future rewards, the variance of the discounted rewards, the effective planning horizon, and the total sum of the discounting. Then, we compare those properties among the previously described discounting methods.

Since not all discounting methods are summable (particularly the cases of hyperbolic and no discounting), we consider the maximum ("infinite") episode length to be 10000 steps. We focus on a characteristic time scale of the environment around 100 steps.

5.4.1 Properties of discounting

Importance of future rewards Properly describing the influence of the future under a specific discounting is challenging. On one hand, individual rewards are typically counted with a smaller weight, as discount vectors $\Gamma^{(t)}$ are usually monotonically decreasing. On the other hand, the longer

the considered time horizon is, the more timesteps it includes, increasing its overall importance. Furthermore, a long time horizon (e.g. 100 steps) directly includes a shorter horizon (e.g. 10 steps), and therefore, the partial sums are not directly comparable. To balance these aspects, we focus on the importance of the first 100 steps using the following expressions:

$$\Gamma_{t_1}^{t_2} = \frac{\sum_{t=t_1}^{t_2} \Gamma^{(t)}}{\sum_{t=0}^{\infty} \Gamma^{(t)}} \tag{5.11}$$

Variance of the discounted rewards The overall objective of the RL agent is maximizing the total (discounted) reward it obtains in the environment. Since both the policy and the environment can be stochastic, the total reward will also be subject to some uncertainty. While the exact rewards, as well as their variances, depend heavily on the exact environment, we make the simplifying assumption that the instantaneous rewards r_t are sampled according to a distribution D with a constant variance of σ^2, e.g. $r_t \sim D = \mathcal{N}(\mu, \sigma^2)$ with an arbitrary, possibly varying, μ. We also assume all rewards to be uncorrelated, which leads to the following expression:

$$\begin{aligned} \mathrm{Var}[\sum_{t=0}^{T} \Gamma^{(t)} r_t] &= \sum_{t=0}^{T} {\Gamma^{(t)}}^2 \mathrm{Var}[r_t] \\ &= \sum_{t=0}^{T} {\Gamma^{(t)}}^2 \sigma^2 = \sigma^2 \sum_{t=0}^{T} {\Gamma^{(t)}}^2 \end{aligned} \tag{5.12}$$

Equation 5.12 shows that the variance of the total discounted reward is proportional to the sum of all the squares of discount factors. While in some cases it is easy to obtain analytically, quite often the expression can be complex and difficult to obtain and analyze; hence, we consider the numerical values, as well as analytical expressions where applicable.

Effective planning horizon For any discounting $\Gamma^{(t)}$, our goal is to have a measure of its effective planning horizon. However, in most cases, we cannot have a clear point beyond which the rewards do not matter and there is not a unique notion of a time horizon that could be specified. Thus, to maintain consistency with the standard notion of a time horizon from exponential discounting, we define the effective time horizon as the timestep T_{eff} after which approximately $\frac{1}{e} (\approx 37\%)$ of the weight of the discounting remains.

Total sum of the discounting Depending on the RL algorithm and the reward normalization method (if any), the magnitude of the total discounted reward might impact the stability of the training, as neural networks typically cannot deal with very large numbers. For this reason, we consider the sum of all rewards under a given discounting.

Consistency It is worth keeping in mind that, as pointed out by Lattimore and Hutter [89], the only time consistent discounting is the exponential discounting. This means that it is possible that other methods cause the agent to change its plans over time. While this has the potential to significantly degrade the performance in some specific environments, that is often not the case in typical RL tasks, as we show in Section 5.5.

5.4.2 Experimental Analysis

To analyze the properties of different discounting methods, we compute them for a set of relevant discounting methods and their parameters. The impact of Beta discounting's η is illustrated in

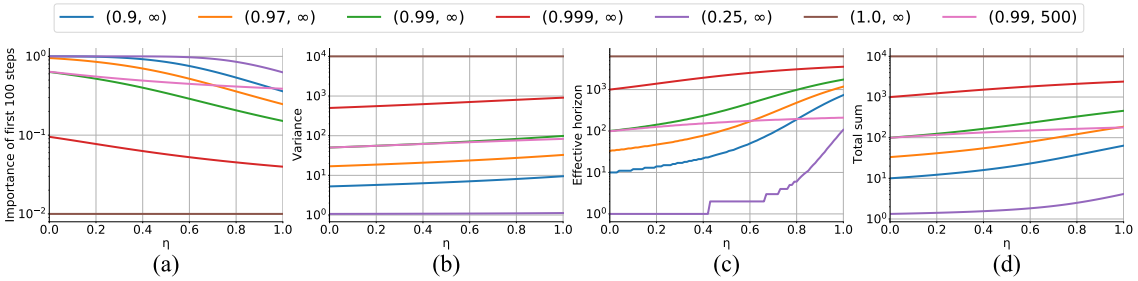


Figure 5.1: Different properties of a discounting, as a function of η, with given (μ, T_{max}) parameters listed in the legend. (a) Importance of the near future (b) Variance measure (c) Effective time horizon (d) Total discounting sum

Figure 5.1(a-d), showing: (a) the importance of first 100 steps Γ_0^{100}, (b) the variance measure, (c) the effective time horizon, (d) the total sum of the discounting – full results are in the supplement. Note that the choice of the discounting is an example of the common bias-variance trade-off. If the real objective is maximizing the total undiscounted reward, decreasing the weights of future rewards inevitably biases the algorithm's value estimation, while simultaneously decreasing its variance.

Using **no discounting**, the rewards from the distant future have a dominant contribution to the total reward estimate since more steps are included. **Exponential discounting** places more emphasis on the short term rewards, according to its γ parameter, while simultaneously decreasing the variance; when $\gamma = 0.99$, it effectively disregards the distant future of $t > 1000$.

In **Beta-weighted discounting** with $\eta > 0$, the future rewards importance, the variance and the effective time horizon increase with η. If μ is adjusted to make T_{eff} similar to exponential discounting's value, the variance decreases significantly, and the balance of different time horizons shifts towards the future, maintaining some weight on the distant future.

With **hyperbolic discounting** (Beta-weighted with $\eta = 1$) the distant future becomes very important, and the effective time horizon becomes very large (in fact, infinite in the limit of an infinitely long episode). To reduce the time horizon to a value close to 100, its μ parameter has to be very small, near $\mu = 0.25$, putting most of the weight on rewards that are close in time, but also including the distant rewards unlike any exponential discounting.

The behavior of **fixed-horizon discounting** is simple – it acts like no discounting, but ignores any rewards beyond T_{max}. Truncating another discounting method results in ignoring the rewards beyond T_{max}, and decreasing the variance and the effective time horizon (see supplemental material).

In summary, by modifying η in Beta-weighted discounting, and T_{max} in the truncated scenario, we can successfully change various aspects of the resulting discounting in a more flexible way than with exponential discounting. In general, changing the discounting method can bias the agent to favor certain timescales and focus on maximizing the rewards that occur within them.

5.4.3 Discussion

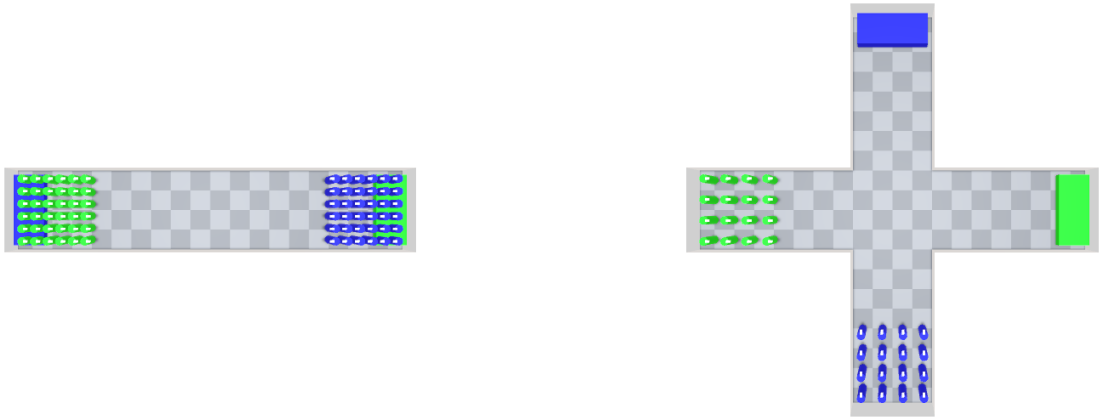
When η is increased, the importance shifts towards the future (Figure 5.1a), variance (Figure 5.1b) and the effective horizon (Figure 5.1c) increase. Introducing a truncation (pink line) decreases the effective horizon and shifts the reward importance towards the near rewards. When truncated at 100 steps, Beta-weighted discounting with $\eta = 0.5$ and hyperbolic discounting ($\eta = 1$) have very

similar properties, indicating their main difference lies in how they deal with the distant future. This is confirmed by comparing the non-truncated versions, where hyperbolic discounting puts a very large weight on the distant future, unlike the Beta-weighted discounting.

5.4.4 Why non-exponential discounting?

A natural question that arises during this discussion is – why do we even want to train agents with non-exponential discounting? As described by Naik et al. [118], optimizing a discounted reward is not equivalent to optimizing the reward itself. The choice of a discounting method affects the optimal policy, or even its existence in the first place. While we do not tackle this problem in this work, we enable larger flexibility in designing the discounting mechanism. This in turn allows researcher to generate more diverse emergent behaviors through the choice of an appropriate discounting – in case that exponential discounting leads to undesirable results.

As mentioned earlier, any discounting other than exponential has the potential for inconsistent behavior. This means that the agent may change its mind on a decision as time passes, without any additional changes to the situation. While this behavior is admittedly irrational, it is commonly exhibited by humans [3]. Therefore, it is important to take this into consideration when creating agents meant to mimic human behavior in applications like video games or social robotics, where human-likeness is important, and potentially not appropriately reflected in the reward function.



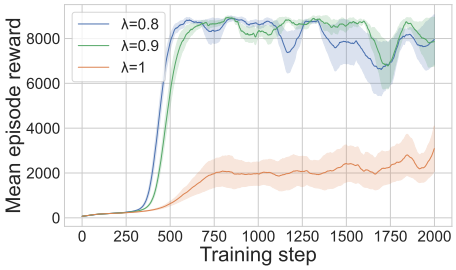
(a) Corridor (b) Crossway

Figure 5.2: Visualizations of the two crowd simulation scenarios used in the experiments. In both cases, each agent needs to reach the opposite end of their respective route, and is then removed from the simulation.

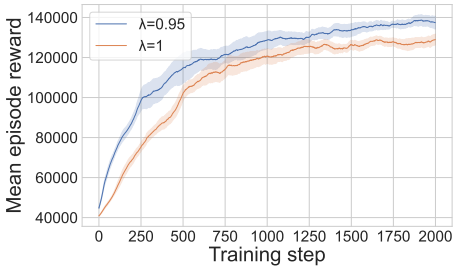
5.5 DRL Experiments

In this section, we evaluate our UGAE for training DRL agents with non-exponential discounting, in both single-agent and multiagent environments. As the baseline, we use non-exponential discounting with regular Monte Carlo (MC) advantage estimation, equivalent to UGAE with $\lambda = 1$. We use Beta-weighted discounting to parametrize the non-exponential discounting with its η value.

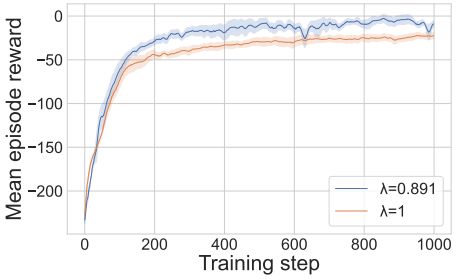
We use four test environments to evaluate our discounting method: InvertedDoublePendulum-v4 and HumanoidStandup-v4 from MuJoCo via Gym [166, 19]; Crossway and Corridor crowd



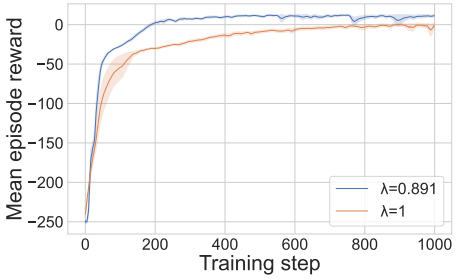
(a) Inverted Double Pendulum, $\eta = 0.8$.



(b) Humanoid Standup, $\eta = 0.5$.



(c) Crowd Crossing, $\eta = 0.8$.



(d) Crowd Corridor, $\eta = 0.5$.

Figure 5.3: Training curves in DRL experiments using non-exponential discounting. All curves are averaged across 8 independent training runs. Shading indicates the standard error of the mean. In all experiments, using λ values that were tuned for optimality with exponential discounting, significantly outperform the MC baseline ($\lambda = 1$). This indicates that UGAE enables translating the benefits of GAE to non-exponential discounting.

simulation scenarios with 50 homogeneous agents each, introduced by Kwiatkowski et al. [87]. The crowd scenarios are displayed in Figure 5.2. We chose these environments because their optimal λ values with exponential GAE are relatively far from $\lambda = 1$ based on prior work. In environments where GAE does not provide benefit over MC estimation, we similarly do not expect an improvement with non-exponential discounting.

Inverted Double Pendulum and Humanoid Standup are both skeletal control tasks with low- and high-dimensional controls, respectively. The former has an 11-dimensional observation space and 1-dimensional action space, whereas the latter has a 376-dimensional observation space, and a 17-dimensional action space. The crowd simulation scenarios use a hybrid perception model combining raycasting to perceive walls, and direct agent perception for neighboring agents for a total of 177-dimensional vector observation and 4-dimensional embedding of each neighbor, as described in Kwiatkowski et al. [87]. They use a 2-dimensional action space with polar velocity dynamics. The episode length is 1000 for MuJoCo experiments, and 200 for crowd simulation experiments.

We train the agents with the PPO algorithm, with hyperparameters based on the RL Baselines Zoo [133] for the MuJoCo environments, and from Kwiatkowski et al. [87] for the crowd environments. It is worth noting that the MuJoCo hyperparameters have been tuned for a prior version of the environments (v3), and thus the results can be different. We use the optimal value of λ in the exponential discounting paradigm, and apply it analogously with UGAE. A single training takes 2-5 hours with a consumer GPU.

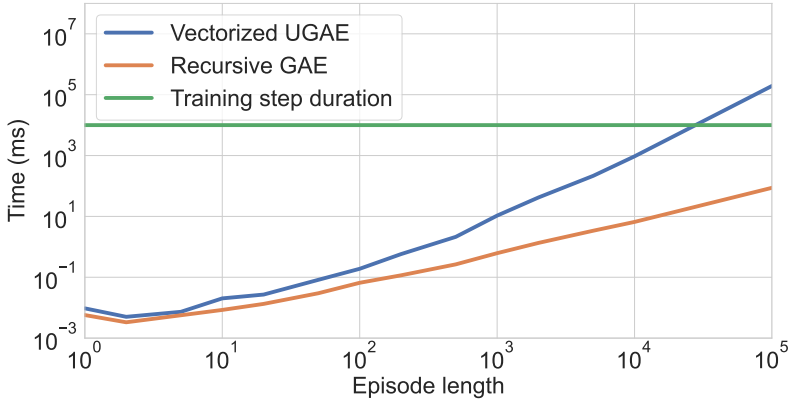


Figure 5.4: Time needed to compute GAE (orange) and UGAE (blue) with a single consumer CPU, on log-log scale. The green line is a reference duration of 10 seconds representing a typical training iteration. While UGAE is more expensive, with typical training step durations, the time to compute its values is negligible.

5.5.1 Results

We show the results in Figure 5.3. In all tested environments, training with UGAE leads to a higher performance compared to the MC baseline, with the largest effect being present in the Inverted Double Pendulum where UGAE achieves a mean episode reward of **8213** $\pm$ **1067**, while MC only achieves **3364** $\pm$ **1078**. The effect is smaller in the Humanoid Standup task, but still significant, with the final rewards being **137300** $\pm$ **3400** and **129000** $\pm$ **2520** respectively. In the crowd scenarios, a more detailed analysis of the emergent behaviors indicates that agents trained with MC fail to maintain a comfortable speed (which is part of the reward function), while UGAE agents are able to efficiently navigate to their goals. This results in rewards of **11.2** $\pm$ **1.457** for UGAE and **-0.156** $\pm$ **2.745** for MC in the Corridor scenario; and **-1.46** $\pm$ **1.50** for UGAE and **-35.15** $\pm$ **8.81** for MC in the Crossway scenario.

5.5.2 Computation time

To estimate the computational impact of our vectorized UGAE formulation as compared to the standard recursive GAE, we generate 16 random episodes with a length between 1 and 100,000 steps, and plot the time needed to compute the advantages as a function of the episode length. The results are in Figure 5.4. For a reference duration of a full training step, we use 10 seconds. It shows that while the computational cost of UGAE (blue) is larger than that of GAE (orange line), it remains insignificant compared to a full training step with episodes shorter than 10^4 steps. For longer episodes, it becomes noticeable, however, this is rarely the case in practice.

5.5.3 Discussion

As our experiments show, using UGAE with episodes of length up to ca. 10^3 steps carries a negligible computational cost, allowing its seamless integration into a PPO training pipeline. At the same time, it enables a performance improvement mirroring that of GAE, but for non-exponential discounting. In conjunction with Beta-weighted discounting, it enables practical and efficient training of agents with non-exponential discounting.

The main limitation of our work lies in the asymptotic complexity of advantage computation.

The time needed to compute the UGAE advantage is negligible with episodes up to around 10^3 steps, and becomes noticeable (although still not overwhelmingly so) at around 10^4 steps. In the rare scenario one needs to compute the advantages for episodes over 10^4 steps, the computation may become too expensive and require truncating the discounting.

Our beta-weighted discounting adds a new hyperparameter, which is a potential challenge, as RL algorithms typically already have a large number of hyperparameters that must be optimized. However, due to the interpretation of η, there is a natural default value of $\eta = 0$ which corresponds to exponential discounting. With the other extreme being $\eta = 1$, this yields a compact range of possible values that can be easily included in a hyperparameter optimization procedure. This also opens the door to further research on automatically tuning the discount factor from a wider family of possibilities as opposed to just exponential methods.

5.6 Conclusions

Our work follows the exciting trend of rethinking the discounting mechanism in RL. In typical applications, our UGAE can be used with negligible overhead, and together with Beta-weighted discounting they provide an elegant way to perform efficient non-exponential discounting. To our knowledge, UGAE is the first method that enables using arbitrary discounting mechanisms in Actor-Critic algorithms. Our experiments show that using non-exponential discounting gives more flexibility in the temporal properties of the RL agent, and thus enables more diverse emergent behaviors. Importantly, this work makes it possible for researchers to empirically investigate different methods of discounting and their relation with various RL problems, including state-dependent discounting. A challenging but valuable contribution would be developing a method to analyze the properties of an environment, and relating them to the ideal discounting method. Finally, developing an analogous method for value-based algorithms like DQN or DDPG would make it possible to use arbitrary discounting with even more state-of-the-art RL algorithms.

Chapter 6

Conclusion

While Reinforcement learning is a powerful tool for simulating human-like crowds, many challenges remain. Designing an appropriate simulation, and applying the appropriate learning algorithm to it, still requires a fair amount of work and expertise. Nevertheless, we introduced four contributions that simplify this line of work for future research:

1. A detailed literature review, providing an understanding of the state of the art of character animation with RL (Chapter 2).
2. An analysis of observation and action spaces in a crowd simulation system, providing an understanding of the basic design choices for the underlying simulation (Chapter 3).
3. A theoretical and empirical investigation into the properties of reward functions, providing an understanding of the reward design for human-like crowds (Chapter 4).
4. A method for non-exponential discounting, providing a practical method to improve the flexibility of reward optimizing algorithms (Chapter 5).

Simulation design As a basic exploration of the topic, we classified and evaluated different types of basic observations and actions that can be used in a crowd simulation system (Chapter 3). This is meant as a stepping stone for more advanced research – after all, how can we build a house if we don't know how to make a solid brick? Our inspiration was the fact that in prior work, every project used something slightly different, without acknowledging the differences, thus harming reproducibility and comparability.

For observations, we analyzed the issue of the reference frame – should it be absolute, or more egocentric (Section 3.2.2)? In principle, both approaches make sense and can be justified, however in practice, they are far from equal. While the details still depend on the scenario we are simulating, egocentric representations tend to perform much better than absolute ones. Our interpretation is that egocentric representations lose some of the information that absolute representations keep, and in this way, they improve the generalization properties of the algorithm. After all, it is not necessary know how to go from 48°42′53″N 2°12′20″E to 48°43′33″N 2°15′32″E, if we know how to go a few kilometers north-east.

There is also the question of the actual information available to the agent – is simple raycasting sufficient, or is it better to give agents the important positions directly (Section 3.2.2)? While our previous result was slightly more human-like, we must resist the temptation of anthropomorphizing

our crowd agents. The intuition for raycasting (which, along with its simplicity, is the probable reason for its ubiquity in prior work) can be as follows. Raycasting is a crude approximation of human vision. Crowd agents are meant to be crude representations of real-world humans. So if their method of perceiving the environment is human-like, the agents themselves will be more human-like. While this reasoning might work in the future, with significantly more capable algorithms, for now, it is out of reach. While raycasting models typically obtain a reasonable performance, they are slower and less reliable than their counterparts using more direct perception. This is because raycasting provides roughly the same information, but in a very obscured manner – so while agents learn to navigate, they also have to learn to see, which overloads their small digital brains.

Finally, we have the actions, or equivalently, the environment dynamics (Section 3.2.3). We identified two main orthogonal axes of designing an action space – whether it is based on polar or Cartesian coordinates; and whether it is based on velocities or accelerations. The first axis again corresponds to our intuitions about how humans do things – most of the time, we only go in one direction, forward, while changing where "forward" actually is. Similarly, following Newtonian mechanics, we tend to move our bodies by applying forces, i.e. accelerations. Velocity-based controls are more flexible and enable crisper motion, but is that really something we need (as researchers), or have (as humans)? Another perspective on this choice is that Cartesian velocity controls are fully holonomic, i.e. they give the agent maximum flexibility. And yet, it often turns out that more restrictive polar or acceleration-based dynamics perform better in practice. This is likely due to their natural fit with egocentric observations – we do not need to know how to go north-east if we know how to turn right a little bit and then go forward.

The main conclusion from this chapter is that *nothing is simple, and nothing is obvious.* While we have observed some regularities in which design choices outperform the others, there is no silver bullet. And because every possible choice has some intuition (and prior work) behind it, we really need to be careful and validate our assumptions – otherwise we risk putting in significant effort to optimize a system, which is being undermined by an inefficient design.

Reward design With a fundamental simulation in place, we then proceeded to investigate the question of designing the right reward function. This is arguably the most important part of designing any RL system, or more broadly, any AI system. We have to ensure that the incentives we give to the agents, lead them to behave the way we want them to behave. If this seems simple and obvious, we recommend that the reader does not interact with any strange spirits trapped in bottles.

Our motivation, similarly to the previous chapter, was the lack of consistency in prior work. Researchers have proposed various approaches, often coming from the principle of "We found that this works well enough." However, there are many designs which might make some sense, appear to work well enough, and yet be fundamentally flawed, causing issues that we did not even know we have to look out for.

In fact, we observed an example of this phenomenon in Chapter 3. The reward function we designed *made sense* – a guiding potential, and an additional term encouraging agents to move at their comfortable velocity. Moving slower or faster leads to a lower reward, so agents should move at just the right speed. However, it is important to keep in mind that the nature of the navigation task disrupts the naive reward design quite a bit. Moving at a slower velocity increases the time required to reach the goal, implicitly penalizing those velocities if the overall rewards are negative. Moreover, due to the discounted utility paradigm common in RL, future rewards

become less important in the learning process, further penalizing slower velocities. With all of these implicit pressures, what happens if we also penalize slow velocities explicitly? As we found out, we obtain bike-like agents that do their best to maintain their velocity, even if that means making loops in place while waiting for an obstacle to pass.

With this experience, we set out to create a more principled design for the reward function. The overarching goal is creating *human-like* agents, which begs the question – what does it mean to be human-like? To make this task tractable, we settled on a working definition using the concept of energy efficiency. Prior work suggests that humans tend to act in a way that minimizes the perceived effort, so this approach has a better theoretical foundation beyond seeming correct. It is also sufficiently well-defined to be practical – even though a precise estimation of human energy usage is rather difficult, we can use a simple approximation, and then refine it as needed.

We identified two main problems with using the (negative) energy usage directly as the reward function. The first one is the **local problem**. When training an RL agent from scratch, it begins by randomly taking actions, and adjusting its action distributions to favor higher rewards. However, if reaching the goal is out of reach for such an agent (e.g. due to a long distance), the high reward actions tend to be ones minimizing energy in the simplest way – by not moving. Then, we have the **global problem**. Due to the discounted utility paradigm commonly used in RL, in certain cases, it turns out that standing still is actually the optimal policy. The effect of discounted utility is that agents are biased for getting reward sooner than later, and that can be obtained by using less energy in the short term, even at a detriment to the long-term navigation goal. If the discount factor is too low, then simply standing still will be the optimal policy, because the distant reward of navigation success is too far in time.

Our main proposed solution to both of these problems is adding a potential term. This by itself is not novel, but for the first time, we analytically derive the exact magnitude of this potential. We found that there is only one coefficient for which the discount factor does not affect the effective optimal velocity. This is a very useful property, as it allows us to again treat the discount factor as a hyperparameter of the learning algorithm, with less of an impact on the decision process. We also showed that curriculum learning approaches are highly effective for this task, and make it possible to actually optimize just the energy (without a potential) in the final stages of the training – as long as the discount factor is set to 1.

Overall, in Chapter 4 we found a more principled way of designing the reward function for training human-like agents with RL. We provided theoretical insights justifying our designs, and validated them empirically, showing what approaches work, and what approaches fail. While this does not entirely solve the question of reward design for life-like agents, it contributes to the construction of a foundation upon which RL-based crowd simulation can be built, without having to worry about hidden assumptions and suboptimal designs.

Reward optimization In Chapter 5, the final chapter of this thesis, we continue the direction of investigating the reward functions, this time from the perspective of the learning algorithm. As we discussed in Chapter 4, some complications in RL-based crowd simulation arise due to the discounted utility paradigm. Notably, modern RL algorithms can only really use exponential discounting, which is theoretically well-justified for rational agents, but also highly restrictive for agents which need not be perfectly rational.

The motivating question was fairly simple – can we use the same modern RL algorithms, but use non-exponential discounting instead of the usual exponential approach? All of our prior

experiments were performed using Proximal Policy Optimization (PPO) with Generalized Advantage Estimation (GAE), and to our surprise, no prior work connected these algorithms with non-exponential discounting.

To close this gap, we introduced the Universal Generalized Advantage Estimation (UGAE) algorithm. It serves as a computationally efficient method of estimating the advantage values following the GAE approach, but using any non-exponential discounting, while GAE is only restricted to exponential discounting. We also showed how any distribution over discount factors $\gamma \in [0, 1]$ can be reinterpreted as a new, non-exponential discounting.

In conjunction with UGAE, we introduced Beta-weighted discounting. "Beta" refers to the Beta distribution, which weighs individual discount factors. This distribution has the interesting property of continuously interpolating between exponential and hyperbolic discounting, depending on the values of its parameters. This gives us a parametrized family of non-exponential discounting methods, which can be easily computed by using well-known properties about the Beta distribution, and has an intuitive interpretation as being a transitional state between exponential and hyperbolic discounting.

The main role of this chapter is opening the door to new, exciting research involving non-exponential discounting. After all, humans are not perfectly rational and tend to exhibit more hyperbolic temporal discounting, so approaches like this may become instrumental for imitating human behavior – including human behavior within a crowd.

Perspectives

The work in this thesis serves as a solid and principled foundation for RL-based crowd simulation systems, but the field is far from solved. There are many directions that can be explored in the future, building upon these methods. Here we aim to provide an overview of the potential future of the field, and how this thesis will contribute to it.

Every aspect of the foundational simulation design choices we discussed can be taken even further and improved for the purpose of increased realism. While in this work we used a simple permutation-invariant representation of nearby agents, a better performance may be attainable using the attention mechanism [172], which recently revolutionized the field of Natural Language Processing. Similarly, action spaces may be refined to be more realistic models of human movement, e.g. by modelling individual footsteps, or potentially even the entire human body. Finally, the reward function used to encourage "human-likeness" may be refined to better reflect real human behavior, either by better modelling of perceived effort or via data-driven methods.

Going beyond the fundamentals, it is worth keeping in mind that often, a crowd cannot be reduced to a collection of individuals – instead, it is composed of smaller structures, such as couples, families, and groups of friends. Sometimes, the structures may even have smaller substructures, like a city tour consisting of multiple families, navigating through a crowd of locals. It is possible that a hierarchical structure would be suitable to model such phenomena, with an individual moving within a family the same way that a family moves within a crowd. This would make it possible to choose the right level of detail for the simulation depending on the needs at any given moment, which is particularly valuable for interactive media like video games. To the best of our knowledge, such approaches have not been evaluated with RL, so this paves a clear path for future work.

Finally, crowd simulation is commonly used across the entertainment industry and beyond – large scenes in movies, video games, but also urban design and architecture. Each of these domains

carries its own set of unique challenges and adaptations, so as RL methods become more efficient and reliable, it will be worthwhile to refine them into variants suited for each of these applications. Eventually, RL may become the standard method for animating human-like crowds, and we hope that this thesis will be a step towards that future.

Appendix A

Non-exponential reward discounting

A.1 Proofs

Notation

- $\Gamma^{(t)}$ – general discount factor at step t
- r_t – reward at step t
- $V(s)$ – value estimate of a state s
- $\Gamma(z) = \int_0^\infty x^{z-1}e^{-x}dx$ – the usual Gamma function
- $B(\alpha, \beta) = \int_0^1 t^{\alpha-1}(1-t)^{\beta-1}dt = \frac{\Gamma(\alpha)\Gamma(\beta)}{\Gamma(\alpha+\beta)}$
- $\Delta(X)$ – set of probability distributions on the set X

Theorem 1. ***UGAE: GAE with arbitrary discounting***

Consider $\boldsymbol{r}_t = [r_{t+i}]_{i\in\mathbb{N}}$, $\boldsymbol{V}_t = [V(s_{t+i})]_{i\in\mathbb{N}}$, $\boldsymbol{\Gamma} = [\Gamma^{(i)}]_{i\in\mathbb{N}}$, $\boldsymbol{\Gamma}' = [\Gamma^{(i+1)}]_{i\in\mathbb{N}}$, $\boldsymbol{\lambda} = [\lambda^i]_{i\in\mathbb{N}}$. *We define the GAE with arbitrary discounting as:*

$$\tilde{A}_t^{UGAE(\Gamma,\lambda)} := -V(s_t) + (\boldsymbol{\lambda} \odot \boldsymbol{\Gamma}) \cdot \boldsymbol{r}_t + (1-\lambda)(\boldsymbol{\lambda} \odot \boldsymbol{\Gamma}') \cdot \boldsymbol{V}_{t+1} \tag{A.1}$$

If $\Gamma^{(t)} = \gamma^t$*, this is equivalent to the standard GAE advantage.*

Proof. Recall that we defined the k-step advantage as

$$\tilde{A}_t^{(k)} := -V(s_t) + \sum_{l=0}^{k-1} \Gamma^{(l)} r_{t+l} + \Gamma^{(k)} V(s_{t+k}).$$

With this, we expand the expression for UGAE as

$$\tilde{A}_t^{UGAE(\Gamma,\lambda)} = (1-\lambda)(\tilde{A}_t^{(1)} + \lambda\tilde{A}_t^{(2)} + \lambda^2\tilde{A}_t^{(3)} + \ldots) \tag{A.2}$$

$$= (1-\lambda)\Big[-V(s_t) + r_t + \Gamma^{(1)}V(s_{t+1}) - \lambda V(s_t) \tag{A.3}$$

$$+\lambda r_t + \lambda\Gamma^{(1)}r_{t+1} + \lambda\Gamma^{(2)}V(s_{t+2}) + \ldots\Big] \tag{A.4}$$

$$= (1-\lambda)\left[-\sum_{l=0}^{\infty}(\lambda^l)V(s_t) + \sum_{l=0}^{\infty}(\lambda^l)r_t + \sum_{l=1}^{\infty}(\lambda^l)\Gamma^{(1)}r_1 + \ldots\right. \tag{A.5}$$

$$\left.+\Gamma^{(1)}V(s_{t+1}) + \lambda\Gamma^{(2)}V(s_{t+2}) + \ldots\right] \tag{A.6}$$

$$= (1-\lambda)\left[\frac{-V(s_t)}{1-\lambda} + \frac{r_t}{1-\lambda} + \frac{\lambda\Gamma^{(1)}r_{t+1}}{1-\lambda} + \ldots\right. \tag{A.7}$$

$$\left.+\Gamma^{(1)}V(s_{t+1}) + \lambda\Gamma^{(2)}V(s_{t+2}) + \ldots\right] \tag{A.8}$$

$$= -V(s_t) + \sum_{l=0}^{\infty}\lambda^l\Gamma^{(l)}r_{t+l} + (1-\lambda)\sum_{l=0}^{\infty}\lambda^l\Gamma^{(l+1)}V(s_{t+l+1}) \tag{A.9}$$

$$= -V(s_t) + (\boldsymbol{\lambda}\odot\boldsymbol{\Gamma})\cdot\boldsymbol{r}_t + (1-\lambda)(\boldsymbol{\lambda}\odot\boldsymbol{\Gamma}')\cdot\boldsymbol{V}_{t+1} \tag{A.10}$$

showing the validity of Equation 5 in the main manuscript. To reduce it to standard GAE with exponential discounting, it is sufficient to replace $\Gamma^{(\cdot)}$ with $\gamma^{\cdot}$ in the second line of Equation A.2 and follow the proof from Schulman et al. [145].

□

Theorem 2. ***UGAE added bias***

Consider an arbitrary summable discounting $\Gamma^{(t)}$ in an environment where the reward is bounded by $R \in \mathbb{R}$. The additional bias, defined as the discrepancy between the UGAE and Monte Carlo value estimations, is finite.

Proof. The goal is to find a finite bound on the difference between the empirical (Monte Carlo) value estimate used for bootstrapping the advantage estimation, and the value estimation used in UGAE, in the infinite time limit. This can be expressed as follows:

$$\left|\sum_{l=0}^{\infty}\Gamma^{(l+1)}\lambda^l\hat{V}(s_{t+l+1}) - \sum_{l=0}^{\infty}\lambda^l V_{l+1}^{\Gamma}(s_{t+l+1})\right| \tag{A.11}$$

where V_{l+1}^{Γ} is the true value as discounted with Γ, l steps after the step for which we compute the advantage, and $\hat{V}$ is the UGAE estimate:

$$V_k^{\Gamma}(s_t) = \mathbb{E}\sum_{t'}\Gamma^{(k+t')}r_{t'} \tag{A.12}$$

$$\hat{V}_k(s_t) = \mathbb{E}\sum_{t'}\Gamma^{(t')}r_{t'} \tag{A.13}$$

With this we can expand the expression in Equation A.11 as follows:

$$\left|\sum_{l=0}^{\infty}\Gamma^{(l+1)}\lambda^l\hat{V}(s_{t+l+1}) - \sum_{l=0}^{\infty}\lambda^l V_{l+1}^{\Gamma}(s_{t+l+1})\right| = \tag{A.14}$$

$$= \left|\sum_{l=0}^{\infty}\lambda^l\left[\Gamma^{(l+1)}\hat{V}(s_{t+l+1}) - V_{l+1}^{\Gamma}(s_{t+l+1})\right]\right| = \tag{A.15}$$

$$= \left|\sum_{l=0}^{\infty}\lambda^l\left[\Gamma^{(l+1)}\mathbb{E}\sum_{t'}\Gamma^{(t')}r_{t'} - \mathbb{E}\sum_{t'}\Gamma^{(l+1+t')}r_{t'}\right]\right| = \tag{A.16}$$

$$= \mathbb{E}\left|\sum_{l=0}^{\infty}\lambda^l\sum_{t'}\left[\Gamma^{(l+1)}\Gamma^{(t')} - \Gamma^{(l+1+t')}\right]r_{t'}\right| \leq \tag{A.17}$$

$$\leq \mathbb{E}\left|\sum_{l=1}^{\infty}\lambda^{(l-1)}\sum_{t'}\left[\Gamma^{(l)}\Gamma^{(t')} - \Gamma^{(l+t')}\right]\right| R \tag{A.18}$$

We now focus on the key expression of the last line, which we denote as δ_l^{Γ}:

$$\begin{aligned}\delta_l^{\Gamma} &= \sum_{t'}\Gamma^{(l)}\Gamma^{(t')} - \Gamma^{(l+t')} \leq \\ &\leq \sum_{t'}\Gamma^{(l)}\Gamma^{(t')} \leq \\ &\leq \max_t \Gamma^{(t)}\sum_{t'}\Gamma^{(t')} \leq \infty\end{aligned}$$

This shows that δ_l^{Γ} is finite for any summable discounting Γ and for every value of l. Because the δ_l^{Γ} terms are summed with an exponentially decreasing factor $\lambda^{(l-1)}$ in Equation A.18, the total difference in Equation A.11 must also be finite, completing the proof.

□

Theorem 3. ***Beta-weighted discounting***

Consider $\alpha, \beta \in [0, \infty)$. The following equations hold for the Beta-weighted discount vector parametrized by α, β:

$$\Gamma^{(t)} = \prod_{k=0}^{t-1}\frac{\alpha + k}{\alpha + \beta + k} \tag{A.19}$$

$$\Gamma^{(t+1)} = \frac{\alpha + t}{\alpha + \beta + t}\Gamma^{(t)} \tag{A.20}$$

Proof. As mentioned in the paper, if we use an effective discount factor obtained by weighing individual values according to some probability distribution $w \in \Delta([0, 1])$, the effective discount factor at step t is given by the distribution's raw moment $\Gamma^{(t)} = m_t$, where

$$m_t = \int_0^1 w(\gamma)\gamma^t d\gamma$$

Consider the Beta distribution. Its probability distribution function [71] is given by the following expression:

$$f(x; \alpha, \beta) = \frac{1}{B(\alpha, \beta)}x^{\alpha-1}(1-x)^{\beta-1}$$

The raw moments can be obtained as follows:

$$\begin{aligned}
\Gamma^{(t)} = m_t &= \int_0^1 x^t f(x;\alpha,\beta) \\
&= \int_0^1 x^t \frac{1}{B(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1} \\
&= \frac{1}{B(\alpha,\beta)} \int_0^1 x^{(\alpha+t)-1}(1-x)^{\beta-1} \\
&= \frac{1}{B(\alpha+\beta)} B(\alpha+t,\beta) \\
&= \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \times \\
&\quad \times \frac{\Gamma(\alpha)\cdot\alpha\cdot(\alpha+1)\cdot\ldots\cdot(\alpha+t-1)\cdot\Gamma(\beta)}{\Gamma(\alpha+\beta)\cdot(\alpha+\beta)\cdot(\alpha+\beta+1)\cdot\ldots\cdot(\alpha+\beta+t-1)} \\
&= \frac{\alpha\cdot(\alpha+1)\cdot\ldots\cdot(\alpha+t-1)}{(\alpha+\beta)\cdot(\alpha+\beta+1)\cdot\ldots\cdot(\alpha+\beta+t-1)} \\
&= \prod_{k=0}^{t-1} \frac{\alpha+k}{\alpha+\beta+k}
\end{aligned}$$

which proves Equation A.19. We then consider the recurrence between consecutive $\Gamma^{(\cdot)}$ values

$$\begin{aligned}
\Gamma^{(t+1)} &= \prod_{k=0}^{t} \frac{\alpha+k}{\alpha+\beta+k} \\
&= \frac{\alpha+t}{\alpha+\beta+t} \prod_{k=0}^{t-1} \frac{\alpha+k}{\alpha+\beta+k} \\
&= \frac{\alpha+t}{\alpha+\beta+t} \Gamma^{(t)}
\end{aligned}$$

proving Equation A.20 and completing our proof of Theorem 1.

□

Lemma 4. ***Special cases of Beta-weighted discounting***

Consider a discounting $\Gamma^{(t)}$ *given by the Beta-weighted discounting parametrized by* $\mu \in (0,1), \eta \in (0,1]$. *The following is true:*

- *if* $\eta \to 0$, *then* $\Gamma^{(t)} = \mu^t$, *i.e. it is equal to exponential discounting*
- *if* $\eta = 1$, *then* $\Gamma^{(t)} = \frac{\mu}{\mu+(1-\mu)t} = \frac{1}{1+t/\alpha}$, *i.e. it is equal to hyperbolic discounting*

Proof. Remember that $\mu = \frac{\alpha+\beta}{\beta}, \eta = \frac{1}{\beta}$. Let us first consider $\eta \to 0$, i.e. $\beta \to \infty$ so that $\frac{\alpha}{\alpha+\beta} = const$. Note that this also implies $\alpha \to \infty$. Consider the expression for $\Gamma^{(t)}$:

$$\Gamma^{(t)} = \prod_{k=0}^{t-1} \frac{\alpha+k}{\alpha+\beta+k}$$

As α and β grow arbitrarily high, the bounded values of k become negligible, and the expression can be reduced to

$$\Gamma^{(t)} = \prod_{k=0}^{t-1} \frac{\alpha}{\alpha+\beta} = \prod_{k=0}^{t-1} \mu = \mu^t.$$

For $\eta = 1$, we can reuse the expression obtained in the proof of Lemma 5. As shown there, with $\beta = \eta = 1$, the effective discount factor is

$$\Gamma^{(t)} = \frac{1}{1+t/\alpha}$$

which with $k = \frac{1}{\alpha}$, becomes the usual hyperbolic discounting:

$$\Gamma^{(t)} = \frac{1}{1+kt}$$

thus completing the proof.

□

Lemma 5. ***Beta-weighted discounting summability***

Given the Beta-weighted discount vector $\Gamma^{(t)} = \prod_{k=0}^{t-1} \frac{\alpha+k}{\alpha+\beta+k}$, $\alpha \in [0,\infty)$, $\beta \in [0,\infty)$, *the following property holds:*

$$\sum_{t=0}^{\infty} \Gamma^{(t)} = \begin{cases} \frac{\alpha+\beta-1}{\beta-1} & \text{if } \beta > 1 \\ \infty & \text{otherwise} \end{cases} \tag{A.21}$$

Thus, Beta-weighted discounting is summable iff $\beta > 1$.

Proof. First, we analyze the convergence of Beta-weighted $\Gamma^{(t)}$ depending on α, β.

In particular, we consider the series:

$$S = \sum_{t=0}^{\infty} a_t = \sum_{t=0}^{\infty} \left(\prod_{k=0}^{t-1} \frac{\alpha+k}{\alpha+\beta+k} \right)$$

We then use the Raabe's convergence test [6]. Given a series (a_t) consider the series of terms $b_t = t\left(\frac{a_t}{a_{t+1}} - 1\right)$ and its limit $L = \lim_{t\to\infty} b_t$. There are three possibilities:

- if $L > 1$, the original series converges
- if $L < 1$, the original series diverges
- if $L = 1$, the test is inconclusive

In the case of Beta-weighted discounting, we have:

$$\begin{aligned} \lim_{t\to\infty} b_t &= t \left(\frac{\prod_{k=0}^{t-1} \frac{\alpha+k}{\alpha+\beta+k}}{\prod_{k=0}^{t} \frac{\alpha+k}{\alpha+\beta+k}} - 1 \right) \\ &= \lim_{t\to\infty} \frac{t}{\frac{\alpha+t}{\alpha+\beta+t}} - t \\ &= \lim_{t\to\infty} \frac{\alpha t + \beta t + t^2 - \alpha t - t^2}{\alpha + t} \\ &= \lim_{t\to\infty} \frac{\beta t}{\alpha + t} \\ &= \beta \end{aligned}$$

Thus, we show that Beta-weighted discounting is summable with $\beta > 1$ and nonsummable with $\beta < 1$. For $\beta = 1$, we can rewrite the effective discount factor as:

$$\Gamma^{(t)} = \prod_{k=0}^{t-1} \frac{\alpha + k}{\alpha + \beta + k} \tag{A.22}$$

$$= \prod_{k=0}^{t-1} \frac{\alpha + k}{\alpha + k + 1} \tag{A.23}$$

$$= \frac{\alpha \cdot \cancel{(\alpha+1)} \cdot \ldots \cdot \cancel{(\alpha+t-2)} \cdot \cancel{(\alpha+t-1)}}{\cancel{(\alpha+1)} \cdot \cancel{(\alpha+2)} \cdot \ldots \cdot \cancel{(\alpha+t-1)} \cdot (\alpha+t)} \tag{A.24}$$

$$= \frac{\alpha}{\alpha + t} \tag{A.25}$$

$$= \frac{1}{1 + t/\alpha} \tag{A.26}$$

The series $\sum_{t=0}^{\infty} \frac{1}{1+t/\alpha}$ is a general harmonic series and therefore divergent, completing the proof of convergence.

To obtain the exact value, we use the following Taylor expansion

$$\frac{1}{1-x} = 1 + x + x^2 + ...$$

By evaluating the expected value of this expression with the Beta distribution's probability density function $w(x)$, we obtain the desired sum of all discount factors:

$$\begin{aligned}
\mathbb{E}(\frac{1}{1-X}) &= \int_0^1 \frac{1}{1-x} f(x;\alpha,\beta)dx \\
&= \int_0^1 w(x) + xw(x) + x^2 w(x) + \ldots dx \\
&= \int_0^1 x^0 w(x)dx + \int_0^1 x^1 w(x)dx + \ldots \\
&= \Gamma^{(0)} + \Gamma^{(1)} + \ldots = \sum_{t=0}^{\infty} \Gamma^{(t)}
\end{aligned}$$

This expression can be expanded as follows:

$$\begin{aligned}
\mathbb{E}(\frac{1}{1-X}) &= \int_0^1 \frac{1}{1-x} f(x;\alpha,\beta)dx \\
&= \int_0^1 (1-x)^{-1} \frac{1}{B(\alpha,\beta)} x^{\alpha-1}(1-x)^{\beta-1} \\
&= \frac{1}{B(\alpha,\beta)} \int_0^1 x^{\alpha-1}(1-x)^{(\beta-1)-1} \\
&= \frac{B(\alpha,\beta-1)}{B(\alpha,\beta)} \\
&= \frac{\Gamma(\alpha)\Gamma(\beta-1)}{\Gamma(\alpha+\beta-1)} \frac{\Gamma(\alpha+\beta)}{\Gamma(\alpha)\Gamma(\beta)} \\
&= \frac{\cancel{\Gamma(\alpha)}\cancel{\Gamma(\beta-1)}}{\cancel{\Gamma(\alpha+\beta-1)}} \frac{(\alpha+\beta-1)\cancel{\Gamma(\alpha+\beta-1)}}{(\beta-1)\cancel{\Gamma(\alpha)}\cancel{\Gamma(\beta-1)}} \\
&= \frac{\alpha+\beta-1}{\beta-1}
\end{aligned}$$

completing the proof.

□

A.2 Beta-weighted Discounting Properties

In Table A.1 we present the values of the properties described in Section 5 for a set of discounting methods. For each of them, we list their normalized partial sums Γ_0^{10}, Γ_{10}^{100}, Γ_{100}^{1000}, Γ_{1000}^{10000}, the variance measure, the effective time horizon, and the total sum of the first 1000 steps.

Table A.1: The values of different metrics for a chosen set of discounting method and their parameters.

Discounting method	Γ_0^{10}	Γ_{10}^{100}	Γ_{100}^{1000}	Γ_{1000}^{10000}	Variance $\sum_{t=0}^{10000} \Gamma^{(t)2}$	T_{eff}	Total $\sum_{t=0}^{1000} \Gamma^{(t)}$
No discounting	0.001	0.009	0.090	0.900	10000	6322	1000
Exponential $\gamma = 0.99$	0.096	0.538	0.366	0.000	50.25	100	100
Exponential $\gamma = 0.999$	0.010	0.085	0.537	0.368	500.25	1000	632.3
Exponential $\gamma = 0.97$	0.263	0.690	0.0480	0.000	16.92	33	33.3
Beta-weighted $\mu = 0.99$, $\eta = 0.5$	0.049	0.293	0.509	0.149	66.67	323	166.1
Beta-weighted $\mu = 0.97$, $\eta = 0.5$	0.135	0.476	0.334	0.055	22.23	110	61.7
Hyperbolic $\mu = 0.99$	0.021	0.130	0.370	0.479	98.53	1741	238.8
Hyperbolic $\mu = 0.25$	0.439	0.188	0.187	0.187	1.12	107	3.3
Fixed-horizon $T_{max} = 100$	0.100	0.900	0.000	0.000	100	64	100
Fixed-horizon $T_{max} = 160$	0.062	0.562	0.375	0.000	160	102	160
Truncated Exponential $\gamma = 0.99$, $T_{max} = 100$	0.151	0.849	0.000	0.000	43.52	51	63.4
Truncated Exponential $\gamma = 0.99$, $T_{max} = 500$	0.096	0.542	0.362	0.000	50.25	99	99.3
Truncated Beta-weighted $\mu = 0.99$, $\eta = 0.5$, $T_{max} = 100$	0.143	0.857	0.000	0.000	47.11	54	69.4
Truncated Hyperbolic $\mu = 0.99$, $T_{max} = 100$	0.138	0.862	0.000	0.000	50.13	55	69.4
Truncated Hyperbolic $\mu = 0.99$, $T_{max} = 500$	0.054	0.335	0.612	0.000	83.13	210	178.6

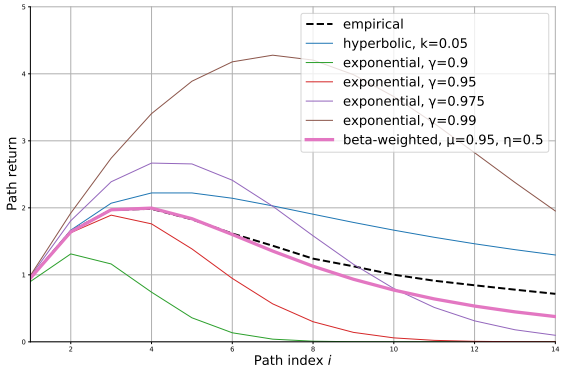


Figure A.1: Pathworld environment results under different discounting schemes. Hyperbolic [36] and exponential (various curves) discountings fail to approximate the empirical (dashed) value. Instead, the proposed Beta-weighted discounting approximates it much better, despite its different functional form.

A.3 Pathworld experiments

Table A.2: Values of the Mean Square Error for different discounting methods on the Pathworld environment, summed across the first 14 paths $i \in \overline{1, 14}$. Lower is better.

Discounting method	discount factor†	η	MSE
Exponential	0.990	0	3.962
Exponential	0.950	0	0.446
Exponential	0.975	0	0.242
Hyperbolic‡	0.05	1	0.250
Beta-Weighted	0.95	0.5	**0.032**

†: factor is γ for exponential
k for hyperbolic
μ for Beta-weighted
‡: Results obtained by our re-implementation of [36]

We showcase the utility of our method on a simple toy environment called the Pathworld introduced by Fedus et al. [36]. Our goal is to show that Beta-weighted discounting can accurately model the presence of unknown risk in an environment, even without being designed to a priori match the functional form of the risk distribution.

A.3.1 Setup

In Pathworld, the agent takes a single action and observes a single reward after some delay. The actions (i.e. paths) are indexed by natural numbers. When taking the i-th path, the agent receives a reward $r = i$ after $d = i^2$ steps. Each path may also be subject to some hazard. In each episode, a risk λ is sampled from the uniform distribution $U([0, 2k])$ for a given parameter k. Given the risk λ, at each timestep on the path, the agent has a chance $(1 - e^{-\lambda})$ of dying, and thus not collecting any reward in that episode. The task of the agent is to use experience gathered without risk, and use the discounting to accurately predict a path's value when evaluated with an unknown risk.

A.3.2 Pathworld results

If the risk is sampled from the Dirac Delta distribution $\mathcal{H} = \delta(\lambda - \lambda_0)$, the optimal discounting method is exponential discounting, i.e. $\Gamma^t = \gamma^t$ with $\gamma = e^{-\lambda}$. As shown in [36], if the risk is sampled from an exponential distribution $\mathcal{H} = \frac{1}{k}\exp(-\lambda/k)$, the optimal discounting scheme is their hyperbolic discounting $\Gamma^t = \frac{1}{1+kt}$.

As shown in Lemma 4, Beta-weighted discounting subsumes both exponential (μ=γ, η→0) and hyperbolic discounting (α=$\frac{1}{k}$, η=1). Thus, it directly models scenarios whose optimal discounting is exponential or hyperbolic.

A more interesting scenario is when the functional form is different, e.g. when the risk is sampled from a uniform distribution $\mathcal{H} \sim U([0, 2\lambda_\mu])$. Figure A.1 and Table A.2 report the results. We observe that Beta-weighted discounting (with μ chosen to fit the mean of the true risk distribution, and η chosen heuristically to decrease the variance of the Beta distribution) successfully outperforms all baselines, indicating that using Beta-weighted discounting enabled by the proposed UGAE allows better modelling of unknown risk distributions in environments where the risk phenomenon makes discounting necessary.

Appendix B

Reward function design

B.1 Discounting-invariant reward

Intermediate equations in 4.3.3. Derivatives and solutions computed using sympy.

Here we provide the intermediate stages of the computation in 4.3.3. All computations are performed using sympy – finding the derivatives and roots of equations.

First, we reiterate the discounted sum of rewards with a discount factor γ and a guiding potential with the coefficient $c_p\sqrt{e_s e_w}$. (Equation 4.13)

$$R^\gamma = \int_0^T e^{t\ln\gamma}\left(-e_s - e_w v^2 + c_p\sqrt{e_s e_w}v\right)dt = \frac{1-\gamma^{\frac{d}{v}}}{-\ln\gamma}\left(-e_w v^2 + c_p\sqrt{e_s e_w} - e_s\right)$$

We differentiate the expression on the right hand side w.r.t. v, looking for a stationary point that corresponds to the optimal velocity v^*:

$$\frac{dR^\gamma}{dv} = F(v,\gamma) = -d\gamma^{\frac{d}{v}}\left(\frac{c_p\sqrt{e_s e_w}}{v} - \frac{e_s}{v^2} - e_w\right) - \frac{\left(1-\gamma^{\frac{d}{v}}\right)\left(c_p\sqrt{e_s e_w} - 2e_w v\right)}{\log(\gamma)} = 0$$

While we cannot solve this analytically for v^*, we know that arguments for which $F(v^*,\gamma)=0$ correspond to optimal velocities with a given γ. We are now interested in finding arguments where $\frac{dv^*}{d\gamma} = 0$, i.e. changes in the discount factor do not affect the optimal velocity. We find that by computing the implicit derivative:

$$\frac{dv}{d\gamma} = \frac{v\left(-d^2\gamma^{\frac{d+2v}{v}}\left(-c_p v\sqrt{e_s e_w} + e_s + e_w v^2\right)\log(\gamma)^2 - dv^2\gamma^{\frac{d+2v}{v}}\left(c_p\sqrt{e_s e_w} - 2e_w v\right)\log(\gamma) + v^3\gamma^2\left(\gamma^{\frac{d}{v}} - 1\right)\left(c_p\sqrt{e_s e_w} - 2e_w v\right)\right)}{\gamma^3\left(dv^2\gamma^{\frac{d}{v}}\left(c_p\sqrt{e_s e_w} - 2e_w v\right)\log(\gamma) + d\gamma^{\frac{d}{v}}\left(d\left(-c_p v\sqrt{e_s e_w} + e_s + e_w v^2\right)\log(\gamma) - v\left(c_p v\sqrt{e_s e_w} - 2e_s\right)\right)\log(\gamma) + 2e_w v^4\left(\gamma^{\frac{d}{v}} - 1\right)\right)\log(\gamma)} = 0$$

Although this expression is even more complex, it turns out to be solvable analytically for c_p:

$$c_p = \frac{d^2 e_s\gamma^{\frac{d}{v}}\log(\gamma)^2 + d^2 e_w v^2\gamma^{\frac{d}{v}}\log(\gamma)^2 - 2de_w v^3\gamma^{\frac{d}{v}}\log(\gamma) + 2e_w v^4\gamma^{\frac{d}{v}} - 2e_w v^4}{v\sqrt{e_s e_w}\left(d^2\gamma^{\frac{d}{v}}\log(\gamma)^2 - dv\gamma^{\frac{d}{v}}\log(\gamma) + v^2\gamma^{\frac{d}{v}} - v^2\right)} = 2$$

After substituting numerical values and $v = v^* = \sqrt{\frac{e_s}{e_w}}$, we get the result $c_p = 2$. This means that if the potential has a coefficient of $c_p = 2$, the optimal velocity will not change with the

discount factor.

B.2 Algorithmic details

The performance of agents trained with PPO tends to significantly depend on the exact implementation details of the algorithm [35, 9, 66]. Beyond a set of typical choices used in our implementation (all of which can be found in the training code), we use a non-standard modification that we call "rewind", inspired by TRPO.

When performing gradient updates with a given batch of data, the algorithm typically keeps changing the policy until a predefined number of updates elapses. Alternatively, if the KL divergence between the behavior policy and the learned policy exceeds a predefined threshold, the process is stopped immediately to obtain a fresh batch of data.

While this approach typically works sufficiently well for maintaining the on-policy assumption of the policy gradient theorem, sometimes a single gradient update leads to a significant drop in the performance, which would then take many training iterations to recover. To counteract this effect, we save the policy parameters before each gradient update. If the KL divergence criterion is triggered, the policy is rolled back to that saved state, ensuring that a single batch of data never leads to an excessive change to the policy.

B.3 Reward implementation details

Due to various differences in the basic simulation setup, including but not limited to the design choices described by Kwiatkowski et al. [87], we were unable to fully reproduce some of the results from prior work. Here, we describe the differences between the reward functions described in other papers, and our implementations.

Lee et al. [93] use the function they named FLOOD, which linearly penalizes velocities above $1.5\,\mathrm{m/s}$ and below $-0.5\,\mathrm{m/s}$. Because our simulation does not allow backwards movement, this is reduced to a linear penalty to velocities exceeding the optimal velocity (which varies by agent).

Work by Xu and Karamouzas [186] focuses on using knowledge distillation for more human-like behavior, but a key component of their reward function deals with the agents maintaining the right speed. The expression listed in the paper is $w_v \exp(\sigma_v ||\boldsymbol{v} - \boldsymbol{v}^*||)$, with $w_v = 0.08$ and $\sigma_v = 0.85$. Notice, however, that this structure would incentivize large deviations from the optimal velocity by maximizing $||\boldsymbol{v} - \boldsymbol{v}^*||$. Due to the monotonicity of the exponential function, exactly one of these parameters must be negative to optimize the behavior in the correct dimension. The source code provided with the paper indicates that $w_v = 0.02$ and $\sigma_v = -0.85$, but in our experiments the reverse convention achieves significantly better results, i.e. $w_v < 0$ and $\sigma_v > 0$. Furthermore, in our experiments we use $w_v = -10$ together with adjusted goal and collision rewards, because values closer to the original ones failed to converge to reliable goal-seeking behavior.

Bibliography

[1] F. Abdolhosseini, H. Y. Ling, Z. Xie, X. B. Peng, and M. van de Panne. On Learning Symmetric Locomotion. In *Motion, Interaction and Games*, MIG '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 978-1-4503-6994-7. doi: 10.1145/3359566.3360070. URL https://doi.org/10.1145/3359566.3360070. event-place: Newcastle upon Tyne, United Kingdom.

[2] J. Achiam. Spinning Up in Deep Reinforcement Learning. 2018.

[3] G. Ainslie and N. Haslam. Hyperbolic discounting. In *Choice over time*, pages 57–92. Russell Sage Foundation, New York, NY, US, 1992. ISBN 978-0-87154-558-9.

[4] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama. Optuna: A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 2623–2631, Anchorage AK USA, July 2019. ACM. ISBN 978-1-4503-6201-6. doi: 10.1145/3292500.3330701. URL https://dl.acm.org/doi/10.1145/3292500.3330701.

[5] S. A. Alexander. The Archimedean trap: Why traditional reinforcement learning will probably not yield AGI. *Journal of Artificial General Intelligence*, 11(1):70–85, Jan. 2020. ISSN 1946-0163. doi: 10.2478/jagi-2020-0004. URL http://arxiv.org/abs/2002.10221. arXiv: 2002.10221.

[6] S. A. Ali. The mth Ratio Test: New Convergence Tests for Series. *The American Mathematical Monthly*, 115(6):514–524, June 2008. ISSN 0002-9890. doi: 10.1080/00029890.2008.11920558. Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/00029890.2008.11920558.

[7] E. Alonso, M. Peter, D. Goumard, and J. Romoff. Deep Reinforcement Learning for Navigation in AAA Video Games. *arXiv:2011.04764 [cs]*, Nov. 2020. URL http://arxiv.org/abs/2011.04764. arXiv: 2011.04764.

[8] R. Amit, R. Meir, and K. Ciosek. Discount Factor as a Regularizer in Reinforcement Learning. In *International Conference on Machine Learning*, pages 269–278. PMLR, Nov. 2020. ISSN: 2640-3498.

[9] M. Andrychowicz, A. Raichuk, P. Stańczyk, M. Orsini, S. Girgin, R. Marinier, L. Hussenot, M. Geist, O. Pietquin, M. Michalski, S. Gelly, and O. Bachem. What Matters In On-Policy Reinforcement Learning? A Large-Scale Empirical Study. *arXiv:2006.05990 [cs, stat]*, June 2020. URL http://arxiv.org/abs/2006.05990. arXiv: 2006.05990.

[10] G. Arechavaleta, J.-P. Laumond, H. Hicheur, and A. Berthoz. On the nonholonomic nature of human locomotion. *Autonomous Robots*, 25(1-2):25–35, Aug. 2008. ISSN 0929-5593, 1573-7527. doi: 10.1007/s10514-007-9075-2. URL http://link.springer.com/10.1007/s10514-007-9075-2.

[11] O. Arikan and D. A. Forsyth. Interactive motion generation from examples. *ACM Transactions on Graphics (TOG)*, 21(3):483–490, 2002.

[12] M. Bain and C. Sammut. A Framework for Behavioural Cloning. In *Machine Intelligence 15, Intelligent Agents [St. Catherine's College, Oxford, July 1995]*, pages 103–129, GBR, Jan. 1999. Oxford University. ISBN 978-0-19-853867-7.

[13] M. G. Bellemare, W. Dabney, and R. Munos. A Distributional Perspective on Reinforcement Learning. In *International Conference on Machine Learning*, pages 449–458. PMLR, July 2017. URL http://proceedings.mlr.press/v70/bellemare17a.html. ISSN: 2640-3498.

[14] R. Bellman. A Markovian Decision Process. *Journal of Mathematics and Mechanics*, 6(5): 679–684, 1957. ISSN 0095-9057. Publisher: Indiana University Mathematics Department.

[15] R. E. Bellman. *Dynamic Programming*. Dover Publications, Inc., USA, 1957. ISBN 978-0-486-42809-3.

[16] K. Bergamin, S. Clavet, D. Holden, and J. R. Forbes. Drecon: data-driven responsive control of physics-based characters. *ACM Transactions On Graphics (TOG)*, 38(6):1–11, 2019.

[17] D. S. Bernstein, S. Zilberstein, and N. Immerman. The complexity of decentralized control of Markov decision processes. In *Proceedings of the Sixteenth conference on Uncertainty in artificial intelligence*, UAI'00, pages 32–37, San Francisco, CA, USA, June 2000. Morgan Kaufmann Publishers Inc. ISBN 978-1-55860-709-5.

[18] J. Bradbury, R. Frostig, P. Hawkins, M. J. Johnson, C. Leary, D. Maclaurin, and S. Wanderman-Milne. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

[19] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba. OpenAI Gym. *arXiv:1606.01540 [cs]*, June 2016.

[20] J. Bruneau, A.-H. Olivier, and J. Pettré. Going Through, Going Around: A Study on Individual Avoidance of Groups. *IEEE Transactions on Visualization and Computer Graphics*, 21(4):9, Apr. 2015. doi: 10.1109/TVCG.2015.2391862. URL https://hal.inria.fr/hal-01149960.

[21] D. Budden, M. Hessel, J. Quan, S. Kapturowski, K. Baumli, S. Bhupatiraju, A. Guy, and M. King. RLax: Reinforcement Learning in JAX, 2020. URL http://github.com/deepmind/rlax.

[22] M. Carroll, R. Shah, M. K. Ho, T. L. Griffiths, S. A. Seshia, P. Abbeel, and A. Dragan. On the Utility of Learning about Humans for Human-AI Coordination. *arXiv:1910.05789 [cs, stat]*, Oct. 2019. URL http://arxiv.org/abs/1910.05789.

[23] P. S. Castro, S. Moitra, C. Gelada, S. Kumar, and M. G. Bellemare. Dopamine: A Research Framework for Deep Reinforcement Learning. 2018. URL http://arxiv.org/abs/1812.06110.

[24] N. Chentanez, M. Müller, M. Macklin, V. Makoviychuk, and S. Jeschke. Physics-based motion capture imitation with deep reinforcement learning. In *Proceedings of the 11th annual international conference on motion, interaction, and games*, pages 1–10, 2018.

[25] F. Chollet and others. Keras, 2015. URL https://keras.io.

[26] R. Choudhury, G. Swamy, D. Hadfield-Menell, and A. D. Dragan. On the Utility of Model Learning in HRI. In *Proceedings of the 14th ACM/IEEE International Conference on Human-Robot Interaction*, HRI '19, pages 317–325. IEEE Press, 2019. ISBN 978-1-5386-8555-6. event-place: Daegu, Republic of Korea.

[27] P. F. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei. Deep Reinforcement Learning from Human Preferences. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 4302–4310, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 978-1-5108-6096-4. event-place: Long Beach, California, USA.

[28] A. Clegg, W. Yu, J. Tan, C. K. Liu, and G. Turk. Learning to Dress: Synthesizing Human Dressing Motion via Deep Reinforcement Learning. *ACM Trans. Graph.*, 37(6), Dec. 2018. ISSN 0730-0301. doi: 10.1145/3272127.3275048. URL https://doi.org/10.1145/3272127.3275048. Place: New York, NY, USA Publisher: Association for Computing Machinery.

[29] S. Coros, P. Beaudoin, and M. Van de Panne. Robust task-based control policies for physics-based characters. In *ACM SIGGRAPH Asia 2009 papers*, pages 1–9. 2009.

[30] R. Coulom. Efficient selectivity and backup operators in Monte-Carlo tree search. In *Proceedings of the 5th international conference on Computers and games*, CG'06, pages 72–83, Berlin, Heidelberg, May 2006. Springer-Verlag. ISBN 978-3-540-75537-1.

[31] E. Coumans and Y. Bai. *PyBullet, a Python module for physics simulation for games, robotics and machine learning*. 2016. URL http://pybullet.org.

[32] S. Daftry, J. A. Bagnell, and M. Hebert. Learning Transferable Policies for Monocular Reactive MAV Control. *CoRR*, abs/1608.00627, 2016. URL http://arxiv.org/abs/1608.00627. _eprint: 1608.00627.

[33] B. C. Daniel, R. Marques, L. Hoyet, J. Pettré, and J. Blat. A Perceptually-Validated Metric for Crowd Trajectory Quality Evaluation. *Proceedings of the ACM on Computer Graphics and Interactive Techniques*, 4(3):1–18, Sept. 2021. ISSN 2577-6193. doi: 10.1145/3480136. URL https://dl.acm.org/doi/10.1145/3480136.

[34] P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, Y. Wu, and P. Zhokhov. OpenAI Baselines, 2017. URL https://github.com/openai/baselines. Publication Title: GitHub repository.

[35] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry. Implementation Matters in Deep Policy Gradients: A Case Study on PPO and TRPO.

arXiv:2005.12729 [cs, stat], May 2020. URL http://arxiv.org/abs/2005.12729. arXiv: 2005.12729.

[36] W. Fedus, C. Gelada, Y. Bengio, M. G. Bellemare, and H. Larochelle. Hyperbolic Discounting and Learning over Multiple Horizons. *arXiv:1902.06865 [cs, stat]*, Feb. 2019. arXiv: 1902.06865.

[37] M. Fortunato, M. G. Azar, B. Piot, J. Menick, I. Osband, A. Graves, V. Mnih, R. Munos, D. Hassabis, O. Pietquin, C. Blundell, and S. Legg. Noisy Networks for Exploration. *arXiv:1706.10295 [cs, stat]*, June 2017. URL http://arxiv.org/abs/1706.10295. arXiv: 1706.10295 version: 1.

[38] S. Fujimoto, H. van Hoof, and D. Meger. Addressing Function Approximation Error in Actor-Critic Methods. *arXiv:1802.09477 [cs, stat]*, Oct. 2018. URL http://arxiv.org/abs/1802.09477. arXiv: 1802.09477.

[39] V. G. Goecks, G. M. Gremillion, V. J. Lawhern, J. Valasek, and N. R. Waytowich. Integrating Behavior Cloning and Reinforcement Learning for Improved Performance in Dense and Sparse Reward Environments. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '20, pages 465–473, Richland, SC, 2020. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-1-4503-7518-4. event-place: Auckland, New Zealand.

[40] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016.

[41] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Networks. *Commun. ACM*, 63(11):139–144, Oct. 2020. ISSN 0001-0782. doi: 10.1145/3422622. URL https://doi.org/10.1145/3422622. Place: New York, NY, USA Publisher: Association for Computing Machinery.

[42] S. Guadarrama, A. Korattikara, O. Ramirez, P. Castro, E. Holly, S. Fishman, K. Wang, E. Gonina, N. Wu, E. Kokiopoulou, L. Sbaiz, J. Smith, G. Bartók, J. Berent, C. Harris, V. Vanhoucke, and E. Brevdo. TF-Agents: A library for Reinforcement Learning in TensorFlow, 2018. URL https://github.com/tensorflow/agents.

[43] D. Guo, L. Tang, X. Zhang, and Y.-C. Liang. Joint Optimization of Handover Control and Power Allocation Based on Multi-Agent Deep Reinforcement Learning. *IEEE Transactions on Vehicular Technology*, 69(11):13124–13138, Nov. 2020. ISSN 1939-9359. doi: 10.1109/TVT.2020.3020400. Conference Name: IEEE Transactions on Vehicular Technology.

[44] J. K. Gupta, M. Egorov, and M. Kochenderfer. Cooperative Multi-agent Control Using Deep Reinforcement Learning. In G. Sukthankar and J. A. Rodriguez-Aguilar, editors, *Autonomous Agents and Multiagent Systems*, pages 66–83, Cham, 2017. Springer International Publishing. ISBN 978-3-319-71682-4.

[45] S. J. Guy, J. Chhugani, S. Curtis, P. Dubey, M. Lin, and D. Manocha. PLEdestrians: A Least-Effort Approach to Crowd Simulation. *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, page 10 pages, 2010. ISSN 1727-5288. doi: 10.2312/SCA/SCA10/119-128. URL http://diglib.eg.org/handle/10.2312/SCA.SCA10.119-128. Artwork Size: 10 pages ISBN: 9783905674279 Publisher: The Eurographics Association.

[46] D. Ha, A. Dai, and Q. V. Le. HyperNetworks. *arXiv:1609.09106 [cs]*, Dec. 2016. URL http://arxiv.org/abs/1609.09106. arXiv: 1609.09106.

[47] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor. *arXiv:1801.01290 [cs, stat]*, Jan. 2018. URL http://arxiv.org/abs/1801.01290. arXiv: 1801.01290 version: 1.

[48] P. Hämäläinen, S. Eriksson, E. Tanskanen, V. Kyrki, and J. Lehtinen. Online motion synthesis using sequential monte carlo. *ACM Transactions on Graphics (TOG)*, 33(4):1–12, 2014.

[49] P. Hämäläinen, J. Rajamäki, and C. K. Liu. Online control of simulated humanoids using particle belief propagation. *ACM Transactions on Graphics (TOG)*, 34(4):1–13, 2015.

[50] E. A. Hansen, D. S. Bernstein, and S. Zilberstein. Dynamic programming for partially observable stochastic games. In *Proceedings of the 19th national conference on Artifical intelligence*, AAAI'04, pages 709–715, San Jose, California, July 2004. AAAI Press. ISBN 978-0-262-51183-4.

[51] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-2649-2. Number: 7825 Publisher: Nature Publishing Group.

[52] B. Haworth, G. Berseth, S. Moon, P. Faloutsos, and M. Kapadia. Deep Integration of Physical Humanoid Control and Crowd Navigation. pages 1–10, Oct. 2020. doi: 10.1145/3424636.3426894.

[53] H. He. The State of Machine Learning Frameworks in 2019. *The Gradient*, 2019. URL https://thegradient.pub/state-of-ml-frameworks-2019-pytorch-dominates-research-tensorflow-dominates-industry/.

[54] J. Heek, A. Levskaya, A. Oliver, M. Ritter, B. Rondepierre, A. Steiner, and M. v. Zee. Flax: A neural network library and ecosystem for JAX, 2020. URL http://github.com/google/flax.

[55] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger. Deep Reinforcement Learning that Matters. *arXiv:1709.06560 [cs, stat]*, Sept. 2017. URL http://arxiv.org/abs/1709.06560. arXiv: 1709.06560 version: 1.

[56] T. Hennigan, T. Cai, T. Norman, and I. Babuschkin. Haiku: Sonnet for JAX, 2020. URL http://github.com/deepmind/dm-haiku.

[57] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver. Rainbow: Combining Improvements in Deep Reinforcement Learning. *arXiv:1710.02298 [cs]*, Oct. 2017. URL http://arxiv.org/abs/1710.02298. arXiv: 1710.02298.

[58] A. Hill, A. Raffin, M. Ernestus, A. Gleave, A. Kanervisto, R. Traore, P. Dhariwal, C. Hesse, O. Klimov, A. Nichol, M. Plappert, A. Radford, J. Schulman, S. Sidor, and Y. Wu. Stable Baselines, 2018. URL https://github.com/hill-a/stable-baselines. Publication Title: GitHub repository.

[59] G. Hinton, O. Vinyals, and J. Dean. Distilling the Knowledge in a Neural Network. *arXiv:1503.02531 [cs, stat]*, Mar. 2015. URL http://arxiv.org/abs/1503.02531. arXiv: 1503.02531.

[60] J. Ho and S. Ermon. Generative Adversarial Imitation Learning. *arXiv:1606.03476 [cs]*, June 2016. URL http://arxiv.org/abs/1606.03476. arXiv: 1606.03476.

[61] S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8): 1735–1780, Nov. 1997. ISSN 0899-7667, 1530-888X. doi: 10.1162/neco.1997.9.8.1735. URL http://www.mitpressjournals.org/doi/10.1162/neco.1997.9.8.1735.

[62] J. Hu, H. Wu, S. A. Harding, S. Jiang, and S.-w. Liao. RIIT: Rethinking the Importance of Implementation Tricks in Multi-Agent Reinforcement Learning. *arXiv:2102.03479 [cs]*, Feb. 2021. URL http://arxiv.org/abs/2102.03479. arXiv: 2102.03479.

[63] K. Hu, M. B. Haworth, G. Berseth, V. Pavlovic, P. Faloutsos, and M. Kapadia. Heterogeneous Crowd Simulation using Parametric Reinforcement Learning. *IEEE Transactions on Visualization and Computer Graphics*, pages 1–1, 2022. ISSN 1077-2626, 1941-0506, 2160-9306. doi: 10.1109/TVCG.2021.3139031. URL https://ieeexplore.ieee.org/document/9665307/.

[64] S. Hu, J. Hu, and S.-w. Liao. Noisy-MAPPO: Noisy Credit Assignment for Cooperative Multi-agent Actor-Critic methods. *arXiv:2106.14334 [cs]*, June 2021. URL http://arxiv.org/abs/2106.14334. arXiv: 2106.14334 version: 1.

[65] S. Huang, R. Dossa, and C. Ye. CleanRL: High-quality Single-file Implementation of Deep Reinforcement Learning algorithms, 2020. URL https://github.com/vwxyzjn/cleanrl/. Publication Title: GitHub repository.

[66] S. Huang, R. F. J. Dossa, A. Raffin, A. Kanervisto, and W. Wang. The 37 Implementation Details of Proximal Policy Optimization. In *ICLR Blog Track*, 2022. URL https://iclr-blog-track.github.io/2022/03/25/ppo-implementation-details/.

[67] R. Hughes, J. Ondřej, and J. Dingliana. Holonomic collision avoidance for virtual crowds. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '14, pages 103–111, Goslar, DEU, July 2015. Eurographics Association.

[68] M. Hüttenrauch, A. Sosic, and G. Neumann. Deep Reinforcement Learning for Swarm Systems. *arXiv:1807.06613 [cs, stat]*, June 2019. URL http://arxiv.org/abs/1807.06613. arXiv: 1807.06613.

[69] M. Isogawa, Y. Yuan, M. O'Toole, and K. Kitani. Optical Non-Line-of-Sight Physics-Based 3D Human Pose Estimation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7011–7020, Seattle, WA, USA, June 2020. IEEE. ISBN 9781728171685. doi: 10.1109/CVPR42600.2020.00704. URL https://ieeexplore.ieee.org/document/9157058/.

[70] Y. Jiang, T. Van Wouwe, F. De Groote, and C. K. Liu. Synthesis of Biologically Realistic Human Motion Using Joint Torque Actuation. *ACM Trans. Graph.*, 38(4), July 2019. ISSN 0730-0301. doi: 10.1145/3306346.3322966. URL https://doi.org/10.1145/3306346.3322966. Place: New York, NY, USA Publisher: Association for Computing Machinery.

[71] N. L. Johnson, S. Kotz, and N. Balakrishnan. *Continuous univariate distributions*. Wiley series in probability and mathematical statistics. Wiley, New York, 2nd ed edition, 1994. ISBN 978-0-471-58495-7 978-0-471-58494-0.

[72] A. L. Jones. A Clearer Proof of the Policy Gradient Theorem. *andyljones.com*, 2020.

[73] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange. Unity: A General Platform for Intelligent Agents. *arXiv:1809.02627 [cs, stat]*, May 2020. URL http://arxiv.org/abs/1809.02627. arXiv: 1809.02627.

[74] L. P. Kaelbling, M. L. Littman, and A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134, May 1998. ISSN 0004-3702. doi: 10.1016/S0004-3702(98)00023-X. URL https://www.sciencedirect.com/science/article/pii/S000437029800023X.

[75] S. Kakade. A natural policy gradient. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, pages 1531–1538, Cambridge, MA, USA, Jan. 2001. MIT Press.

[76] A. Karpathy. I've been using PyTorch a few months now and I've never felt better. [...], May 2017. URL https://twitter.com/karpathy/status/868178954032513024.

[77] L. Kidzin'ski, S. P. Mohanty, C. Ong, Z. Huang, S. Zhou, A. Pechenko, A. Stelmaszczyk, P. Jarosik, M. Pavlov, S. Kolesnikov, S. Plis, Z. Chen, Z. Zhang, J. Chen, J. Shi, Z. Zheng, C. Yuan, Z. Lin, H. Michalewski, P. Miłoś, B. Osiński, A. Melnik, M. Schilling, H. Ritter, S. Carroll, J. Hicks, S. Levine, M. Salathé, and S. Delp. Learning to Run challenge solutions: Adapting reinforcement learning methods for neuromusculoskeletal environments. *arXiv:1804.00361 [cs, stat]*, Apr. 2018. URL http://arxiv.org/abs/1804.00361. arXiv: 1804.00361.

[78] L. Kidziński, S. P. Mohanty, C. Ong, J. Hicks, S. Francis, S. Levine, M. Salathé, and S. Delp. Learning to Run challenge: Synthesizing physiologically accurate motion using deep reinforcement learning. In S. Escalera and M. Weimer, editors, *NIPS 2017 Competition Book*. Springer, Springer, 2018.

[79] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. Hamrick, J. Grout, S. Corlay, P. Ivanov, D. Avila, S. Abdalla, C. Willing, and J. d. team. Jupyter Notebooks - a publishing format for reproducible computational workflows. In F. Loizides and B. Scmidt, editors, *Positioning and Power in Academic Publishing: Players, Agents and Agendas*, pages 87–90. IOS Press, 2016. URL https://eprints.soton.ac.uk/403913/.

[80] L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. *Proceedings of ACM SIGGRAPH 2002, July*, pages 473–482, 2002.

[81] V. C. V. Kumar, S. Ha, and C. K. Liu. Learning a unified control policy for safe falling. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3940–3947, 2017. doi: 10.1109/IROS.2017.8206246.

[82] A. Kwiatkowski. RedTachyon/coltra-rl: Reward paper, July 2023. URL `https://zenodo.org/record/8126793`.

[83] A. Kwiatkowski. RedTachyon/CrowdAI: Thesis version, Aug. 2023. URL `https://zenodo.org/record/8205093`.

[84] A. Kwiatkowski, E. Alvarado, V. Kalogeiton, C. K. Liu, J. Pettré, M. van de Panne, and M. Cani. A Survey on Reinforcement Learning Methods in Character Animation. *Computer Graphics Forum*, 41(2):613–639, May 2022. ISSN 0167-7055, 1467-8659. doi: 10.1111/cgf.14504. URL `https://onlinelibrary.wiley.com/doi/10.1111/cgf.14504`.

[85] A. Kwiatkowski, V. Kalogeiton, J. Pettré, and M.-P. Cani. Reward Function Design for Crowd Simulation via Reinforcement Learning, Sept. 2023. URL `http://arxiv.org/abs/2309.12841`. arXiv:2309.12841 [cs].

[86] A. Kwiatkowski, V. Kalogeiton, J. Pettré, and M.-P. Cani. UGAE: A Novel Approach to Non-exponential Discounting, Feb. 2023. URL `http://arxiv.org/abs/2302.05740`. arXiv:2302.05740 [cs].

[87] A. Kwiatkowski, V. Kalogeiton, J. Pettré, and M.-P. Cani. Understanding reinforcement learned crowds. *Computers & Graphics*, 110:28–37, Feb. 2023. ISSN 00978493. doi: 10.1016/j.cag.2022.11.007. URL `https://linkinghub.elsevier.com/retrieve/pii/S0097849322002035`.

[88] M. Lanctot, E. Lockhart, J.-B. Lespiau, V. Zambaldi, S. Upadhyay, J. Pérolat, S. Srinivasan, F. Timbers, K. Tuyls, S. Omidshafiei, D. Hennes, D. Morrill, P. Muller, T. Ewalds, R. Faulkner, J. Kramár, B. De Vylder, B. Saeta, J. Bradbury, D. Ding, S. Borgeaud, M. Lai, J. Schrittwieser, T. Anthony, E. Hughes, I. Danihelka, and J. Ryan-Davis. OpenSpiel: A Framework for Reinforcement Learning in Games. *arXiv:1908.09453 [cs]*, Sept. 2020. URL `http://arxiv.org/abs/1908.09453`. arXiv: 1908.09453.

[89] T. Lattimore and M. Hutter. Time Consistent Discounting. In J. Kivinen, C. Szepesvári, E. Ukkonen, and T. Zeugmann, editors, *Algorithmic Learning Theory*, Lecture Notes in Computer Science, pages 383–397, Berlin, Heidelberg, 2011. Springer. ISBN 978-3-642-24412-4. doi: 10.1007/978-3-642-24412-4_30.

[90] J. Lee and K. H. Lee. Precomputing avatar behavior from human motion data. *Graphical models*, 68(2):158–174, 2006.

[91] J. Lee, J. Chai, P. S. Reitsma, J. K. Hodgins, and N. S. Pollard. Interactive control of avatars animated with human motion data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 491–500, 2002.

[92] J. Lee, M. X. Grey, S. Ha, T. Kunz, S. Jain, Y. Ye, S. S. Srinivasa, M. Stilman, and C. K. Liu. Dart: Dynamic animation and robotics toolkit. *The Journal of Open Source Software*, 3(22), 2018.

[93] J. Lee, J. Won, and J. Lee. Crowd simulation by deep reinforcement learning. In *Proceedings of the 11th Annual International Conference on Motion, Interaction, and Games*, pages 1–7, Limassol Cyprus, Nov. 2018. ACM. ISBN 978-1-4503-6015-9. doi: 10.1145/3274247.3274510. URL https://dl.acm.org/doi/10.1145/3274247.3274510.

[94] S. Lee, M. Park, K. Lee, and J. Lee. Scalable muscle-actuated human simulation and control. *ACM Transactions On Graphics (TOG)*, 38(4):1–13, 2019.

[95] S. Lee, S. Lee, Y. Lee, and J. Lee. Learning a family of motor skills from a single motion clip. *ACM Transactions on Graphics (TOG)*, 40(4):1–13, 2021.

[96] Y. Lee, K. Wampler, G. Bernstein, J. Popović, and Z. Popović. Motion fields for interactive character locomotion. *ACM Transactions on Graphics*, 29(6):138:1–138:8, Dec. 2010. ISSN 0730-0301. doi: 10.1145/1882261.1866160. URL https://doi.org/10.1145/1882261.1866160.

[97] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. Gonzalez, M. Jordan, and I. Stoica. RLlib: Abstractions for Distributed Reinforcement Learning. In J. Dy and A. Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 3053–3062. PMLR, July 2018. URL http://proceedings.mlr.press/v80/liang18b.html.

[98] J. Liang, V. Makoviychuk, A. Handa, N. Chentanez, M. Macklin, and D. Fox. GPU-Accelerated Robotic Simulation for Distributed Reinforcement Learning. *arXiv:1810.05762 [cs]*, Oct. 2018. URL http://arxiv.org/abs/1810.05762. arXiv: 1810.05762.

[99] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv:1509.02971 [cs, stat]*, Sept. 2015. URL http://arxiv.org/abs/1509.02971. arXiv: 1509.02971 version: 1.

[100] H. Y. Ling, H. Yu Ling, F. Zinno, G. Cheng, and M. van de Panne. Character Controllers Using Motion VAEs. *ACM Trans. Graph*, 39(4):12, 2020. doi: 10.1145/3386569.3392422. URL https://doi.org/10.1145/3386569.3392422.

[101] D. Liu, Z. Wang, B. Lu, M. Cong, H. Yu, and Q. Zou. A Reinforcement Learning-Based Framework for Robot Manipulation Skill Acquisition. *IEEE Access*, 8:108429–108437, 2020. ISSN 2169-3536. doi: 10.1109/ACCESS.2020.3001130. Conference Name: IEEE Access.

[102] L. Liu and J. Hodgins. Learning to schedule control fragments for physics-based characters using deep q-learning. *ACM Transactions on Graphics (TOG)*, 36(3):1–14, 2017.

[103] L. Liu and J. Hodgins. Learning basketball dribbling skills using trajectory optimization and deep reinforcement learning. *ACM Transactions on Graphics (TOG)*, 37(4):1–14, 2018.

[104] L. Liu, K. Yin, M. van de Panne, T. Shao, and W. Xu. Sampling-based contact-rich motion control. In *ACM SIGGRAPH 2010 papers*, pages 1–10. 2010.

[105] L. Liu, M. V. D. Panne, and K. Yin. Guided learning of control graphs for physics-based characters. *ACM Transactions on Graphics (TOG)*, 35(3):1–14, 2016.

[106] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan. Towards Optimally Decentralized Multi-Robot Collision Avoidance via Deep Reinforcement Learning. *arXiv:1709.10082 [cs]*, May 2018. arXiv: 1709.10082.

[107] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. *arXiv:1706.02275 [cs]*, Mar. 2020. URL `http://arxiv.org/abs/1706.02275`.

[108] Y.-S. Luo, J. H. Soeseno, T. P.-C. Chen, and W.-C. Chen. CARL: Controllable Agent with Reinforcement Learning for Quadruped Locomotion. *ACM Trans. Graph.*, 39(4), July 2020. ISSN 0730-0301. doi: 10.1145/3386569.3392433. URL `https://doi.org/10.1145/3386569.3392433`. Place: New York, NY, USA Publisher: Association for Computing Machinery.

[109] L.-K. Ma, Z. Yang, X. Tong, B. Guo, and K. Yin. Learning and exploring motor skills with spacetime bounds. In *Computer Graphics Forum*, volume 40, pages 251–263. Wiley Online Library, 2021.

[110] M. Macklin. Differentiable Physics Simulation for Learning and Robotics. In *GTC 2021*, 2021.

[111] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015. URL `http://tensorflow.org/`.

[112] J. Merel, S. Tunyasuvunakool, A. Ahuja, Y. Tassa, L. Hasenclever, V. Pham, T. Erez, G. Wayne, and N. Heess. Catch & carry: reusable neural controllers for vision-guided whole-body tasks. *ACM Transactions on Graphics (TOG)*, 39(4):39–1, 2020.

[113] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb. 2015. ISSN 1476-4687. doi: 10.1038/nature14236. URL `https://www.nature.com/articles/nature14236`.

[114] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 1928–1937, New York, NY, USA, June 2016. JMLR.org.

[115] I. Mordatch, E. Todorov, and Z. Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics*, 31(4), 2012. ISSN 07300301. doi: 10.1145/2185520.2185539.

[116] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, and I. Stoica. Ray: A Distributed Framework for Emerging AI Applications. *arXiv:1712.05889 [cs, stat]*, Sept. 2018. URL `http://arxiv.org/abs/1712.05889`. arXiv: 1712.05889.

[117] L. Mourot, L. Hoyet, F. L. Clerc, F. Schnitzler, and P. Hellier. A survey on deep learning for skeleton-based human animation. *arXiv preprint arXiv:2110.06901*, 2021.

[118] A. Naik, R. Shariff, N. Yasui, H. Yao, and R. S. Sutton. Discounted Reinforcement Learning Is Not an Optimization Problem. *arXiv:1910.02140 [cs]*, Nov. 2019. arXiv: 1910.02140.

[119] A. Y. Ng, D. Harada, and S. J. Russell. Policy Invariance Under Reward Transformations: Theory and Application to Reward Shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pages 278–287, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1-55860-612-2.

[120] S. Park, H. Ryu, S. Lee, S. Lee, and J. Lee. Learning predict-and-simulate policies from unorganized human motion data. *ACM Transactions on Graphics (TOG)*, 38(6):1–11, 2019.

[121] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d. Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

[122] X. B. Peng and M. van de Panne. Learning locomotion skills using deeprl: Does the choice of action space matter? In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 1–13, 2017.

[123] X. B. Peng, G. Berseth, and M. van de Panne. Dynamic Terrain Traversal Skills Using Reinforcement Learning. *ACM Trans. Graph.*, 34(4), July 2015. ISSN 0730-0301. doi: 10.1145/2766910. URL `https://doi.org/10.1145/2766910`. Place: New York, NY, USA Publisher: Association for Computing Machinery.

[124] X. B. Peng, G. Berseth, and M. van de Panne. Terrain-Adaptive Locomotion Skills Using Deep Reinforcement Learning. *ACM Trans. Graph.*, 35(4), July 2016. ISSN 0730-0301. doi: 10.1145/2897824.2925881. URL `https://doi.org/10.1145/2897824.2925881`. Place: New York, NY, USA Publisher: Association for Computing Machinery.

[125] X. B. Peng, G. Berseth, K. Yin, and M. Van De Panne. DeepLoco: Dynamic Locomotion Skills Using Hierarchical Deep Reinforcement Learning. *ACM Trans. Graph.*, 36(4), July 2017. ISSN 0730-0301. doi: 10.1145/3072959.3073602. URL `https://doi.org/10.1145/3072959.3073602`. Place: New York, NY, USA Publisher: Association for Computing Machinery.

[126] X. B. Peng, P. Abbeel, S. Levine, and M. van de Panne. DeepMimic: Example-Guided Deep Reinforcement Learning of Physics-Based Character Skills. *ACM Trans. Graph.*, 37

(4), July 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201311. URL https://doi.org/10.1145/3197517.3201311. Place: New York, NY, USA Publisher: Association for Computing Machinery.

[127] X. B. Peng, A. Kanazawa, J. Malik, P. Abbeel, and S. Levine. Sfv: Reinforcement learning of physical skills from videos. *ACM Transactions On Graphics (TOG)*, 37(6):1–14, 2018.

[128] X. B. Peng, E. Coumans, T. Zhang, T.-W. Lee, J. Tan, and S. Levine. Learning agile robotic locomotion skills by imitating animals. *arXiv preprint arXiv:2004.00784*, 2020.

[129] X. B. Peng, Z. Ma, P. Abbeel, S. Levine, and A. Kanazawa. AMP: Adversarial Motion Priors for Stylized Physics-Based Character Control. *ACM Trans. Graph.*, 40(4), July 2021. ISSN 0730-0301. doi: 10.1145/3450626.3459670. URL https://doi.org/10.1145/3450626.3459670. Place: New York, NY, USA Publisher: Association for Computing Machinery.

[130] S. Pitis. Rethinking the Discount Factor in Reinforcement Learning: A Decision Theoretic Approach. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33:7949–7956, July 2019. ISSN 2374-3468, 2159-5399. doi: 10.1609/aaai.v33i01.33017949. URL https://aaai.org/ojs/index.php/AAAI/article/view/4795.

[131] D. Premack and G. Woodruff. Does the chimpanzee have a theory of mind? *Behavioral and Brain Sciences*, 1(4):515–526, 1978. doi: 10.1017/S0140525X00076512.

[132] N. C. Rabinowitz, F. Perbet, H. F. Song, C. Zhang, S. M. A. Eslami, and M. Botvinick. Machine Theory of Mind. *arXiv:1802.07740 [cs]*, Mar. 2018. URL http://arxiv.org/abs/1802.07740.

[133] A. Raffin. *RL Baselines3 Zoo*. GitHub, 2020. URL https://github.com/DLR-RM/rl-baselines3-zoo. Publication Title: GitHub repository.

[134] A. Raffin, A. Hill, M. Ernestus, A. Gleave, A. Kanervisto, and N. Dormann. *Stable Baselines3*. GitHub, 2019. URL https://github.com/DLR-RM/stable-baselines3. Publication Title: GitHub repository.

[135] J. Rajamäki and P. Hämäläinen. Augmenting sampling based controllers with machine learning. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 1–9, 2017.

[136] T. Rashid, M. Samvelyan, C. S. de Witt, G. Farquhar, J. Foerster, and S. Whiteson. QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. *arXiv:1803.11485 [cs, stat]*, June 2018. URL http://arxiv.org/abs/1803.11485. arXiv: 1803.11485.

[137] T. Rashid, G. Farquhar, B. Peng, and S. Whiteson. Weighted QMIX: Expanding Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. In H. Larochelle, M. Ranzato, R. Hadsell, M.-F. Balcan, and H.-T. Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020. URL https://proceedings.neurips.cc/paper/2020/hash/73a427badebe0e32caa2e1fc7530b7f3-Abstract.html.

[138] D. Reda, T. Tao, and M. van de Panne. Learning to locomote: Understanding how environment design matters for deep reinforcement learning. In *Motion, Interaction and Games*, pages 1–10. 2020.

[139] S. Ross, G. Gordon, and D. Bagnell. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. In G. Gordon, D. Dunson, and M. Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, volume 15 of *Proceedings of Machine Learning Research*, pages 627–635, Fort Lauderdale, FL, USA, Apr. 2011. PMLR. URL `http://proceedings.mlr.press/v15/ross11a.html`.

[140] R. Y. Rubinstein. *Simulation and the Monte Carlo Method*. John Wiley & Sons, Inc., USA, 1st edition, 1981. ISBN 978-0-471-08917-9.

[141] A. Santoro, D. Raposo, D. G. T. Barrett, M. Malinowski, R. Pascanu, P. Battaglia, and T. Lillicrap. A simple neural network module for relational reasoning. *arXiv:1706.01427 [cs]*, June 2017. URL `http://arxiv.org/abs/1706.01427`.

[142] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. Prioritized Experience Replay. *arXiv:1511.05952 [cs]*, Feb. 2016. URL `http://arxiv.org/abs/1511.05952`. arXiv: 1511.05952.

[143] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz. Trust Region Policy Optimization. In *International Conference on Machine Learning*, pages 1889–1897. PMLR, June 2015. ISSN: 1938-7228.

[144] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal Policy Optimization Algorithms. *arXiv:1707.06347 [cs]*, Aug. 2017.

[145] J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *arXiv:1506.02438 [cs]*, Oct. 2018.

[146] A. Seth, M. Sherman, J. A. Reinbolt, and S. L. Delp. OpenSim: a musculoskeletal modeling and simulation framework for in silico investigations and exchange. *Procedia IUTAM*, 2:212–232, 2011. ISSN 22109838. doi: 10.1016/j.piutam.2011.04.021. URL `https://linkinghub.elsevier.com/retrieve/pii/S2210983811000228`.

[147] C. R. Shelton. Balancing multiple sources of reward in reinforcement learning. In *Proceedings of the 13th International Conference on Neural Information Processing Systems*, NIPS'00, pages 1038–1044, Cambridge, MA, USA, Jan. 2000. MIT Press.

[148] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller. Deterministic policy gradient algorithms. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages I–387–I–395, Beijing, China, June 2014. JMLR.org.

[149] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm. *arXiv:1712.01815 [cs]*, Dec. 2017. URL `http://arxiv.org/abs/1712.01815`.

[150] D. Silver, S. Singh, D. Precup, and R. S. Sutton. Reward is enough. *Artificial Intelligence*, 299:103535, Oct. 2021. ISSN 0004-3702. doi: 10.1016/j.artint.2021.103535. URL https://www.sciencedirect.com/science/article/pii/S0004370221000862.

[151] G. Snook. Simplified 3D Movement and Pathfinding Using Navigation Meshes. In M. DeLoura, editor, *Game Programming Gems*, pages 288–304. Charles River Media, 2000.

[152] N. Soares. The Value Learning Problem. In R. V. Yampolskiy, editor, *Artificial Intelligence Safety and Security*, pages 89–97. Chapman and Hall/CRC, First edition. | Boca Raton, FL : CRC Press/Taylor & Francis Group, 2018., 1 edition, July 2018. ISBN 978-1-351-25138-9. doi: 10.1201/9781351251389-7. URL https://www.taylorfrancis.com/books/9781351251372/chapters/10.1201/9781351251389-7.

[153] K. Son, D. Kim, W. J. Kang, D. E. Hostallero, and Y. Yi. QTRAN: Learning to Factorize with Transformation for Cooperative Multi-Agent Reinforcement Learning. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 5887–5896. PMLR, June 2019. URL http://proceedings.mlr.press/v97/son19a.html.

[154] Stack Overflow. Stack Overflow Developer Survey 2020, 2020. URL https://insights.stackoverflow.com/survey/2020.

[155] R. H. Strotz. Myopia and Inconsistency in Dynamic Utility Maximization. *The Review of Economic Studies*, 23(3):165–180, 1955. ISSN 0034-6527. doi: 10.2307/2295722. Publisher: [Oxford University Press, Review of Economic Studies, Ltd.].

[156] L. Sun, J. Zhai, and W. Qin. Crowd Navigation in an Unknown and Dynamic Environment Based on Deep Reinforcement Learning. *IEEE Access*, 7:109544–109554, 2019. ISSN 2169-3536. doi: 10.1109/ACCESS.2019.2933492. Conference Name: IEEE Access.

[157] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel. Value-Decomposition Networks For Cooperative Multi-Agent Learning. *arXiv:1706.05296 [cs]*, June 2017. URL http://arxiv.org/abs/1706.05296.

[158] R. S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3(1):9–44, Aug. 1988. ISSN 1573-0565. doi: 10.1007/BF00115009. URL https://doi.org/10.1007/BF00115009.

[159] R. S. Sutton and A. G. Barto. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, USA, 1st edition, 1998. ISBN 978-0-262-19398-6.

[160] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. A Bradford Book, Cambridge, MA, USA, 2018. ISBN 978-0-262-03924-6.

[161] R. S. Sutton, D. McAllester, S. Singh, and Y. Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, pages 1057–1063, Cambridge, MA, USA, Nov. 1999. MIT Press.

[162] Y. Tassa, T. Erez, and E. Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913, 2012. doi: 10.1109/IROS.2012.6386025.

[163] J. K. Terry, N. Grammel, A. Hari, L. Santos, and B. Black. Revisiting Parameter Sharing In Multi-Agent Deep Reinforcement Learning. *arXiv:2005.13625 [cs, stat]*, Nov. 2020. URL `http://arxiv.org/abs/2005.13625`. arXiv: 2005.13625 version: 5.

[164] J. K. Terry, B. Black, M. Jayakumar, A. Hari, R. Sullivan, L. Santos, C. Dieffendahl, N. L. Williams, Y. Lokesh, C. Horsch, and P. Ravi. PettingZoo: Gym for Multi-Agent Reinforcement Learning. *arXiv:2009.14471 [cs, stat]*, Feb. 2021. URL `http://arxiv.org/abs/2009.14471`. arXiv: 2009.14471.

[165] J. K. Terry, N. Grammel, B. Black, A. Hari, C. Horsch, and L. Santos. Agent Environment Cycle Games. *arXiv:2009.13051 [cs, stat]*, May 2021. URL `http://arxiv.org/abs/2009.13051`. arXiv: 2009.13051.

[166] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, Oct. 2012. doi: 10.1109/IROS.2012.6386109. ISSN: 2153-0866.

[167] W. Toll and J. Pettré. Algorithms for Microscopic Crowd Simulation: Advancements in the 2010s. *Computer Graphics Forum*, 40(2):731–754, May 2021. ISSN 0167-7055, 1467-8659. doi: 10.1111/cgf.142664. URL `https://onlinelibrary.wiley.com/doi/10.1111/cgf.142664`.

[168] F. Torabi, G. Warnell, and P. Stone. Behavioral Cloning from Observation. In *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, IJCAI'18, pages 4950–4957. AAAI Press, 2018. ISBN 978-0-9992411-2-7. event-place: Stockholm, Sweden.

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

[170] A. Treuille, Y. Lee, and Z. Popović. Near-optimal character animation with continuous control. In *ACM SIGGRAPH 2007 papers*, pages 7–es. 2007.

[171] H. van Hasselt, A. Guez, and D. Silver. Deep Reinforcement Learning with Double Q-learning. *arXiv:1509.06461 [cs]*, Dec. 2015. URL `http://arxiv.org/abs/1509.06461`. arXiv: 1509.06461.

[172] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention Is All You Need. *arXiv:1706.03762 [cs]*, Dec. 2017. URL `http://arxiv.org/abs/1706.03762`.

[173] J. Wang, Z. Ren, T. Liu, Y. Yu, and C. Zhang. QPLEX: Duplex Dueling Multi-Agent Q-Learning. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL `https://openreview.net/forum?id=RcmkOxxIQV`.

[174] T. Wang, Y. Guo, M. Shugrina, and S. Fidler. Unicon: Universal neural controller for physics-based character motion. *arXiv preprint arXiv:2011.15119*, 2020.

[175] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas. Dueling Network Architectures for Deep Reinforcement Learning. In *International Conference on Machine Learning*, pages 1995–2003. PMLR, June 2016. URL http://proceedings.mlr.press/v48/wangf16.html. ISSN: 1938-7228.

[176] C. J. C. H. Watkins and P. Dayan. Q-learning. *Machine Learning*, 8(3):279–292, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992698. URL https://doi.org/10.1007/BF00992698.

[177] J. Weng, H. Chen, D. Yan, K. You, A. Duburcq, M. Zhang, H. Su, and J. Zhu. Tianshou: a Highly Modularized Deep Reinforcement Learning Library. *arXiv:2107.14171 [cs]*, July 2021. URL http://arxiv.org/abs/2107.14171. arXiv: 2107.14171.

[178] K. Werling, D. Omens, J. Lee, I. Exarchos, and C. K. Liu. Fast and Feature-Complete Differentiable Physics for Articulated Rigid Bodies with Contact. mar 2021. URL https://arxiv.org/abs/2103.16021v3.

[179] M. W. Whittle. *Gait analysis: an introduction*. Butterworth-Heinemann, Elsevier, Edinburgh, 4th ed., reprinted edition, 2008. ISBN 978-0-7506-8883-3.

[180] R. J. Williams. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine Language*, 8(3-4):229–256, May 1992. ISSN 0885-6125. doi: 10.1007/BF00992696.

[181] J. Won and J. Lee. Learning body shape variation in physics-based characters. *ACM Transactions on Graphics (TOG)*, 38(6):1–12, 2019.

[182] J. Won, D. Gopinath, and J. Hodgins. A scalable approach to control diverse behaviors for physically simulated characters. *ACM Transactions on Graphics (TOG)*, 39(4):33–1, 2020.

[183] Y. Wu, E. Mansimov, S. Liao, R. Grosse, and J. Ba. Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. *arXiv:1708.05144 [cs]*, Aug. 2017. URL http://arxiv.org/abs/1708.05144. arXiv: 1708.05144.

[184] Z. Xie, H. Y. Ling, N. H. Kim, and M. van de Panne. ALLSTEPS: Curriculum-driven Learning of Stepping Stone Skills. In *Proc. ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 2020.

[185] D. Xu, X. Huang, Z. Li, and X. Li. Local motion simulation using deep reinforcement learning. *Transactions in GIS*, 24(3):756–779, 2020. ISSN 1467-9671. doi: 10.1111/tgis.12620. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/tgis.12620. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/tgis.12620.

[186] P. Xu and I. Karamouzas. Human-Inspired Multi-Agent Navigation using Knowledge Distillation. *arXiv:2103.10000 [cs]*, Mar. 2021. URL http://arxiv.org/abs/2103.10000. arXiv: 2103.10000.

[187] Y. Yang, J. Hao, B. Liao, K. Shao, G. Chen, W. Liu, and H. Tang. Qatten: A General Framework for Cooperative Multiagent Reinforcement Learning. *CoRR*, abs/2002.03939, 2020. URL https://arxiv.org/abs/2002.03939. _eprint: 2002.03939.

[188] Z. Yin, Z. Yang, M. Van De Panne, and K. Yin. Discovering Diverse Athletic Jumping Strategies. *ACM Trans. Graph.*, 40(4), July 2021. ISSN 0730-0301. doi: 10.1145/3450626.3459817. URL https://doi.org/10.1145/3450626.3459817. Place: New York, NY, USA Publisher: Association for Computing Machinery.

[189] C. Yu, A. Velu, E. Vinitsky, Y. Wang, A. Bayen, and Y. Wu. The Surprising Effectiveness of PPO in Cooperative, Multi-Agent Games. *arXiv:2103.01955 [cs]*, July 2021. URL http://arxiv.org/abs/2103.01955. arXiv: 2103.01955.

[190] W. Yu, G. Turk, and C. K. Liu. Learning Symmetric and Low-Energy Locomotion. *ACM Trans. Graph.*, 37(4), July 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201397. URL https://doi.org/10.1145/3197517.3201397. Place: New York, NY, USA Publisher: Association for Computing Machinery.

[191] Y. Yuan and K. Kitani. Residual force control for agile human behavior imitation and extended motion synthesis. *arXiv preprint arXiv:2006.07364*, 2020.

[192] Y. Yuan, S.-E. Wei, T. Simon, K. Kitani, and J. M. Saragih. SimPoE: Simulated Character Control for 3D Human Pose Estimation. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2021, virtual, June 19-25, 2021*, pages 7159–7169. Computer Vision Foundation / IEEE, 2021. URL https://openaccess.thecvf.com/content/CVPR2021/html/Yuan_SimPoE_Simulated_Character_Control_for_3D_Human_Pose_Estimation_CVPR_2021_paper.html.

[193] M. Zaheer, S. Kottur, S. Ravanbakhsh, B. Poczos, R. R. Salakhutdinov, and A. J. Smola. Deep Sets. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper/2017/hash/f22e4747da1aa27e363d86d40ff442fe-Abstract.html.

[194] V. Zambaldi, D. Raposo, A. Santoro, V. Bapst, Y. Li, I. Babuschkin, K. Tuyls, D. Reichert, T. Lillicrap, E. Lockhart, M. Shanahan, V. Langston, R. Pascanu, M. Botvinick, O. Vinyals, and P. Battaglia. Relational Deep Reinforcement Learning. *arXiv:1806.01830 [cs, stat]*, June 2018. URL http://arxiv.org/abs/1806.01830.

[195] M. Zhou, Z. Liu, P. Sui, Y. Li, and Y. Y. Chung. Learning Implicit Credit Assignment for Cooperative Multi-Agent Reinforcement Learning. *Advances in Neural Information Processing Systems*, 33:11853–11864, 2020. URL https://papers.nips.cc/paper/2020/hash/8977ecbb8cb82d77fb091c7a7f186163-Abstract.html.

[196] B. D. Ziebart, A. Maas, J. A. Bagnell, and A. K. Dey. Maximum Entropy Inverse Reinforcement Learning. In *Proceedings of the 23rd National Conference on Artificial Intelligence - Volume 3*, AAAI'08, pages 1433–1438. AAAI Press, 2008. ISBN 978-1-57735-368-3. event-place: Chicago, Illinois.

[197] H. Zou, H. Su, S. Song, and J. Zhu. Understanding Human Behaviors in Crowds by Imitating the Decision-Making Process. *arXiv:1801.08391 [cs]*, Jan. 2018. URL http://arxiv.org/abs/1801.08391. arXiv: 1801.08391.

DOCTORAL SCHOOL

Titre: Simulation de foules avec l'apprentissage par renforcement

Mots clés: simulation de foules, apprentissage par renforcement, apprentissage automatique, intelligence artificielle, apprentissage par renforcement multi-agents

Résumé: Simuler le comportement des foules constitue une composante clé de la création d'espaces numériques immersifs. Les méthodes traditionnelles, bien qu'efficaces, sont souvent limitées dans leur capacité à reproduire fidèlement la complexité du comportement humain. Récemment, l'apprentissage par renforcement (RL) a émergé comme une nouvelle approche pour surmonter ce défi. Cependant, de nombreux détails de la simulation des foules par RL peuvent sembler négligeables, mais s'avèrent avoir un impact majeur, incluant la simulation physique sous-jacente, les modèles d'observations et de dynamiques, et les détails de l'algorithme RL lui-même.

Cette thèse vise à mettre en lumière ces détails cruciaux et leurs effets sur les foules virtuelles formées par RL. Notre objectif est d'établir une compréhension des choix de conception pertinents qui permettraient la création de simulations de foules plus réalistes.

Dans la première partie, nous nous concentrons sur l'évaluation de l'impact des divers choix de conception sur la performance d'apprentissage et la qualité du comportement résultant. Nos expériences avec le Deep RL montrent que les contrôles non holonomiques avec une variante d'observations égocentriques produisent de meilleurs résultats par rapport aux autres alternatives plus simples.

Ensuite, nous examinons les détails de la conception de la fonction de récompense pour simuler des foules semblables aux humains. Nos expériences montrent qu'une minimisation directe de l'utilisation d'énergie, lorsqu'elle est couplée à un potentiel de guidage correctement calibré, permet de générer des comportements de foule plus efficaces.

Enfin, nous explorons le mécanisme d'escompte dans le RL. Nous présentons l'algorithme UGAE, une nouvelle solution qui permet l'utilisation d'algorithmes RL modernes avec un escompte arbitraire. Nous démontrons que UGAE surpasse la base de référence de Monte Carlo en utilisant à la fois des critères de référence RL standard et des scénarios de simulation de foule. Ceci ouvre la voie à de futures méthodes de simulation de foule utilisant un escompte non exponentiel.

Dans l'ensemble, cette recherche apporte des éclairages essentiels sur la dynamique des foules formées par RL, et contribue significativement au développement de nouvelles techniques et à l'amélioration des techniques existantes pour la simulation de foule.

Title: Simulating Crowds with Reinforcement Learning

Keywords: crowd simulation, reinforcement learning, machine learning, artificial intelligence, multiagent reinforcement learning

Abstract: Simulating crowd behavior is an important aspect of creating immersive digital environments, be it for video games or other virtual experiences. Traditional methods lead to satisfactory results but are often limited in their capacity to accurately emulate the complexity of human behavior. Recently, Reinforcement Learning (RL) has emerged as a new approach to tackle this problem. However, there are many details of RL-driven crowd simulation that may seem irrelevant, but turn out to be rather impactful. This includes the underlying physics simulation, models of observations and dynamics, and details of the RL algorithm optimizing the crowd's behavior.

This thesis aims to shed light on these critical details and their effects on virtual crowds trained with RL. Our overarching objective is to establish an understanding of relevant design choices, enabling the creation of more realistic crowd simulations.

In the first part of the thesis, we focus on evaluating how various design choices of the foundational crowd simulation impact both the learning performance and the overall quality of the resulting behavior. We present a classification of observation methods and dynamics, and evaluated their impact with DRL experiments. This shows that nonholonomic controls with a variant of egocentric observations produce better results compared to other, simpler alternatives.

Following this, we investigate the details of reward function design for simulating human-like crowds. We explore different reward functions, providing theoretical insights on their properties, and evaluate them empirically in different scenarios. Our experiments show that directly minimizing energy usage, when paired with a properly scaled guiding potential, are effective in producing more efficient crowd behaviors.

In the final part of the thesis, we explore the discounting mechanism in RL. We present the Universal Generalized Advantage Estimation (UGAE) algorithm, a novel solution that enables using modern RL algorithms with arbitrary discounting. We also introduce Beta-weighted discounting to parameterize non-exponential discounting methods. We demonstrate that UGAE outperforms the Monte Carlo baseline using both standard RL benchmarks and crowd simulation scenarios. This paves the way to future crowd simulation methods using non-exponential discounting, which may help overcome some of the challenges identified in our previous work.

This work, combined, provides critical insights into the dynamics of reinforcement learned crowds, and contributes significantly to the development of new and improved techniques for crowd simulation.

Institut Polytechnique de Paris
91120 Palaiseau, France