Towards Learning Human-Like and Efficient Multi-Agent Path Finding

Ariel Kwiatkowski, Vicky Kalogeiton, Julien Pettré, Marie-Paule Cani

Towards Learning Human-Like and Efficient Multi-Agent Path Finding

Ariel Kwiatkowski[1], **Vicky Kalogeiton**[1], **Julien Pettré**[2], **Marie-Paule Cani**[1]

[1]LIX, École Polytechnique/CNRS, Institut Polytechnique de Paris, Palaiseau, France
{ariel.kwiatkowski, vicky.kalogeiton, marie-paule.cani}@polytechnique.edu
[2]INRIA Rennes, France
julien.pettre@inria.fr

Abstract

Simulating trajectories of virtual crowds is a commonly encountered task in computer graphics. It significantly overlaps with the broader field of multiagent path finding, having the same central goal, but with different desired characteristics of motion. Several recent works have applied Reinforcement Learning methods to animate virtual crowds, however they often make quite different design choices when it comes to the fundamental simulation setup. Each of these choices comes with a reasonable justification for its use, so it is not obvious what is their real impact, and how they affect the results. In this work, we build upon our recent research (Kwiatkowski et al. 2022b) where we study the impact of these arbitrary design choices in terms of their impact on the learning performance, as well as the quality of the resulting motion. We extend it with a more in-depth analysis of the reward function, its structure and properties. We introduce a simple framework for modelling the reward function that enables studying its properties without performing a relatively costly RL training. We also show some of our findings on how certain specific reward functions succeed or fail at producing believable behavior in different scenarios.

1 Introduction

Multiagent navigation is a task that is common across many diverse applications. From warehouse robot path-finding, to autonomous driving and virtual crowd generation, it is necessary to generate trajectories moving agents between two (or more) points in a way that is collision-free and efficient. Each domain also has its unique characteristics – autonomous cars have a very specific set of movements available to them, and warehouse robots are similarly constrained by their hardware. In both of these cases it is also crucial that any collisions are eliminated, even at the cost of longer and less efficient trajectories, due to the high cost of collisions. On the other hand, simulating virtual crowds is much more forgiving – it is not restrained by the physical world, and the cost of individual collisions may be as trivial as slightly worse immersion in games or movies.

Whatever the domain, there is always a trade-off to be made. Even in the domains where collisions are very costly, they are not infinitely so. Similarly to human locomotion, both pedestrian and motorized, where collisions and car crashes still occur – some margin of error must be allowed. Otherwise, if a lack of collision is infinitely more important than any other goal, the only fully reliable strategy for all participants is not to move at all. A natural way to express the competing objectives is to give them numerical values, and each trajectory is then rated based on the sum of those values.

This formulation naturally yields itself to the Reinforcement Learning (RL) approach. At its core, RL is a study of sequential decision making, with the goal of optimizing a scalar objective over time. It does not require any additional structure like differentiability or having a known environment model, which makes it a natural option for learning to act in complex environments.

An RL task can be described with three main components: states, actions and rewards. When an agent takes an action, it observes the new state of the environment, and receives a reward indicating the quality of that action. In this work, we focus on this last component by investigating the properties of the reward function. A commonly used structure of a reward function is a weighted sum of several terms, and so we explore both the possible terms, as well as their weights.

In this work, we specifically focus on generating virtual crowds using multiagent reinforcement learning. On the one hand, this gives us a lot of flexibility on how to design and train the RL agents. Occasional collisions are not a big issue both during training and deployment, and their impact can be significantly mitigated by simple rendering tricks (e.g. rendering agents as smaller than their collision radius). On the other hand, each of these design choices may have subtle and difficult to predict consequences, and therefore we must be mindful about them.

An often overlooked component of the behavior of virtual crowds is the velocity at which the agents are moving. In other domains there are typically external considerations that set the possible and preferred velocities, like mechanical capabilities and speed limits on roads. Simulated agents, however, can move in arbitrary ways, which means that their motion must be deliberately designed via an appropriate action space, as well as a reward that incentivizes desired (often human-like) motion.

In this work, we aim to show what structures of the reward function will exhibit desirable qualitative and quantitative properties in the resulting RL agents. We use theoretical modelling of several simplified scenarios which can indicate

potential problems with some choices of reward functions, as well as empirical evaluation of agents trained with different variants, in terms of reward-independent metrics such as the number of collisions and energy usage.

This work overlaps with our recently published work on the wider design choices in RL-trained crowd simulations (Kwiatkowski et al. 2022b). We summarize the main findings in the following sections, and extend the parts related to the reward function design.

2 Related work

Microscopic simulation of virtual crowds has garnered significant research interest in recent years. Various non-learning techniques have been introduced in the previous decade (Toll and Pettré 2021), which typically involve designing rule-based systems to update each agent's velocity based on their context. There also exists a body of work using RL for controlling virtual characters, including crowd scenarios (Kwiatkowski et al. 2022a). In this section we describe the elements of related work which are the most relevant to applying RL in crowd simulation.

Reinforcement Learning. RL is a study of sequential decision making, where one or more agents act in an environment, affecting its state, and receiving rewards. Modern RL widely uses neural network-based algorithms (Sutton and Barto 2018), using them e.g. as a policy function, mapping observations to actions taken by the agent. This network is then optimized with a procedure based on the Policy Gradient Theorem, as introduced by the REINFORCE algorithm (Sutton et al. 1999). A more modern version that follows the same principle is the Proximal Policy Optimization (PPO) algorithm, introduced by Schulman et al. (2017). PPO is now the de facto standard on-policy algorithm used in many DRL applications due to its simplicity and efficiency.

Crowd Simulation. Microscopic simulation of crowds is typically done by either using **force-based** methods (Helbing and Molnár 1995) where the positions of nearby agents, obstacles, as well as the agent's destination, all contribute to Social Forces that drive the acceleration of the agent at the next time step, or alternatively **velocity-based** methods such as Optimal Reciprocal Collision Avoidance (ORCA) (van den Berg et al. 2011). The latter construct obstacles in the agent's velocity space, which correspond to velocities that would result in a collision if the agent were to take them. This leads to effective solutions to collision avoidance, able to capture anticipation behaviours in crossing scenarios. Recently, **vision-based** and **data-driven** algorithms were explored as well, promising even more human-like results (Toll and Pettré 2021).

Crowd Simulation via DRL. There is a number of prior papers which train DRL agents on the task of crowd simulation. Long et al. (2018) use a multiagent robotic navigation setup, which shares certain properties with crowd simulation. Lee, Won, and Lee (2018) train an RL agent on a variety of crowd scenarios, showing that a single trained model can be used to control multiple agents acting in a shared environment, on a diverse range of scenarios. Sun, Zhai, and Qin (2019) train groups of agents by making them follow specially-trained leader agents. Xu et al. (2020) combine DRL with an ORCA layer that ensures collision-free movement.

DRL is also used to generate more interesting, higher-quality trajectories. Xu and Karamouzas (2021) use real-world human trajectory data to train a supervised model judging the human-likeness of a generated trajectory. Then, the output of that model is used as an additional component of the reward function, encouraging agents to act in a human-like manner. Hu et al. (2022) use a parametric RL approach to generate heterogeneous behaviors with a single shared policy network. Each agent has its own preferred velocity value, and is trained to move according to it. Similarly, Panayiotou et al. (2022) vary the weights in the reward functions of different agents in order to give them unique and configurable personalities.

While each of these prior works tackles the same problem of crowd simulation, many elements of their fundamental setup differ in potentially significant ways, which makes direct comparison infeasible.

3 Environment Design Choices

We identify three crucial 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.

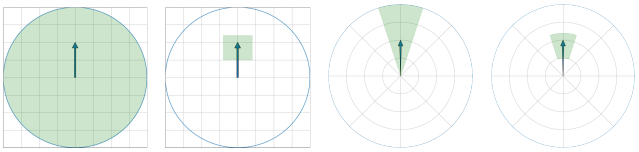
3.1 Problem Formulation

We model the problem of crowd simulation as a Partially Observable Stochastic Game (POSG) (Hansen, Bernstein, and Zilberstein 2004). 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 (note that we use the notation ΔX to mean the space of probability distributions over the set X).

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 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 format agents should receive this information, or in fact, what the information should be.



(a) Cartesian Velocity. (b) Cartesian Acceleration. (c) Polar Velocity. (d) Polar Acceleration.

Figure 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.

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). For the reference frame, we have three representations: Absolute, Relative, and Egocentric. The details of each of these methods are described in (Kwiatkowski et al. 2022b).

3.3 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 (Hughes, Ondřej, and Dingliana 2015), 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. (2008) 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.

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. Each of these choices is described in detail in (Kwiatkowski et al. 2022b), and visualized in Figure 1.

4 Reward-independent metrics

Before we move on to the reward function design, it is worth considering general metrics which quantitatively describe the motion of trained agents. Typically, when developing RL algorithms used in a fixed set of benchmarks, evaluation is conceptually simple – we measure the average reward obtained on each benchmark, potentially aggregating those performances into a single measure, and as long as it is statistically robust. This typically enough to prove the algorithm's performance. However, when discussing the design of the reward function, it is not that simple, as different reward functions are not directly comparable. Instead, it is useful to have one or more general metrics that quantitatively describe the agent's behavioral characteristics.

In this work, we use reward-independent metrics both for comparing the performance of agents trained using different reward functions, and as a inspiration for reward function design. We measure the following metrics, averaged across all agents:

1. Energy usage (in two variants)
2. Total trajectory length (distance)
3. Total trajectory duration (time)
4. Number of collisions
5. Success rate
6. Average speed
7. Average deviation from the preferred speed

When comparing reward functions with an identical structure, we additionally track the values of each term with the respective coefficient set to 1 (although some of them will often directly overlap with the metrics stated above).

4.1 Energy usage

Energy is a valuable metric, functioning as a measure of the overall movement efficiency. Following the hypothesis stated by Bruneau, Olivier, and Pettré (2015) that human tend to follow trajectories that minimize their energy expenditure, as well as the general intuition, we believe energy to be a good coarse estimate of the human-likeness of the agent's movement.

We focus on pedestrian locomotion, and for this reason we use an appropriate model of energy usage. The commonly used model is one known from gait analysis literature (Whittle 2008), and since then used in many RL-based crowd simulation systems (Whittle 2008; Guy et al. 2010; Godoy et al. 2020; Xu and Karamouzas 2021; Kwiatkowski et al. 2022b). It estimates the energy usage per unit time, per unit mass as:

$$E/\Delta t = e_s + e_w\ v^2 \tag{1}$$

where v is the speed of the agent, e_s and e_w are parameters specific to each person, whose values for a "standard human" are $e_s = 2.23$ and $e_w = 1.26$. Note that for simplicity, we assume a fixed mass and thus omit the agent's mass m in the following derivations. To get the real numerical values, it is enough to scale the results by the desired mass in kilograms.

This expression is deceptively similar to the standard equation for kinetic energy $T = \frac{1}{2}mv^2$, but its interpretation is not that simple. Kinetic energy is something that an agent *has* during motion, whereas Equation 1 refers to the energy that it *uses* over time.

In fact, this expression seems to entirely disregard the kinetic energy that an agent gains while accelerating, since it depends only on the momentary velocity, ignoring its changes. Instead, the energy usage equation can be easily obtained with a simple model of motion with linear damping, which uses some constant energy for general body maintenance, and some energy for overcoming that damping. Consider an agent moving in a straight line, at a constant velocity $\vec{v}$. It is slowed down by a damping force opposite to its motion $\vec{F} = -\frac{e_w}{\Delta t}\vec{v}$, so to maintain a constant velocity, the agent has to exert a force equal in magnitude. From Newtonian mechanics we know that work is the product of the force and the distance across which the force was applied, which means that the work over time is:

$$W/t = \frac{F\ s}{t} = F\ v = e_w\ v^2 \tag{2}$$

Combined with a constant energy usage of e_s, this leads to the expression in Equation 1. It notably lacks any dependency on the acceleration, or on the actual kinetic energy of the agent.

We propose to extend this model based on physical principles. When an agent accelerates, it uses energy to maintain its metabolism and to overcome the damping, the same way as in Equation 1. But it may also apply an additional force so that its velocity increases over time. To estimate the total energy usage, we analyze the acceleration resulting from applying a certain force in any direction relative to the current velocity, leading to the more general expression:

$$E/\Delta t = e_s + |\vec{v}\cdot\vec{a} + e_w\vec{v}_0\cdot\vec{v}| \tag{3}$$

where $\vec{v}_0$ is the velocity from the previous timestep and $\vec{a} = (\vec{v} - \vec{v}_0)/\Delta t$.

It is easy to see that this model is consistent with Equation 1 when $\vec{v}_0 = \vec{v}$. When the agent moves with a positive acceleration in the direction of the movement, it takes the following simple form:

$$E/\Delta t = e_s + e_w\ v^2 + av \tag{4}$$

During deceleration, the model is more nuanced. There is a velocity threshold $v_p := (1 - e_w\Delta t)v_0$ corresponding to the natural damping. As long as the deceleration maintains the velocity above v_p, the energy usage decreases, since the agent can simply putting in effort to overcome the damping (**passive deceleration**). However, if the agent decides to stop more abruptly, it must once again use additional energy (**active deceleration**).

As is clear by looking at the equations, the practical difference between the two energy formulations (Equation 1, and Equation 3) appears when agents accelerate or decelerate during their trajectories. Moving at a constant speed will result in the exact same energy usage under the two models, but if an agent instead decides to speed up and slow down throughout its trajectory, the new model will indicate a higher energy usage.

Note that the physical principles underlying our reasoning are all based on continuous time, whereas computer-based systems operate in discrete time. For simplicity, we used above a simple discretization model where the energy cost is computed based on the velocity in the "current" and previous timestep, instead of performing interpolation between it and the previous timestep. A more precise estimate may be obtained by averaging the velocity between two timesteps, or using the equivalent of the trapezoidal rule in integration.

It is also important to state here that this model is still a vast oversimplification of human locomotion. In reality, we do not move by applying abstract forces to our center of mass – we move our feet and let gravity propel us forward. That being said, we believe that this formulation is more accurate than the simple model which only considers constant linear motion, and can be beneficial for computational applications like training RL agents.

5 Reward functions

We base our analysis on the reward structure that we introduced in Kwiatkowski et al. (2022b), which encompasses several reward terms already used in the literature on RL-based crowd animation. It includes the following components:

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

where c_g, c_p, c_v, c_e, c_c, c_t are arbitrary (typically positive) coefficients, v_0 is the preferred speed, and d_t is the distance from the goal at timestep t. Additionally, we consider the following alternative terms:

1. Alternative for the comfortable speed reward used by (Xu and Karamouzas 2021), $R_v = -c_v \exp(c_s||\mathbf{v} - \mathbf{v}_0||)$, where $\mathbf{v}$ is the current velocity vector, and $\mathbf{v}_0$ is a vector pointing from the agent to its goal, with a magnitude equal to v_0. This term can be used to simultaneously replace R_v and R_t since it never reaches 0 due to the exponential function.
2. A modification of the comfortable speed reward which only penalizes velocities exceeding the preferred value $R_v = -c_v\ \delta_{v>v_0}|v - v_0|^{c_e}$
3. An additional term adding a penalty for the total distance travelled $R_d = -c_d|v|$

In this section, we describe our theoretical framework for analyzing the properties of various reward structures and their coefficients. We describe the modelling approach, as well as the test behaviors which we model. The findings based on this approach are in Section 7.2.

5.1 Reward modelling

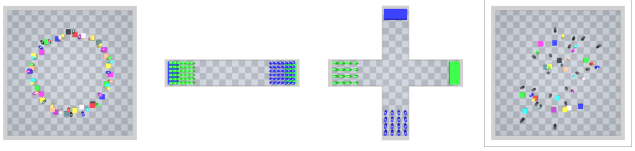
In order to investigate the properties of a given reward function, it is useful to have a simplified model of the decision process, one that can be quickly evaluated on a variety of different scenarios. Without this, every reward function would have to be evaluated by training a new agent, or at least evaluating a fixed policy on the full environment, which is likely to be computationally expensive when repeated many times. For this reason, in this work we use two simple models of the decision process, continuous and discrete, that enable faster iteration and deeper understanding of the properties of a given reward function.

Continuous reward model While in the RL setting, rewards are assigned on a step-by-step basis with discrete time, the components of the reward function largely correspond to continuous phenomena, which significantly simplifies investigating some of their properties analytically.

Although it is convenient in certain cases, using continuous models has two significant disadvantages. The first is the fact that it does not exactly match the process we are modelling. While the continuity of the physical reality is a subject of debate, our simulations have to work with discrete time. The second is the fact that to obtain valuable insights with this model, we have to be able to actually solve the integrals, which are often famously intractable or highly complex. For this reason, it is valuable to also have a less elegant, but more flexible alternative.

Discrete reward model The discrete time model is much closer to the reality of RL training. We emulate the decision process step-by-step, and assign rewards according to the known reward function, factoring in any implementation details that are relevant in the RL setting.

This model has the disadvantage of generally not being tractable analytically, since evaluating arbitrary finite series tends to be more difficult than integrals. However, it is very easy to evaluate them numerically by simply enumerating all the timesteps in an episode, leading to a more flexible modelling framework.



(a) Circle scenario. (b) Corridor scenario. (c) Crossing scenario. (d) Random scenario.

Figure 2: 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.

5.2 Test behaviors

We use both continuous and discrete models on a set of simple, yet informative behaviors designed to probe specific aspects of a given reward function. We describe three of them in this section, but it is likely that more will be added in the course of this work.

Constant linear motion The simplest test behavior we consider is an agent moving in a straight line, with a constant velocity. This provides a simple check whether the reward function (especially combined with a temporal discounting) properly incentivizes moving at the preferred velocity, as opposed to e.g. standing still or moving at full speed.

Moving at a slower effective speed Even though the agents should generally move at their preferred speed, sometimes external factors (such as avoiding collisions) lead to a situation where they want to move at a lower speed. In this scenario we consider two strategies – either actually slowing down, or artificially lengthening the trajectory (e.g. by moving in a loop) to keep moving at the preferred speed. In pedestrian motion, the former is generally preferred, although the latter may lead to behavior resembling a bicycle.

Not reaching the goal When the goal is relatively far away compared to the maximum time allotted to the agent it is useful to check what the reward or energy usage is with relatively efficient navigation, as compared to the agent simply standing still. This is to ensure that the implementation details of the RL setting (such as the time limit) do not adversely affect the global properties of the reward function.

6 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 2). 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

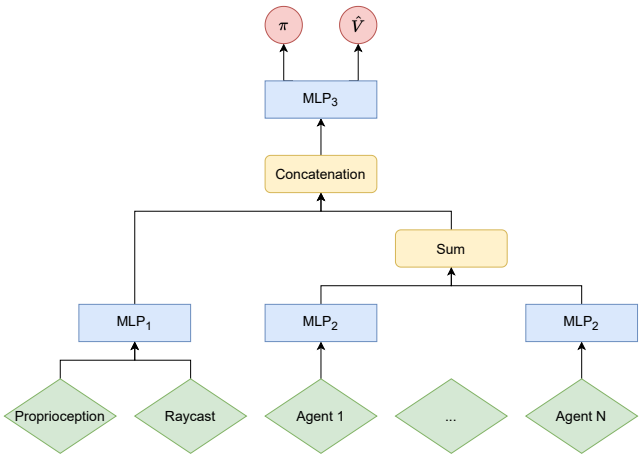


Figure 3: The neural architecture used as the policy. Green blocks represent inputs, blue blocks represent feed-forward neural networks, yellow blocks represent vector operations, red blocks represent outputs. Depending on the observation model used, certain elements of the architecture are disabled.

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, with each agent being represented as a circle with a 0.4 meter radius. In both Circle and Random, there are optional small obstacles placed randomly in the scene, represented as immovable agents.

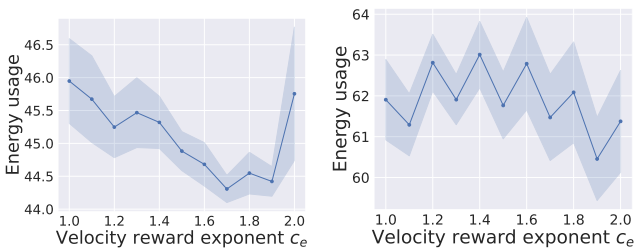
6.1 Policy Optimization

In this work, we train RL agents using PPO with Generalized Advantage Estimation (GAE) (Schulman et al. 2018) to estimate the advantages. The agents are trained in an independent paradigm with parameter sharing, i.e. they share the same policy network, but each agent takes a decision on 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.

6.2 Network Architecture

In order to appropriately process the Agent Perception observations, we use a neural architecture depicted in Figure 3. It is inspired by prior work such as Deep Sets (Zaheer et al. 2017) and Mean Embedding (Hüttenrauch, Sosic, and Neumann 2019), and extends the architecture used by Xu and Karamouzas (2021).

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



(a) Circle 30 scenario (b) Crossway 50 scenario

Figure 4: Comparison of energy usage in agents trained with a different exponent in the velocity term of the reward function. Lower is better.

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 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.

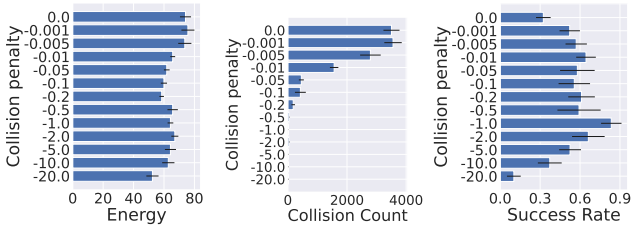
In our current work, we are also evaluating a similar architecture where the embeddings are instead processed by a cross attention module inspired by the Transformer model (Vaswani et al. 2017). It can potentially improve the performance in crowded scenes, where the irrelevant agents serve as noise impeding the navigation process.

7 Results

In this section, we describe the two types of experimental results obtained for this work. First, we briefly summarize the conclusions of Kwiatkowski et al. (2022b), focusing on the empirical findings relevant to the rewards. Then, we discuss the results of applying our modelling methodology detailed in Section 5.1, and the insights they lead to regarding the reward function design.

7.1 Experimental results

The main goal of the empirical experiments was establishing the performance implications of different dynamics and action models, as measured by the total reward and the energy consumption. The overall conclusion however is that there are no strong regularities. The best observation and action spaces depend heavily on the specific environment, but as



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

Figure 5: 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.

a general rule, relative or egocentric observations, together with non-holonomic actions work better than global observations and holonomic actions. The random scenario is an exception due to its specific structure where the goal can be in any location relative to the agent, whereas in all other scenarios, the goal is generally in a "forward" direction for each agent. There is, however, a consistent trend of agents using Direct Agent Perception outperforming their raycasting counterparts.

Additionally, we have two sets of experiments specifically probing the properties of the reward function. In Figure 4 we train agents with different values of the c_e coefficient described previously. In both scenarios, there is an optimum around $c_e = 1.9$ where training to optimize that reward results in more energy-efficient motion. In Figure 5 we train agents with a varying collision penalty R_c. We can see that if the penalty is too low or too high, the agents fail to consistently navigate to their goals – either by getting stuck in the middle, or by refusing to move to avoid any risk of collisions. It is also worth noticing the fact that the global minimum of energy usage in this graph corresponds to the agents not moving at all, which is an issue further investigated in the following section.

7.2 Simulation results

Energy as reward A very tempting possibility at this point may be using energy optimization directly as the reward. After all, if we believe that it is a good metric for the desired behavior of trained agents, maybe we can use the flexibility of the RL framework to optimize it directly?

The answer is – maybe. In principle it is definitely possible, seeing as we can compute the energy for all actions taken by the agent. However, using the (negative) energy as a reward function has certain properties that make it challenging to optimize directly, and may require some additional thought.

To understand this phenomenon, let us consider a continuous model of the constant linear motion scenario. If an agent moves in a straight line crossing a distance l with a fixed velocity v, then the total energy it uses under this model is:

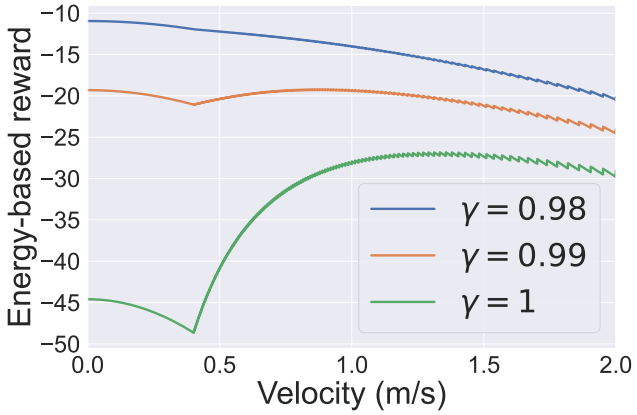


Figure 6: Discounted negative energy usage as a function of the velocity in the constant linear motion scenario, using a discrete model, with different discount factors.

$$E = \int_0^T (e_s + e_w\, v^2)dt = \int_0^{l/v} (e_s + e_w\, v^2)dt \tag{5}$$

$$= e_s \frac{l}{v} + l\, e_w\, v = l\, (e_s\, v^{-1} + e_w\, v) \tag{6}$$

Notice that when we increase the velocity, the body of the integral increases as well. At the same time, the length of the integral decreases. These two aspects are balanced against each other, with the optimal velocity:

$$v^* = \sqrt{\frac{e_s}{e_w}} \tag{7}$$

To relate this to the practical navigation task, consider an RL training from the beginning on a nontrivial scenario with multiple agents. This is common practice with a standard reward function and enables a simple training procedure with a stationary environment. Agents will typically learn by first approaching the goal, then actually reaching it in time, and then adjusting their trajectories to minimize collisions. When training with energy as the reward, the first stage is likely to fail, which means that the upper limit of the integral is some constant T_{max}. If the agent cannot reach its goal immediately, it will only learn that it is more energy efficient to stand still until the time limit runs out – a local optimum when evaluating the energy usage. The real gain from navigating to the goal at just the right speed comes from the balance between not spending too much time in motion, and not using too much energy per unit time. If the agent ends up not reaching the goal, then they receive the same negative reward as if they stood still, and more – because they used energy to move.

Furthermore, when considering energy as the reward, we must pay close attention to other seemingly irrelevant details, such as the discount factor and the timestep. Commonly used discounting turns out to be a complicating factor. In prior work, Naik et al. (2019) argue that the typical exponential discounting can change the optimal behavior that

the agent is trying to learn. This phenomenon is rather accepted in the field as something that is theoretically problematic, but practically inconsequential, so a vast majority of implementations only include exponential discounting.

In this case, using exponential discounting with an otherwise reasonable discount factor like $\gamma = 0.99$ in certain scenarios leads to a situation where standing still turns from being a tricky local optimum, to a global optimum. Consider a discrete model where the agent again chooses a velocity and commits to it throughout the entire trajectory. Let the total distance from the goal be $d = 8\,m$, the timestep $\Delta t = 0.1\,s$, and the maximum episode length (in timesteps) $T = 200$. In Figure 6 we show the energy-based discounted reward values with different discount factors, as a function of the velocity. The undiscounted curve has the correct global properties with a globally optimal velocity $v^* = 1.33\frac{m}{s}$. With $\gamma = 0.99$, this value changes to $v^* = 0.88\frac{m}{s}$, and with $\gamma = 0.98$ the globally optimal policy is for the agents to stay still, which is obviously undesirable. Regardless of of the discount factor, we also see the local optimum around $v = 0$ which corresponds to velocities which are too low for the agent to reach its destination before the time limit.

This analysis indicates that using energy as reward requires additional attention to the training setup. A promising approach for the local optimum issue might be a curriculum learning approach, where the agents are initially trained on small, empty scenarios, which are then progressively increased, and additional agents are added on. We could also train the agents with a different reward function so that they learn to successfully navigate to their goals, and then switch the reward to the energy. The issue with the global optimum might require a different discounting approach. Either using $\gamma = 1$ or using the average reward setting (Naik et al. 2019) can retain the correct global properties, but could lead to an increased variance in the rewards. Alternatively, using an appropriately shaped nonexponential discounting could keep the discounted rewards finite, while preserving the global properties. Finally, significantly increasing the maximum time limit can potentially work to solve this issue as well, but at a significant computational cost.

Moving at an effective slower speed Consider a continuous model of the basic reward structure. In this scenario, the agent wants to travel a distance d in a time T such that it needs to move slower than its preferred velocity $v_0 > \frac{d}{T}$. It can either move in a straight line at the velocity $v = \frac{d}{T}$, or artificially extend its trajectory (e.g. by looping, or by walking on a curved line). Considering just the value of the velocity-related reward $R_v = -c_v|v - v_0|^{c_e}$, it is simple to see that moving at exactly the preferred will incur no penalty, whereas an agent moving slower but in a straight line will have a nonzero penalty, meaning that the longer way will be preferable.

This can potentially be addressed via three different modifications. If the velocity-related reward factors in the direction towards the goal, then a curved trajectory will also lead to a penalty. Removing the penalty for moving too slowly will instead cause both trajectories to yield an equal reward, potentially causing agents to take the simpler option. Penalizing the total distance travelled also directly penalizes the longer trajectory, and may be effective depending on the coefficients. We aim to investigate these three modifications in terms of their similarity to the energy usage, as well as empirically, by measuring the total curvature of agent trajectories when trained with each variant. Alternatively, using the energy as a reward directly eliminates this issue, at the cost of the previously discussed complications.

Not reaching the goal Going back to Figure 5, we see that the seemingly poorly-performing agents with a steep collision penalty, end up using the least energy out of all the investigated variants. This is caused by the agents having to travel a relatively long distance compared to the maximum time allowed for a single episode. If the time required to reach the goal at the preferred velocity is close to the maximum episode duration, then staying still is indeed less energy consuming in the scope of an episode – the energy used for metabolism (e_s) is the same, and the energy used for movement (e_v) is lower without any movement.

This can be remedied in two ways. Extending the time limit may suffice, but depending on the implementation, it can proportionally increase the training time. Alternatively, we propose adding a penalty at the end of episode to each agent which has not reached its goal. Its magnitude should be at least such that standing still, never outperforms efficient navigation in terms of the total energy usage (counted together with said penalty).

8 Conclusions

In this work, we build upon our prior 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.

Our current work focuses on designing the reward function for RL training. We introduce a theoretical and numerical framework for modelling the navigation problem in a way that enables analyzing the properties of a reward function, without having to go through a costly training process. We use it to identify both present and potential flaws in the reward structure, and to validate potential improvements before implementing them in the full simulation.

This aspect of our work is still at an early stage. We will continue modelling various failure cases inspired by our experiments, train agents with new and improved reward functions, and repeat this process to obtain a solid understanding of what matters in designing an effective reward for human-like multiagent navigation. In the future, we consider using an Inverse RL approach to learn the parameters of the reward function from real data, or given enough data – learn the whole reward function.

References

Arechavaleta, G.; Laumond, J.-P.; Hicheur, H.; and Berthoz, A. 2008. On the nonholonomic nature of human locomotion. *Autonomous Robots*, 25(1-2): 25–35.

Bruneau, J.; Olivier, A.-H.; and Pettré, J. 2015. Going Through, Going Around: A Study on Individual Avoidance of Groups. *IEEE Transactions on Visualization and Computer Graphics*, 21(4): 9.

Godoy, J.; Guy, S. J.; Gini, M.; and Karamouzas, I. 2020. C-Nav: Distributed coordination in crowded multi-agent navigation. *Robotics and Autonomous Systems*, 133: 103631.

Guy, S. J.; Chhugani, J.; Curtis, S.; Dubey, P.; Lin, M.; and Manocha, D. 2010. PLEdestrians: A Least-Effort Approach to Crowd Simulation. *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, 10 pages. Artwork Size: 10 pages ISBN: 9783905674279 Publisher: The Eurographics Association.

Hansen, E. A.; Bernstein, D. S.; and Zilberstein, S. 2004. Dynamic programming for partially observable stochastic games. In *Proceedings of the 19th national conference on Artifical intelligence*, AAAI'04, 709–715. San Jose, California: AAAI Press. ISBN 978-0-262-51183-4.

Helbing, D.; and Molnár, P. 1995. Social force model for pedestrian dynamics. *Physical Review E*, 51(5): 4282–4286.

Hu, K.; Haworth, M. B.; Berseth, G.; Pavlovic, V.; Faloutsos, P.; and Kapadia, M. 2022. Heterogeneous Crowd Simulation using Parametric Reinforcement Learning. *IEEE Transactions on Visualization and Computer Graphics*, 1–1.

Hughes, R.; Ondřej, J.; and Dingliana, J. 2015. Holonomic collision avoidance for virtual crowds. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '14, 103–111. Goslar, DEU: Eurographics Association.

Hüttenrauch, M.; Sosic, A.; and Neumann, G. 2019. Deep Reinforcement Learning for Swarm Systems. *arXiv:1807.06613 [cs, stat]*. ArXiv: 1807.06613.

Kwiatkowski, A.; Alvarado, E.; Kalogeiton, V.; Liu, C. K.; Pettré, J.; van de Panne, M.; and Cani, M. 2022a. A Survey on Reinforcement Learning Methods in Character Animation. *Computer Graphics Forum*, 41(2): 613–639.

Kwiatkowski, A.; Kalogeiton, V.; Pettré, J.; and Cani, M.-P. 2022b. Understanding reinforcement learned crowds. ArXiv:2209.09344 [cs].

Lee, J.; Won, J.; and Lee, J. 2018. Crowd simulation by deep reinforcement learning. In *Proceedings of the 11th Annual International Conference on Motion, Interaction, and Games*, 1–7. Limassol Cyprus: ACM. ISBN 978-1-4503-6015-9.

Long, P.; Fan, T.; Liao, X.; Liu, W.; Zhang, H.; and Pan, J. 2018. Towards Optimally Decentralized Multi-Robot Collision Avoidance via Deep Reinforcement Learning. *arXiv:1709.10082 [cs]*. ArXiv: 1709.10082.

Naik, A.; Shariff, R.; Yasui, N.; Yao, H.; and Sutton, R. S. 2019. Discounted Reinforcement Learning Is Not an Optimization Problem. *arXiv:1910.02140 [cs]*. ArXiv: 1910.02140.

Panayiotou, A.; Kyriakou, T.; Lemonari, M.; Chrysanthou, Y.; and Charalambous, P. 2022. CCP: Configurable Crowd Profiles. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Proceedings*, 1–10. Vancouver BC Canada: ACM. ISBN 978-1-4503-9337-9.

Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.; and Abbeel, P. 2018. High-Dimensional Continuous Control Using Generalized Advantage Estimation. *arXiv:1506.02438 [cs]*.

Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; and Klimov, O. 2017. Proximal Policy Optimization Algorithms. *arXiv:1707.06347 [cs]*.

Sun, L.; Zhai, J.; and Qin, W. 2019. Crowd Navigation in an Unknown and Dynamic Environment Based on Deep Reinforcement Learning. *IEEE Access*, 7: 109544–109554. Conference Name: IEEE Access.

Sutton, R. S.; and Barto, A. G. 2018. *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book. ISBN 978-0-262-03924-6.

Sutton, R. S.; McAllester, D.; Singh, S.; and Mansour, Y. 1999. Policy gradient methods for reinforcement learning with function approximation. In *Proceedings of the 12th International Conference on Neural Information Processing Systems*, NIPS'99, 1057–1063. Cambridge, MA, USA: MIT Press.

Toll, W.; and Pettré, J. 2021. Algorithms for Microscopic Crowd Simulation: Advancements in the 2010s. *Computer Graphics Forum*, 40(2): 731–754.

van den Berg, J.; Guy, S. J.; Lin, M.; and Manocha, D. 2011. Reciprocal n-Body Collision Avoidance. In Siciliano, B.; Khatib, O.; Groen, F.; Pradalier, C.; Siegwart, R.; and Hirzinger, G., eds., *Robotics Research*, volume 70, 3–19. Berlin, Heidelberg: Springer Berlin Heidelberg. ISBN 978-3-642-19456-6 978-3-642-19457-3. Series Title: Springer Tracts in Advanced Robotics.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention Is All You Need. *arXiv:1706.03762 [cs]*.

Whittle, M. W. 2008. *Gait analysis: an introduction*. Edinburgh: Butterworth-Heinemann, Elsevier, 4th ed., reprinted edition. ISBN 978-0-7506-8883-3.

Xu, D.; Huang, X.; Li, Z.; and Li, X. 2020. Local motion simulation using deep reinforcement learning. *Transactions in GIS*, 24(3): 756–779. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/tgis.12620.

Xu, P.; and Karamouzas, I. 2021. Human-Inspired Multi-Agent Navigation using Knowledge Distillation. *arXiv:2103.10000 [cs]*. ArXiv: 2103.10000.

Zaheer, M.; Kottur, S.; Ravanbakhsh, S.; Poczos, B.; Salakhutdinov, R. R.; and Smola, A. J. 2017. Deep Sets. In *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc.