GREIL-Crowds: Crowd Simulation with Deep Reinforcement Learning and Examples

PANAYIOTIS CHARALAMBOUS, CYENS - Centre of Excellence, Cyprus
JULIEN PETTRÉ, Univ Rennes, Inria, CNRS, IRISA, France
VASSILIS VASSILIADES, CYENS - Centre of Excellence, Cyprus
YIORGOS CHRYSANTHOU, CYENS - Centre of Excellence, Cyprus
NURIA PELECHANO, Universitat Politecnica de Catalunya (UPC), Spain

Simulating crowds with realistic behaviors is a difficult but very important task for a variety of applications. Quantifying how a person balances between different conflicting criteria such as goal seeking, collision avoidance and moving within a group is not intuitive, especially if we consider that behaviors differ largely between people. Inspired by recent advances in Deep Reinforcement Learning, we propose Guided REinforcement Learning (GREIL) Crowds, a method that learns a model for pedestrian behaviors which is guided by reference crowd data. The model successfully captures behaviors such as goal seeking, being part of consistent groups without the need to define explicit relationships and wandering around seemingly without a specific purpose. Two fundamental concepts are important in achieving these results: (a) the per agent state representation and (b) the reward function. The agent state is a temporal representation of the situation around each agent. The reward function is based on the idea that people try to move in situations/states in which they feel comfortable in. Therefore, in order for agents to stay in a comfortable state space, we first obtain a distribution of states extracted from real crowd data; then we evaluate states based on how much of an outlier they are compared to such a distribution. We demonstrate that our system can capture and simulate many complex and subtle crowd interactions in varied scenarios. Additionally, the proposed method generalizes to unseen situations, generates consistent behaviors and does not suffer from the limitations of other data-driven and reinforcement learning approaches.

CCS Concepts: • **Computing methodologies** → **Sequential decision making**; **Real-time simulation**; **Reinforcement learning**; **Animation**.

Additional Key Words and Phrases: crowd simulation, data-driven methods, user control, crowd authoring, reinforcement learning.

ACM Reference Format:
Panayiotis Charalambous, Julien Pettré, Vassilis Vassiliades, Yiorgos Chrysanthou, and Nuria Pelechano. 2023. GREIL-Crowds: Crowd Simulation with Deep Reinforcement Learning and Examples. *ACM Trans. Graph.* 42, 4, Article 1 (August 2023), 15 pages. https://doi.org/10.1145/3592459

Authors' addresses: Panayiotis Charalambous, CYENS - Centre of Excellence, Dimarchou Lellou Demetriadi 23, Nicosia, 1016, Cyprus, p.charalambous@cyens.org.cy; Julien Pettré, Univ Rennes, Inria, CNRS, IRISA, , Rennes, France, julien.pettre@inria.fr; Vassilis Vassiliades, CYENS - Centre of Excellence, Dimarchou Lellou Demetriadi 23, Nicosia, 1016, Cyprus, v.vassiliades@cyens.org.cy; Yiorgos Chrysanthou, CYENS - Centre of Excellence, Dimarchou Lellou Demetriadi 23, Nicosia, 1016, Cyprus, y.chrysanthou@cyens.org.cy; Nuria Pelechano, Universitat Politecnica de Catalunya (UPC), Barcelona, Spain, npelechano@cs.upc.edu.

0730-0301/2023/8-ART1 $15.00
https://doi.org/10.1145/3592459

1 INTRODUCTION

Crowds are an important part of our daily lives; we interact with them in streets, workplaces, shopping malls, football stadiums or concerts. Therefore, one important element in the compositing of lively and believable virtual scenes in applications such as games and movies is the presence of realistic ambient crowds. The dynamics of crowd motion greatly affect the ambiance, and are thus a crucial element of computer generated environments used in computer games, movies, training and architectural visualizations. As these applications continue to strive towards higher levels of realism and scene complexity, there is an increasing need for more believable crowd simulations.

Research in microscopic crowd modelling has focused on finding the main principles by which a person moves within a crowd, and on progressively removing simulation artefacts, such as deadlocks or jerky trajectories leading to easily noticeable animation imperfections. A side effect of their success, is that most recent simulation algorithms tend to generate an overly fluid look and feel with characters exhibiting perfect collision avoidance and goal seeking. However, real people in crowds exhibit more complex trajectories. For example, in streets, people move in groups, they may suddenly change direction, show hesitation, or generate accidental bumps. Simulation algorithms clearly loose these details which do not result from collision avoidance or goal-reaching tasks.

It would be an endless goal to model and simulate all these little causes for recovering trajectory details. One solution is to follow a data-driven approach. Data-driven techniques can capture the variety and detail of human trajectories without the need to explain their cause, by looking at a data-base of real-world examples. Different types of crowds can be simulated by simply using different sets of input examples. However, existing solutions show a number of limitations: they can only simulate situations that are found in the original data; they need a considerable amount of training input causing a heavy overhead in some cases; and since they do not take into account history and future consequences the overall trajectories tend to exhibit inconsistencies. Importantly, differences between simulated and original data are bound to happen; these lead to errors in prediction. In an environment with multiple agents, these errors accumulate over time resulting in trajectories with large deviations from the intended.

In this article we propose a novel data-driven crowd approach based on a well known Deep Reinforcement Algorithm (Double Deep Q-learning) [van Hasselt et al. 2016] that addresses several of the previously mentioned issues. It leverages the ability of reinforcement learning to learn good policies for sequential decision

problems, by optimizing a cumulative reward function, in our case for learning more consistent trajectories. This is done in a two-stage framework; first a novelty detection algorithm is used to learn a reward function from input crowd, which is then used during training to find an optimal policy for individual agents in the crowd. During the learning process, new example data are generated that enrich the policy and enable it to handle situations not present in the original data, something lacking in previous data-driven methods that rely on supervision. The main contributions are:

- A complete learning framework that can reproduce plausible and consistent crowd behaviours from a relatively small set of input crowd data.
- A data-driven novelty detection based reward function that takes into account the plausibility of a given state.

In Section 2 we give an overview of previous work, we then give an overview of the GREIL framework (Section 3), followed by a detailed description of the methods and learning approach (Section 4), we demonstrate results (Section 5) and finish with some discussion about the method, limitations and future work (Section 6).

2 RELATED WORK

There are many crowd simulation techniques that can broadly be categorized as macroscopic or microscopic. In the macroscopic approach, crowds are modelled as an active matter, for which variation of density in space and time is computed, with no distinction of individuals. They generally fail to simulate variety in motion and behaviours. In this work we focus on microscopic approaches - and more specifically data-driven methods which are the most relevant to our work. We refer the reader to [Pelechano et al. 2016] for a more comprehensive discussion on crowd simulation techniques.

Data-Driven Crowds. Recently, data-driven crowd simulation methods have emerged as an attractive alternative to manually defining the crowd simulation model. The promise in these approaches is that agents will "learn" how to behave from real-world examples, keeping the natural crowd ambiance with a wide range of complex individual behaviours without the effort of defining an explicit behavioural model. One of the earliest data-driven techniques for groups of characters employed a motion graph approach for synthesizing group behaviour [Lai et al. 2005]. In order to be able to build a tractable motion graph, this method makes the assumption that the input follows a well-defined behaviour model, such as a flocking system with a restricted configuration space. Graph based simulation was also employed by Kwon et al. [Kwon et al. 2008] for guiding a single group of agents to navigate together. These methods are impractical for general human pedestrian crowds due to the high degree of behaviour variation.

In several data-driven methods, trajectories extracted from videos of crowds are stored in databases alongside some representation of the local state as an indication of (state, action) response [Lee et al. 2007; Lerner et al. 2007, 2010; Zhao et al. 2017]. During simulation, agents match simulated states to the ones stored in the database and act accordingly. Charalambous and Chrysanthou [Charalambous and Chrysanthou 2014] introduced the Temporal Perception Pattern (TPP) to represent agent state and the Perception-Action-Graph (PAG) to do simulation and improve upon the performance of data-driven crowd simulations. Ju et al. [Ju et al. 2010] take a different approach; input data that represent different styles of crowds are blended together to generate new crowd animations. Metoyer and Hodgins [Metoyer and Hodgins 2003] allow the user to define specific examples of behaviours, while Musse et al. [Musse et al. 2006] extract paths from a video for a specific environment. All of these methods are in essence supervised learning approaches, and therefore they are highly dependent on the amount and variability of the training data; i.e., they do not generalize well to unseen states. Even small differences between observed and input data states give errors in agent reactions; these errors accumulate over time resulting in undesirable behaviours. Additionally, most of these approaches are greedy; agents react to the current state observation without taking into account long term consequences of their actions.

As an alternative to storing example behaviours in databases, some techniques use observations of real people to learn parameter values for simulators. Several works for example estimate collision avoidance and anticipation parameters by examining motion capture data in a controlled environment and propose prediction based approaches for crowd steering [Paris et al. 2007; Pettré et al. 2009; van Basten et al. 2009]. Moussaïd et al. [Moussaïd et al. 2010] used data from videos of real crowds to modify Helbing's social forces model [Helbing and Molnár 1995] to handle group formations in a more realistic way. In the work of Courty and Corpetti [Courty and Corpetti 2007] a macroscopic approach is followed; the crowd is seen as a continuous flow and the captured data is used to define the guiding vector field. Li et al. [Li et al. 2012] follow a procedural approach where periodic motion patches of crowds are concatenated together to create large crowd animations. Looking a bit further away, biology researchers proposed using input from stereoscopic videos of Starling birds to estimate a statistical model of their massive and complex flocking behaviour [Hildenbrandt et al. 2010]. In all of these techniques, the examples are used to refine an underlying behaviour model therefore they are still bound by the limitations of the underlying model.

A recent class of systems predict *trajectories* for each individual agent in a simulation instead of instantaneous actions like velocity or acceleration. Early works in this area retrieve sub-trajectories from databases of real-world data[Lerner et al. 2007]. More recent work take advantage of Deep Learning techniques such as LSTMs[Alahi et al. 2016] or GANs[Amirian et al. 2019; Gupta et al. 2018] that were trained on real world trajectories to generate local trajectory responses of agents. Other methods use variational autoencoders [Mangalam et al. 2021; Salzmann et al. 2020]. Contrary to these methods we do not generate local trajectories and we do not utilize generative networks to generate the actions of the agents.

Several researchers proposed data-driven methods to analyse simulated crowds. Some of these methods rank the capability of simulators to capture some behaviour by comparing simulated states to the ones found in reference data [Charalambous et al. 2014; Guy et al. 2012; Kapadia et al. 2011; Wang et al. 2017]. Wolinski et al. [Wolinski et al. 2014] propose a genetic algorithm approach to find the set of optimal parameters that a simulator must use to get desired crowd behaviour based on some metrics; this in turn allows for fairer comparison between simulation systems. Getting inspiration from

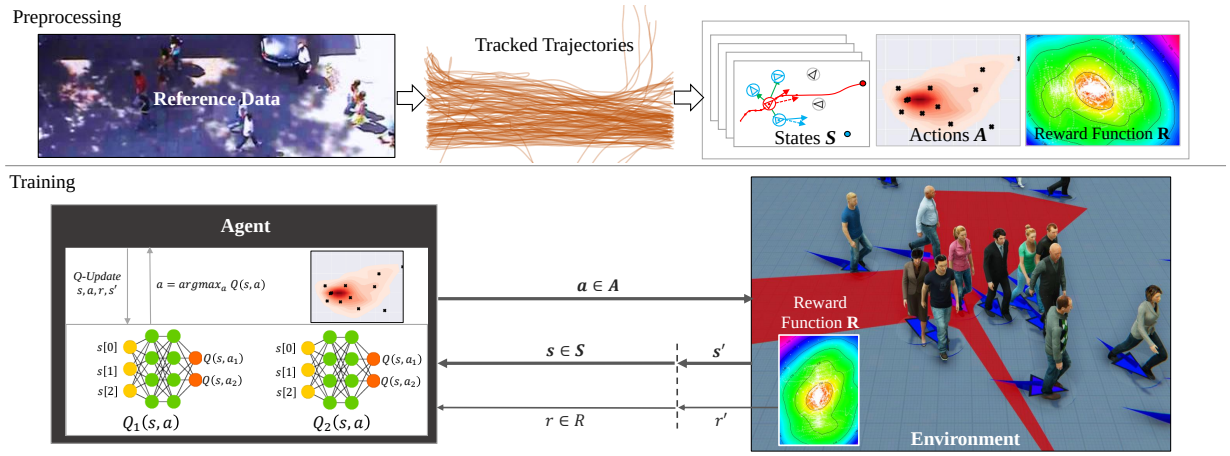


Fig. 1. *Overview of GREIL-Crowds.* (top) At the *pre-processing* stage, spatio-temporal trajectories of real world crowds are extracted from videos. These are then processed to find agent centric states and actions, and to define a data-driven reward function $R(s, a, s')$. (bottom) During *training*, $R(s, a, s')$ is used by the Double Deep Q-Learning algorithm to find the optimal policy function for the agents. A replay buffer memory combined with a data-driven training strategy are crucial elements in achieving efficient policies.

these approaches, we define a novel reward function that evaluates simulated behaviour as compared to reference data using novelty detection [Charalambous et al. 2014]; we therefore integrate crowd evaluation in the training loop.

Reinforcement Learning (RL) (or Optimal Control). RL is a learning framework that is used to find optimal strategies in sequential decision-making problems [Sutton and Barto 2018]. Mnih et al. [Mnih et al. 2015] introduced the Deep Q-Learning (DQN) framework that allowed agents to learn and play Atari games at human level. Later, van Hasselt et al. [van Hasselt et al. 2016] improved learning and stability of that system by introducing Double Deep Q-Learning (DDQ). The work on DQN inspired many system designs to handle complex control problems; of particular relevance to this paper are works from the computer animation community [Liu and Hodgins 2017; Peng et al. 2018, 2017; Treuille et al. 2007]. Treuille et al. [Treuille et al. 2007] and Peng et al. [Peng et al. 2017] for example demonstrate characters that navigate environments and avoid collisions with moving obstacles. Several authors proposed RL approaches to learn crowd simulation policies [Godoy et al. 2015; Henry et al. 2010; Lee et al. 2018; Martinez-Gil et al. 2011]; Long et al. [Long et al. 2018] for example introduced a multi-agent deep reinforcement learning approach to learn optimal policies for robot navigation. Many of these RL systems for crowds make simplifying assumptions or use very simple manually defined reward functions that aim to capture specific aspects of crowds such as collision avoidance and goal reaching [Lee et al. 2018; Long et al. 2018]; this results in agents with highly efficient movement that do not resemble actual crowds. Recently, Hu et al. [Hu et al. 2021] introduced the use of control parameters such as preferred speed as input to the policy to generate heterogeneous behaviours for crowd simulations. Panayiotou et al. [Panayiotou et al. 2022] introduced Configurable Crowd Profiles (CCP), an RL based method where the weights of different components of the reward function are added in the inputs to the policy; this allows the learning of a single policy for heterogeneous agent behaviours that mix several basic behaviours. Additionally, this approach allows for run-time control of agent parameters without the need of retraining the learned models. Our aim is to capture many behaviours that are present in the reference crowd data and therefore define a reward function that is based on actual crowd data. We get inspiration for the design of our Deep Q-Learning framework from the works by Mnih et al. [Mnih et al. 2015] and van Hasselt et al. [van Hasselt et al. 2016].

The proposed system is different in many ways to other data-driven methods. Firstly, it has better generalization properties, does not suffer from the problems of supervised learning agents and is less bound by the amount of data. Secondly, an optimal policy for sequential decision-making is found and therefore agents take into account consequences of actions that they took many learning steps before whereas most of the other approaches are reactive or optimize at a very short time horizon. Additionally, our approach has high run-time performance and is memory efficient since during simulation time only the trained network is needed; forward passes through the network return actions and training data can be discarded. Performance is improved significantly since we also utilize a large timestep between decisions. Moreover, we use novelty detection on actual crowd data as the basis of our reward function; agents tend to move in areas of the state space that are more beneficial for them. Finally, the proposed system captures many behaviours that are present in actual crowds which make them feel more natural such as social interactions, collision avoidance, sudden changes in direction, slight bumping, etc.; in contrast to other systems which optimize a particular aspect of the crowd.

3 SYSTEM OVERVIEW

The goal of this work is to create control policies able to simulate realistic looking ambient crowds using as reference real-world crowds

for applications in domains such as video games, movies and architectural visualizations. Our system is split in two distinct phases: *pre-processing* where a data-driven reward function is computed and *training* where the optimal policy is estimated (Fig. 1). In order to achieve good crowd simulation performance, numerous issues had to be considered. The biggest issue is defining a proper reward function; in environments like the Atari games [Mnih et al. 2015] for example, a reward can easily be determined by the game's score. However, crowd simulation is a multiagent interaction problem by nature; this implies that a proper reward function should consider multiple criteria such as satisfying desired velocities, avoiding collisions, and moving with other agents in groups (e.g, families, tourists, etc.).

Pre-processing. Given some reference crowd data such as tracked data of people from a street or a controlled environment [Ju et al. 2010; Lerner et al. 2007] (Section 4.1) we extract the state/action spaces and the reward function. States include both intrinsic and extrinsic features that could affect a person's behaviour such as the agents current velocity, desired velocity and positions/velocities of neighbouring agents (Section 4.2). The action space consists of all possible *acceleration vectors* in the local coordinate system of agents; these accelerations are applied continuously between decision-making steps (Section 4.3). We construct a scalar reward function that is based on novelty detection; this function evaluates a state based on how much of an inlier it is as compared to the reference data (Section 4.4). The key intuition behind the selection of such a reward function is that when people move, they tend to stay in parts of the state space that they feel comfortable in, giving preference to more common states than abnormal ones; however, abnormal states are also acceptable with a low reward since these are normal behaviours that are not visited often, but are present in the example data.

Training. During training, the DDQ algorithm [van Hasselt et al. 2016] is adapted to find an optimal policy π^* that maps an individual agent's state to an action (see Section 4.6). The policy function is a 4-layer fully connected neural network (a *Q-Network*) that computes the cumulative reward the agent is expected to receive at any given state (Section 4.5) for all possible actions; during simulation an agent simply selects the action with the highest expected reward. The DDQ algorithm trains two such networks to stabilize training and allow for agents to learn more complex tasks. Our novel learning strategy involves the simulation of agents in an environment consisting of the original tracked crowd data; this generates experiences (states) that are close to the original data (Section 4.6). We additionally allow for exploration to improve the quality of the policy.

The following section provides a detailed description of our methodology and how we approached these issues.

4 LEARNING AGENT BASED CROWD BEHAVIOR

In the Reinforcement Learning setting, an agent interacts with an environment over a sequence of episodes trying to maximize a cumulative reward. We formulate the navigation task of an individual agent through a crowd as a Markov Decision Process (MDP) which is represented by the tuple $(S, A, R, T, \gamma, \rho)$ where S is the continuous state space, A is a *discrete* set of actions, R is a scalar reward function, T is the transition model, $\gamma \in [0, 1]$ is a discount factor and ρ is the initial state distribution.

A state $s \in S$ in our agent based simulation system consists of intrinsic and extrinsic factors that influence the navigation behavior of an agent; e.g., the agent's velocity and goal and neighborhood formation (Section 4.2). An action $a \in A$ is a $2D$ *acceleration vector* with respect to the local coordinate system of an agent; instead of having a continuous action space we sample the distribution of acceleration values from some reference crowd data to get a discrete set of actions (Section 4.3). The transition function $T : (s, a) \mapsto s'$ describes the probability we transition to state $s' \in S$ given that we executed action $a \in A$ in state $s \in S$. A reward function $R : S \times A \times S \mapsto \mathbb{R}$ (for convenience $r := R(s, a, s')$) evaluates the transition (s, a, s') given the agent task; in our case the task is to *behave similarly to reference crowd data* (Section 4.4).

A policy $\pi : S \rightarrow A$ defines a direct mapping from continuous states to discrete actions. We note that in the more general case, continuous state and action spaces could be used. This would require other approaches to solve the problem such as policy gradient or Actor-Critic methods. Since we employ Q-Learning and our action space is low dimensional ($|A| = 2$), we do a discrete approximation of A. An agent applies actions from the policy π continuously; these leads to sequences of (s_i, a_i, r_i, s_{i+1}) tuples where $a_t = \pi(s_t)$, $r_t = R(s_t, a_t, s_{t+1})$ and $s_{t+1} = T(s_t, a_t)$. Any given state $s \in S$ can be evaluated using the value function $V_\pi(s) = \sum_{t=0}^{\infty} \gamma^t r_t$ over the entire trajectory seen after this state. The discount factor γ gives less weight to future states, keeps the value function bounded and implicitly defines a finite time horizon. For the experiments shown in this work we set $\gamma = .9$. The goal of Reinforcement Learning is to find the optimal policy π^*. Q-Learning is one of the most popular model-free approaches to find π^* [Sutton and Barto 2018; Watkins and Dayan 1992]. The goal in this formulation is to learn a Q-value function $Q^\pi(s, a) = r + \gamma V^\pi(s')$ that gives the expected reward assuming an agent takes action $a = \pi(s)$. The optimal Q-value function $Q^*(s, a)$ satisfies the Bellman equation:

$$Q^*(s, a) = r + \gamma max_{a'} Q^*(s', a'). \tag{1}$$

Having calculated $Q^*(s, a)$, the optimal policy at any given state s is:

$$\pi^*(s) = argmax_a Q^*(s, a). \tag{2}$$

Since the state space S is continuous, the Q-value function is approximated by a function approximator; typically a neural network (i.e., *Q-Network*). The goal of Q-Learning therefore is to learn the optimal parameters θ (weights and biases) of the neural network $\tilde{Q}(s, a; \theta) \approx Q(s, a)$ and the approximate optimal policy is $\pi^*(s) = argmax_a \tilde{Q}^*(s, a; \theta)$.

In the following paragraphs, we give more detailed descriptions of states, actions our data-driven reward function, a description of the policy network and our modifications to the DDQ algorithm [van Hasselt et al. 2016].

4.1 Crowd Data

We consider a person's trajectory in a video of real world crowds as a near optimal *expert demonstration* of an individual's behaviour in a crowd. These tracked data consist of a set $\{T_i\}$ of spatio-temporal trajectories of individual people $T_i = \{\mathbf{p}_{t,i}\}$; each $2D$ point $\mathbf{p}_{t,i}$ is a projection of the Center of Mass (CoM) of each individual on the ground. We can then pre-process $\{T_i\}$ to extract sequences of agent-centric states and actions (Sections 4.2 and 4.3) to generate the set $\mathcal{D} : \{\tau_i\} \sim \pi^*$ where $\tau_i = \{s_1, a_1, \ldots, s_t, a_t, \ldots, s_T\}$; these data can be considered as sample demonstrations of the optimal policy π^* of the agents in the crowd. These data are then used to extract state and action distributions, a reward function and sequences of actions (Sections 4.2 and 4.6).

4.2 Agent State

All characters in the crowd are represented as $2D$ circular agents; we set their radius $r = .3m$ for all experiments in this paper. At any given moment, an agent collects partial observations of the environment $\{O_i\}$ w.r.t. its local coordinate system (LCS). Then, at any given time t, these are used to update the belief of the agent about the state s_t of the environment; i.e., $s_t = f(\{O_i : i \leq t\})$. Observations contain a) the positions and velocities of the agent and its neighbours (we set a maximum search distance of $R_s = 5m$) and b) the agent's goal position $\mathbf{p}_g \in \mathbb{R}^2$.

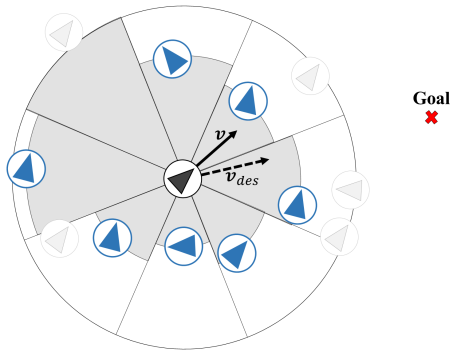


Fig. 2. *Agent State.* The agent state consists of the distances of the nearest neighbours in each of 11 arcs around the agent (we show 8 for clarity) and their relative velocities to the agent; i.e., grey agents are not considered. The radial region is aligned to the current velocity of the agent; when the agent is not moving, this is aligned to the desired velocity; this defines the local coordinate system of the agent.

We designed a state feature vector that aims to capture various behaviours such as grouping, goal seeking and collision avoidance. First, we define an LCS that is aligned to the current velocity $\mathbf{v}$ of an agent; when an agent is not moving, we define the LCS w.r.t. the desired velocity $\mathbf{v}_d$. The state $s \in S \in \mathbb{R}^{36}$ of an agent is the union $s = s_a \cup s_g \cup s_n$ of all relevant information (Fig. 2). $s_a = \{|\mathbf{v}|\} \in \mathbb{R}$ is the internal state of the agent and contains the magnitude of the current velocity. $s_g = \{\mathbf{v} - \mathbf{v_d}\} \in \mathbb{R}^2$ is the relative velocity of the agent as compared to its desired velocity. $s_n = \{(d_i, \mathbf{v}_i - \mathbf{v})\} \in \mathbb{R}^{33}$ is the neighbourhood information and contains distances and relative velocities of closest neighbours in predefined arcs; a representation inspired by Lee et al. [Lee et al. 2007] was chosen. We tested various configurations (8-32 arcs) and found that 11 arcs gave good policies in short training time; we find the closest neighbour in each arc. Besides the difference in the number of arcs (11 instead of 8), the main differences are that a) both distances and differences in average velocity (past $1s$) from the closest agent are stored in each arc and b) the neighbourhood radius R_s is much larger ($5m$ instead of $1m$); this helps agents to anticipate the movement of other agents. In arcs without any agents present, the closest distance is set to R_s and the velocity difference to $\mathbf{v}_0 = (0, 0)$. This results in a state space $S \in \mathbb{R}^{36}$. We expect that the training algorithm will learn which features of the state are important and when; e.g., closer agents will be more important in dense situations or nearby agents that are moving with similar velocities might affect reactions to potential groups.

We note that all features are scaled to have zero mean and unit variance given the reference data; the scaling parameters are stored and later used during training and simulation for all the new states that are generated on the fly. Following recent work for planning control fragments of physically based characters [Liu and Hodgins 2017], we additionally record the index of the last action to help exploration during training (Section 4.3); this index is not part of the input to the policy network.

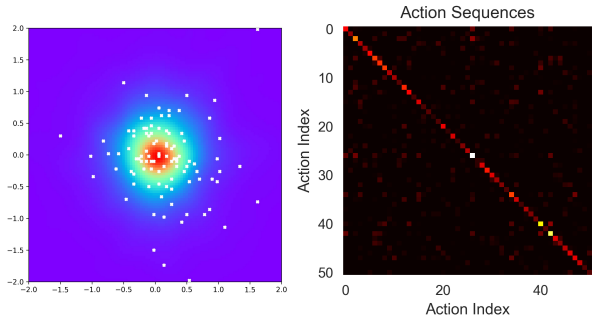
4.3 Actions

We define agent actions to be acceleration vectors $\{\mathbf{a}_i\} \in \mathbb{R}^2$ parallel to the ground w.r.t. the LCS of the agent; an acceleration (including $\mathbf{a}_0 = (0, 0)$) is always applied on each agent at each update step. Instead of manually defining the actions, these are sampled from the distribution of acceleration values of the source data (Fig. 3). To calculate these values from the reference data, we find the acceleration $\mathbf{a}_t$ of any given person at any given timestep t; we estimate values over $.5s$ length windows. We then use Kernel Density Estimation (KDE) in a grid of 100 * 100 values over the extent of the values on each axis to estimate the PDF of the actions. We subsequently sample 50 values out of the KDE; these are the actions to be used by each agent.

To accelerate learning, we go through all the action trajectories of the data and find all possible sequences of actions (Fig. 3b). In many of the datasets, we found that people tend to apply the same action for many consecutive timesteps; moreover, even when switching between actions, these were very similar (i.e., people tend to transition smoothly between nearby actions). We use these sequences during exploration in the DDQ algorithm to improve performance, similarly to the approach by Liu and Hodgins [Liu and Hodgins 2017].

4.4 Data-Driven Reward Function

A reward function $R : S \times A \times S \mapsto \mathbb{R}$ defines the agent task in the RL domain. In the case of crowds, manually defining the reward function is a difficult task, people take into account many – often conflicting – factors at the same time such as moving towards a goal, avoiding collisions and staying with nearby friends/family. The problem is enhanced when we consider the fact that to get realistic ambient crowd behaviour we are interested in also capturing



(a) Action Space (b) Action Sequences

Fig. 3. *Actions.* (a) To get a discrete set of actions A, we sample the distributions of acceleration vectors from the input data ($|A| = 50$ for the experiments in this work). (b) This histogram shows how humans in the Pedestrian dataset switched between two consecutive actions; the diagonality of the graph indicates that people prefer to keep the same action for several consecutive timesteps.

subtleties or even unpredictable behaviour that are not typically simulated by many crowd simulation systems, such as people suddenly standing still, changing directions without any obvious reasons and even bumping slightly onto each other.

Therefore, instead of manually defining a reward function R, we propose to extract an implicit representation from the crowd data. We propose to use *novelty detection* to characterize how much of an inlier an agent state is. The intuition behind this approach is that people tend to move in parts of the state space S that they feel comfortable in; sometimes people tend to prefer very inlier situations (e.g., move straight) while some other times they prefer rare situations (e.g., stop and answer the phone). Therefore, the reward function R should handle all of these cases appropriately while at the same time aim in satisfying the distributions of rewards in the input data; this will ensure that many behaviours appear and the overall feel of the simulation is similar to the input data. We base the design of R on a distance based novelty detection algorithm.

4.4.1 Novelty detection with k-LPE. We choose to use Localized p-value estimation (k-LPE) proposed by [Zhao and Saligrama 2009] as the basis of the novelty detection algorithm; this algorithm is based on the construction of the k-Nearest Neighbour Graph of the data (kNNG) and does not require tuning complicated parameters or defining function approximation classes and can adapt to local structure [Zhao and Saligrama 2009]. Each point $\mathbf{s}_i$ on the graph (of the states in our case), is assigned a score value $R_S(\mathbf{s}_i) \in [0, 1]$ based on the distances to its k-nearest neighbors. Assuming the k-nearest neighbors of $\mathbf{s}_i$ is the ordered list of points $kNN(\mathbf{s}_i) = \{\mathbf{s}_{i1}, \mathbf{s}_{i2}, \ldots, \mathbf{s}_{ik}\}$, the score for each point is then defined as the distance to the farthest nearest neighbor $\mathbf{s}_{ik}$:

$$R_S(\mathbf{s}_i) = d(\mathbf{s}_i, \mathbf{s}_{ik}) \tag{3}$$

We experimented with various distance functions (e.g., Euclidean, Mahalanobis and Manhattan) and found that in most cases they performed the same. For all the experiments demonstrated in this work we use the euclidean distance. Given a new test point (i.e., in this work from training), an anomaly score $p_K(\mathbf{s}_t)$ is estimated based on the following formula [Zhao and Saligrama 2009]:

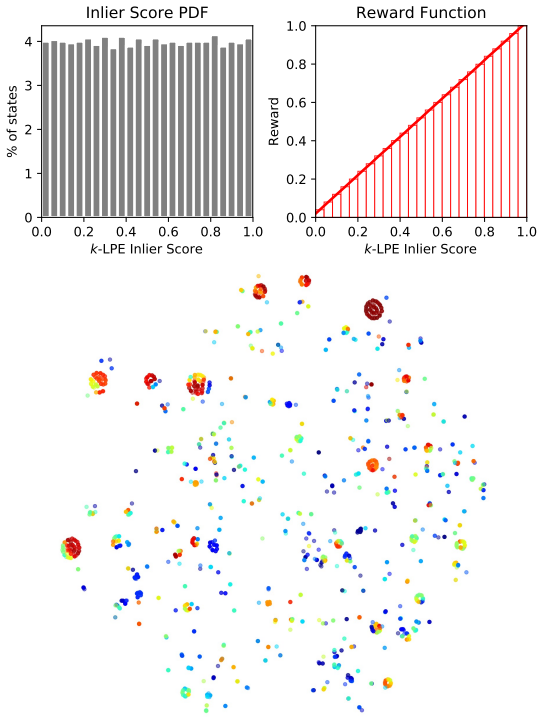


Fig. 4. *Reward Function.* (top) The reward function construction for the flocking dataset. We find the inlier score of each input state $s \in S$ using k-LPE; this always distributes scores *uniformly* in the $[0, 1]$ range. We define the reward function to be the cumulative histogram of the inlier data; this gives a score that is directly proportional to the k-LPE score. (bottom) t-SNE projection of the states in the flocking dataset; warm colours indicate high rewards.

$$p_K(\mathbf{s}_t) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}_{\{R_S(\mathbf{s}_t) \leq R_S(\mathbf{s}_i)\}} \tag{4}$$

where $\mathbb{I}_{\{.\}}$ is the indicator function, i.e., it returns 1 if the condition is valid and 0 otherwise; this maps the data in a *uniform distribution* over the interval $[0, 1]$ with 0 indicating complete inliers and 1 extreme outliers (Fig. 4). Intuitively, Equation 4 can be thought of as indicating what portion of existing points on the kNNG of the training data have worse score than the testing point. Or more simply, how much of the training data is more anomalous than a given piece of testing data.

4.4.2 Reward function. To construct the reward function , we first find the inlier scores $\mathcal{I} = \{1 - p_K(s) : \forall s \in S\}$ of all states in the input data and construct their Cumulative Density Function (CDF): $P_{\mathcal{I}}(\mathcal{I} \leq 1 - p_K(s))$. Since inlier scores are uniformly distributed in $[0, 1]$, the CDF is *linear* (Fig. 4). We then set the reward function to:

$$R(s, a, s') = P(\mathcal{I} \leq 1 - p_K(s')), \tag{5}$$

i.e., when evaluating a transition (s, a, s') only the landing state s' is considered. Intuitively, this choice of a reward function helps

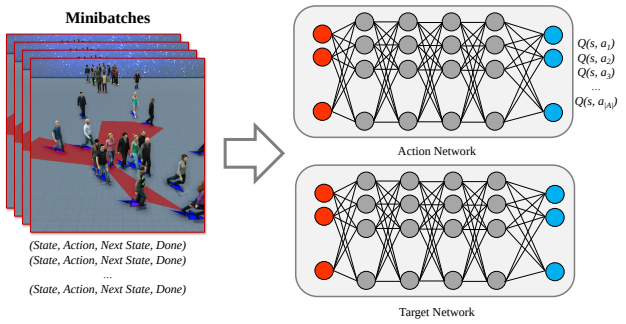


Fig. 5. *The Q-Networks.* The Q-Network is used at each simulation step of both training and simulation. During training, minibatches of experiences are use to train the network. The output $Q(s, a)$ is the expected cumulative reward for each possible action; the action with the highest reward is selected and applied on the agent.

agents to visit both rare and common states proportionally to the probability they were generated by the source state distribution. Additionally, all possible states $s \in S$ get rewards in the $[0, 1]$ range; this helps in being able to capture abnormal behaviors in our simulations that would not be easy to do with a manually defined reward function.

4.5 The Q-Network

We use a relatively small fully connected feed forward Neural Network (NN) to represent the parameterized Q-value function $\tilde{Q}(s, a; \theta) \approx Q(s, a)$ (Fig. 5); the parameters vector θ consist of all the weights and biases of the NN. The input of $\tilde{Q}(s, a; \theta)$ is a vector $s \in \mathbb{R}^{|S|}$ that represents the current agent state ($|S| = 36$ in all experiments) and outputs a vector of dimensionality $|A|$; i.e., it outputs $[Q(s, a) : \forall a \in A]$ given a state $s \in S$. This choice was inspired by Mnih et al. [Mnih et al. 2015] and allows to retrieve the Q-value for all actions at the same time for any given state; the selected action is the one with the highest Q-value. The network has four fully connected hidden layers; the first two consist of 128 nodes each and the other two of 64. All of them have a sigmoid activation function - we also experimented with ReLU activations which had almost identical performance.

4.6 Learning Algorithm

We build upon the work of van Hasselt et al. [van Hasselt et al. 2016] and employ the DDQ algorithm. To improve performance and stability we a) use an experience replay memory and b) maintain two Q-networks. The *experience replay memory* M is a finite queue (Table 1) of experiences that is used to break correlations between consecutive observations in samples observed in training sequences. During training, experiences $(s, a, s', r, done)$ are stored in M and then minibatches of experiences are sampled from M to do the Q-updates; we use the Adam optimizer for these updates. The *done* variable indicates if the agent reached a terminal state. We set a learning rate of $1e-5$ at the beginning of training, and half it every 10000 episodes of training.

4.7 Learning strategy

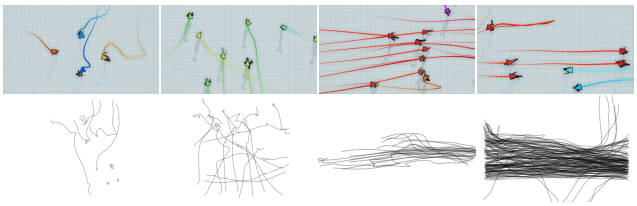
We initially considered initializing several agents in an environment and letting them learn optimal strategies to replicate the behaviours in reference data. However, this is not a good strategy; running a complete multi-agent simulation with all agents initially acting completely randomly is wasteful and might not lead to good policies. Instead, we take a more guided strategy where we use the reference data as part of the training environment; we cut segments of the reference data (i.e., trajectories in a range), we select randomly one of these trajectories to initialize a simulated agent and we let the rest to follow their original trajectories (i.e., these are *playback* agents). For the simulated agent, we set starting and end positions, intended speed and initial velocities using three choices; a) we leave the reference data values, b) we jitter them using Gaussian noise or c) we select random values in acceptable ranges. During training, the simulated agent takes actions in one of the following ways; a) by following the currently learned policy, b) by choosing a completely random action or c) by selecting an action based on the statistics of the action sequences from the reference data (see Fig. 3). The playback agents simply follow reference trajectories. After each simulation step, experiences are stored in the experience replay memory and minibatches of 64 experiences are sampled to update the Q-networks. A list of all the hyperparameters and the values we used for this work can be seen Table 1.

Table 1. Default values for hyperparameters.

Parameter	Value	Description
r	$.3m$	Agent Radius
R_s	$5m$	Maximum search distance for neighbours
H	$10s$	Maximum time for agent training
γ	.90	Discount factor
M	64000	Experience replay memory size
M_{min}	6400	Minimum experiences before training starts
E_e	1000	Episodes of that E_e is linearly changed
e_r	$1.0 - .1$	Random exploration linearly changed in E_e
e_g	.4	Guided exploration
batch size	64	Number of experiences used per update
k	100	Number of nearest neighbours used by k-LPE
freq	$5Hz$	Agent update frequency (training+simulation)
α	$1e^{-4}$	Initial learning rate
α_e	10000	Episode period to drop α
α_d	.5	Drop factor of α every α_e

5 RESULTS

The training system was implemented in Python using the Keras Deep Learning framework. The simulation timestep was set to $.04s$ (i.e., $25Hz$ which matches the frame rate of most of the input crowd videos). Each individual agent takes a decision every $.2s$; for all in-between timesteps the agent follows the action that was returned by the policy and collects observations to update its belief for the state. Agents are split into 5 subsets; decision-making is interleaved between simulation steps and agents in the same set take decisions in a random order; i.e., $1/5$ of the agents takes decisions per timestep. During training, we periodically export the Q-networks and various statistics to check for convergence. For all the demonstrated cases, simulation was faster than real-time; agents simply collect state information and query the Q-network to get acceleration values to apply.



(a) Chat (b) Flock-1 (c) Flock-2 (d) Pedestrians

Fig. 6. *Crowd Data.* We used several datasets to demonstrate that our method can capture a variety of behaviours. (a) (b) Chat and Flock-1 are from Ju et al. [Ju et al. 2010], (c) Flock-2 from Charalambous et al. [Charalambous et al. 2014] and (d) Pedestrians from Lerner et al. [Lerner et al. 2007].

To evaluate our approach, we train a set of different policies using different crowd datasets. We quantitatively evaluate GREIL by measuring several statistics from different simulations and compare them against the input data (e.g., see Figures 7 and 9). These statistics help in getting a macroscopic view of how GREIL behaves with respect to data, such as fundamental diagrams of speed and flow with respect to density, social interactions such as walking with other agents (by measuring for example the distance to the closest naighbour) or goal oriented behaviour (by measuring deviations from a straight path to the goal). We note here that even though the fundamental diagrams are more informative in very dense crowd situations which we do not handle in this work, they were included in the experimental evaluation for the sake of completeness since it is a standard way of comparing macroscopically crowds in the literature[Bellomo and Dogbe 2011; Siebel and Mauser 2006]. Detailed discussion on the results can be found in the following sections. We additionally provide visualizations of simulations to get a more qualititative view of the results.

The final crowd visualizations were done in the Unity Game Engine; animated results can be seen in the supplemented video and are a better indicator of the quality of the learned policies. It is important to note that each agent acts independently and there is no information in the state describing groups or the pose of a character and therefore animations shown in the videos such as talking, waving hands, etc. were added for aesthetic reasons.

5.1 Data / Learned Policies

We trained policies for several crowd datasets; in this work we demonstrate results using datasets from [Charalambous et al. 2014; Ju et al. 2010; Lerner et al. 2007] (see Fig. 6). Chat is a dataset of 8 agents moving around, stopping and talking to each other. The Flocking datasets (Flock-1, Flock-2) consist of 16 and 24 agents respectively and demonstrate crowds moving together in roughly the same direction; i.e., they have similar desired velocities. Finally, the pedestrian dataset is the more complex one since it has 148 people with mixed behaviors such as walking by themselves and/or in small groups of 2-4 people entering and/or leaving groups, social interactions, abnormal yet acceptable behaviors such as stopping suddenly and changing directions, it mixes slow and fast walkers, etc. Fig. 7 summarizes the fundamental diagrams of speed $|\mathbf{v}|$ and density specific speed $J = |\mathbf{v}| * d$ (i.e., flow) with respect to density d [Siebel and Mauser 2006]. Speed is measured in a window of $1.0s$ whereas density is measured per agent in a radius of $3.6m$ which is the social distance as defined in the proxemic model of Hall [1963]. Since these datasets are not very dense, flow has essentially a linear relationship to density and pedestrians move mostly with constant speed. We note that the Flock-2 and the Pedestrians datasets are tracked from real-world ambient crowds in Cyprus whereas the other datasets are captured in a controlled environment in South Korea, which can partially explain the differences in absolute values in speed and densities.

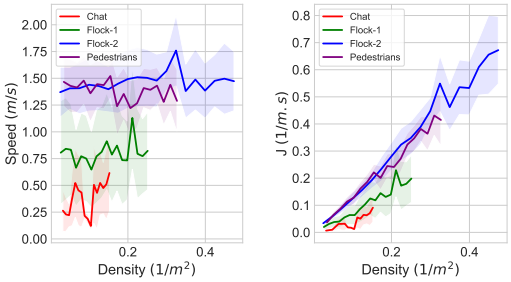


Fig. 7. *Properties of training datasets.* Fundamental diagrams of pedestrian dynamics for all the training datasets used in this paper.

Fig. 8 summarizes the learning curves for all datasets; they show the average Q-Values for each minibatch being updated over time. Shaded values show these values per update step whereas the thick lines represent a moving average of 200 update steps. Recall that the reward function returns values in the $[0, 1]$ range; having $\gamma = .9$ and assuming we simulate for an infinite horizon, the expected cumulative reward would be around 10 (from Equation 2). Since the k-LPE based reward function (Equation 5) maps rewards uniformly in the $[0, 1]$ range, the average reward per state is .5; therefore it was expected that these curves would converge around 5.

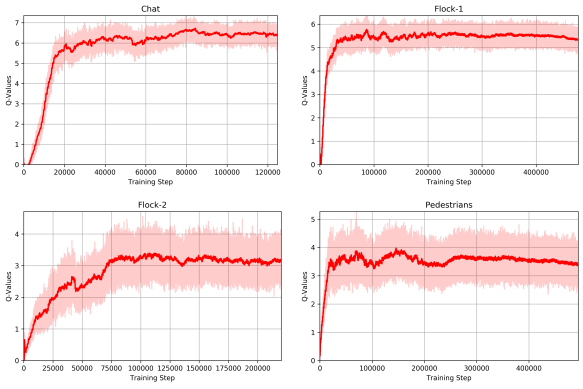


Fig. 8. *Training Curves.* Training converges after a few thousand training steps – the thick lines indicate a moving average of the Q-value function.

In Fig. 10 we demonstrate the progression in learning for the Flock-2 dataset. The demonstration simulation for this policy consists of the 24 agents having the same start and end positions as the source data. Initially, the agents learn quickly that the optimal strategy is to move together but they do this without caring about their targets; gradually they learn that the optimal strategy is to move together towards their goals.

5.2 Simulations in similar conditions

Given the training trajectories, we define a set of scenarios based on the original data as a starting point. Each agent has the same initial and ending positions as a tracked person from the input; the desired speed is set to the person's trajectory average speed. As expected, all agents successfully reach their goals following the learned policy; in Fig. 11 we demonstrate the simulated trajectories against the input trajectories of the pedestrian dataset. In the following paragraphs, we demonstrate the same policy applied to different environments.

We additionally found that the fundamental diagrams are similar to the input, however, there is some difference in the spectrum of values covered in our simulations with respect to the input data (Fig. 9). For example, in the replication of the Flock-2 scenario (Fig. 9, top row), we do not have densities over .13 $agents/m^2$. We do however find that agents keep similar distances to their closest neighbors and have low deviation from the intended path, indicating similar behavior to the input(lefmost column). This behaviour is consistent when using other datasets such as the Pedestrian one (Fig. 9, bottom row). When trying to replicate the input data, we get a fundamental diagram that is quite similar to the input. A more interesting observation in this experiment is that agents end up having low distances to the nearest neighbours; this indicates grouping behaviour which is a dominant behaviour in the input data. Additionally, we find that agents have similar deviation from the desired path. We note once again that in our fundamental diagrams we do not get bottleneck situations; here, we would observe agents slowing down as density around them increased. This is due to the fact that we do not have very dense data and simulations in our experiments. We provide these evaluations for completeness as they are a common practice in the transportation and crowd simulation literature[Siebel and Mauser 2006; Wolinski et al. 2014].

To demonstrate that we do not overfit, we also train a controller using the 148 trajectories from the pedestrian dataset and then initialize a simulation with 137 trajectories from a later stage of the same dataset (ground truth). Snapshots of the ground truth can be seen on the top row of Fig. 12; the bottom row demonstrates snapshots of the simulation at the same exact points in time (please consult the video). We notice that the simulation gives consistent groups and individuals with collision avoidance and navigation demonstrating that we can simulate crowds under similar conditions to the input.

5.3 Sandbox Simulations

To demonstrate the emergent properties of the learned policies, a sandbox toric-space scenario environment is initialized with 40 agents in a square region of $20m * 20m$. Agents are initialized at random positions and random desired velocities; when an agent leaves from one side of the environment it appears on the opposite side. As expected, using different policies resulted in different behaviors. In the flocking dataset, agents flock and move together after just a few seconds; with Flock-1 agents are closer and move relatively slow whereas with Flock-2 they move faster and more spread out (Fig. 13). Fig. 14 summarizes some statistics for all the sandbox simulations. Macroscopically, it seems that the simulations match the input data. However, getting into more depth, it seems that the more data the better the capabilities of the policy to capture the desired behaviours.

In the case of the Pedestrian policy, many behaviors emerge; agents move with different flows, they split into multiple subgroups, some move alone, some move slowly, some stop abruptly and change directions and so forth. Importantly, these behaviors are persistent (i.e., they last for long periods of time) resulting in more believable behaviors (please consult the video). This is a key result since agents act individually and none of them has a knowledge of the behavior they should follow (besides the control policy); this is decided from the initial conditions and the states that agents perceive over time. That is, if it is for the benefit of an agent to stay with other agents (i.e., larger future rewards), then it will stay with them for long periods of time. The Chat scenario on the other hand, failed in this scenario - agents simply started running around the scene without stopping next to each other; the assumption here is that the scenario is much more complex than the input data and a better training strategy could be explored. There are some methods from the Inverse Reinforcement Learning literature that demonstrate how to learn from a few expert trajectories that we plan to explore as future work [Finn et al. 2016] to handle this issue.

5.4 Comparison to data-driven methods

We compare our method to the Perception Action Graph (PAG) method which is a supervised learning based approach [Charalambous and Chrysanthou 2014]. There is a fundamental problem with supervised learning based approaches; they need large amounts of data to generalize well. Even having large amounts of data, differences between simulation and training states are bound to exist. Because of this, the actions returned by the learned models have errors; in a sequential decision making setting, these errors accumulate over time. This gets worse in a multi-agent simulation environment where the perception of any agent about the state of the environment is affected by the erroneous actions of other agents. Additionally, agents do not consider the future consequences of their actions; actions are selected greedily. In GREIL on the other hand, novel states are simulated and visited during training and are evaluated by the future outcomes of simulations. These properties of GREIL result in a robust data-driven crowd simulation system that captures a variety of behaviors and generalize to more difficult situations. To demonstrate these differences, we create two scenarios; a long corridor ($200m$ length) and a crossing scenario that has double the agents as compared to the input data (296 agents).

PAG uses as state representation called Temporal Perception Patterns (TPP). For our comparisons, we use a TPP with a field of view $\theta = 120^\circ$ and a temporal window of $1s$. We also used a threshold of .95 for the construction of the graph (please refer to the work

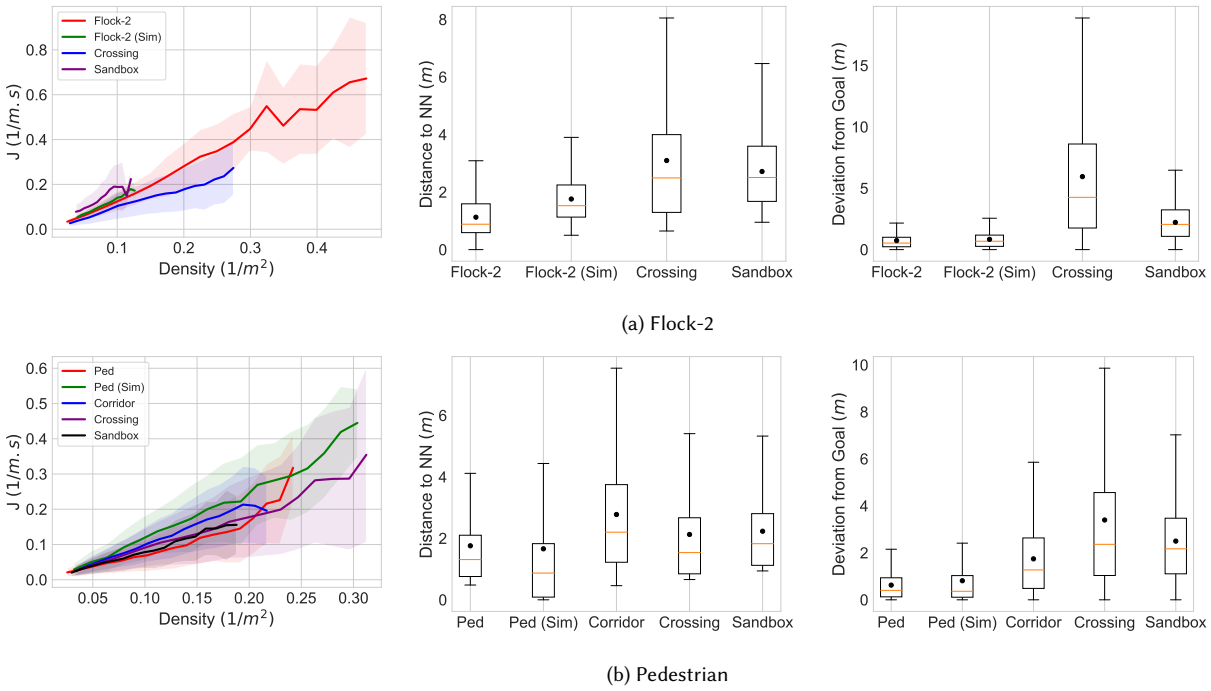


Fig. 9. *Simulation statistics.* We quantitatively evaluate different simulations against real world data that were used to train GREIL policies. These include the fundamental diagram of density specific speed, distance to the nearest neighbor, and deviation from the goal. The leftmost columns in the box plots represent the reference data, whereas the other plots represent simulations ran by trained policies that used the reference data.

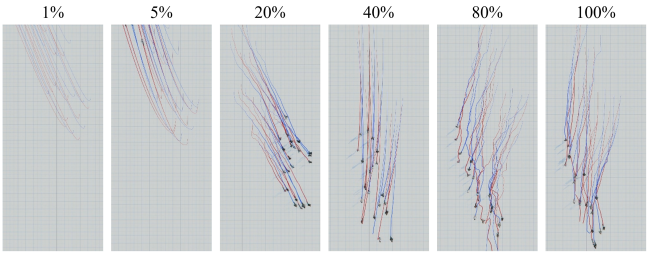


Fig. 10. *Training Progress.* The learned policy for the Flock-2 dataset over time. Initially the agents learn quickly to group and move together but fail to reach their goals; over time they learn to move towards their goals.

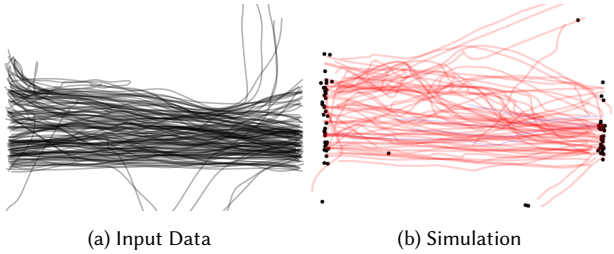
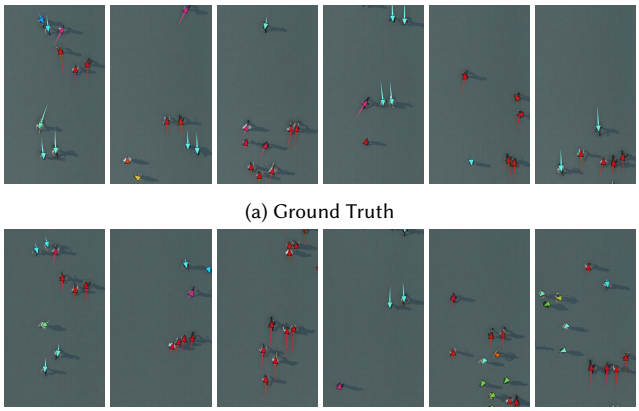


Fig. 11. *Replicating the input.* (a) Data used to learn a policy. (b) A simulation of the original agent configuration using the learned policy.

by Charalambous and Chrysanthou [Charalambous and Chrysanthou 2014] for technical details). In the long corridor (Fig. 15), PAG introduces errors over time whereas in the crossing scenario errors are accumulated both spatially and temporally (Fig. 16); notice the chaotic nature of the PAG trajectories. GREIL agents on the other hand move efficiently and in groups towards their targets. Even though both methods capture macroscopic behaviour such as the relationship of speed to density quite well (Fig. 17), GREIL demonstrates better generalization and stability performance w.r.t. deviation from intended paths in complex situations.

5.5 Comparison to a baseline RL method

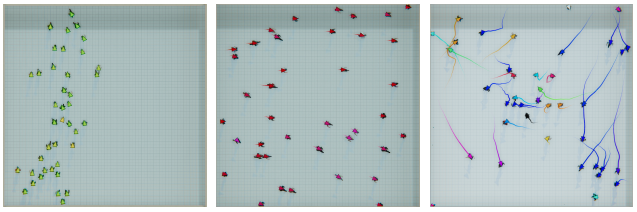
We also compare against a variation of a state-of-the-art representative reinforcement learning system for collision avoidance in the robotics domain[Long et al. 2018]; the main change being that we use our own state representation. We note that we opted not to compare against the recent Configurable Crowd Profiles approach [Panayiotou et al. 2022]; this would only make sense if this method was tuned to real-world data. However, this is not an easy task. CCP mixes four specific behaviors; to have a fair comparison we would need to find optimal parameters for each agent. These parameters would probably need to change over time. By itself this is a very challenging problem. If we leave standard parameters, this would



(a) Ground Truth

(b) GREIL

Fig. 12. *Comparison to ground truth.* (top) Actual crowd data against (bottom) simulation with GREIL. Colors indicate the current velocity of agents. GREIL manages to simulate similar behavior without overfitting to the data.



(a) Flock-1 (b) Flock-2 (c) Pedestrians

Fig. 13. *Sandbox Simulations.* Agents adapt to the learned policies after a few seconds (colors indicate current velocities). In (a) and (b) all agents move together with different speeds and densities. In (c) on the other hand we get a multitude of behaviors; groups, individuals, slow/fast walkers, etc.

be similar to [Long et al. 2018] with a slightly different constant reward function.

The reward function is defined as $r = r_c + r_g$; r_c is the punishment for collision and r_g is the reward for moving towards the goal. We set a sparse reward $r_c = -1$, $r_g = 1$ when the agent collides or reaches a goal and a dense reward signal $r_g = 2.5(||\mathbf{p}_{t-1} - \mathbf{g}|| - ||\mathbf{p}_t - \mathbf{g}||)$ at any other given moment; $\mathbf{p}_t$ and $\mathbf{g}$ are the agent's position at time t and goal position respectively. r_g motivates the agent to move continuously towards the goal. The training curve for these parameters can be seen in Fig. 18.

This approach, being RL based, manages to navigate the agents more efficiently than PAG and, at the trajectory level at least, very similar to GREIL (Fig. 16). Examining the simulations though, we can see that the generated behaviors are quire different. Because of the simple nature of the reward function, agents move as fast as possible towards their targets whilst avoiding collisions; no other interaction is handled. There are no groups, wandering agents or other emergent behavior; defining a manual reward function to capture these would not be an easy task. GREIL on the other hand navigates agents towards their targets whilst respecting the input data behaviors. Please consult the video for the animated comparisons.

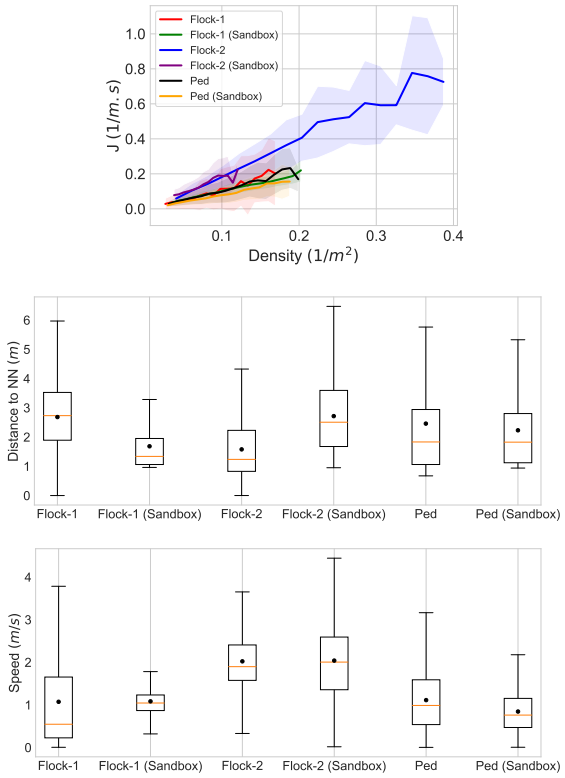
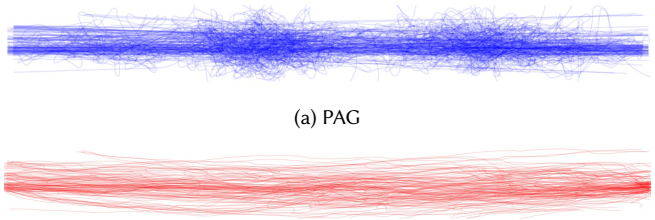


Fig. 14. (top) The fundamental diagrams of the sandbox scenarios indicate that macroscopically, the simulations match the input data. However, (middle, bottom) the sandbox simulations with the flocking data do not match the full spectrum of the input data, whereas the simulations with the pedestrian dataset have a better coverage.



(a) PAG

(b) GREIL

Fig. 15. Supervised learning methods such as PAG [Charalambous and Chrysanthou 2014] can result in non consistent simulations because of the accumulation of errors over time and over multiple agents' actions.

5.6 Generalization Simulations

To demonstrate the applicability of the learned policies on different scenarios, we simulated different environments with the same policies. In Fig. 19, we demonstrate resulting trajectories from the application of the Pedestrian policy in 5 different scenarios. Dots indicate agent goals, red lines indicate agents who reached their

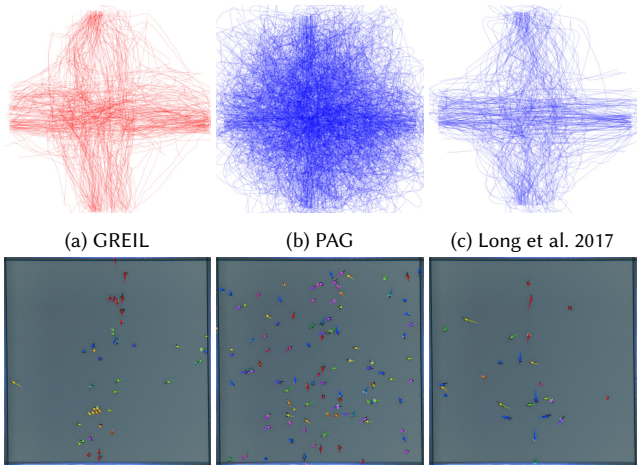


Fig. 16. GREIL can handle situations that were different from the training data; this simulation has double the agent density of the source data. The agents are colored based on current moving direction. Supervised learning approaches (PAG) have difficulties to generalize to unseen situations, whereas RL methods produce similar trajectories but do not capture aspects of the data such as groups (notice the dynamically formed clusters of similar colored agents in the left figure).

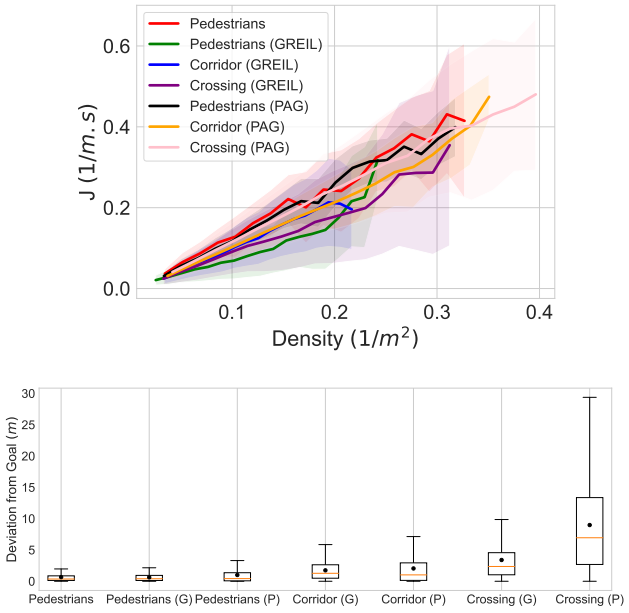


Fig. 17. Quantitative comparison of a supervised learning method (PAG[Charalambous and Chrysanthou 2014]) against GREIL. Macroscopically, both methods are quite similar, however PAG struggles in difficult scenarios (e.g., Crossing) due to error accumulation.

goals and blue those who failed to do so before leaving the bounding box of the simulation environment. These simulations manage to capture both common and uncommon behaviors that are present in the data (Fig. 20); social groups that stay together, individuals,

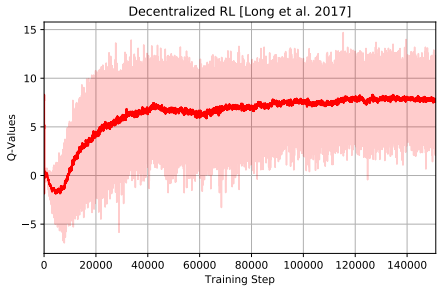


Fig. 18. Convergence of our baseline implementation of Long et al.[2018].

wandering agents (which reach their goal after some time), agents joining and/or leaving groups, etc. Please refer to the accompanied video to see animated results of the simulations.

Additionally, Fig. 21 shows the sequence of actions that two of the agents took during the simulation when using this policy; it is clear that each agent applied only a handful of actions for prolonged periods of time. This indicates consistency in decision-making and behavior and that different subsets of agents use different (small) subsets of the action space indicating heterogeneity in behavior; heterogeneity is implicitly influenced by the initial conditions of the agents (e.g., goals, position, preferred speed).

6 DISCUSSION/FUTURE DIRECTIONS

In this paper, we leverage fairly recent advances in Deep RL, namely the Double Deep Q-learning algorithm [van Hasselt et al. 2016], to find policies for crowd agents. We use a novel reward function structure that is based on Novelty Detection and more specifically k-LPE; this function gives a reward to a state that is essentially proportional to the probability it came from the distribution of input states. We demonstrate results from trained controllers that manage to capture and simulate both common behaviors such as social groups, individuals and collision avoidance and rare cases such as sudden stops and changes in direction, joining/leaving groups, etc. Moreover, the agent behavior is more persistent than previous data-driven methods.

Learning Strategy. While we effectively managed to create agents that display a wide variety of behaviors, there are situations where the policies are not robust, such as, when small amount of data is available. It is possible, however, to combine other improvements to increase the performance as well as the robustness of our approach [Hessel et al. 2018]. For example, using a prioritized experience replay memory, instead of uniformly sampling from it [Schaul et al. 2015], a dueling architecture that better generalizes across actions [Wang et al. 2016], multi-step learning [Sutton and Barto 2018], or learning a distribution of returns instead of the expected return [Bellemare et al. 2017].

Continuous Actions. The training algorithm that we employ requires the actions to be discrete. An alternative approach would be to use continuous actions. In such cases, actor-critic (e.g., see [Mnih et al. 2016]) or policy search methods (e.g., see [Levine and Koltun 2013]) could be an alternative. In addition, multimodal behaviors

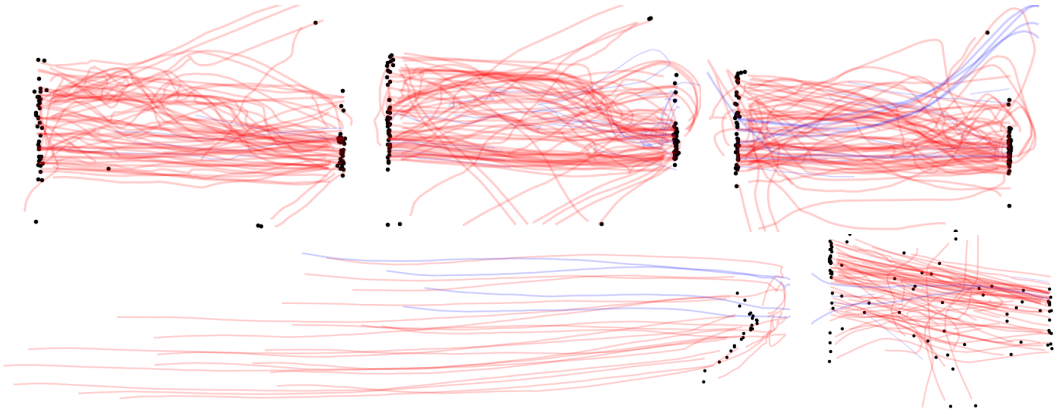
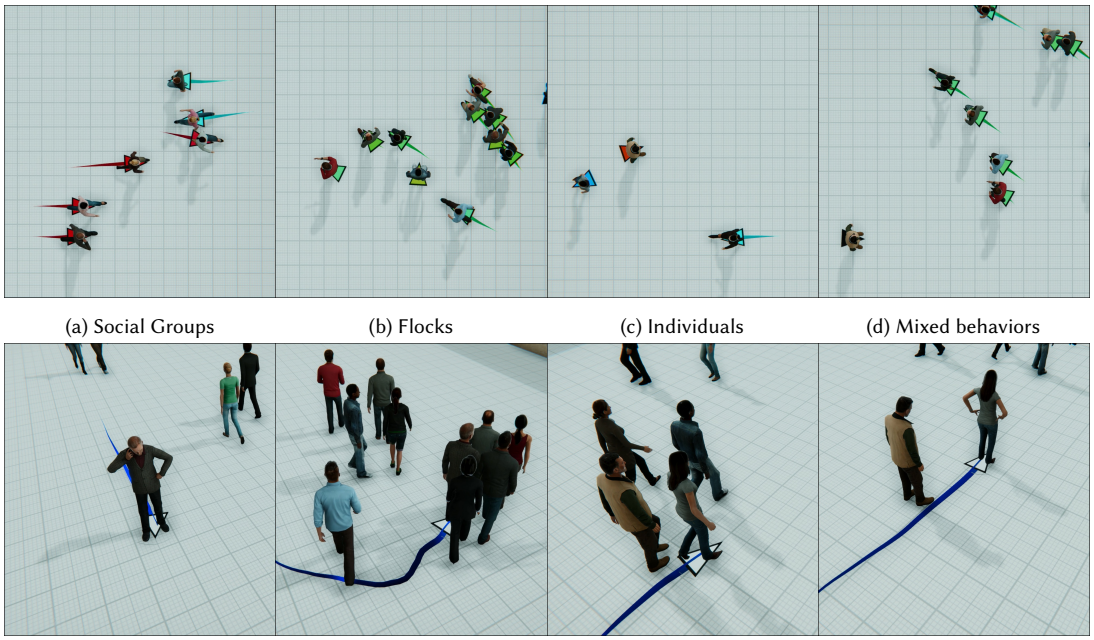


Fig. 19. Different simulations under the same policy (Pedestrians). The dots indicate goal positions; red trajectories indicate agents who reached their goals and blue those who reached the boundaries of the environment before reaching their goals.



(a) Social Groups (b) Flocks (c) Individuals (d) Mixed behaviors

(e) Sudden Stops (f) Joining Groups (g) Leaving Groups (h) Unexpected behavior

Fig. 20. Results from the same simulation - all agents use the pedestrian controller. The policy is able to capture and simulate simultaneously (a-d) social groups, flocks, individuals, mixed behaviours; (e-f) and various behaviors such as agents suddenly standing still, joining/leaving groups, etc. Please watch to the supplemented video for the animated results.

could be captured by considering a soft Bellman equation [Haarnoja et al. 2017].

Reward Function. We designed an intuitive reward function that aims at guiding agents in places of the state space real people prefer. Our method is based on novelty detection; since this is a nearest neighbor based approach, it has some limitations such as search speed especially in higher dimensions. Additionally, these methods do not scale well with large amounts of data. An alternative approach we would like to explore, is to learn the underlying reward function concurrently to the policy learning procedure. This is the subject of Inverse Reinforcement Learning methods [Abbeel and Ng 2004; Finn et al. 2016; Wulfmeier et al. 2015; Ziebart et al. 2008]. We are interested to see how our approach compares to these and also explore methods proposed by these approaches to learn from a limited set of expert trajectories (for example in [Finn et al. 2016]).

Heterogeneous Crowds. In this work, we learn a policy for all the agents in a given dataset. This allows to learn for example crowd behaviors in a train station or a street during rush hour or at night.

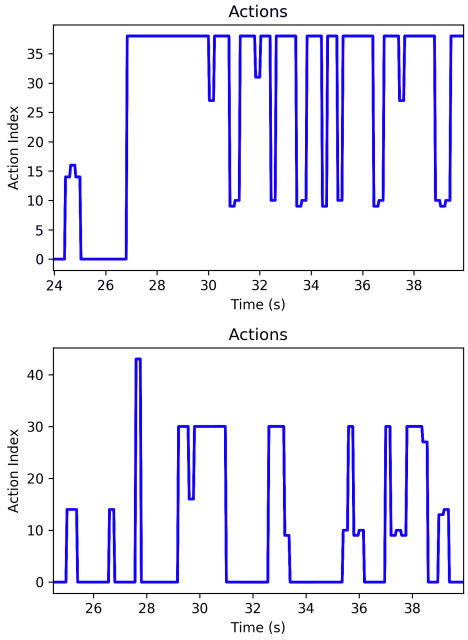


Fig. 21. **Actions Sequences.**

If we want to learn finer details of individualized behavior (e.g., children, athletic, social, etc.), we need to be able and identify input trajectories that represent similar behaviors and learn "profile" based policies. Some recent work that could help us reach this goal has shown that it is possible to control simulated behavior by adding a set of control signals in the input of a single policy [Panayiotou et al. 2022].

RL to control macroscopic features of the crowd. We did not discuss how and when to initialize agents in a given environment; a simple way to approach this is to initialize agents around the borders of the simulation environment at certain intervals and give them some goals. We argue, that a more clever and robust way is to develop a controller (i.e., an agent) that observes the state of the entire simulation and acts accordingly by spawning agents with specific agendas. The state can consist for example of macroscopic patterns such as densities, flows etc. We are interested in developing these kinds of controllers and Reinforcement Learning seems to be a very promising approach to do so.

Applications to other domains. Our method is applied on ambient crowds for game or movie scenarios; in these scenarios we care mostly for the appearance of realistic crowds. A large body of work in the crowd simulation community also considers applications such as evacuations or autonomous driving. Given appropriate data, we would like to explore the applicability of our method to these cases; we believe that with specific modifications to the state and actions spaces, our method could potentially work.

ACKNOWLEDGMENTS

This project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No 739578 and the Government of the Republic of Cyprus through the Deputy Ministry of Research, Innovation and Digital Policy. This work has received funding from the Spanish Ministry of Science and Innovation (PID2021- 122136OB-C21).

REFERENCES

Pieter Abbeel and Andrew Y. Ng. 2004. Apprenticeship Learning via Inverse Reinforcement Learning. In *Proceedings of the Twenty-first International Conference on Machine Learning* (Banff, Alberta, Canada) *(ICML '04)*. ACM, New York, NY, USA, 1–. https://doi.org/10.1145/1015330.1015430

Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. 2016. Social lstm: Human trajectory prediction in crowded spaces. (2016), 961–971.

Javad Amirian, Jean-Bernard Hayet, and Julien Pettré. 2019. Social ways: Learning multi-modal distributions of pedestrian trajectories with gans. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*. 0–0.

Marc G Bellemare, Will Dabney, and Rémi Munos. 2017. A distributional perspective on reinforcement learning. (2017), 449–458.

Nicola Bellomo and Christian Dogbe. 2011. On the modeling of traffic and crowds: A survey of models, speculations, and perspectives. *SIAM review* 53, 3 (2011), 409–463.

Panayiotis Charalambous and Yiorgos Chrysanthou. 2014. The PAG Crowd: A Graph Based Approach for Efficient Data-Driven Crowd Simulation. *Computer Graphics Forum* 33, 8 (2014), 95–108. https://doi.org/10.1111/cgf.12403

Panayiotis Charalambous, Ioannis Karamouzas, Stephen J. Guy, and Yiorgos Chrysanthou. 2014. A Data-Driven Framework for Visual Crowd Analysis. *Computer Graphics Forum* 33, 7 (2014), 41–50. https://doi.org/10.1111/cgf.12472

N. Courty and T. Corpetti. 2007. Crowd motion capture. *Computer Animation and Virtual Worlds* 18, 4-5 (2007), 361–370.

Chelsea Finn, Sergey Levine, and Pieter Abbeel. 2016. Guided cost learning: Deep inverse optimal control via policy optimization. In *International Conference on Machine Learning*. 49–58.

Julio E Godoy, Ioannis Karamouzas, Stephen J Guy, and Maria Gini. 2015. Adaptive learning for multi-agent navigation. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*. International Foundation for Autonomous Agents and Multiagent Systems, 1577–1585.

Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. 2018. Social gan: Socially acceptable trajectories with generative adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2255–2264.

Stephen J. Guy, Jur van den Berg, Wenxi Liu, Rynson Lau, Ming C. Lin, and Dinesh Manocha. 2012. A Statistical Similarity Measure for Aggregate Crowd Dynamics. *ACM Trans. Graph.* 31, 6, Article 190 (Nov. 2012), 11 pages. https://doi.org/10.1145/2366145.2366209

Tuomas Haarnoja, Haoran Tang, Pieter Abbeel, and Sergey Levine. 2017. Reinforcement learning with deep energy-based policies. In *International conference on machine learning*. PMLR, 1352–1361.

Edward T Hall. 1963. A system for the notation of proxemic behavior. *American anthropologist* 65, 5 (1963), 1003–1026.

Dirk Helbing and Péter Molnár. 1995. Social force model for pedestrian dynamics. *Phys. Rev. E* 51, 5 (May 1995), 4282–4286. https://doi.org/10.1103/PhysRevE.51.4282

Peter Henry, Christian Vollmer, Brian Ferris, and Dieter Fox. 2010. Learning to navigate through crowded environments. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE, 981–986.

Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. 2018. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

H. Hildenbrandt, C. Carere, and CK Hemelrijk. 2010. Self-organized aerial displays of thousands of starlings: a model. *Behavioral Ecology* 21, 6 (2010), 1349.

Kaidong Hu, Brandon Haworth, Glen Berseth, Vladimir Pavlovic, Petros Faloutsos, and Mubbasir Kapadia. 2021. Heterogeneous crowd simulation using parametric reinforcement learning. *IEEE Transactions on Visualization and Computer Graphics* (2021).

Eunjung Ju, Myung Geol Choi, Minji Park, Jehee Lee, Kang Hoon Lee, and Shigeo Takahashi. 2010. Morphable Crowds. *ACM Trans. Graph.* 29, 6, Article 140 (Dec. 2010), 10 pages. https://doi.org/10.1145/1882261.1866162

Mubbasir Kapadia, Matt Wang, Shawn Singh, Glenn Reinman, and Petros Faloutsos. 2011. Scenario space: characterizing coverage, quality, and failure of steering algorithms. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Vancouver, British Columbia, Canada) *(SCA '11)*. ACM, New

York, NY, USA, 53–62. https://doi.org/10.1145/2019406.2019414
T. Kwon, K.H. Lee, J. Lee, and S. Takahashi. 2008. Group motion editing. In *ACM Transactions on Graphics (TOG)*, Vol. 27. ACM, 80.
Yu-Chi Lai, Stephen Chenney, and ShaoHua Fan. 2005. Group motion graphs. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation* (Los Angeles, California). 281–290.
Jaedong Lee, Jungdam Won, and Jehee Lee. 2018. Crowd simulation by deep reinforcement learning. In *Proceedings of the 11th Annual International Conference on Motion, Interaction, and Games*. 1–7.
Kang Hoon Lee, Myung Geol Choi, Qyoun Hong, and Jehee Lee. 2007. Group Behavior from Video: A Data-driven Approach to Crowd Simulation. In *Proceedings of the 2007 ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (San Diego, California) *(SCA '07)*. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 109–118. http://dl.acm.org/citation.cfm?id=1272690.1272706
Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. 2007. Crowds by Example. *Computer Graphics Forum* 26, 3 (2007), 655–664. https://doi.org/10.1111/j.1467-8659.2007.01089.x
Alon Lerner, Yiorgos Chrysanthou, Ariel Shamir, and Daniel Cohen-Or. 2010. Context-Dependent Crowd Evaluation. *Computer Graphics Forum* 29, 7 (2010), 2197–2206. https://doi.org/10.1111/j.1467-8659.2010.01808.x
Sergey Levine and Vladlen Koltun. 2013. Guided policy search. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*. 1–9.
Yi Li, Marc Christie, Orianne Siret, Richard Kulpa, and Julien Pettré. 2012. Cloning Crowd Motions. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (Lausanne, Switzerland) *(SCA '12)*. Eurographics Association, Goslar Germany, Germany, 201–210. http://dl.acm.org/citation.cfm?id=2422356.2422385
Libin Liu and Jessica Hodgins. 2017. Learning to schedule control fragments for physics-based characters using deep q-learning. *ACM Transactions on Graphics (TOG)* 36, 3 (2017), 29.
Pinxin Long, Tingxiang Fan, Xinyi Liao, Wenxi Liu, Hao Zhang, and Jia Pan. 2018. Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. In *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 6252–6259.
Karttikeya Mangalam, Yang An, Harshayu Girase, and Jitendra Malik. 2021. From goals, waypoints & paths to long term human trajectory forecasting. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 15233–15242.
Francisco Martinez-Gil, Miguel Lozano, and Fernando Fernández. 2011. Multi-Agent Reinforcement Learning for Simulating Pedestrian Navigation. In *International Workshop on Adaptive and Learning Agents*. 53.
Ronald A. Metoyer and Jessica K. Hodgins. 2003. Reactive Pedestrian Path Following from Examples. In *CASA '03: Proceedings of the 16th International Conference on Computer Animation and Social Agents (CASA 2003)*. IEEE Computer Society, Washington, DC, USA, 149.
Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. 2016. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*. 1928–1937.
Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529–533.
Mehdi Moussaïd, Niriaska Perozo, Simon Garnier, Dirk Helbing, and Guy Theraulaz. 2010. The walking behaviour of pedestrian social groups and its impact on crowd dynamics. *PloS one* 5, 4 (2010), e10047.
S. R. Musse, C. R. Jung, A. Braun, and J. J. Junior. 2006. Simulating the Motion of Virtual Agents Based on Examples. In *ACM/EG Symposium on Computer Animation, Short Papers*. Vienna, Austria.
Andreas Panayiotou, Theodoros Kyriakou, Marilena Lemonari, Yiorgos Chrysanthou, and Panayiotis Charalambous. 2022. CCP: Configurable Crowd Profiles. In *ACM SIGGRAPH 2022 Conference Proceedings*. 1–10.
Sebastien Paris, Julien Pettre, and Stephane Donikian. 2007. Pedestrian Reactive Navigation for Crowd Simulation: a Predictive Approach. *Computer Graphics Forum* 26, 3 (2007), 665–674.
Nuria Pelechano, Jan M Allbeck, Mubbasir Kapadia, and Norman I Badler. 2016. *Simulating heterogeneous crowds with interactive behaviors*. CRC Press.
Xue Bin Peng, Pieter Abbeel, Sergey Levine, and Michiel Van de Panne. 2018. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. *ACM Transactions On Graphics (TOG)* 37, 4 (2018), 1–14.
Xue Bin Peng, Glen Berseth, KangKang Yin, and Michiel Van De Panne. 2017. Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 41.
Julien Pettré, Jan Ondrej, Anne-Hélène Olivier, Armel Crétual, and Stéphane Donikian. 2009. Experiment-based Modeling, Simulation and Validation of Interactions between Virtual Walkers. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. 189–198.
Tim Salzmann, Boris Ivanovic, Punarjay Chakravarty, and Marco Pavone. 2020. Trajectron++: Dynamically-feasible trajectory forecasting with heterogeneous data. In *European Conference on Computer Vision*. Springer, 683–700.
Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2015. Prioritized experience replay. *arXiv preprint arXiv:1511.05952* (2015).
Florian Siebel and Wolfram Mauser. 2006. On the fundamental diagram of traffic flow. *SIAM J. Appl. Math.* 66, 4 (2006), 1150–1162.
Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.
Adrien Treuille, Yongjoon Lee, and Zoran Popović. 2007. Near-optimal Character Animation with Continuous Control. *ACM Trans. Graph.* 26, 3, Article 7 (July 2007). https://doi.org/10.1145/1276377.1276386
B. van Basten, S. Jansen, and I. Karamouzas. 2009. Exploiting motion capture to enhance avoidance behaviour in games. *Motion in Games* (2009), 29–40.
Hado van Hasselt, Arthur Guez, and David Silver. 2016. Deep Reinforcement Learning with Double Q-Learning.
He Wang, Jan Ondrej, and Carol O'Sullivan. 2017. Trending Paths: A New Semantic-Level Metric for Comparing Simulated and Real Crowd Data. *IEEE Transactions on Visualization and Computer Graphics* 23, 5 (May 2017), 1454–1464. https://doi.org/10.1109/TVCG.2016.2642963
Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. 2016. Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*. PMLR, 1995–2003.
Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3-4 (1992), 279–292.
D. Wolinski, S. J. Guy, A.-H. Olivier, M. Lin, D. Manocha, and J. Pettré. 2014. Parameter estimation and comparative evaluation of crowd simulations. *Computer Graphics Forum* 33, 2 (2014), 303–312. https://doi.org/10.1111/cgf.12328
Markus Wulfmeier, Peter Ondruska, and Ingmar Posner. 2015. Maximum entropy deep inverse reinforcement learning. *arXiv preprint arXiv:1507.04888* (2015).
M Zhao, W Cai, and SJ Turner. 2017. CLUST: Simulating Realistic Crowd Behaviour by Mining Pattern from Crowd Videos. In *Computer Graphics Forum*. Wiley Online Library.
M. Zhao and V. Saligrama. 2009. Anomaly detection with score functions based on nearest neighbor graphs. In *Advances in Neural Information Processing Systems*.
Brian D. Ziebart, Andrew Maas, J. Andrew Bagnell, and Anind K. Dey. 2008. Maximum Entropy Inverse Reinforcement Learning. In *Proc. AAAI*. 1433–1438.