May 2021

Learning Multi-Agent Navigation from Human Crowd Data

Foram Joshi
Clemson University, foram2494@gmail.com

Learning Multi-agent navigation from human crowd data

A Thesis
Presented to
the Graduate School of
Clemson University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science
Computer Science

---

by
Foram Joshi
May 2021

Accepted by:
Dr. Ioannis Karamouzas, Committee Chair
Dr. James Z. Wang
Dr. Victor B. Zordan

Abstract

The task of safely steering agents amidst static and dynamic obstacles has many applications in robotics, graphics, and traffic engineering. While decentralized solutions are essential for scalability and robustness, achieving globally efficient motions for the entire system of agents is equally important. In a traditional decentralized setting, each agent relies on an underlying local planning algorithm that takes as input a preferred velocity and the current state of the agent's neighborhood and then computes a new velocity for the next time-step that is collision-free and as close as possible to the preferred one. Typically, each agent promotes a goal-oriented preferred velocity, which can result in myopic behaviors as actions that are locally optimal for one agent is not necessarily optimal for the global system of agents. In this thesis, we explore a human-inspired approach for efficient multi-agent navigation that allows each agent to intelligently adapt its preferred velocity based on feedback from the environment. Using supervised learning, we investigate different egocentric representations of the local conditions that the agents face and train various deep neural network architectures on extensive collections of human trajectory datasets to learn corresponding life-like velocities. During simulation, we use the learned velocities as high-level, preferred velocities signals passed as input to the underlying local planning algorithm of the agents. We evaluate our proposed framework using two state-of-the-art local methods, the ORCA method, and the PowerLaw method.

Qualitative and quantitative results on a range of scenarios show that adapting the preferred velocity results in more time- and energy-efficient navigation policies, allowing agents to reach their destinations faster as compared to agents simulated with vanilla ORCA and PowerLaw.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Real-time goal-directed navigation of multiple agents has many applications in robotics, graphics, and traffic engineering. Whether Roombas are cleaning the floor, animated pedestrians walking through a virtual world, or robots delivering parts for packaging in Amazon warehouses, the agents should be able to sense their surroundings and react accordingly to avoid collisions while successfully executing their tasks. Despite not colliding, though, it is often important for the agents to navigate efficiently, for example, to save resources (i.e., battery life) or reach critical locations in a timely manner. However, conflicting constraints and the need to operate in dynamic environments make the problem very challenging. Besides, due to real-time requirements, each agent typically needs to compute its motion independently, with limited or no communication with the other agents. State-of-the-art decentralized techniques for multi-agent navigation can provide formal guarantees about the collision-free behavior of the agents and can be extended to account for motion and sensing uncertainty allowing implementation on actual robots [1, 9, 59, 63]. However, even though, such techniques generate locally efficient motions for each agent, the global behavior of the agents can be far from efficient. This is because actions that are optimal for

one agent are not necessarily optimal for the entire system of agents. Consequently, most of the existing decentralized planners are very conservative leading to inefficient agents that focus on not colliding and often forget about their tasks at hand.

In this project, we seek to develop a new, human-inspired approach for efficient mobile agent navigation. Humans know when they have to be polite and yield to others and when to take decisive actions, efficiently performing complex navigation tasks without collisions such as walking in densely packed environments. The goal of this project is to enable such behavior to mobile agents by taking advantage of the large amounts of human crowd data available today. Importantly, and as opposed to recent learning techniques for multi-robot navigation [15, 12], we will use human data to enhance the decision-making process of the agents rather than replace their underlying planning routines. Consequently, the proposed work lead to agents that exhibit efficient, human-like, behavior while still guaranteeing robust and collision-free navigation.

This project aims to improve the global efficiency of decentralized multiagent systems by improving the decision-making of individual agents based on data of real human interactions. Our work follows the traditional multi-agent navigation paradigm, where each agent navigates independently by running a continuous cycle of sensing and acting. At each cycle, the agent senses its nearby agents and computes a new collision-free velocity that is as close as possible to a preferred velocity indicating the agent's desired direction of motion and speed. While an extensive amount of work exists on finding locally optimal collision-free velocities, little effort has been put to adapt the input preferred velocity. Traditionally, the preferred velocity used is the unit vector pointing towards the agent's goal scaled by its preferred speed. However, opting to move fast towards the goal while ignoring the aggregate motion of its neighbors can lead to myopic behavior and decrease the efficiency of the overall

system in the long run. In this project, we hypothesize that the best way to teach agents what preferred velocities to take is by letting them learn these velocities from human crowds. Our overall approach is divided into two phases as shown in Figure 1.1. First, in an offline training phase, we train a deep learning model to learn preferred velocities from pedestrian datasets. Second, in the online simulation phase, where each agent queries the trained network to retrieve a preferred velocity, which is inputted into a local planning algorithm to get a collision-free velocity.

In the training phase, we use five publicly available pedestrian datasets which contain crowd interaction examples across a variety of real-world scenarios. We explore three state-descriptors, namely LiDAR, minimum predicted distance (MPD) [49], and a latent-space attention to encode the local conditions that the agent faces. The neural network architectures are trained to learn the instantaneous velocity taken by the human as a function of the local interaction conditions.

In the simulation phase, each agent in an iterative manner observes its local neighborhood and queries the trained neural network to find its new target velocity. The target velocity is not taken directly but rather is used as the preferred velocity in a collision-avoidance subroutine. We tested the applicability of our proposed approach on two popular algorithms for local collision avoidance, the ORCA [63] introduced in robotics and the Power-law [31]. We qualitatively and quantitatively compare the simulation results produced by vanilla collision-avoidance methods as well as the combined approach proposed above. Our results show that using learned velocities as the input preferred velocity improves the time efficiency of the global system of agents when compared to vanilla local planners. The primary focus of this thesis is as follows:

1. Investigate how the efficiency of the multi-agent system is impacted when each

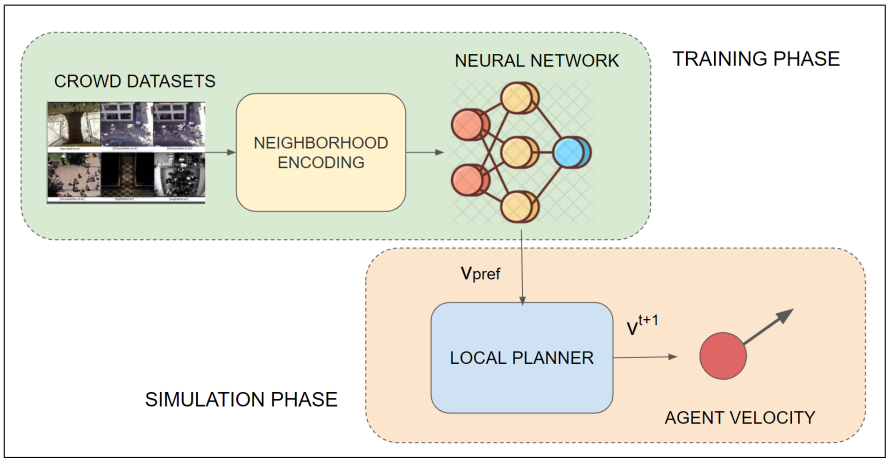

Figure 1.1: Overview of the proposed method. In the training phase, we compute local neighborhood representations of individual humans from real-world crowd datasets. We then train a neural network to learn to map these neighborhood encodings into velocities taken by the human at the next timestep. In the simulation phase, each agent observes its local neighborhood and queries the trained neural network to obtain a human-inspired velocity which is passed into the local planner as the preferred velocity. The local planner outputs a collision free velocity which the agent takes in the simulation.

agent selects its preferred velocity based on the state of its local neighborhood, compared to moving greedily towards their goal.

2. Investigate various state-descriptors in terms of their efficacy of representing the instantaneous as well as dynamic factors of the agent's local neighborhood.

3. Evaluate the efficacy of the proposed human-inspired framework for multi-agent navigation.

The rest of the thesis is organized as follows. In Chapter 2, we review highly relevant work on multi-agent navigation. In Chapter 3, we present our problem formulation for multi-agent navigation. We further elaborate on the pedestrian datasets that we used for training (3.1), and explain in detail our three proposed methods for capturing the per-agent local conditions, namely the LiDAR scan, the Miminum

Predicted Distance scan, and an attention-based latent encoding method (3.2). In Chapter 4, we provide qualitative results comparing simulation results using two collision-avoidance methods (4.1), and enumerate the quantitative metrics we used for evaluation and discuss how the different encoding-based approaches perform (4.2). Finally, in section 5, we discuss the results and plans about future work.

Chapter 2

Related Work

Our work focuses on the decentralized multi-agent navigation domain, where the aim is to steer a system of agents from their starting position to the goal position while avoiding collision with static and dynamic obstacles in the scene. In particular, our work follows a traditional decentralized paradigm where each agent follows a cycle of sensing and acting inside the environment on its own, without any communication from its neighbors, and the goal is to improve the global efficiency of the entire system of agents. Two general approaches for addressing the problem are *planning-based* and *learning-based*. Planning-based approaches are procedure-driven methods that can provide theoretical guarantees about navigation. In contrast, learning-based algorithms are data-driven methods that tend to generalize well to a large variety of scenarios, but unlike the planning methods, they do not guarantee collision-free behavior. Our work adopts a hybrid approach where we use machine learning to learn high-level navigation control signals from pedestrian data and combine it with a planning-based subroutine to produce collision-free motion.

2.1 Local Planning Algorithms for Collision Avoidance

Multi-agent motion planning algorithms aim to find a sequence of valid actions for each agent to move them from starting to goal state without colliding with any static obstacles or other agents present in the environment. Local planning algorithms select these actions based on the agent's local surroundings, typically based on feedback from their sensors. Existing local planning approaches that rely on social forces and rule-based techniques have been successfully applied to various multi-agent domains and have been shown to generate human-like collision avoidance behavior [25, 50, 52, 54]. Geometric local planners based on the concepts of velocity obstacles and time to collision [63, 31] are also widely applicable as they provide formal guarantees about the collision-free behavior of the agents and can be extended to account for motion and sensing uncertainty allowing implementation on actual robots [1, 26, 67, 64].

In this work, two popular local planning frameworks have been explored. Optimal Reciprocal Collision Avoidance (ORCA) [63] is a highly scalable geometric local planner that uses the concept of velocity obstacles and for provable collision-free navigation. One limitation of ORCA is that agents can behave very timidly and not exhibit urgency to reach their goal position, leading to high simulation times. The other collision avoidance framework we consider in this thesis is the PowerLaw [31, 33], which is a predictive force-based approach that calculates collision-free velocities using the time-to-collision with respect to neighboring agents. The PowerLaw's formulation is derived from human pedestrian data and is known to produce human-like navigation. We are treating these frameworks as "black-boxes" in our work, and in theory, they can be replaced with any other local planner that takes some information about

an agent's local neighborhood as well as a preferred velocity as input and computes a target velocity that avoids collision and is as close as possible to the preferred. The preferred velocity passed into the local planner is typically the direction vector pointing from the agent's current position towards the goal, scaled by the agent's preferred speed, hence encouraging agents to exhibit goal-oriented behavior. This often results in locally greedy actions such that each agent attempts to make maximum progress towards its goal while ignoring its neighbors. Such actions, while locally optimal, may lead to long-term consequences that negatively impact the system's efficiency at a global level. In this thesis, we propose to address this issue by learning preferred velocities from human pedestrian data.

2.2 Machine Learning for Multiagent Navigation

Numerous recent works [8, 12, 15, 43, 70, 13] have used reinforcement learning, where the task is formulated as a multi-agent Markov Decision Process, and the agent interacts with the environment to learn a policy to map its observable state into actions that maximize the rewards. There have also been work that focuses on planning under uncertainty with Bayesian approaches [34, 2], and learning through imitation learning like [72, 60]. Inverse reinforcement learning approaches have also been explored [37, 38, 27] to predict human trajectories and promote socially compliant navigation in virtual agents. Behavior cloning approaches, that rely on expert demonstrations to train artificial agents have previously attempted to learn policies by mapping from state to actions [6, 48] in self-driving cars, as well as in multi-robot navigation [14, 44]. A recent work [71] has combined knowledge distillation [29] in neural networks and deep reinforcement learning to shape reward functions from human trajectory data. In [5], the authors train multiple machine learning policies by

dividing the possible space of states an agent can encounter into steering contexts, which they define as collections of situations selected for their qualitative similarity. In [40], the authors track pedestrians in crowd videos to create example scenarios queried by agents during simulation to generate human-like trajectories and interactions. Long et al. [44] uses deep learning to train a collision-avoidance network on a synthetic dataset containing examples of collision-free velocities generated by ORCA in randomly generated scenarios. In contrast to the aforementioned approaches, we aim to learn high-level actions directly from human crowd pedestrian data through supervised learning and combine them with a geometric collision avoidance framework. This allows for human-inspired multi-agent navigation while still providing guarantees about collision-free motion.

Our work also investigates how different state-space representation is conducive for efficient training. Various representations have been proposed in the literature, such as static representations of the neighborhood like distance-based LiDAR scans, occupancy grids, RGB-D images, or anticipatory metrics like the minimum predicted distance (MPD) [49]. Many recent deep learning approaches learn directly from raw neighborhood data, such as a list of neighbor positions and velocities relative to the agent [7, 44]. Communication between agents for collaborative navigation is also a heavily researched area, where the states of each agent can also be influenced by communication signals received from nearby agents [28].

Chapter 3

Methods

3.1 Problem Formulation

We are given n agents $A_1...A_n$ moving in a scene. Each agent A_i enters the environment at a specific time t_i^0 as a and has a specific goal position g_i to reach. In this work, we assume that all agents are holonomic discs with a fixed radius r, and have a commonly preferred speed s. At time-step t in the environment, each active agent A_i has a global position $\mathbf{p_i^t}$ and a velocity $\mathbf{v_i^t}$, and can sense the static obstacles and neighboring agents within a given sensing radius r. After observing its local neighborhood $\mathcal{N}(A_i^t)$, the agent must take a velocity $\mathbf{v_i^{(t+1)}}$ at the next time-step that (a) makes progress towards its goal, (b) avoids collisions with the neighboring agents and obstacles in the scene. This velocity $\mathbf{v_i^{(t+1)}}$ is typically selected by a collision avoidance algorithm that takes as input the positions and relative velocities of the agent's neighbors and a preferred velocity $\mathbf{v_{pref}}$, and outputs a collision-free velocity that is closest to $\mathbf{v_{pref}}$. Traditionally, $\mathbf{v_{pref}}$ is treated as the vector that points towards the agent's goal, scaled by the agent's preferred speed. Here we propose to instead use as $\mathbf{v_{pref}}$, the velocity that a human would have taken if facing the same

Dataset	***Description***	***Pedestrian count***	***Frames (10fps)***
Oneway01 and 02 [30]	Pedestrians walking down a hallway (unidirectional flow)	46	2357
Zara01 and 02 [40]	Pedestrians interactions at a commercial street. (bidirectional flow)	352	58257
Students [40]	Pedestrians interactions at a college campus. (bidirectional flow)	434	70306

Table 3.1: Pedestrian datasets used for training

interaction conditions in the real world. To do this, we use supervised learning to train neural networks on human trajectory data and learn a mapping from the local conditions of the human to their preferred velocity, i.e. $E(\mathcal{N}(A_i^t)) \mapsto \mathbf{v_i^{(t+1)}}$. During simulation, the agent observes it's local conditions and queries the trained neural network to retrieve a human-preferred velocity. Note that the agent does not take the learned velocity directly, and instead inputs it as the preferred velocity $\mathbf{v_{pref}}$ into it's underlying local planning algorithm which then returns a collision-free velocity that is maximally aligned to the inputted preferred velocity.

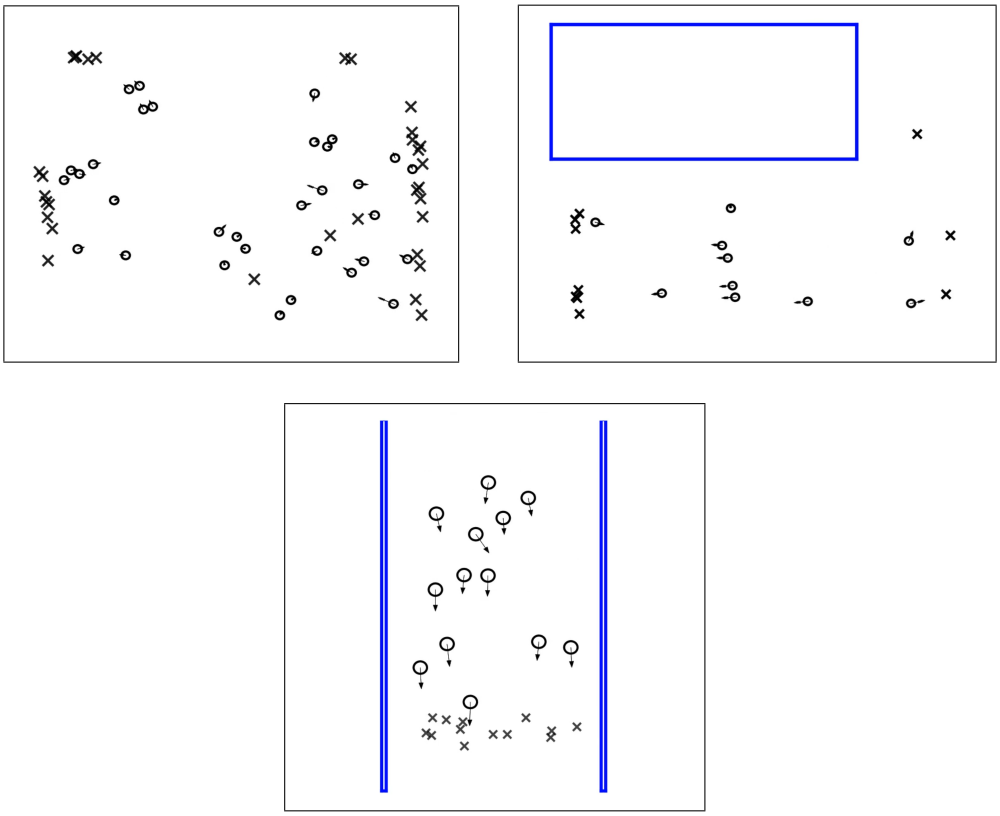

Figure 3.1: The datasets used for training, top left - `Students`, top right - `Zara`, bottom - `Oneway`

3.2 Learning Human-Like Velocities

We seek to learn a function that maps an agent's local conditions to the velocity that it takes. In polar coordinates, each velocity $\mathbf{v_i^t}$ can be represented as a tuple of speed (magnitude) s_i^t and deviation θ_i^t. Note that the deviation θ_i^t refers to the angle of steering that an agent makes with respect to its goal direction. Since our eventual goal is to use human-preferred velocities as preferred-velocity inputs into collision avoidance subroutines, we only learn the human data's deviation values. The choice

of only learning the deviation is based on the assumption that agents will always want to move at their preferred speeds and that human speeds can be noisy, dependent on social, external, or other unobserved factors. Hence, we are learning the mapping $H : E(\mathcal{N}(A_i^t)) \mapsto \theta_i^{(t+1)}$ using a deep neural network, where E is an encoding of the local neighborhood of the agent. In Appendix B, we show results from our experiments by learning human preferred velocities, that includes both speed and magnitude.

This chapter discusses three different strategies for the encoding function E and corresponding network architectures H that can learn human-preferred deviations from raw pedestrian data. Once the neural network H is trained, it can collaborate with vanilla collision avoidance methods to produce efficient collision-free velocities. In particular, our framework supports any local collision-avoidance planning algorithm $P(\mathcal{N}(A_i^t), \mathbf{v_{pref}^t}) \mapsto \mathbf{v_i^{t+1}}$, that can compute a collision-free velocity v^{t+1} by accepting the agent's local neighborhood $\mathcal{N}(A_i^t)$ and preferred velocity $\mathbf{v_{pref}^t}$ as input. Typically, $\mathbf{v_{pref}^t}$ is treated as the unit vector from the agent's current position to the goal position scaled by the preferred speed s. Still, we hypothesize that learning preferred velocities from human beings will help the agents react according to their neighborhood's dynamic features and take actions that optimize their trajectories and improve the global system's efficiency.

3.2.1 Distance Based Scan - LiDAR Scan

This is a static approach to encode an agent's neighborhood that considers the space that is not occluded by other agents or static obstacles in the field of view of the agent. Note that the scan is performed in the local neighborhood representation $\mathcal{N}(A_i)$, such that it generalizes for all agents in our datasets. The lidar-based neighborhood encoding E_{lidar} : [-θ, θ] $\mapsto$ [0,r], which denotes a discrete mapping from

angular samples to the distance to the closest obstacle (static or dynamic) computed using the LiDAR approach. We ignore any obstacles beyond the agent's sensing radius r. A major limitation of the LiDAR scan is its static nature, preventing from capturing the neighboring agents' velocity information. To address this, we stack previous frames together and reformulate $E^t_{hist\_lidar} = (E^t_{lidar}, ..., E^{t-k}_{lidar}) \in \mathbb{R}^{k \times (f/res)}$, where k is the length of frame history, f is the field of view and res is the angle resolution of the LiDAR.

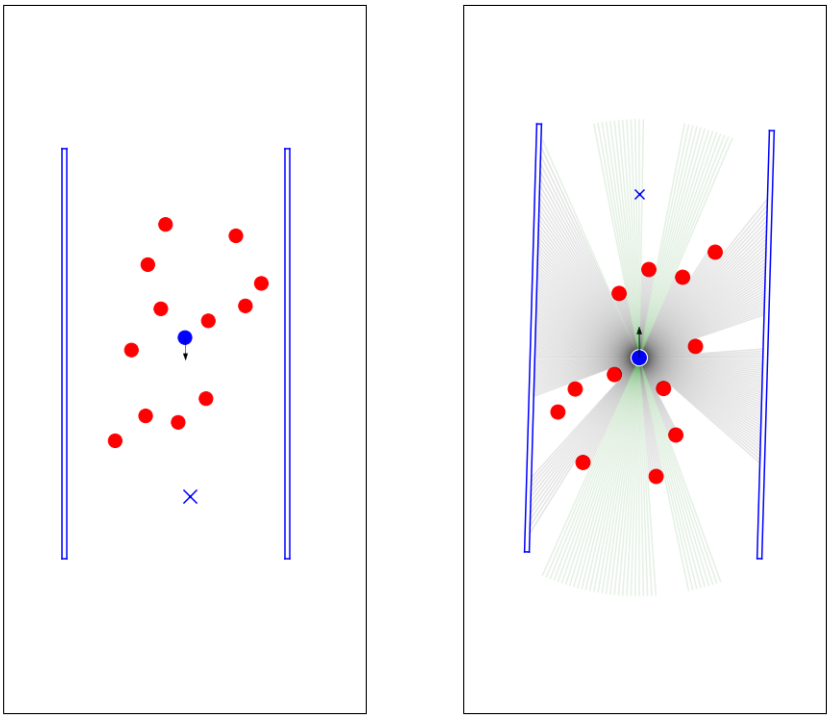

Figure 3.2: The left figure shows a frame from the `oneway` scene. The blue 'X' sign denotes the goal position of the blue agent. The red agents are other agents moving in the scene (their velocities are not shown for simplicity). On right, the lidar-scan in the local coordinate frame of the ego (blue) agent is shown. The new space is centered at zero with the Y-axis pointing towards the goal.

3.2.2 Minimum Predicted Distance (MPD)

Unlike LiDAR, which gives a static representation of the agent's neighborhood, MPD [46, 32, 49] is an anticipatory metric that considers the dynamic nature of the neighborhood by accounting for the relative displacements and velocities between the agent and its nearby neighbors. We can capture both how the local density changes around the agent based on positions and how it is likely to change in the near future from relative velocities. Similar to the distance-based map, to capture how MPD varies spatially, we use an egocentric representation where now the neighborhood encoding, $E_{MPD} : [-\theta, \theta] \mapsto [0, r]$, denotes a discrete mapping between evenly-spaced orientations along with the agent's FOV and MPD values with respect to the closest object along with each orientation that can be found using ray tracing. Then, we compute the MPD as the closest distance between the two interacting parties assuming a linear extrapolation of their current velocities. Formally, given an agent A, the MPD to its n^{th} neighbor is computed as:

$$\min_{\tau \geq 0} \|(\mathbf{p} - \mathbf{p_n}) + (\mathbf{v} - \mathbf{v_n})\tau\|)$$

where p and v denote the current position and velocities of the agent A, and p_n and v_n denote the position and goal velocity of the neighbor, respectively. τ is the time of closest approach between A and the neighbor.

The MPD-based formulation allows us to capture the agent's local conditions using just the current state observation. With this information, MPD is still inherently anticipatory in nature, accounting for the agents' expected future interactions through their relative positions and velocities.

Figure 3.3, explains the idea of how MPD values are captured from the point of view of the blue agent. We see that there are four neighboring red agents around the blue

agent. The length of black and green lines at a particular angle represents the agent's MPD value with respect to the closest agent along that ray. The green lines show no agent/obstacle, and the MPD value is set to the sensing radius, while the black lines indicate that the agent senses an impending collision. Each ray signifies the MPD value with respect to the closest neighbor along that ray. The relative velocity and relative positions of the ego-agent with respect to agent 1 suggest a close encounter in the near future. So, the MPD value with respect to this neighbor is quite low, denoting the low distance to the closest approach with respect to this neighbor (a zero value will suggest that a linear extrapolation of their current velocity will result in a collision). Agent 2 is moving parallel to the ego-agent in the opposite direction, and there is no sign of a collision. The MPD value with respect to agent 2, therefore, is just the distance between them when they have the same x-coordinate. Note that for agent 3, the time of closest approach τ is negative, signifying that the nearest approach would have happened in the past. For this case, we assume that the minimum predicted distance is the current euclidean distance between the two since they will diverge in the future. Finally, since the MPD for agent 4 is greater than the agent's sensing radius, this agent's reading is not included in the scan.

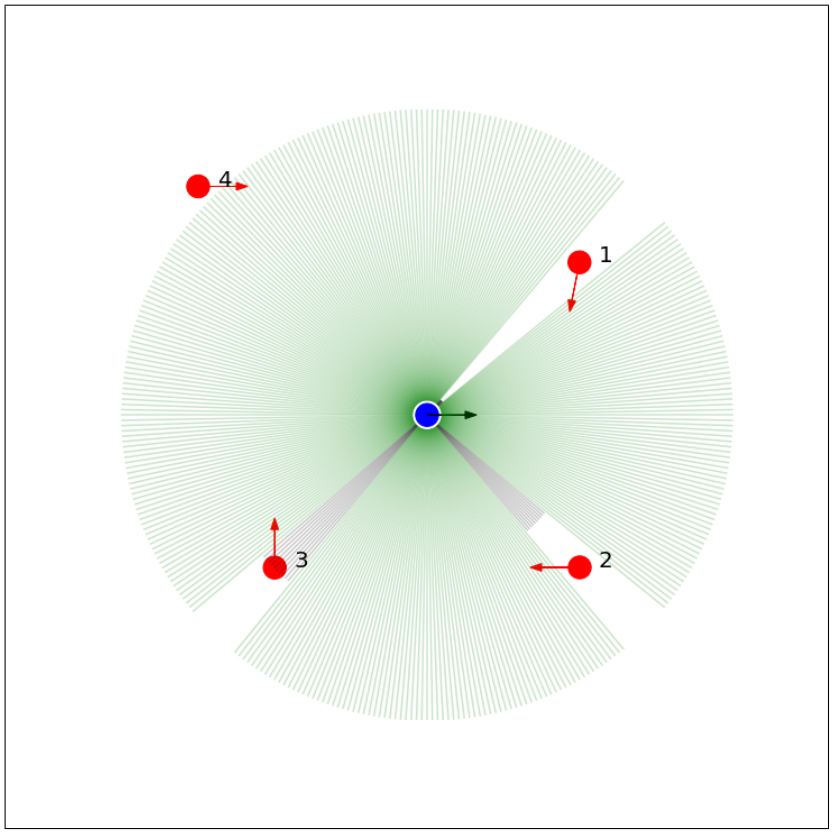

Figure 3.3: MPD Scan. Notice how the MPD value is low for agent 1 which seems to be headed towards a collision with the ego-agent in the near future. On the other hand, since the blue agent is diverging from agent 3, the distance of closest approach is their distance at the current frame. For agent 2, the MPD value is the distance between the two when they pass each other in the future.

3.2.3 Attention-based Latent Representation

Unlike LiDAR and MPD, where we generate a spatial or anticipatory embedding of the agent's neighborhood, this descriptor represents the neighborhood encoding E as a generic neural network, where the input are the states from a fixed number of nearby neighbors and the output are the latent variable ϕ (fixed-size vector) which

describes local crowd conditions from the perspective of the agent.

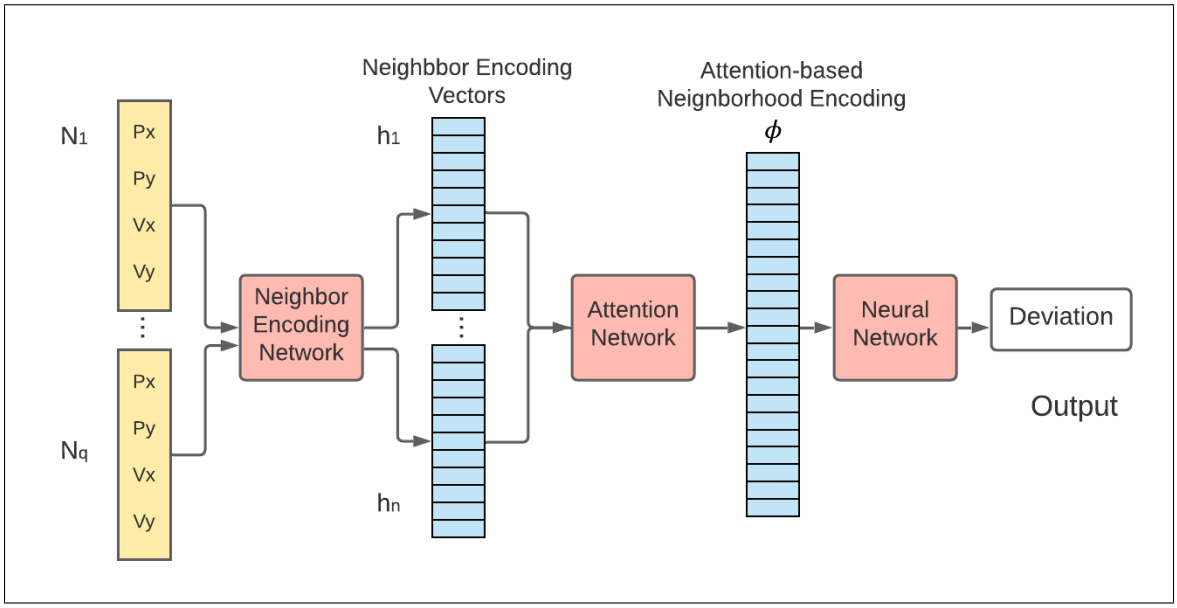

Figure 3.4: Overview of the attention-based encoding.

For a given agent A_i, we obtain its neighborhood $\mathcal{N}(A_i) \in \mathbb{R}^{q\times 4}$ that contains the state information (p_x, p_y, v_x, v_y) of each of the q nearest neighbors expresses in the agent's local coordinate system. We then process each neighbor with a neighbor encoding network J that maps each neighbor to a fixed-length embedding space. We aim to learn a neural attention score for each neighbor from these embedding vectors. The basic principle of attention is to find importance scores for each item in a list of *keys* depending on a *query*. In our case, the list of *keys* are the neighbor embeddings $[h_0...h_q]$, and the query is an order-invariant representation of the neighborhood, such as the mean $\frac{1}{q}\sum_j h_j$. We concatenate the order-invariant mean vector with individual neighbor embeddings and learn a scalar attention score for each neighbor j, that is $a_j = A(\Big[\ h_j \odot \frac{1}{q}\sum_j h_j)\ \Big])$. Finally, we normalize the attention scores

(using a softmax operation) and obtain an order-invariant latent embedding vector ϕ for the neighborhood as a weighted sum of all neighbor encodings. A similar type of architecture has been used recently with great success to train visual recognition systems as agents move around scenes and robot-crowd interaction scenarios [7].

Algorithm 1: Attention-Based Encoding

---

input : Agent neighborhood in local coordinate space $\mathcal{N}(A_i) \in \mathbb{R}^{q \times 4}$

output: A fixed length neighbor encoding vector ϕ

1. Map to a fixed length embedding for each neighbor j in $\mathcal{N}(A_i)$:

$$h_j = J(\mathcal{N}(A_i)_k), \text{ where } J \text{ is a neural network.}$$

2. Calculate attention scores for each neighbor:

$$a_j = A(\left[h_j \odot \frac{1}{q}\sum_j h_j)\right])$$

where $\odot$ is the concatenation operator and A is a feed forward network;

3. Normalize the attention scores for each neighbor

$$a_j = softmax(a_j \forall j)$$

4. Compute neighborhood encoding: $\phi = \sum_j a_j h_j$

3.3 Training

3.3.1 Data Preprocessing

We used five pedestrian datasets (Table 3.1) to train our neural network, namely `oneway01, oneway02` , `zara01, zara02,` and `students` . Each dataset contains a list of timestamps, instantaneous positions, and velocities of individuals in the scene. Since some of our latter processing steps (the LiDAR scan, for example) depends on frame sequences, we resample all datasets to have a fixed timestep difference of 0.1 seconds. All datasets also contain static obstacles represented as axis-aligned bounding boxes. To extract dynamic information from the data, we simulate the data and reconstruct the scene by processing each row chronologically. We treat the last known position of a pedestrian as his goal, and each pedestrian is removed from the simulation once reaches the goal. At each timestep, we retrieve all the active pedestrians' global positions and then construct a local view of each agent. The local coordinate frame's origin is the current position of the ego-agent, and the Y-axis points toward the agent's goal. We transform the entire scene (neighboring agents and static obstacles) to this local system and obtain the raw neighborhood information $\mathcal{N}(A_i)$ for the ego-agent. This process is repeated for each active agent in each timestep. There are many irregular movements in the crowd datasets, especially `zara02` and `students`. Some pedestrians stand still for large amounts of time and or move aimlessly around the scene (without trying to reach their final positions). To clean these examples, we prune examples where the pedestrian's instantaneous speed is below a certain threshold (0.25 m/s) and if their deviation angle is outside a certain range ($\pm$ 1.25 radians).

3.3.2 Implementation Details

The training has been conducted in a supervised fashion using scenarios containing `oneway01, oneway02, zara01, zara02,` and `students` datasets which cumulatively consists of 130,000 training examples. The dataset is divided into a train-test ratio of 80:20. For better generalization and inclusion of all training examples during training, we created stratified mini-batches to ensure that the sampled mini-batch contained an equal number of training examples from each scenario in the combined dataset.

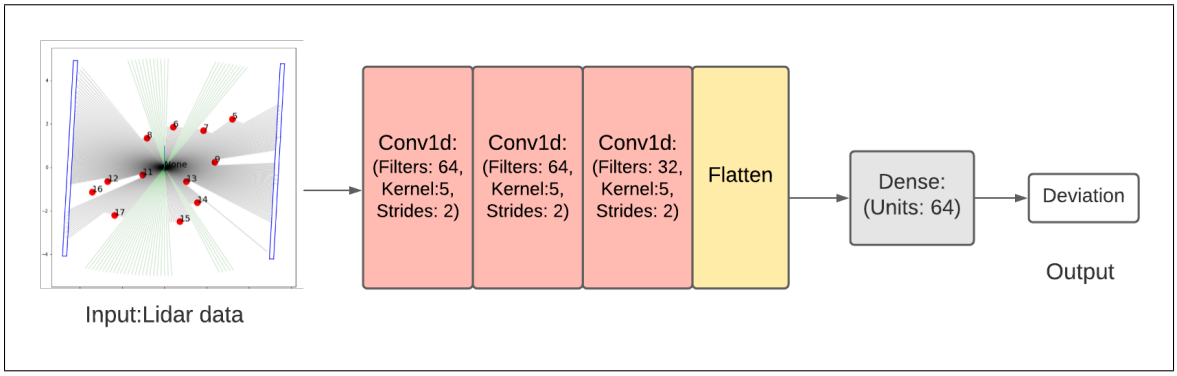

Figure 3.5: Network Architecture for LiDAR-based training

1. **LiDAR**:

   The sensing radius r for calculating the LiDAR scan was kept as 5m. The field of view f angle set to 240°, and the angular resolution is set to 1. We use a frame history length of 4 spaced with 0.1 second intervals to add dynamic information to the scans. The input passed to the neural network is the LiDAR values with historical scans. We then give the input to three 1D-Convolutional layers, each consisting of 64 filters, kernel size of 5, and strides as 2 with activation function as the rectified linear unit (ReLU). The third convolution layer's output

is then flattened and fed through one fully connected layer containing 64 units. The final dense layer contains one neuron optimized to predict the deviation. We trained this network with mini-batches of 64 examples using the Adaptive Moment Estimation (ADAM) algorithm with a learning rate of 0.0001. We used early stopping to prevent over-fitting when the model performed poorly on a test dataset. We used dropout on every layer to limit over-fitting, and a drop rate of 0.5 resulted in the best generalization of the test data.

2. **MPD**:

   The network architecture and training method is same as LiDAR. The only difference is that we do not use historic frames for MPD.

3. **Latent-Attention**:

   The neighbor encoding network H is a two-layer feed-forward neural network, each of 64 units and activation function as the hyperbolic tangent. The attention network A is a linear layer that outputs unnormalized attention scores. The final dense layer accepts the attention-encoded features as input and predicts the deviation values. Like the other methods, we trained this network using ADAM Optimizer as well with a learning rate of $5e^{-4}$.

The Figure 3.6 shows the training and testing plots of the 3 methods. The test losses for each method reaches close to 10^{-3} which attests that our neighborhood descriptors have a strong correlation with the deviations chosen by the humans.

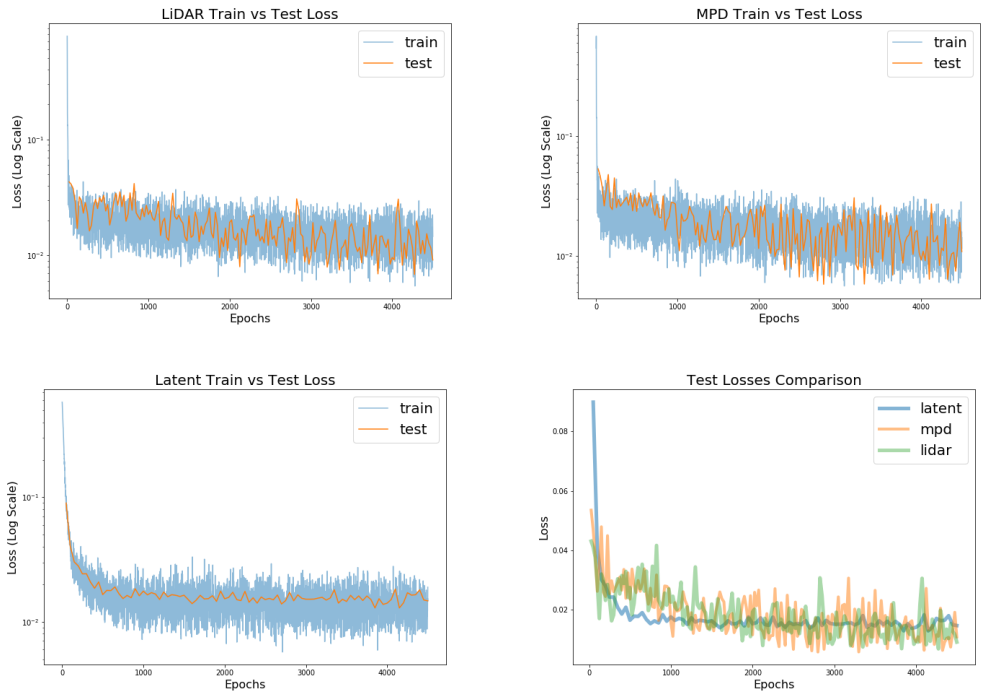

Figure 3.6: Train vs Test losses for the LiDAR, MPD, and Latent approaches. It seems that both MPD and LiDAR achieve lower test losses than the Latent method, although the Latent method seems less noisy compared to the other two.

3.3.3 Model Interpretability

Before deploying any machine learning model, especially highly non-linear structures like neural networks, it is crucial to understand the fairness and the explainability of the network. This section presents a simple gradient-based optimization algorithm to interpret our models and help us understand how the network is making decisions. The algorithm will input a target deviation angle, and the objective of this algorithm is to create a saliency map [57, 45] of LiDAR values that will make the trained network predict the input deviation angle. In other words, the algorithm

tries to create an inverse mapping from the deviation to the LiDAR values.

Algorithm 2: Towards Model Interpretability

input : A trained neural network $\Psi : \mathbb{X} \in \mathbb{R}^{FOV} \mapsto \theta$ that maps from a radial input (MPD or LiDAR) to human-preferred deviation,
A query deviation angle θ^*

output: A saliency map $\mathbb{X}^* \in \mathbb{R}^{FOV}$ $s.t.$ $\Psi(\mathbb{X}^*) \rightarrow \theta^*$

Randomly initialize $\mathbb{X}^*$,

Initialize $\sigma = 0.1, l_2 = 10^{-3}, lr = 10^{-1}$

for *k steps* **do**

Forward pass through Ψ and obtain the current prediction $\hat{\theta} = \Psi(\mathbb{X}^*)$;

Randomly perturb the target deviation θ^* with mean centered gaussian noise. $\dot{\theta} = \theta^* + \mathcal{N}(0,\ \sigma)$;

Calculate loss $\mathcal{L} = (y - y*)^2 + l_2 \left\|\mathbb{X}^*\right\|$;

Calculate gradients of loss $\mathcal{L}$ with respect to $\mathbb{X}^*.\nabla = \dfrac{\delta \mathcal{L}}{\delta \mathbb{X}^*}$;

Optimize $\mathbb{X}^*$ with one step of gradient descent. $\mathbb{X}^* = \mathbb{X}^* - lr * \nabla$

end

The optimization algorithm is presented in Algorithm 2. Given a neural network Ψ and a query deviation angle θ^*, we aim to find a radial input $\mathbb{X}$ such that $\Psi(\mathbb{X})$ is very close to θ^*. At each step, we aim to reduce the loss between the queried deviation angle and the network's output with respect to our current best guess for $\mathbb{X}$. We add an L2 normalization loss to the radial scan to penalize high values in the scan. Finally, we compute the gradients of the total loss with respect to $\mathbb{X}$ and perform gradient descent. We also recommend using Gaussian filtering for smoothing

the output scans. Alternatively, instead of optimizing for 240 rays, one could optimize for a lower number (say, 24) and then up-sample to retrieve the entire field-of-view.

We have trained the network with one frame LiDAR data on pedestrian datasets (`oneway01, oneway02, zara01, zara02 and students`). We then capture an importance map with each ray of LiDAR scan for a range of deviations. In Figure 3.7 the black lines represent if the saliency map expects an obstacle along that ray, while green lines represent little or no obstruction. The blue arrow is the target deviation input into the optimization algorithm. To interpret the Figure 3.7, the aim here is that when an input of -1 radians is passed as deviation angle to the network, we want to generate a LiDAR scan that would make the network output the deviation angle as -1 radians. We see that for input with extreme deviation angle (e.g., -1 and 1), the network creates a scan representing obstacles/neighbors at the agent's front and opposite sides. We notice the agent tries to predict a deviation that is away from the approaching obstacles to avoid the collision. For inputting the deviation angle as 0, we would expect a scan with no obstructions, which is happening in the third row of the Fig 3.7. The network correctly returns a nearly empty scan. These experiments show that the neural network is learning useful patterns from the pedestrian datasets that align with human intuition.

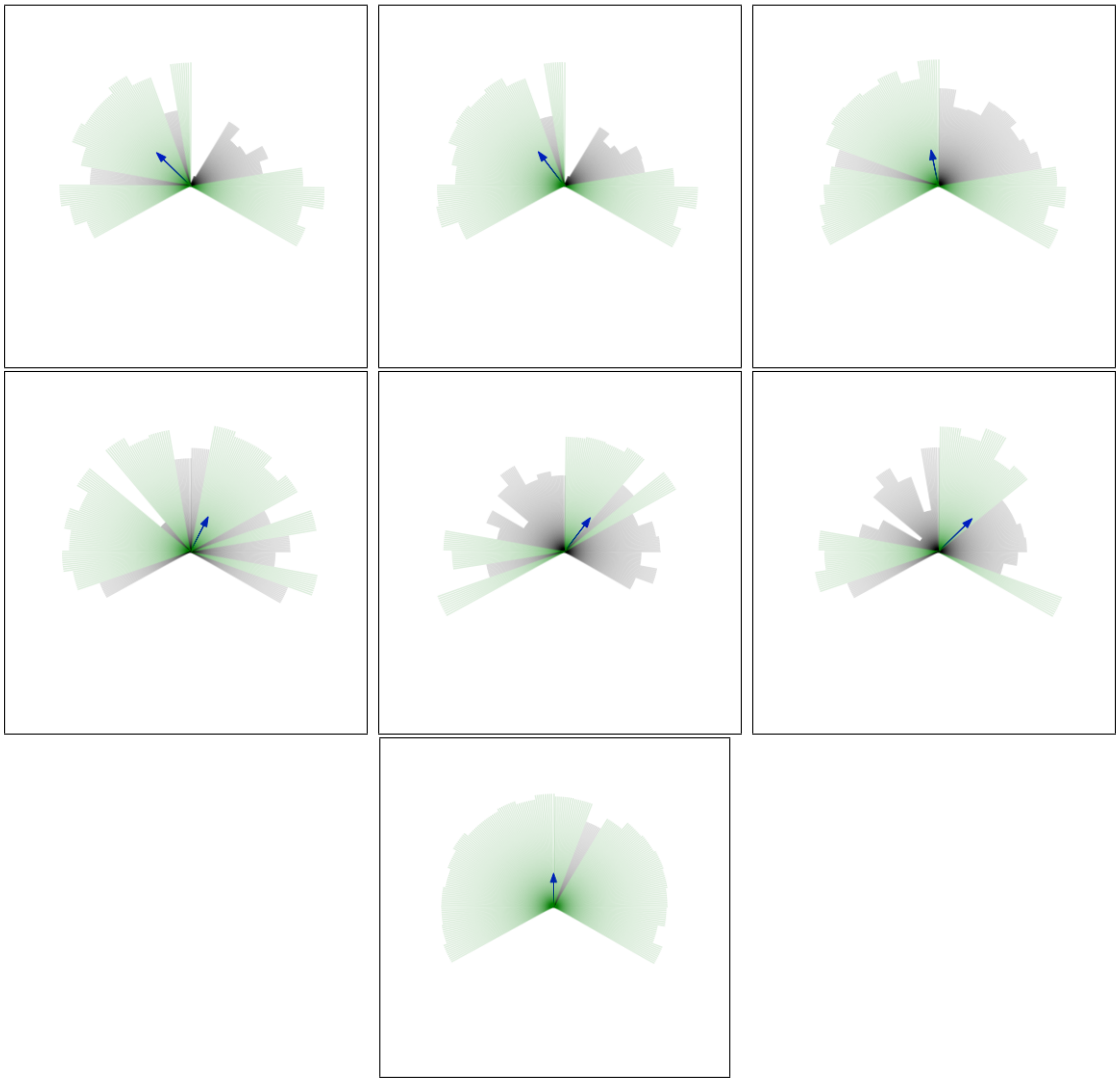

Figure 3.7: The 1^{st} row shows the LiDAR scans for input deviation -1, -0.5, -0.2 radians (from left to right). The 2^{nd} row shows the LiDAR scans for deviation input 0.2, 0.5, 1 radians. The 3^{rd} row shows the LiDAR scan for input deviation of 0 radians. Note that the scans are produced with a constraint on its L2-norm, so it's biased towards producing the minimum perturbations (i.e. black rays) possible.

3.4 Simulation

In 3.3 we elaborate on how we use supervised learning on pedestrian datasets to train a neural network that maps the local conditions of a pedestrian to the deviation (from the goal vector) that he took. We now seek to integrate this neural network to steer virtual agents towards their goals in multi-agent navigation scenarios. However, using the human-preferred velocities directly does not guarantee collision-free navigation. Instead, we propose to use the learned velocities as the input preferred velocity $\mathbf{v_{pref}}$ to the agent's underlying collision avoidance subroutine $\mathbb{C}$.

Our simulation algorithm follows the typical multiagent navigation paradigm. At timestep $t = 0$, the scene is initialized with each disk-shaped agent A_i at their starting location with predefined goal positions $\mathbf{g_i}$. At each timestep, each agent observes it's local neighborhood and formulates the appropriate representation (LiDAR, MPD, or Latent). This representation is then processed by the trained neural network to output a human-preferred deviation θ_i^{t+1}. The agent's preferred velocity for the next timestep is defined as a tuple of it's preferred speed s and preferred deviation θ_i^{t+1}. Finally, the subroutine $\mathbb{C}$ inputs the preferred velocity $\mathbf{v_i^{t+1}}_{\mathbf{pref}}$ and the relative positions and velocities of the agent's nearest neighbors (typically within a predefined sensing radius) and computes a collision free velocity $\mathbf{v_i^{t+1}}$ closest to the input preferred velocity. The agent's location is then updated using standard Euler integration, i.e. $\mathbf{x_i^{(t+1)}} = \mathbf{x_i^t} + \mathbf{v_i^{t+1}}\Delta t$, where Δt is the simulation timestep. We serially update each agent's location following the above steps. Agents are removed from the scene when they reach within 0.1m distance from their goal.

Algorithm 3: Simulation Algorithm

input : A pre-trained neural network $\Psi : f(E(\mathcal{N}(A_i))) \mapsto \theta$

Collision avoidance algorithm $\mathbb{C}(\mathcal{N}(A_i),\ \mathbf{v_{pref}}) \mapsto \mathbf{v_i^{(t+1)}}$

Initial positions $(\mathbf{x_1^0}, ...\mathbf{x_N^0})$, goal positions $(\mathbf{g_1}...\mathbf{g_N})$, radius $(r_1, ...r_N)$, and preferred speeds $(s_1, ...s_N)$ of N agents.

Initialize time t = 0

Initialize list of agents A containing disks A_i of radius r_i at position $\mathbf{x_i}$

while $\|A\| \neq 0$ **do**

  for $i = 1 \ldots N$ **do**

    Create a local coordinate mapping function $\mathbb{L}$ that transforms from the global space into the agent's local space such that the origin is at $\mathbf{x_i^t}$ and Y-axis as $\dfrac{\mathbf{g_i} - \mathbf{x_i^t}}{\|\mathbf{g_i} - \mathbf{x_i^t}\|}$;

    Compute local state $\mathcal{N}(A_i)$ containing the relative position and velocities of neighboring agents within sensing r_{sense} in $\mathbb{L}$

    Compute observation $E(\mathcal{N}(A_i))$ using the encoding strategy E ;

    Query the neural network to get preferred deviation $\theta_i^{t+1} = \Psi(E(\mathcal{N}(A_i)))$;

    Preferred velocity $\mathbf{v_i^{t+1}}_{\mathbf{pref}} = [s_i, \theta_i^{t+1}]$ (in polar coordinates);

    Collision free velocity $\mathbf{v_i^{t+1}} = \mathbb{C}(\mathbf{v_{i\,pref}^t}, E(\mathcal{N}(A_i)))$;

    Convert $\mathbf{v_i^{t+1}}$ to global space with $\mathbb{L}^{-1}$

  end

  for $i = 1 \ldots N$ **do**

    Update each agent's global position $\mathbf{x_i^{t+\Delta t}} = \mathbf{x_i^t} + \mathbf{v_i^{t+1}}\Delta t$;

    Remove agent A_i from A if $\|\mathbf{x_i} - \mathbf{g_i}\| \leq 0.1$ m ;

  end

  $t := t + \Delta t$;

end

Chapter 4

Results

In this section, we evaluate the performance of our proposed framework (Figure 1.1) using the ORCA [Appendix C.1] and the Power Law [Appendix C.2] methods as the underlying local planners to compute the final collision-free velocity of each agent. . We consider the following four scenarios:

1. **Hallway**: Two groups of six agents each start at opposite sides of a narrow and short hallway and have to swap positions.

2. **Circle**: Twelve agents initialized on the perimeter of a circle trying to reach the opposite end of the circle.

3. **Two-groups at 90°**: Two groups of 15 agents each cross paths perpendicularly in an obstacle-free environment.

4. **Crowd**: 48 agents are places in a square region and have to reach randomly assigned goals.

The starting configurations of each scenario are shown in Figure 4. All simulations were produced using same test conditions across all methods. The preferred speed

for each agent is set to 1.3 m/s and the maximum speed to 1.5 m/s. The agent radius for the `circle` and `hallway` scenario is set to 0.25m and 0.3m, respectively, and for the `crowd` and `Two-groups at 90°`, it is set to 0.2m. Selecting the agent's radius to a larger value for the `circle` and `hallway` scenarios makes navigation tasks more difficult and allows us to assess the encoding methods' value. For ORCA and PowerLaw, the velocities are updated every 0.1 seconds and 0.02 seconds, respectively. The network is queried for preferred velocities every 0.1 seconds, consistent with the training conditions in the pedestrian datasets.

During simulation, every agent spawns in the virtual environment at its designated start position. At each step of the simulation, each agent observes neighboring agents' states in a 5m sensing radius around the agent and converts the neighborhood into the agent's local coordinate system. The neighborhood is then encoded with the corresponding method (LiDAR, MPD, or Latent-based) and then given as input into a trained neural network. The neural network outputs a preferred deviation based on the input neighborhood. We input this deviation and the agent's preferred speed into the underlying local planner as the next timestep's preferred velocity. The local planning algorithm considers the states of the agent's neighbors and static obstacles and outputs a collision-free velocity that is maximally aligned to the preferred velocity. The positions of all agents are updated according to the planner's velocity using standard Euler integration. We remove agents from the scene after they reach a 0.5m radius surrounding their goal positions. In our experiments, we use the Python versions of the official PowerLaw and ORCA libraries as implemented in [http://motion.cs.umn.edu/PowerLaw] and [https://gamma.cs.unc.edu/RVO2/], respectively.

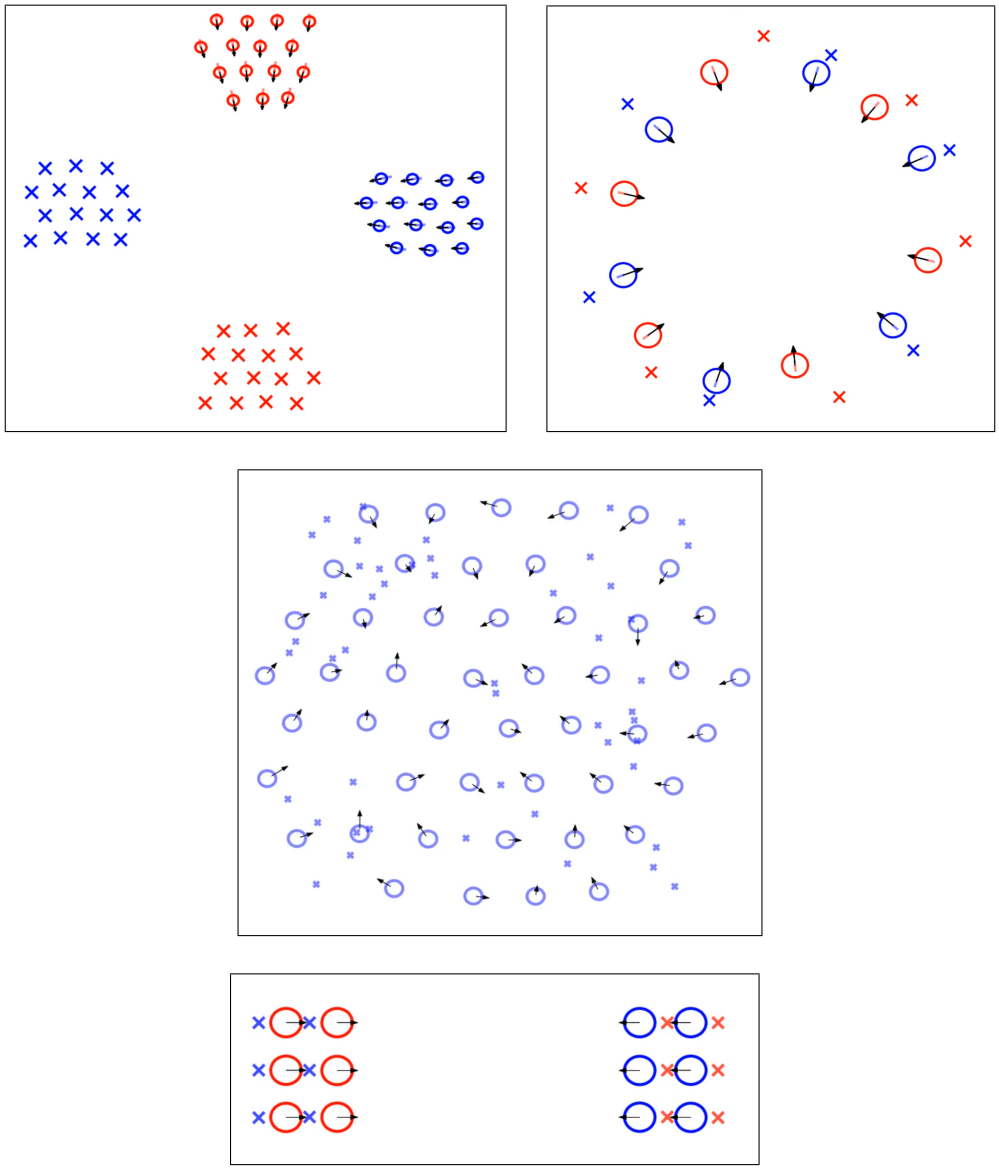

Figure 4.1: The above group of images show the test scenarios where we evaluate our approaches. Top Left: `Two-groups at 90°`, Top Right: `Circle`, Middle: `Crowd`, Bottom: `Hallway`. The images show the starting locations of the agents and the cross-hairs show their goal positions in the environment.

4.1 Evaluation Metrics

We use the following metrics to evaluate the performance of the simulations :

1. **Simulation Time**: This can be defined as the time taken by the entire simulation to complete, i.e., from the beginning until the last agent reaches its goal. Lower simulations times are better.

2. **Average Speed**: This denotes the average speed of each agent in the simulation. A speed equal to 1.3 m/s is preferable which is the preferred agent speed set in all of our simulations.

3. **Collision Rate**: This is a ratio of the total number of collisions to the total number of frames in the simulation. Two agents are said to be colliding at a particular timestep if their centers' distance is smaller than the sum of their radii.

4. **Interaction Overhead**: We use the interaction overhead metric as defined in [20]. For each agent, the interaction overhead calculates the additional time it took to reach its goal compared to the theoretical best time possible. In other words, this measures the extra time that the agent spent avoiding collisions with all other agents and static obstacles in the scene. An interaction overhead of 0 would mean that all the agents reached their goals in the best possible travel time. Formally, the interaction overhead for the set of agents A is defined as follows:

$$Interacion\ Overhead = TTime(A) - MinTTime(A)$$

where $TTime(A) = \mu TimeToGoal(A) + 3\sigma TimeToGoal(A)$ such that $TimeToGoal(A)$ are a set of travel completion times, and $\mu(.)$ and $\sigma(.)$ denote the mean and stan-

dard deviations respectively and $MinTTime(A) = \mu MinTimeToGoal(A) + 3\sigma MinTimeToGoal(A)$; such that $MinTimeToGoal(A)$ are a set of theoretical best travel times assuming that each agent followed a straight line path to the goal at their preferred speed.

5. **Energy Efficiency**: We use the energy efficiency metric as defined in [18, 21]. The energy efficiency of an agent at a particular frame is defined as the ratio of the progress made by the agent towards the goal to the total energy expended at that frame.

   The progress made by an agent A_i towards the goal at a particular time-step is calculated as follows:

$$Progress\ (A_i) = \mathbf{v} \cdot \frac{\mathbf{g} - \mathbf{p}}{\|\mathbf{g} - \mathbf{p}\|}$$

   where $\mathbf{g}, \mathbf{p}$ are the goal position and current position respectively and $\mathbf{v}$ is the instantaneous velocity of the agent.

   The energy expended by the agent at a particular timestep is calculated as:

$$Energy\ (A_i) = b + c\|\mathbf{v}\|^2,$$

   where the first term b denotes the power consumed by the agent due its processing and sensing operations, while the second term $c\|\mathbf{v}\|^2$ corresponds to the kinetic energy spent by the agent while moving. Following the work of [19], we have opted to pick b=1 and c=2.25.

   Finally the energy efficiency for the agent is:

$$Energy\ Efficiency\ (A_i) = \frac{Progress(A_i)}{Energy(A_i)}.$$

   The total energy efficiency of a simulation is calculated by taking the mean of the energy efficiencies of each agent.

4.2 Results and Analysis

Table 4.1 shows evaluation results of traditional ORCA and ORCA combined with LiDAR, MPD, and Attention encoding. In all scenarios, using learned preferred velocities performs better than naively picking the goal direction. In the `circle` and `hallway` scenario, our methods outperform vanilla ORCA by a sizeable margin. Not only do they complete the tasks faster with low interaction overhead, but they also lead to higher average speed and energy efficiency. This suggests that the agents can traverse towards their goal with less deviation compared to vanilla methods. This can be confirmed with the trajectory plots discussed later in the section. For the `Two-groups at 90°` scenario, the proposed methods only marginally outperform vanilla ORCA. This can be attributed to the fact that the scenario's dense interactions are not well presented in the datasets we chose for training. As future work, it will be interesting to see how training on more densely packed crowd data affects the network's performance in such interactions.

Table 4.2 shows results from our experiments with the PowerLaw. It can be noticed that for most scenarios, the difference in performance between the vanilla PowerLaw and modified versions is not as distinct as in ORCA. This is because PowerLaw is elaborated from crowd interaction data and hence simulations generally exhibit some of the efficiency found in human motion. Still, learning preferred velocities shows an improvement in almost all scenarios in comparison to the vanilla method. In the `Two-groups at 90°` scenario, which requires the densest agent interactions among our testing scenarios, we noticed that the our approaches can reduce the number of collisions than vanilla power law while still improving the energy efficiency and interaction overhead of the system.

While all encoding methods improve vanilla local planning approaches, the

MPD metric consistently generates energy-efficient and low overhead navigation. The anticipatory nature of MPD seems to generalize on a wide variety of scenarios containing both sparse and dense scenarios, different neighbor radius and relative velocities, and a variety of neighbor interaction scenarios absent in the original dataset.

Figures 4.2 and 4.3 compare the vanilla ORCA versus ORCA + MPD, in the `Hallway` scenario. In the ORCA simulation, the two groups of agents approach each other head-on and spend some time to find a collision-free velocity that makes progress towards the goal. Note that, for vanilla ORCA, we are perturbing the goal velocity with random gaussian noise (0 mean, 10^{-4} standard deviation) to avoid deadlock situations common in ORCA. On the other hand, when the preferred velocity is queried using our MPD network, the agents can pick preferred velocities by observing other agents' actions. This allows to avoid the deadlock scenario in the ORCA simulation and accelerates the simulation time.

In Figures 4.4 and 4.5, the simulations for vanilla ORCA and LiDAR-assisted ORCA is shown for the `circle` scenario. The latter completes the task in 7.6 seconds compared to the 11.7 seconds of the former method, with higher energy efficiency and lower interaction overhead. The agents' trajectories in the vanilla ORCA simulation show high curvatures, particularly in the middle of the simulation when multiple agents flock in the center searching for space. The trajectories produced by the hybrid approach are much smoother and decisive.

Figures 4.6 and 4.7, the simulations for vanilla PowerLaw and MPD+PowerLaw is shown for the `Two-groups at 90°` scenario (agent-radius set to 0.2m). In the vanilla method, it is noticeable that the two groups of agents start pushing each other towards the environment's bottom-right corner. Both parties greedily try to reach their destinations at each timestep without reacting to what the neighbors are trying to achieve. This behavior eventually results in multiple agents taking a longer

path to the goal and a simulation time of 19 seconds with an interaction overhead of 11.11 seconds. In contrast, the MPD-assisted preferred velocities show lesser deviation and complete the simulation in 16 seconds, with a lower interaction overhead of 6.326 seconds.

In Figures 4.9 and 4.8, we show the speed distribution plots with PowerLaw and ORCA variants. We ignore the first two and last two frames of each agent's trajectory, calculate their speed per frame, and plot a density distribution graph. The Y-axis denotes the density ($density = counts/(\sum(counts) * bin\_width)$, where counts are the number of data points in each bin), such that the area under the histogram integrates to 1. Although the improvement is less pronounced when using PowerLaw (due to its inherent capabilities of producing human-like motion) compared to ORCA, we can still see that LiDAR can improve the performance of vanilla PowerLaw. For ORCA-variants, we can see that all methods improve the vanilla implementation to produce higher speeds in agents. The vanilla ORCA plot shows a high density in low-speed regions (around 0.6 m/s) highlighted by the red circle in Figure 4.9 that is absent in the hybrid plots, suggesting that some agents stops or significantly slow down during their trajectory due to possible congestions.

In Figures 4.11 and 4.10, we compare the interaction overheads of each method on different scenarios. In most scenes, the adaptive preferred speed improves the performance of the vanilla methods (except LiDAR+ORCA in `Two-Groups at 90°`, and PowerLaw+Latent in `circle`). The MPD method seems to consistently produce the best gain in the ORCA scenarios. Although its performance gain is less pronounced in PowerLaw compared to ORCA, it still shows improvement over vanilla PowerLaw in all scenarios.

Scene	***Method***	***Sim Time***	***Average Speed***	***Interaction Overhead***	***Energy Efficiency***
Circle12	ORCA	11.7	0.971	8.53	0.302
	LiDAR	8.50	1.098	4.34	**0.342**
	MPD	**8.1**	**1.118**	**4.102**	**0.341**
	Latent	**8.1**	1.043	4.10	0.335
Hallway	ORCA	14.2	0.659	10.73	0.211
	LiDAR	8.7	1.111	3.837	0.349
	MPD	**7.8**	**1.16**	**1.24**	**0.366**
	Latent	8.1	1.144	2.546	0.356
Two-group at 90°	ORCA	14.3	**1.244**	3.5	0.368
	LiDAR	14.9	1.172	5.04	0.364
	MPD	**13.8**	1.237	3.45	**0.371**
	Latent	**13.7**	**1.247**	**3.18**	0.368
Crowd	ORCA	12	0.932	8.08	0.293
	LiDAR	9.6	1.064	3.88	0.335
	MPD	**9.4**	**1.104**	**3.02**	**0.347**
	Latent	**9.3**	**1.087**	**3.27**	**0.342**

Table 4.1: **ORCA** Experiment Results. Collision statistics are not shown since no collisions were observed.

Scene	*Method*	*Sim Time*	*Average Speed*	*Interaction Overhead*	*Energy Efficiency*
Circle12	PowerLaw	9.38	0.995	4.683	0.337
	LiDAR	**7.83**	**1.123**	3.06	**0.363**
	MPD	**7.71**	**1.05**	**2.710**	0.352
	Latent	9.65	0.991	6.328	0.333
Hallway	PowerLaw	12.38	0.72	6.952	0.257
	LiDAR	**8.75**	**1.032**	**3.651**	**0.347**
	MPD	**8.72**	0.987	3.448	0.343
	Latent	**8.7**	1.017	**3.567**	**0.347**
Two-group at 90°	PowerLaw	16.70	1.08	8.404	0.345
	LiDAR	**15.83**	**1.12**	**6.074**	**0.358**
	MPD	16.70	1.085	7.347	0.353
	Latent	16.17	1.089	6.601	0.350
Crowd	PowerLaw	10.65	0.874	5.067	0.324
	LiDAR	**9.84**	0.887	4.701	**0.329**
	MPD	9.96	0.877	4.83	0.321
	Latent	10	**0.918**	**4.324**	**0.332**

Table 4.2: **PowerLaw** Experiment Results. Collision statistics are not shown since no collisons were encountered.

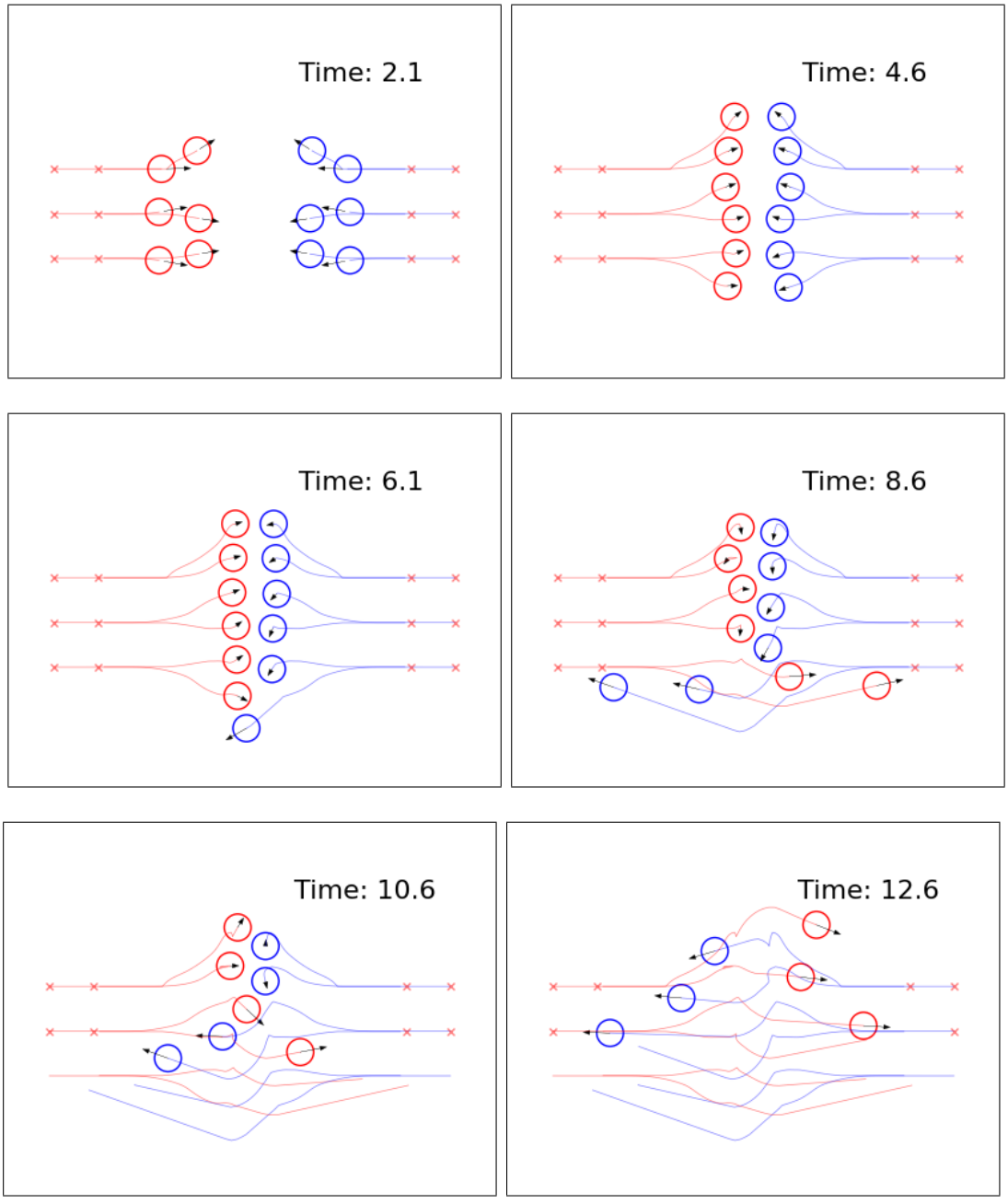

Figure 4.2: **Vanilla ORCA** simulation on the `Hallway` scenario. The agents reach a deadlock situation which increases the interaction overhead of the simulation.

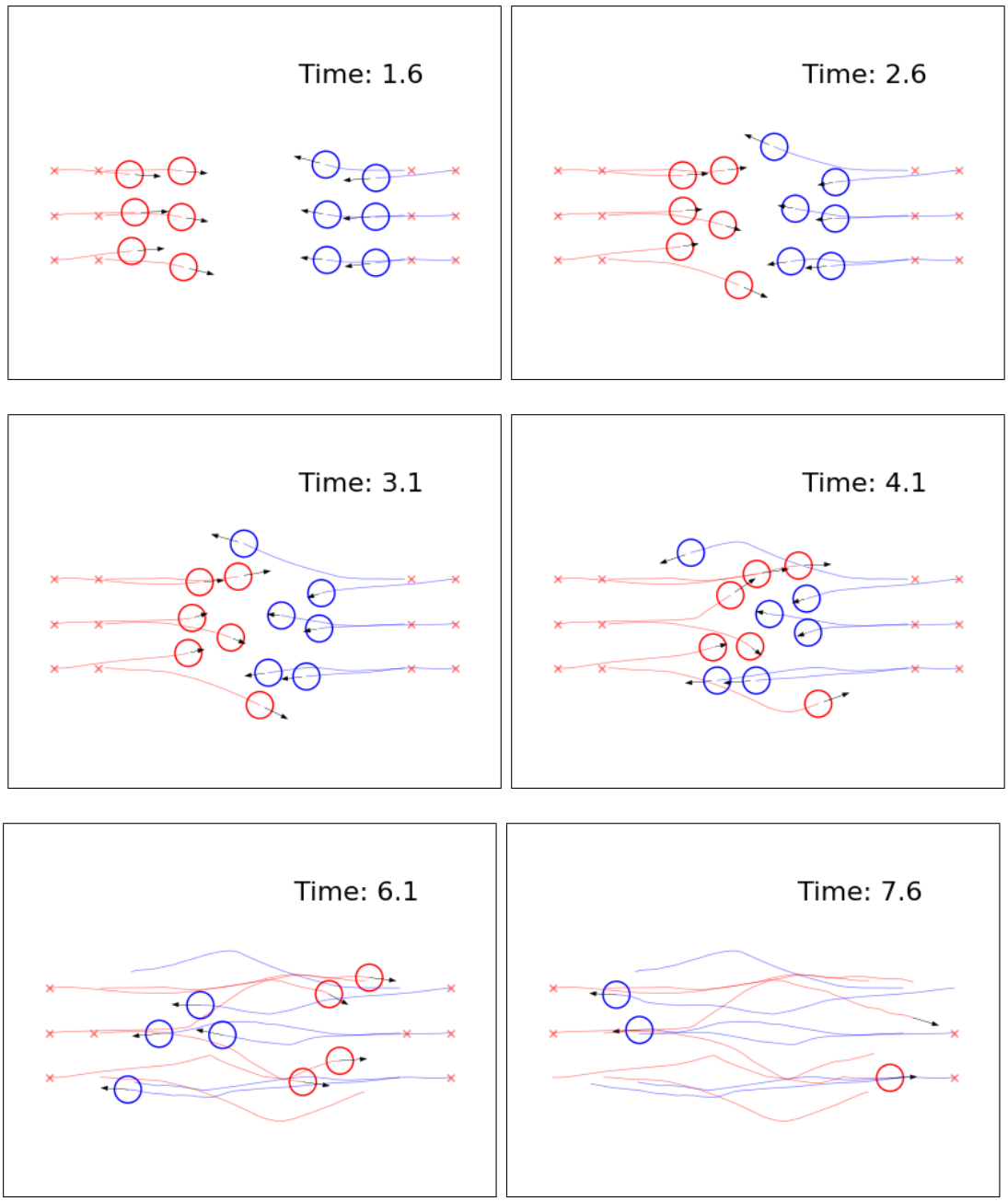

Figure 4.3: **ORCA** simulation in the `Hallway` scenario with preferred deviations obtained from **MPD encoding**. Notice in particular, how the agents in the wing react to each other's actions by moving into pockets of spaces created by each other's motion.

Figure 4.4: **Vanilla ORCA** simulation on 12 agents `Circle` scenario with agent-radius set to 0.25m. The last three agents exhibit high rotation and end up travelling a longer and convoluted distances to the goal with extreme curvatures.

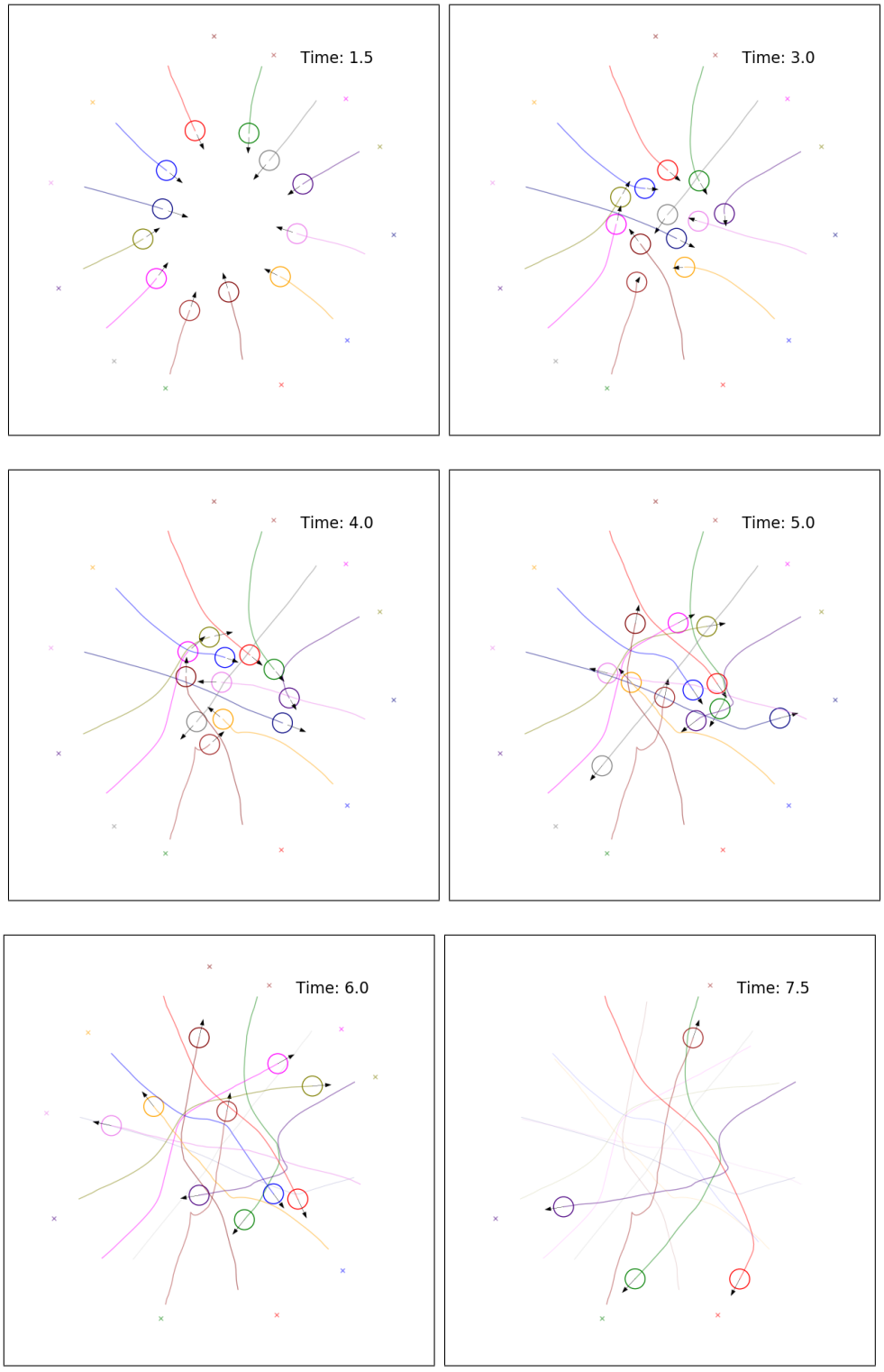

Figure 4.5: **ORCA** simulation with preferred velocities obtained from **MPD** scans on `Circle` scenario (agent radius = 0.25m). Unlike 4.4, the agents reach the goal faster with smoother trajectories.

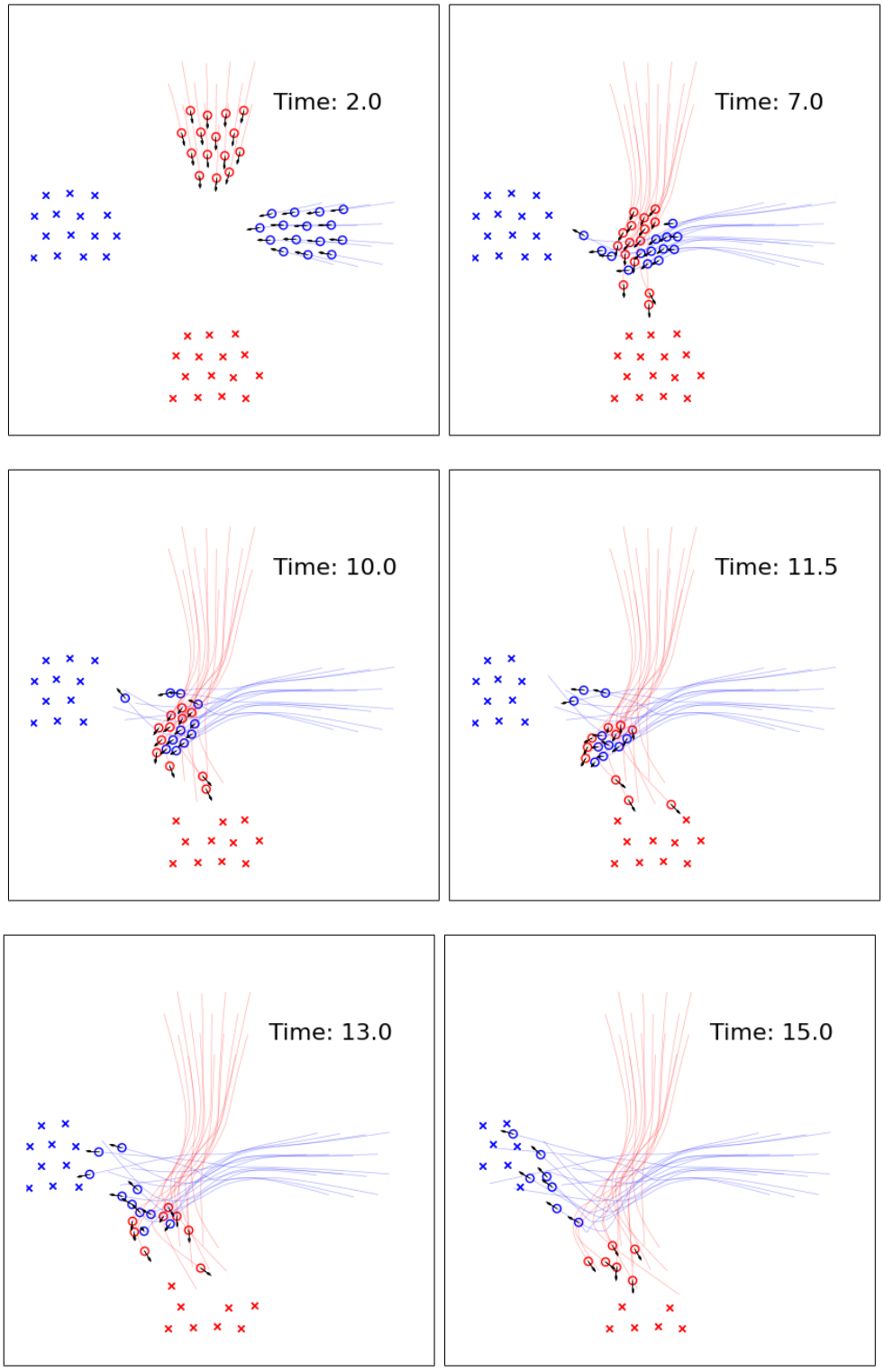

Figure 4.6: **Vanilla Powerlaw** simulation on `Two-groups at 90°` scenario. Notice from 7 to 11 seconds how all agents try to reach towards their goal, but end up pushing the mass of agents towards the bottom-left. This causes a large group of agents to deviate from the straight line path and cause a high interaction overhead.

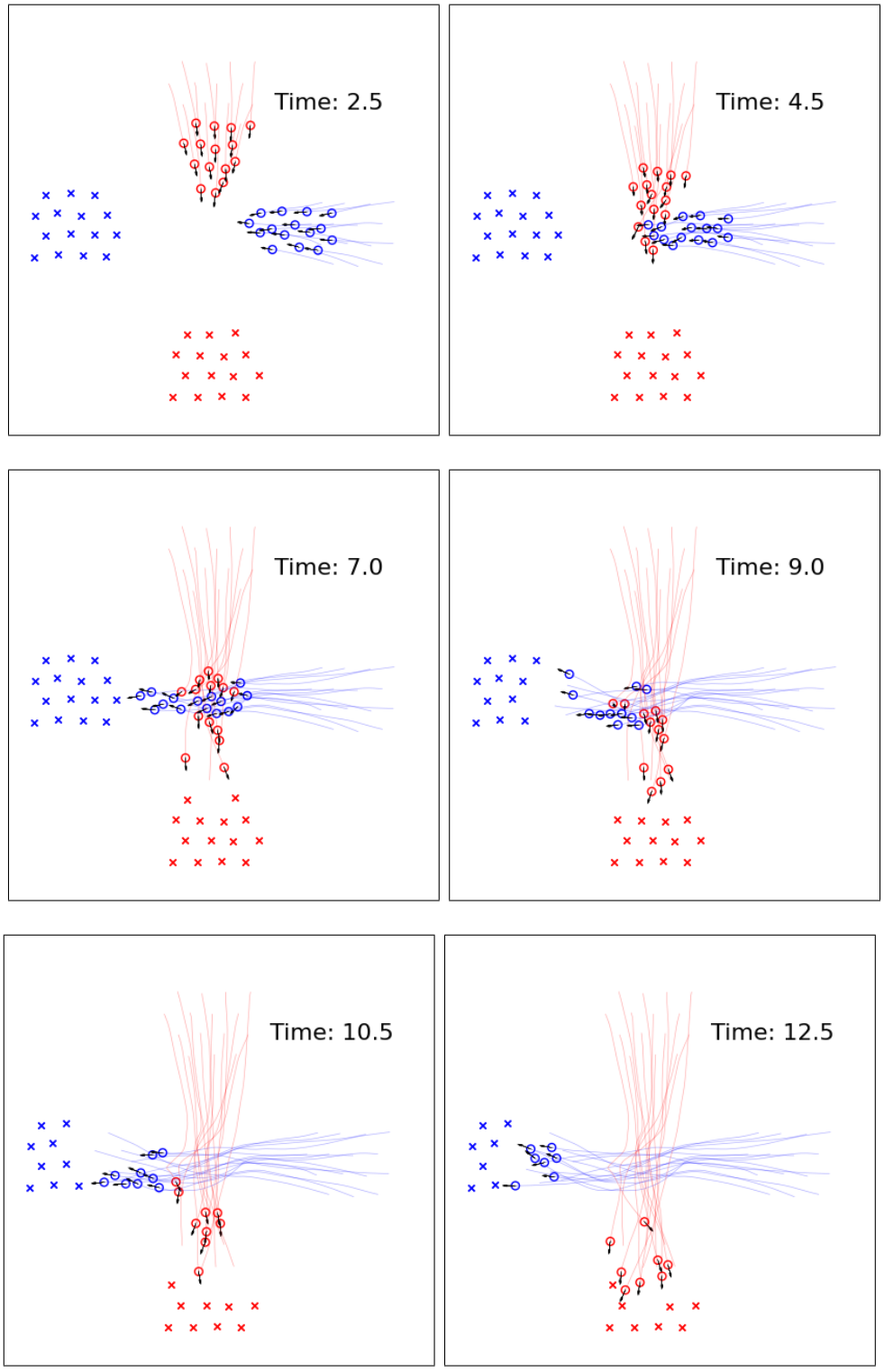

Figure 4.7: **PowerLaw** simulation with **MPD-assisted** preferred velocities on `Two-groups at 90°` scenario. The agents complete the scenario faster as well as take less deviations while reaching the goal compared to vanilla PowerLaw.

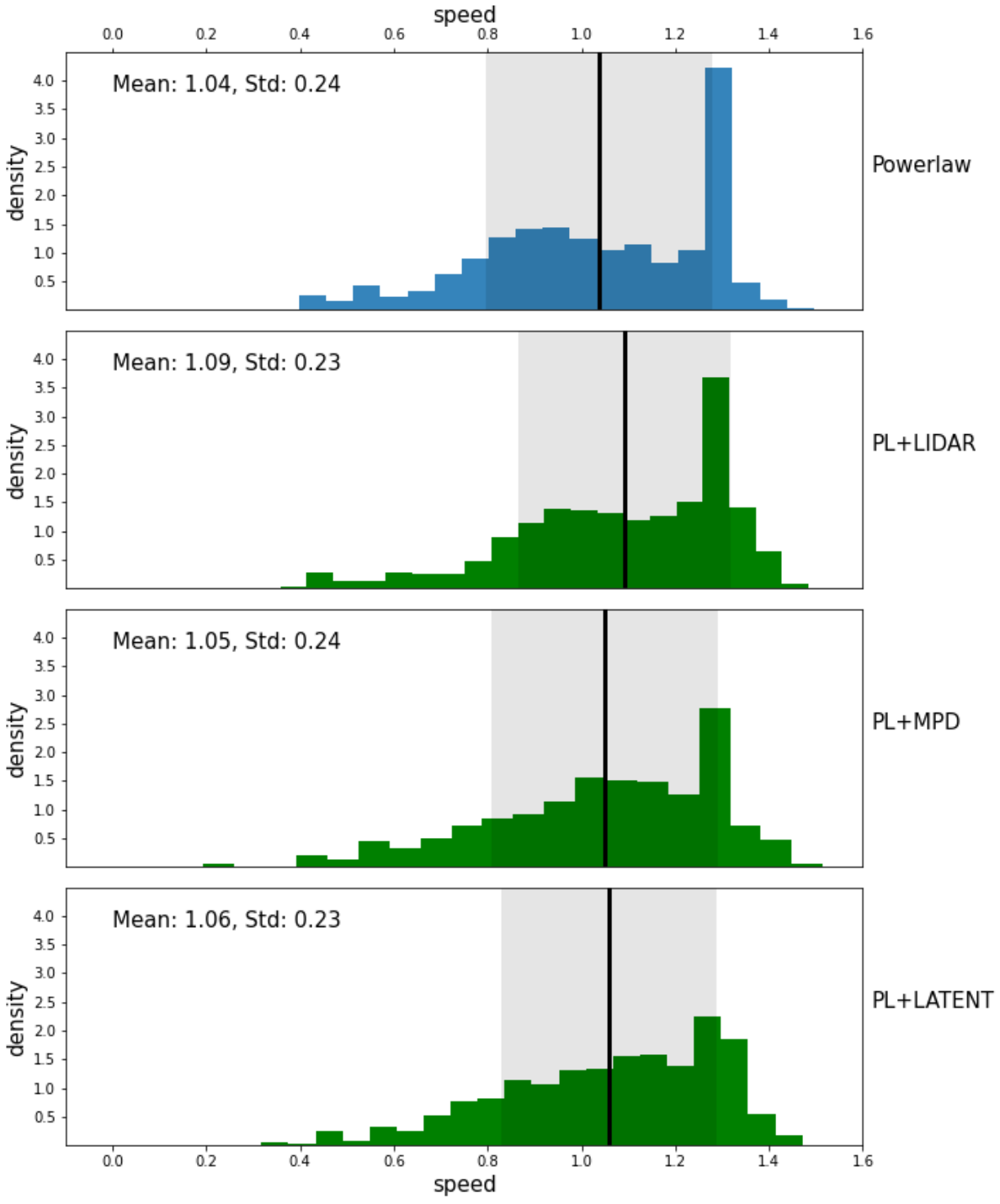

Figure 4.8: Speed distribution histogram for `Two-groups at 90°` scenario with **PowerLaw** variants. The black line denotes the mean speed and the gray box shows the one-standard deviation.

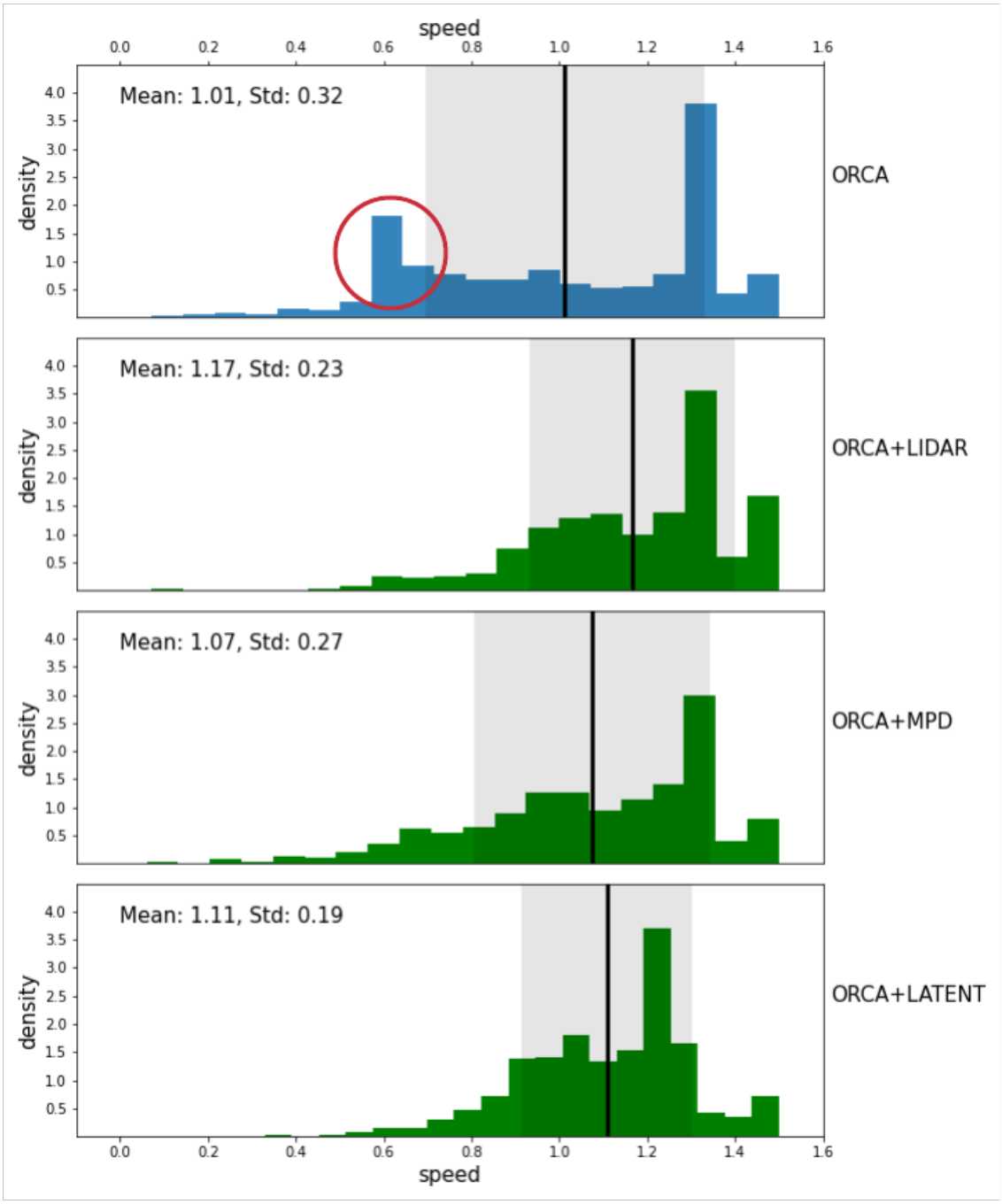

Figure 4.9: Speed distribution histogram for `Two-groups at 90°` scenario with **ORCA variants**.The black line denotes the mean speed and the gray box shows the one-standard deviation. The high density in ORCA at speed = 0.6m/s shows that some agents slow down during certain parts of their trajectories.

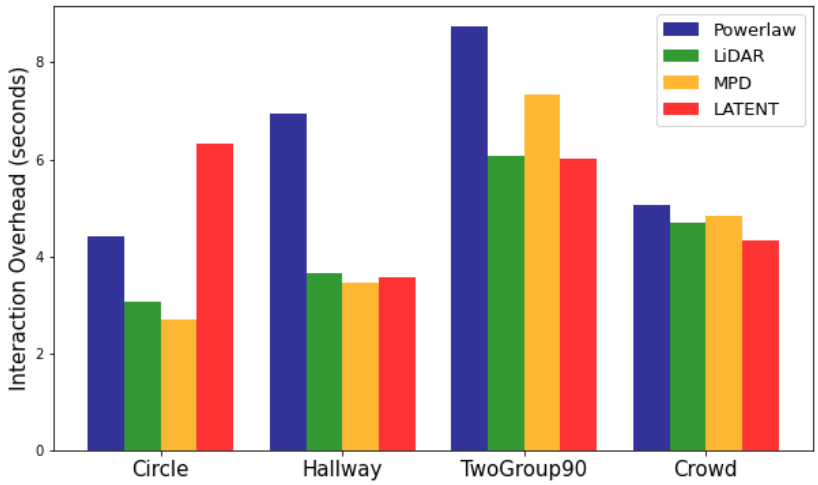

Figure 4.10: A bar graph that shows the comparison of interaction overhead with **PowerLaw and our three methods** on test scenarios. A higher interaction overhead corresponds to agents taking more time to reach their goals.

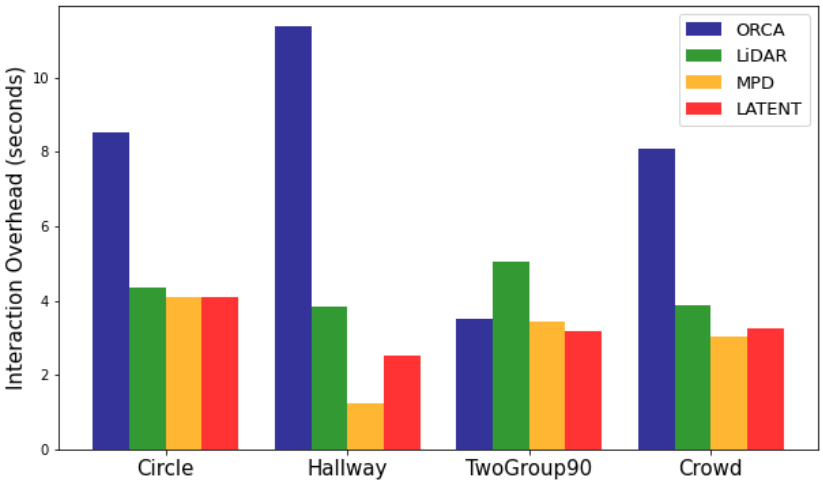

Figure 4.11: A bar graph that shows the comparison of interaction overhead with **ORCA and our three methods** on test scenarios. A higher interaction overhead corresponds to agents taking more time to reach their goals.

Chapter 5

Conclusion and Discussion

In this thesis, we explored a human-inspired approach for learning efficient multi-agent navigation. We used two collision avoidance frameworks, ORCA and the Power Law, that traditionally rely on goal-oriented preferred velocities. We propose a framework that queries a neural network to pick preferred velocities as a function of the agent's local conditions. We train the neural networks to learn human preferred deviations (from the goal direction) from four publicly available crowd datasets and investigate three neighborhood encoding methods that try to capture the static and dynamic properties of the agent's surroundings. The first method is a distance-based LiDAR scan with stacked frames (to provide a sense of time to the encoding). The second method is another radial scan based on the minimum predicted distance (MPD) metric, which measures the distance of the closest approach of the ego-agent with respect to each of its neighbors. The third method we experimented with is a general neural encoding method that does not make any assumptions about how to describe the interaction conditions that an agent faces. We individually encode the states (positions, velocities) of neighboring agents and use an attention module to encode the collection into a fixed-length embedding space.

Our training datasets provide a wide range of real-world crowd interaction scenarios. During training, we create mini-batches by stratifying examples from each scenario, promoting diversity and generalization. We also analyze the trained neural networks by designing a gradient-based interpretation algorithm (for LiDAR and MPD-based networks) that inputs a query input deviation angle and outputs a radial scan that makes the trained network produce *that* deviation. The scans align perfectly with human intuition, as it is noticeable that the network learns to go straight towards the goal when there are no obstacles in front and takes deviations when surrounded by agents.

After we train the neural networks, we use them to pass high-level control signals to the agents' underlying planning algorithms during simulation. Each agent iteratively senses its local neighborhood, queries the neighborhood to obtain a human-preferred deviation, and pass this (along with its preferred speed) into its collision avoidance subroutine. The subroutine then returns a new collision-free velocity which the agent takes at the next time step. Thus, we follow a hybrid approach, where we combine $learning-based$ and $planning-based$ methods to train virtual agents to exhibit human-level navigation patterns and retain the formal guarantees provided by collision-avoidance algorithms.

We test these data-driven agents in multiple diverse scenarios and evaluate their global interaction overhead and energy efficiency. Our results show a promising direction for future research. When human-inspired actions are passed as preferred velocities into the agent's underlying local planner, we observed the time efficiency of the multi-agent system improves, especially when using the ORCA navigation method. We also show an improvement over PowerLaw, a force-based planner derived from crowd interaction data that aligns with our original hypothesis that opting for greedy actions at every timestep (always aligning preferred direction of motion with

the goal direction) can exhibit myopic behavior that can be detrimental to the long-term efficiency of the global system of agents. Instead, our method allows agents to opt for *smarter* target velocities that adapt to its neighborhood dynamics. We present some trajectories from simulations, such as Figures 4.3, 4.5, and 4.7, where we show examples where agents show a tendency to react to each other's actions (like humans do while navigating in a crowded space) and take independent actions without any communication with its neighbors. We demonstrate quantitatively how these actions eventually lead to an improvement in global efficiency.

We found that the MPD method generalizes the best across all our test scenarios and planners both qualitatively and quantitatively. The LiDAR method also shows promising results in a variety of scenarios. Still, conceptually the MPD scan has the edge over LiDAR because, by definition, the former can encode dynamic information of the neighborhood more straightforwardly. While the current formulation of the latent-embedding method has some limitations (no representation of the agents' radius, no simple way of encoding static obstacles), early results with this method appear to be promising. Because of the generality of the encoding scheme, we think training on larger datasets should help improve the performance further. Further experimentation is needed to see how all three method scales in different scenarios that include static obstacles and when using different training datasets.

Limitations:

Since our approach is data-driven, the quality of our results is bounded by the scope and sizes of the datasets and training examples that we use. Our testing scenarios were devoid of static obstacles (we show some preliminary results in Appendix A), and it remains to be seen how adding more closed and densely packed scenarios

affect the simulations. We would also need to make adjustments to the attention-based descriptor in order to account for static obstacles. One method may be to treat static obstacles as a collection of neighbors with zero velocity, and continue using our current formulation. This may be computationally inefficient, so other static-obstacle representations need also to be considered. Still, due to it's generality, we think that the attention-based method has the potential to outperform both the MPD and the LiDAR method, given enough data.

We assume that all our agents are discs with fixed radius in our framework, which may not be the case when we are in a real-world setting. If this method is to be transferred to real robotic systems, it may be worth to explore different shapes and sizes of agents. We also assume that trained agents can behave similarly to the expert pedestrians from our datasets, ignoring the kinodynamic constraints of the real-world robots. Finally, it must be noted that our method does not directly optimize for global efficiency. We follow the decentralized multi-agent navigation paradigm where each neighbor senses and acts for optimizing towards own goals, without communication with its neighbors. We are aware that our approach cannot entirely replace a global planner, when deploying many agents to navigate in a complex environments with dense obstacles, or with densely crowded pedestrians, or for that matter both. Adding a waypoint generation algorithm as a high level planner that inputs new goal locations can also be beneficial for traversing through scenarios with a large number of static obstacles [42].

Future Work:

There are multiple promising directions for future work. One could improve the framework by addressing limitations mentioned above. Although we learn navigation

policies from human data, our primary goal is to increase the efficiency of the global system of agents. It will be interesting to assess the trajectories taken by agents using our approach and compare directly with crowd motion, for example by using the entropy metric [23, 32] to quantitatively analyze the similarity between trajectories observed in our agents and ground truth human trajectories. We can also evaluate on the basis of distance metric [62], novelty detection [74, 71], or unfreezing time [61]. The next phase of our research involves deploying our methods to turtlebots and test how the method translates to real- world applications. Another interesting avenue of research is to deploy robots in a heterogeneous setting and extend our work to socially-intelligent agents that interact with human beings [53, 22, 41]. We also intend to collect laboratory and real-world datasets in such settings, which can be used downstream for supervised learning and facilitating socially-aware multi-robot navigation.

Appendices

Appendix A Learning from dense scenarios with obstacles

In this section, we discuss training MPD and LiDAR based descriptors on dense or closed pedestrian datasets that contain multiple static obstacles. Table 1 shows the description of the two scenarios used for the training. In [32], the authors capture a low dimensional manifold of various scenarios that humans encounter during crowd navigation, and show that human action in closed scenarios like the `bnc` (Figure 1) map to a different space than open scenarios in 3.1. Our early results are promising and shows that learning from dense scenarios can show improvement over vanilla local planners, and that training policies on dense scenarios produce better results in crowded spaces compared to policies trained in open scenarios like `zara`, `students` and `oneway`.

Dataset	***Description***	***Pedestrian count***	***Frames (10fps)***
Bnc-Bottleneck [55]	Pedestrians walking down a constricted hallway.(unidirectional flow)	176	18519
Bncl-Long Bottleneck [55]	Pedestrians walking down a long constricted hallway.(unidirectional flow)	178	31127

Table 1: Pedestrian datasets used for training

For learning human-preferred deviations in closed scenarios, we trained neural networks on `bnc` and `bncl` datasets, having about 50,000 interaction examples in total, which is considerably lower than the open scenarios we trained in Table 4.1 (which contained close to 130,000 interactions). We evaluate these networks on the

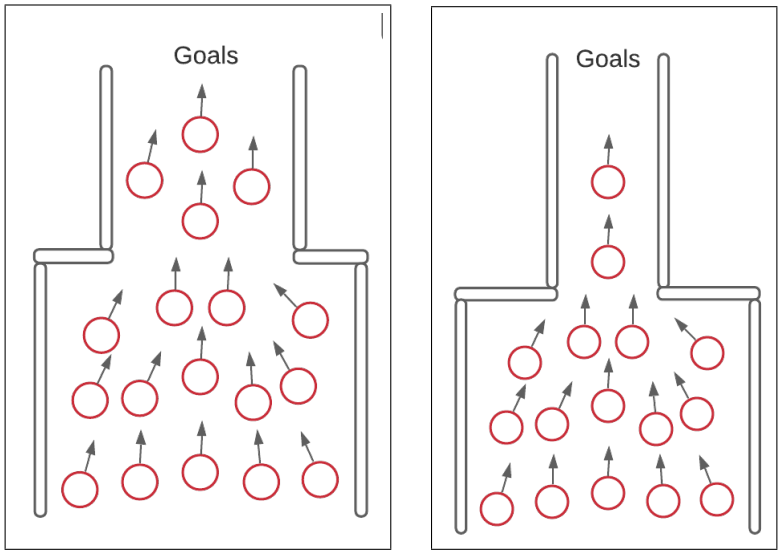

Figure 1: The datasets used for training. Left: Bottleneck, Right: Long bottleneck

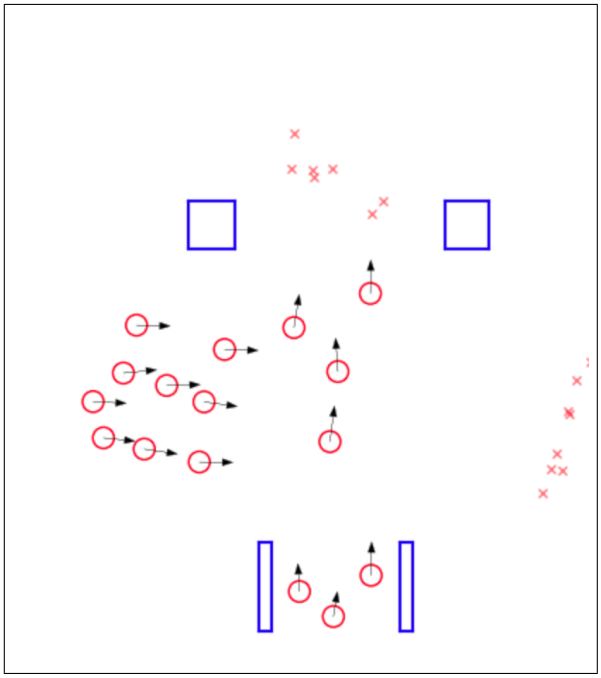

Figure 2: `Gr90`: A "closed" densely packed scenario with static obstacles. For 15 seconds, two groups (with total 100 pedestrians) enter the scene and need to cross each other to reach the other at 90 degrees.

`Two-groups at 90°` dataset (Chapter 4) and the `Gr90` [39] dataset (Figure 2), which consists of two groups with a total of 100 pedestrians trying to cross each other at 90° in the presence of static obstacles in the scene. Table 2 and 3 shows experiment results on both of these scenarios with using ORCA and PowerLaw respectively. Despite the low number of training rows, we can see that the network can still improve the performance of vanilla methods or at least match it. Especially with PowerLaw, our approach has considerably improved the vanilla implementation in both scenarios. The closed-model seems to also improve on the open-models trained for Table 4.1 and 4.2 in the `Two-groups at 90°` scenario. Although these results look promising, they are inconclusive and need more investigation with larger training data and testing scenarios.

Scene	***Method***	***Sim Time***	***Average Speed***	***Interaction Overhead***	***Energy Efficiency***
Gr90	ORCA	**30.4**	**1.23**	3.27	**0.36**
	LiDAR	30.5	**1.23**	**2.79**	**0.36**
	MPD	30.5	1.21	4.15	0.35
Two-groups at 90°	ORCA	14.3	1.244	3.5	0.368
	LiDAR	14.4	**1.27**	3.5	**0.37**
	MPD	**13.9**	1.25	**3.4**	**0.37**

Table 2: **Vanilla ORCA** Experiment Results. Collision statistics are not shown since no collisions were encountered.

Scene	***Method***	***Sim Time***	***Average Speed***	***Interaction Overhead***	***Energy Efficiency***
Gr90	PowerLaw	32.9*	1.19	3.99	0.36
	LiDAR	**30.94**	1.18	**2.73**	**0.37**
	MPD	**30.98**	1.17	**2.77**	**0.37**
Two-groups at 90°	Powerlaw	16.70	1.08	8.404	0.345
	LiDAR	**14.77**	**1.18**	**4.07**	0.36
	MPD	14.89	1.16	4.55	**0.37**

Table 3: **Powerlaw** Experiment Results. (* here represents that 1 agent was stuck at the obstacle at the end of the simulation. Collision statistics are not shown since no collisions were encountered)

Appendix B Learning Speed Information from Pedestrian Datasets

In section 3.3, we train neural networks to learn human-preferred deviations from pedestrian datasets. During simulation, we assumed that the preferred speed of the agent is always fixed and we only incorporate the human-preferred deviation to formulate a new $\mathbf{v_{pref}}$ that we pass as input into local planners. In this section, we provide results where we also learn the magnitude of human-preferred velocities from open-crowd datasets (`oneway01, oneway02, zara01, zara02, students`), compared to just learning deviations, and use $\mathbf{v_{pref}}$ as a tuple of both human-preferred speed and deviation. This means that for certain interaction scenarios, the agent will be able to decide to slow down, as well as go faster compared to its usual preferred speed. We tested this approach in the `circle, hallway,` and `Two-groups at` 90° scenarios using the same experimental conditions as section 4.2. Tables 4 and 5 show corresponding results. The latent method seems to be more stable compared to MPD and LiDAR methods, however the results still seem inconclusive. In certain scenarios, using human-preferred speeds actually shows a minor drop in performance compared to vanilla planner. Comparing with 4.1 and 4.2, we can observe that the performance gain has reduced in these results.

In conclusion, it is unclear if they can increase the global efficiency of the system. This does not necessarily mean that learning human-preferred speed is a bad idea. We hypothesize that learning both speed and deviation will produce more human-like understanding of crowd interactions in virtual agents. Conceptually, slowing down speeds (in dense environments) may result in higher energy efficiency and less deviations from the goal path. We think that training on a larger corpus of data and test on more representative test scenarios can help to properly evaluate the worth

of learning human-speeds.

Scene	***Method***	***Sim Time***	***Average Speed***	***Interaction Overhead***	***Energy Efficiency***
Circle	ORCA	11.70	0.97	8.53	0.30
	Lidar	**9.7**	**1.07**	**4.75**	**0.34**
	MPD	10.60	0.91	7.80	0.30
	Latent	**9.8**	0.97	5.96	0.32
Hallway	ORCA	14.20	0.66	10.73	0.21
	Lidar	9.09	1.02	4.39	0.33
	MPD	8.8	1.056	4.23	0.345
	Latent	**8.3**	**1.01**	**2.89**	**0.355**
Two-groups at 90°	ORCA	**14.3**	**1.24**	**3.5**	0.368
	Lidar	14.99	1.17	5.04	0.363
	MPD	15.6	1.16	5.4	0.365
	Latent	14.8	1.19	4.17	**0.373**

Table 4: **ORCA** Experiment Results using human-like preferred speeds and directions. Collision statistics are not shown since no collisions were encountered.

Scene	***Method***	***Sim Time***	***Average Speed***	***Interaction Overhead***	***Energy Efficiency***
Circle	PowerLaw	9.22	0.995	4.411	0.335
	Lidar	8.54	**1.036**	**3.44**	**0.3514**
	MPD	**8.22**	1.019	3.39	0.349
	Latent	9.43	0.926	5.33	0.323
Hallway	PowerLaw	12.38	0.72	6.952	0.257
	Lidar	10.75	0.97	5.25	0.345
	MPD	10.72	0.94	5.51	0.32
	Latent	**9.23**	**0.98**	**4.24**	**0.36**
Two-groups at 90°	PowerLaw	10.65	0.87	5.07	0.32
	Lidar	10.76	0.96	5.25	0.337
	MPD	10.72	0.94	5.51	0.33
	Latent	**9.24**	**0.97**	**4.24**	**0.342**

Table 5: **PowerLaw** Experiment Results using human-like preferred speeds and directions. Collision statistics are not shown since no collisions were encountered.

Appendix C Local Planners

C.1 Optimal Reciprocal Collision Avoidance

ORCA is a velocity-based approach for collision avoidance that provides guarantees for collision-free motion among multiple holonomic robots. We are given a group of n robots which are represented as disks of radius r_i, each having a target goal position $\mathbf{g_i}$, velocity $\mathbf{v_i}$ and position $\mathbf{x_i}$. At each timestep, the agent also selects a preferred velocity $\mathbf{v_i^{pref}}$, which is typically chosen as the goal direction scaled by its preferred speed. The objective of ORCA is to compute an optimal v_i that is collision-free and maximally aligned to $\mathbf{v_i^{pref}}$.

For robot A_i and neighbor A_j, the velocity obstacle [17] $VO_{i|j}$ is a set of all relative velocities that will lead to a collision between the two agents at some moment in time $\tau >= 0$.

$$VO^{\tau}_{i|j} = \{\bar{\mathbf{v}} \mid \exists t \in [0, \tau], \ \ \tau\bar{\mathbf{v}} \in D(\mathbf{x_j} - \mathbf{x_i}, r_j + r_i)\}$$

where $D(\mathbf{x}, r)$ is a disc centered at $\mathbf{x}$ having radius r.

The set of collision-free velocities for the agent A_i is given by:

$$ORCA^{\tau}_i = D(0, V^{max}) \ \cap_{i \neq j} \ ORCA^{\tau}_{i|j}$$

where $D(0, V_i^{max})$ denotes the set of allowed holonomic velocities for the robot, and $ORCA^{\tau}_{i|j}$ is a set of velocities which move agent A_i at least halfway out of the velocity obstacle between A_i and A_j.

The optimal velocity for the robot can then be found as the velocity in the set $ORCA^{\tau}_i$ that is closest to the preferred velocity $\mathbf{v_i^{pref}}$:

$$\mathbf{v_i^*} = \text{argmin}_{\mathbf{v_i} \in ORCA^{\tau}_i} \|\mathbf{v_i} - \mathbf{v_i^{pref}}\|$$

The above optimization problem can be solved using linear programming in time linear with the number of neighbors that A_i has to avoid. The position of the agent can then be computed at the next timestep using standard Euler integration:

$$\mathbf{x_i} += \mathbf{v_i}\Delta t, \text{ where } \Delta t \text{ is the simulation time.}$$

C.2 Power Law

PowerLaw is a predictive force-based method that computes collision-free velocities using the notion of time-to-collision. In [31], the authors analyzed a variety of publicly available crowd datasets and demonstrated that the interaction energy between a pair of pedestrians follows a PowerLaw distribution as a function of their projected time-to-collision τ. The time to collision τ between agent A_i and neighbor A_j, such that their relative distance $x = x_i - x_j$, relative velocity $v = v_i - v_j$ and combined radius $r = r_i + r_j$, denotes the first time in the future that the two agents collide assuming both agents extrapolate forward with the same velocity. It can be computed by finding the smallest positive root for the quadratic equation related to the necessary condition for disc collision:

$$\|\mathbf{x} + \mathbf{v}\tau\| = r$$

In case no roots exist for τ we consider $\tau = \infty$, which means that the two agents can never collide (for example if their velocities are equal and parallel to each other).

The interaction potential of two pedestrians are mathematically given as:

$$U(\mathbf{x}, \mathbf{v}) = k\tau^{-p}e^{-\tau/\tau_0}$$

where k is the scaling constant that sets the unit for energy, p is the exponent of power law, and the τ_0 models the fact that pedestrians tend to ignore collisions that place too far in the future.

The collision force is calculated as the negative gradient of the potential:

$$\mathbf{F}_C = -\nabla_x U$$
$$\Rightarrow \mathbf{F}_C = \frac{ke^{-\tau/\tau_0}}{\tau^{m+1}}\left(p + \frac{\tau}{\tau_0}\right)\frac{\mathbf{x} + \mathbf{v}\tau}{\sqrt{D}}$$

where D is the discriminant of the quadratic $(\mathbf{x} \cdot \mathbf{v})^2 - \|\mathbf{v}\|^2(\|\mathbf{x}\|^2 - r^2)$

Given the preferred velocity v_{pref} and the current velocity v_i of the agent, the driving force F_{goal} computes the force due to which the agent moves towards its goal. $\mathbf{F}_{goal} = \frac{\mathbf{v_i^{pref}} - \mathbf{v_i}}{\xi}$, where ξ is the time that the agent takes to adapt its velocity to the preferred velocity.

Given the driving force $\mathbf{F}_{goal}$ and the collision avoidance force $\mathbf{F}_C$, we calculate the total force $\mathbf{F} = \mathbf{F}_{goal} + \mathbf{F}_C$. Using Euler integration, the new velocity and position is computed.

$$\mathbf{v} += \mathbf{F}\Delta t$$
$$\mathbf{x} += \mathbf{v}\Delta t$$

where Δt is the time step of the simulation.

Because PowerLaw is derived from crowd data, agents tend to exhibit more human-like behavior. As compared to ORCA, PowerLaw agents are willing to take a step towards a collision if that collision can be solved more efficiently in the short run. This addresses the issues of timid agents commonly encountered with ORCA and leading to agents being more assertive towards reaching their goals.
One of the disadvantage of the PowerLaw is that, as any first-order method, it typically requires a very small time step to guarantee numerical stability. This makes it slower to run than ORCA.

Bibliography

[1] Javier Alonso-Mora, Andreas Breitenmoser, Martin Rufli, Paul Beardsley, and Roland Siegwart. *Optimal Reciprocal Collision Avoidance for Multiple Non-Holonomic Robots*, pages 203–216. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[2] C. Amato, G. Konidaris, G. Cruz, C. A. Maynor, J. P. How, and L. P. Kaelbling. Planning for decentralized control of multiple robots under uncertainty. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1241–1248, 2015.

[3] Tucker Balch. Social entropy: a new metric for learning multi-robot teams. 04 1997.

[4] Tucker Balch. Hierarchic social entropy: An information theoretic measure of robot group diversity. *Autonomous Robots*, 8, 03 2000.

[5] Cory D. Boatright, Mubbasir Kapadia, Jennie M. Shapira, and Norman I. Badler. Generating a multiplicity of policies for agent steering in crowd simulation. *Comput. Animat. Virtual Worlds*, 26(5):483–494, September 2015.

[6] Mariusz Bojarski, D. Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, L. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, X. Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. *ArXiv*, abs/1604.07316, 2016.

[7] C. Chen, Y. Liu, S. Kreiss, and A. Alahi. Crowd-robot interaction: Crowd-aware robot navigation with attention-based deep reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6015–6022, 2019.

[8] Y. F. Chen, M. Everett, M. Liu, and J. P. How. Socially aware motion planning with deep reinforcement learning. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1343–1350, 2017.

[9] Bobby Davis, Ioannis Karamouzas, and Stephen J Guy. Nh-ttc: A gradient-based framework for generalized anticipatory collision avoidance. *arXiv preprint arXiv:1907.05945*, 2019.

[10] Vishnu R Desaraju and Jonathan P How. Decentralized path planning for multi-agent teams with complex constraints. *Autonomous Robots*, 32(4):385–403, 2012.

[11] S. M. Ali Eslami, Danilo Jimenez Rezende, Frederic Besse, Fabio Viola, Ari S. Morcos, Marta Garnelo, Avraham Ruderman, Andrei A. Rusu, Ivo Danihelka, Karol Gregor, David P. Reichert, Lars Buesing, Theophane Weber, Oriol Vinyals, Dan Rosenbaum, Neil Rabinowitz, Helen King, Chloe Hillier, Matt Botvinick, Daan Wierstra, Koray Kavukcuoglu, and Demis Hassabis. Neural scene representation and rendering. *Science*, 360(6394):1204–1210, 2018.

[12] Michael Everett, Yu Fan Chen, and Jonathan P. How. Collision avoidance in pedestrian-rich environments with deep reinforcement learning. *IEEE Access*, 9:10357–10377, 2021.

[13] Tingxiang Fan, Xinjing Cheng, Jia Pan, D. Manocha, and Ruigang Yang. Crowdmove: Autonomous mapless navigation in crowded scenarios. *ArXiv*, abs/1807.07870, 2018.

[14] Tingxiang Fan, Pinxin Long, Wenxi Liu, and Jia Pan. Fully distributed multi-robot collision avoidance via deep reinforcement learning for safe and efficient navigation in complex scenarios. 08 2018.

[15] Tingxiang Fan, Pinxin Long, Wenxi Liu, and Jia Pan. Distributed multi-robot collision avoidance via deep reinforcement learning for navigation in complex scenarios. *The International Journal of Robotics Research*, 39(7):856–892, 2020.

[16] Ernst Fehr and Urs Fischbacher. Social norms and human cooperation. *Trends in cognitive sciences*, 8:185–90, 05 2004.

[17] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17:760–, 07 1998.

[18] Julio Godoy. Machine learning methods for multi robot navigation. In *Proceedings of the 2015 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '15, page 1995–1996, Richland, SC, 2015. International Foundation for Autonomous Agents and Multiagent Systems.

[19] Julio Godoy, Stephen J. Guy, Maria Gini, and Ioannis Karamouzas. C-nav: Distributed coordination in crowded multi-agent navigation. *Robotics and Autonomous Systems*, 133:103631, 2020.

[20] Julio Godoy, Ioannis Karamouzas, Stephen Guy, and Maria Gini. Adaptive learning for multi-agent navigation. *Autonomous Robots*, 3:1577–1585, 12 2018.

[21] Julio Godoy, Ioannis Karamouzas, Stephen J Guy, and Maria Gini. Anytime navigation with progressive hindsight optimization. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 730–735. IEEE, 2014.

[22] Agrim Gupta, Justin Johnson, Li Fei-Fei, Silvio Savarese, and Alexandre Alahi. Social gan: Socially acceptable trajectories with generative adversarial networks. pages 2255–2264, 06 2018.

[23] Stephen J. Guy, Jur van den Berg, Wenxi Liu, Rynson Lau, Ming C. Lin, and Dinesh Manocha. A statistical similarity measure for aggregate crowd dynamics. *ACM Trans. Graph.*, 31(6), November 2012.

[24] Dirk Helbing, Illés Farkas, and Tamás Vicsek. Simulating dynamic features of escape panic. *Nature*, 407:487–490, 09 2000.

[25] Dirk Helbing and Péter Molnár. Social force model for pedestrian dynamics. *Phys. Rev. E*, 51:4282–4286, May 1995.

[26] Daniel Hennes, Daniel Claes, Wim Meeussen, and Karl Tuyls. Multi-robot collision avoidance with localization uncertainty. In *AAMAS*, pages 147–154, 2012.

[27] Peter Henry, Christian Vollmer, Brian Ferris, and Dieter Fox. Learning to navigate through crowded environments. pages 981 – 986, 06 2010.

[28] Dalton Hildreth and Stephen J. Guy. Coordinating multi-agent navigation by learning communication. *Proc. ACM Comput. Graph. Interact. Tech.*, 2(2), July 2019.

[29] Geoffrey Hinton, Oriol Vinyals, and Jeffrey Dean. Distilling the knowledge in a neural network. In *NIPS Deep Learning and Representation Learning Workshop*, 2015.

[30] Eunjung Ju, Myung Geol Choi, Minji Park, Jehee Lee, Kang Hoon Lee, and Shigeo Takahashi. Morphable crowds. *ACM Transactions on Graphics (TOG)*, 29(6):1–10, 2010.

[31] Ioannis Karamouzas, Brian Skinner, and Stephen J Guy. Universal power law governing pedestrian interactions. *Physical review letters*, 113(23):238701, 2014.

[32] Ioannis Karamouzas, Nick Sohre, Ran Hu, and Stephen J. Guy. Crowd space: A predictive crowd analysis technique. *ACM Trans. Graph.*, 37(6), December 2018.

[33] Ioannis Karamouzas, Nick Sohre, Rahul Narain, and Stephen Guy. Implicit crowds: optimization integrator for robust crowd simulation. *ACM Transactions on Graphics*, 36:1–13, 07 2017.

[34] Sammie Katt, Frans A. Oliehoek, and Christopher Amato. Bayesian reinforcement learning in factored pomdps. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems*, AAMAS '19, page 7–15, Richland, SC, 2019. International Foundation for Autonomous Agents and Multiagent Systems.

[35] Beomjoon Kim and Joelle Pineau. Socially adaptive path planning in human environments using inverse reinforcement learning. *International Journal of Social Robotics*, 8:51–66, 2016.

[36] Wee Lit Koh and Suiping Zhou. Modeling and simulation of pedestrian behaviors in crowded places. *ACM Trans. Model. Comput. Simul.*, 21(3), February 2011.

[37] Henrik Kretzschmar, Markus Spies, Christoph Sprunk, and Wolfram Burgard. Socially compliant mobile robot navigation via inverse reinforcement learning. *The International Journal of Robotics Research*, 35(11):1289–1307, 2016.

[38] Markus Kuderer, Henrik Kretzschmar, Christoph Sprunk, and Wolfram Burgard. Feature-based prediction of trajectories for socially compliant navigation. 07 2012.

[39] Gregor Lämmel and Matthias Plaue. Getting out of the way: Collision-avoiding pedestrian models compared to the realworld. In *Pedestrian and Evacuation Dynamics 2012*, pages 1275–1289. Springer, 2014.

[40] Alon Lerner, Yiorgos Chrysanthou, and Dani Lischinski. Crowds by example. In *Computer graphics forum*, volume 26, pages 655–664. Wiley Online Library, 2007.

[41] Yuejiang Liu, Qi Yan, and Alexandre Alahi. Social nce: Contrastive learning of socially-aware motion representations. *ArXiv*, abs/2012.11717, 2020.

[42] Zuxin Liu, B. Chen, Hongyi Zhou, G. Koushik, M. Hebert, and D. Zhao. Mapper: Multi-agent path planning with evolutionary reinforcement learning in mixed dynamic environments. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 11748–11754, 2020.

[43] Pinxin Long, Tingxiang Fan, Xinyi Liao, Wenxi Liu, Hao Zhang, and Jia Pan. Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6252–6259. IEEE, 2018.

[44] Pinxin Long, Wenxi Liu, and Jia Pan. Deep-learned collision avoidance policy for distributed multi-agent navigation. *IEEE Robotics and Automation Letters*, PP, 09 2016.

[45] T. Mundhenk, Barry Chen, and Gerald Friedland. Efficient saliency maps for explainable ai. 11 2019.

[46] Anne-Hélène Olivier, Antoine Marin, Armel Crétual, and Julien Pettré. Minimal predicted distance: A common metric for collision avoidance during pairwise interactions between walkers. *Gait Posture*, 36(3):399–404, 2012.

[47] Mark Pfeiffer, Michael Schaeuble, Juan Nieto, Roland Siegwart, and Cesar Cadena. From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. pages 1527–1533, 05 2017.

[48] Dean Pomerleau. Alvinn: An autonomous land vehicle in a neural network. In D.S. Touretzky, editor, *Proceedings of Advances in Neural Information Processing Systems 1*, pages 305 –313. Morgan Kaufmann, December 1989.

[49] Victoria Rapos, Michael Cinelli, Natalie Snyder, Armel Crétual, and Anne-Hélène Olivier. Minimum predicted distance: Applying a common metric to collision avoidance strategies between children and adult walkers. *Gait Posture*, 72:16–21, 2019.

[50] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, page 25–34, New York, NY, USA, 1987. Association for Computing Machinery.

[51] Craig W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *SIGGRAPH Comput. Graph.*, 21(4):25–34, August 1987.

[52] Craig W Reynolds. Steering behaviors for autonomous characters. In *Game developers conference*, volume 1999, pages 763–782. Citeseer, 1999.

[53] Adarsh Sathyamoorthy, Jing Liang, Utsav Patel, Tianrui Guan, Rohan Chandra, and Dinesh Manocha. Densecavoid: Real-time navigation in dense crowds using anticipatory behaviors. pages 11345–11352, 05 2020.

[54] Adarsh Jagan Sathyamoorthy, Utsav Patel, Tianrui Guan, and D. Manocha. Frozone: Freezing-free, pedestrian-friendly navigation in human crowds. *IEEE Robotics and Automation Letters*, 5:4352–4359, 2020.

[55] Armin Seyfried, Oliver Passon, Bernhard Steffen, Maik Boltes, Tobias Rupprecht, and Wolfram Klingsch. New insights into pedestrian flow through bottlenecks. *Transportation Science*, 43(3):395–406, 2009.

[56] Masahiro Shiomi, Francesco Zanlungo, Kotaro Hayashi, and Takayuki Kanda. Towards a socially acceptable collision avoidance for a mobile robot navigating among pedestrians using a pedestrian model. *International Journal of Social Robotics*, 6(3):443–455, 2014.

[57] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. *arXiv preprint arXiv:1312.6034*, 2013.

[58] Shawn Singh, Mubbasir Kapadia, Petros Faloutsos, and Glenn Reinman. An open framework for developing, evaluating, and sharing steering algorithms. volume 5884, pages 158–169, 11 2009.

[59] Jamie Snape, Jur Van Den Berg, Stephen J Guy, and Dinesh Manocha. Independent navigation of multiple mobile robots with hybrid reciprocal velocity obstacles. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5917–5922. IEEE, 2009.

[60] Jiaming Song, Hongyu Ren, Dorsa Sadigh, and Stefano Ermon. Multi-agent generative adversarial imitation learning. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[61] P. Trautman and A. Krause. Unfreezing the robot: Navigation in dense, interacting crowds. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 797–803, 2010.

[62] Muhammad Usman, Tien-Chi Lee, Ryhan Moghe, Xun Zhang, Petros Faloutsos, and Mubbasir Kapadia. A social distancing index: Evaluating navigational policies on human proximity using crowd simulations. In *Motion, Interaction and Games*, MIG '20, New York, NY, USA, 2020. Association for Computing Machinery.

[63] Jur van den Berg, Stephen J. Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In Cédric Pradalier, Roland Siegwart, and Gerhard Hirzinger, editors, *Robotics Research*, pages 3–19, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.

[64] Jur Van Den Berg, Stephen J Guy, Ming Lin, and Dinesh Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011.

[65] H. Van Dyke Parunak and Sven Brueckner. Entropy and self-organization in multi-agent systems. In *Proceedings of the Fifth International Conference on Autonomous Agents*, AGENTS '01, page 124–130, New York, NY, USA, 2001. Association for Computing Machinery.

[66] Wouter van Toll and Julien Pettre. Synchronizing navigation algorithms for crowd simulation via topological strategies. *Computers Graphics*, 89, 04 2020.

[67] L. Wang, Z. Li, C. Wen, R. He, and F. Guo. Reciprocal collision avoidance for nonholonomic mobile robots. In *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 371–376, 2018.

[68] D. Wolinski, S. J. Guy, A.-H. Olivier, M. Lin, D. Manocha, and J. Pettré. Parameter estimation and comparative evaluation of crowd simulations. *Comput. Graph. Forum*, 33(2):303–312, May 2014.

[69] David Wolinski, S J. Guy, A-H Olivier, Ming Lin, Dinesh Manocha, and Julien Pettré. Parameter estimation and comparative evaluation of crowd simulations. In *Computer Graphics Forum*, volume 33, pages 303–312. Wiley Online Library, 2014.

[70] Yuchen Xiao, Joshua Hoffman, Tian Xia, and Chris Amato. Learning multi-robot decentralized macro-action-based policies via a centralized q-net. *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10695–10701, 2020.

[71] Pei Xu and Ioannis Karamouzas. Human-inspired multi-agent navigation using knowledge distillation. 03 2021.

[72] Fan Yang, Alina Vereshchaka, Changyou Chen, and Wen Dong. Bayesian multi-type mean field multi-agent imitation learning. 12 2020.

[73] Chao Yu, Minjie Zhang, Fenghui Ren, and Xudong Luo. Emergence of social norms through collective learning in networked agent societies. volume 1, pages 475–482, 05 2013.

[74] M. Zhao and Venkatesh Saligrama. Anomaly detection with score functions based on nearest neighbor graphs. In *NIPS*, 2009.