Attention Graph for Multi-Robot Social Navigation with Deep Reinforcement Learning

Erwan Escudie, Laëtitia Matignon, Jacques Saraydaryan

Attention Graph for Multi-Robot Social Navigation with Deep Reinforcement Learning

Erwan Escudie[1], Laetitia Matignon[2] and Jacques Saraydaryan[3]
Project Page: https://perso.liris.cnrs.fr/laetitia.matignon/multisoc.html

***Abstract*— Learning robot navigation strategies among pedestrian is crucial for domain based applications. Combining perception, planning and prediction allows us to model the interactions between robots and pedestrians, resulting in impressive outcomes especially with recent approaches based on deep reinforcement learning (RL). However, these works do not consider multi-robot scenarios. In this paper, we present MultiSoc, a new method for learning multi-agent socially aware navigation strategies using RL. Inspired by recent works on multi-agent deep RL, our method leverages graph-based representation of agent interactions, combining the positions and fields of view of entities (pedestrians and agents). Each agent uses a model based on two Graph Neural Network combined with attention mechanisms. First an edge-selector produces a sparse graph, then a crowd coordinator applies node attention to produce a graph representing the influence of each entity on the others. This is incorporated into a model-free RL framework to learn multi-agent policies. We evaluate our approach on simulation and provide a series of experiments in a set of various conditions (number of agents / pedestrians). Empirical results show that our method learns faster than social navigation deep RL mono-agent techniques, and enables efficient multi-agent implicit coordination in challenging crowd navigation with multiple heterogeneous humans. Furthermore, by incorporating customizable meta-parameters, we can adjust the neighborhood density to take into account in our navigation strategy.**

I. Introduction

Robot navigation in crowded spaces has attracted significant attention in recent years given its numerous potential applications, but it still faces many challenges [1]. Especially understanding pedestrian behavior is crucial to develop effective robot navigation strategies that prioritize human safety. But predicting crowd behavior is difficult and most of approaches intend to learn it from experiment or simulation [2], [3], [4]. Robot social navigation becomes even more challenging when the crowd is dense or the environment complex (obstacles, occlusions, reduced field of view, ...). Another difficulty appears when multiple robots navigate in the same crowd. Then a distinction must be made between a robot or human from the navigation point of view, as robots can coordinate themselves.

Recent approaches [5], [6] use deep reinforcement learning (RL) to build social navigation strategies with the help of a simulated crowd. Lately the works of [7] use deep RL combined with attention [8] and graph-based representations [9] of interactions between robot and humans. This achieved very good performance in dense crowds for a single robot. However, as specified by the authors, this model remains difficult to train as it exhibits unstable learning. Furthermore, its architecture explicitly separates the robot from humans and, as such, cannot easily be extended in its current form to accommodate multi-robot learning.

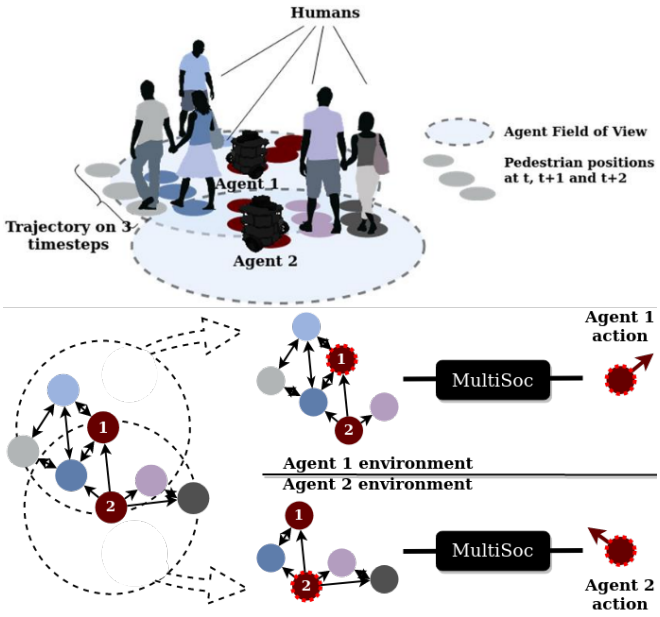

Fig. 1. Overview of MultiSoc process: (Top) Scene with two agents (robots) with 360° field of view (FoV). (Bottom) Each agent applies MultiSoc on a graph of its environment (limited to its FoV) with each entities (human/robot) as a node.

In this work, we propose a model for the learning of multi-robot navigation strategies within crowded environments (cf. Fig.1 (Top)). Main challenges compared to the state of the art are learning coordinated and human-safe navigation strategies for the fleet of robots and managing interactions with both controlled entities (robots) and uncontrolled entities (humans). In our contribution, we highlight the similarity between two approaches that utilize Graph Neural Networks (GNN) [9] to represent, on one hand, human interactions in single-robot social navigation [7], and on the other hand, interactions between agents in multi-robot navigation [10]. Thus GNNs offer a bridge between these two fields which we

[1]Erwan Escudie is with Univ Lyon, LIRIS, UMR5205, CITI Lab., INRIA-INSA Chroma team, Villeurbanne, France
[2]Laetitia Matignon is with Univ Lyon, UCBL, CNRS, INSA Lyon, LIRIS, UMR5205, F-69622, France. laetitia.matignon@univ-lyon1.fr
[3]Jacques Saraydaryan is with CPE Lyon, CITI Lab., INRIA-INSA Chroma team, Villeurbanne, France. jacques.saraydaryan@cpe.fr

leverage in our proposed model, named MultiSoc. MultiSoc uses two GNNs combined with attention mechanisms. First an edge-selector produces a sparse graph of the most interesting interactions between entities; then a crowd coordinator applies node attention to produce a graph representing the influence of each entity on the others. MultiSoc follows Centralized Training Decentralized Execution (CTDE) paradigm. During the learning process with a multi-agent RL algorithm, the model is shared between robots, taking benefit of each robot experience. But at the execution, each robot processes its input through its MultiSoc model and gets as result its action (commands in velocity) (cf. Fig. 1 (Bottom)). The input is a directed graph with information (current and predicted future poses) concerning the entities (robots and humans) in the field of view (FoV) of the robot.

We present empirical results obtained with a multi-agent social navigation simulator that we implemented building upon an existing single-agent one [7]. Our MultiSoc model overcomes the main baseline in deep RL social navigation especially when several robots are involved. Results also demonstrate (i) a better generalization of our model even in more balanced crowd (as many robots as humans) or with heterogeneous human policies; (ii) scalability capacities with different proportions of humans and robots between training and testing; (iii) the usefulness of the density factor we introduced to adapt to the neighborhood density.

The main contributions of this paper are as follows. (i) We propose the first (as far as we know) graph-based interactions model for **multi**-robot social navigation (ii) We introduce a customizable meta-parameter to **adjust the neighborhood density** to take into account in each robot navigation strategy (iii) The experiments demonstrate that our model enables **efficient multi-agent implicit coordination** in challenging crowd navigation and is able to deal with **heterogeneous human policies**.

II. Related Works

A. Social robot navigation

Social robot navigation has inspired a significant amount of research [1]. Early methods [11], [12], [13] consider humans as dynamic but non-responsive obstacles resulting in shortsighted and unnatural robot behaviors. Others plan robot motion conditioned on the predicted future trajectories of humans but suffer from the "freezing robot problem" [14].

One solution is to couple planning and prediction but these suffer from computational intractability [1].

A set of recent approaches address these coupled models with **deep RL**. Efficient robot policies can be trained with interaction awareness encoded in the reward function. The complexity of the coupled models is then transferred in the training. These approaches differ in their way of modeling crowd interactions. Previous ones ignore human-human (H-H) interactions and consider a limited number of robot-human (R-H) interactions [15], [16]. Thus their performance degrades in dense and highly interactive crowds. Most of the following works use **graph-based models** to extract efficient representations of the crowd, and attention mechanisms [17] to infer the relative importance of each interaction. In the graph representation, the nodes represent robot or humans and the edges relations between them. DS-RNN [6] uses a spatio-temporal graph to capture spatial R-H interactions and temporal interactions in the robot's own trajectory. Spatial and temporal interactions are represented with Recurrent Neural Networks (RNN) and attention weights are assigned to spatial edges to infer the most pertinent human neighbors. However, the number of humans is fixed for each learnt model and H-H interactions are not considered. Besides, it suffers dramatic loss when the robot encounters too many kind of obstacles.

Others employed GNNs to learn interactions between the entities. RGL [18] combines Graph Convolutional Networks (GCN) with relational graph learning but does not distinguish between H-H and R-H interactions. G-GCNRL [5] learns human-like attention weights with a GCN trained with human gaze data.

These weights are incorporated into the adjacency matrix of a second GCN containing the policy network of the RL architecture. Lately, Attention-Based Interaction Graph (AttnGraph) [7] emphasizes crowd analysis by prioritizing H-H interactions. Attention mechanisms based on Graph Attention Network (GAT) are applied, first between humans to balance each human trajectory with the others; secondly to include the robot in the analysis. It is interesting to note that this method uses human trajectory predictors, some of which are also based on GNNs [19], [3].

Although the impressive results of AttnGraph compared to other deep RL models in social robot navigation, it is worth mentioning, according to the authors, that it remains difficult to train. Moreover, no approaches consider **multi-robot** navigation with deep multi-agent RL to our knowledge. On its part, AttnGraph architecture explicitly separates the robot from humans and cannot easily be extended to accommodate multi-robot learning.

B. Multi-agent deep reinforcement learning

Significant developments have been made in the field of multi-agent deep RL, analysed in recent surveys [20], [21]. In our context of multi-robot navigation in a crowd, we will focus solely on cooperative approaches. Some solutions are based on value decomposition in CTDE paradigm, as in [22] where global Q-value is decomposed based on individual Q-values. QMIX [23] adds a constraint of monotonicity on the type of function used to merge the Q-values. Concerning policy-gradient algorithms, [24] suggest that Multi-Agent PPO (MAPPO) (cf. §III-C.1) can also be a competitive baseline for cooperative multi-agent RL tasks. A parallel line of work is based on **coordination graphs**. In [25], a purely abstract graph represents each agent as a node, but the relations of each pairs are handled by MultiLayers Perceptrons (MLPs), which is restrictive. Instead, GNNs have been used in multi-agent RL to enhance the cooperation among agents, leveraging the graph whose nodes represent agents' information. In [26] a GCN allows to expand the number of neighbors included in the calculation. DICG [27]

is very similar, except that the structure of the graph is dynamically calculated with attention mechanisms. MAGE-X [10] (detailed in §III-A.3) uses GCN paired with Gumbel Softmax [28], [29] instead of attention to produce a dynamic discrete graph, more realistic for communication than attention.

III. Contribution

We first briefly recall relevant background on specific neural networks used in our model and on two related works at the foundations of our contribution. Then our MultiSoc model is presented in detail as well as the applied RL approach. In the following, agents refer to the robots and entity refers to robot or humans.

A. Preliminary

1) Graph Neural Network: Graph neural network (GNN) [9] is a neural network designed specifically to process and analyze data structured as a graph. Different architectures have emerged, of which the most well-known are GCN [30] as convolutional methods and GAT [31] as spatial methods. Both can be formalized similarly by aggregating information between neighboring nodes. The only difference is the calculation of the coefficients used in the aggregation process. Coefficients are static and directly dependent of the structure of the graph in GCN, while they are calculated dynamically with attention mechanism between neighboring nodes in GAT. This allows more flexibility but at the cost of greater calculation.

2) Gumbel Softmax Transformer: Gumbel Softmax Transformer (GST) [3] is a neural network used for pedestrian motion prediction. Composed of 3 consecutive parts, only the first one, the **Edge Gumbel Selector**, will draw our attention. It computes a dynamical, complete and directed graph of the interactions between pedestrians, with their positions as nodes. A multi-head attention (MHA) produces attention between each possible edges and offers the possibility to connect each nodes with at most N_{head} other nodes, where N_{head} is the number of heads of MHA. The gumbel softmax trick allows to perform discrete clustering compatible with gradient descent. By choosing a low number of heads, we can then produce a sparse graph and vice versa. Tests on different types of crowds show significant differences depending on the number of heads and the density of the crowd.

3) MAGE-X/AttnGraph: We now detail two GNN-based models to better highlight their commonalities and differences, summarized in Fig. 2.

AttnGraph [7] focuses on mono-robot social navigation. It first uses a trajectory predictor (e.g. constant speed trajectory or more sophisticated ones as GST [3]) to dissipate uncertainty in the process. Then a complete graph is constructed, focusing only on the humans and ignoring the robot: each node is composed of the consecutive future positions of each human and each edge represents the visibility between humans. This graph is submitted to an attention module (first GAT). This produces a new graph, with structure defined by attention and features influenced by each neighbors (middle graph in Fig. 2 (Left)). Then, the robot is integrated in the graph submitted to an other GAT, presented in 2 blocks in Fig.2 (Attn + Aggreg) to highlight the parallelism with MAGE-X.

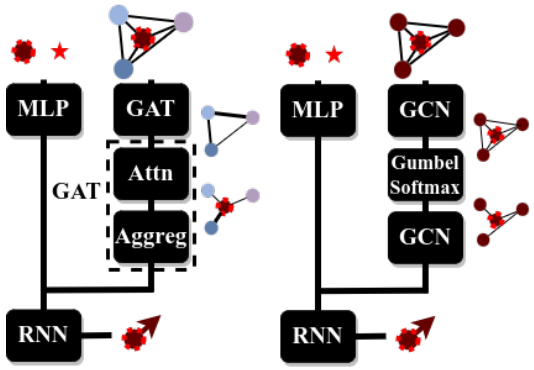

Fig. 2. Oversimplified architectures of AttnGraph [7] and MAGE-X [10]. Agents (robots) are in red and agent of interest is surrounded by dotted line. (left) AttnGraph : At the end of each bloc is represented the graph actually computed and attention (edges width). (right) MAGE-X : At the end of each bloc is represented the graph actually computed.

Thus, the first GAT focuses on the analysis of the interactions between humans. The second GAT focuses on the robot and its links to the crowd (second graph with star structure centered on the robot in Fig. 2 (left)). Finally, the node of the robot is extracted and passes through a RNN, producing the action.

MAGE-X [10] focuses on multi-agent navigation problems. First agents goals are chosen in a centralized way before the navigation. This initial centralized assignment reduces the difficulty of predicting other trajectories, as their behaviors are influenced right the beginning. Moreover, it allows to transform the multi-agent navigation problem to multiple single-agent navigation tasks, in which each agent is required to reach the designated goal while avoiding collisions with others. A comparison can here be made with AttnGraph, in which the only agent (robot) must navigate to its goal while avoiding collisions with others (humans).

During navigation, MAGE-X is decentralized. A complete graph with each agent positions as nodes is submitted to a first GCN (cf. Fig. 2 (Right)). Combined with a gumbel softmax, a discrete clustering on the edges is realized to produce a sparser graph keeping only the most important neighbors of the agent of interest.

We can note here that the edge selection operation is similar (but not identical) to the one performed by the Edge Gumbel Selector of GST. Then, a second GCN analyses the sparser graph and the node of interest is extracted, concatenated with the intrinsic parameters of the agent, to obtain robot action through a RNN.

Thus, as highlighted by Fig.2, the philosophy of these algorithms are close. Both used 2 GNNs and retract more and more the process on the agent/node of interest (encoded by the architecture in AttnGraph and with GCN combined with gumbel softmax for MAGE-X).

However, the nature of the entities made the differences unavoidable. AttnGraph analyses humans and agent sequentially, while MAGE-X can analyse every entities from the beginning because they have the same nature.

B. Our model

We present here our main contribution, MultiSoc, that is a model for learning multi-agent navigation among humans. MultiSoc can be seen as an homotopy between AttnGraph and MAGE-X. From the former, we keep the attention mechanism and the graph with predicted future positions of the entities. From the latter, we improve edge-selection and take up the graph merging early all the entities.

Moreover, unlike AttnGraph where entities ignore each other except for the only robot's consideration of humans, in multi-agent scenarios, interactions and visibilities among controllable agents are crucial for their coordination. That's why in MultiSoc, visibility among entities is critical, and the computation graph is based on this element. Indeed the algorithm has to merge both controllable and uncontrollable entities, all of them interacting with each other. MultiSoc workflow for each agent j is the following (cf. Fig. 3):

- The Edge-Selector applies attention on nodes of a graph composed of predicted positions of each entities in the FoV of agent j. This produces a sparse directed graph with adjustable density given N_{head}.
- The Crowd Coordinator, a GAT with one layer of attention, is applied on the sparse graph to compute node features influenced by neighbors. Meanwhile, the Intrinsic Coordinator produces a broader summary of the constraint applied on the robot (constraint of goal).
- Once the external constraints (Crowd Coordinator) have been correctly represented on the node representing the agent j, this node is extracted and concatenated with the constraint of goal (Intrinsic Coordinator).
- Then a GRU, followed by two MLPs, produces both value and action, with respect to the previous hidden state and the information previously computed.

From a technical perspective, the GNNs included in the architecture allows: (i) A flexible computation taking into account as many entities as wanted. It is worth noting that humans and agents are included in the same graph (and not in the architecture itself as does AttnGraph) (ii) A pseudo centralization in the decision acts on each agent in the graph (as node). By extension of the homogeneous paradigm, all other agents with enough information on their own observations, can be accurately understood by the concerned agent. (iii) An extendable receptive field increases the observation space from which the agent takes its action. As in [26], each GNN layer extends the information perceived and allows nodes to be connected indirectly to more nodes.

1) Input data:

a) Trajectory prediction: As in AttnGraph, the input information for each entity is a prediction of the trajectory. But instead of leaving the choice of the prediction method to the user, we systematically use the "constant speed" prediction. Because humans are highly unpredictable on the long term, a short-term prediction is enough. More importantly, AttnGraph results [7] do not provide great superiority of more complex prediction method for 5 timesteps trajectory. The constant speed method approximates the current speed of an entity by subtracting its current and previous positions. From that, the trajectory at constant speed and for 5 timesteps is calculated for each entity observed by the agent of interest.

b) Input Graph: For the rest of the section, N will be the total number of entities in the simulation (a mask being applied on entities to represent the partial observation).
We obtain, from the trajectory predictor, the approximated future trajectory $T_i = [p_i^t, ..., p_i^{t+5}]$ for each entity i with p_i^t the position of i at timestep t. Along trajectories, we obtain the intrinsic information $w^j = [p, v, g, \theta, r]$ of agent j with p its current position, v its speed, g its goal, θ its angular heading and r its radius. Each agent has a limited FoV (depth and angle) and has access to the visibility matrix $\mathbf{M^j} \in R^{N\times N}$ between each entity in its vision, defined by:

$$M_{i,k}^j = \begin{cases} 1 & \text{if } i \text{ sees } k \text{ and } i \text{ is seen by the agent of interest } j \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

For each agent j, a graph $G^j = (\mathbf{E}, \mathbf{M}^j)$ is then formed where $\mathbf{E}$ is the node set with features e_i of each node $i \in [1, N]$ equal to T_i and a label discriminating entities following their nature (e.g. $label \in \{robot, human\}$). $\mathbf{M^j}$ is the adjacency matrix for j. The state $s_t^j = [w^j, G^j]$ forms the input given to the MultiSoc model, as illustrated in Fig. 1 (Bottom) with a graph limited to each agent FoV.

2) Edge-Selector: The first component of MultiSoc is an Edge-Selector detailed in Fig. 4. It applies attention on nodes of G to produce a sparse directed graph keeping only the most interesting interactions between entities. Edge-Selector is a tractable adjustment of the Edge Gumbel Selector from GST [3], where the attention is applied on the set of possible edges ($N \times N$ edges with N the number of entities). For a supervised algorithm, this computation is not a limit but for RL, which learns online, the speed of the algorithm is crucial. Thus, we designed Edge-Selector based on the attention between nodes only. The remaining mechanisms are similar to the ones found in GST.
First, a MHA with N_{head} heads is applied between every nodes i (cf. Fig. 4 (Left)). For each head k, the input matrix $\mathbf{E}$ with features e_i of all nodes $i \in [1, N]$ is linearly transformed using learned weight matrices: $\mathbf{Q} = \mathbf{E}\mathbf{W}_Q, \mathbf{K} = \mathbf{E}\mathbf{W}_K, \mathbf{V} = \mathbf{E}\mathbf{W}_V$. Then, $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ are divided along the features axis into N_{head} parts such that $\mathbf{Q} = [\mathbf{Q}^1..\mathbf{Q}^{N_{head}}], \mathbf{K} = [\mathbf{K}^1..\mathbf{K}^{N_{head}}], \mathbf{V} = [\mathbf{V}^1..\mathbf{V}^{N_{head}}]$. The scaled dot-product attention is computed as follows for each head:

$$\mathbf{A}^k = Softmax(\frac{\mathbf{Q}^k(\mathbf{K}^k)^T}{\sqrt{d_n}})\mathbf{V}^k \tag{2}$$

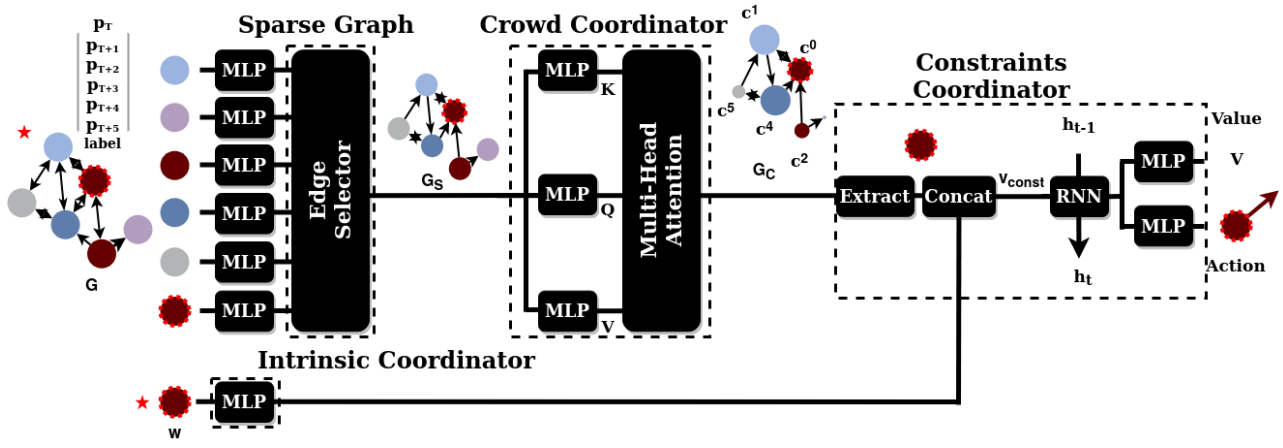

Fig. 3. Overview of the MultiSoc architecture. For the agent of interest (surrounded by dotted line), the input is its intrinsic information and a graph limited to its FoV. Each node of the graph is composed of the current and consecutive predicted positions of the observed entities, and by a label discriminating entities following their nature.

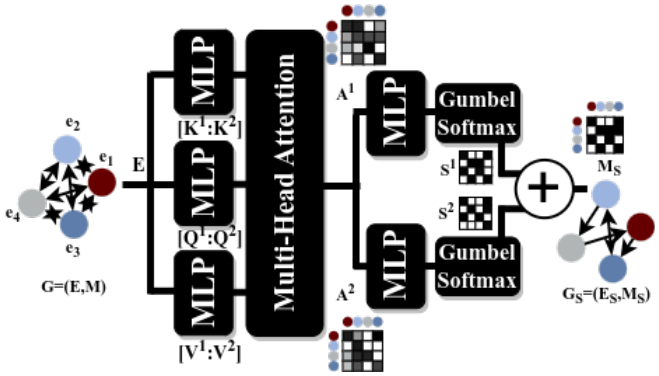

Fig. 4. Overview of the Edge-Selector architecture producing a sparse graph G_S with a MHA module with 2 heads.

where d_n is the dimension of the nodes. Thus MHA applied to all node features e_i of the graph G produces attention score $a^k_{i,j}$ between agents i and j for the k^{th} head.

As for GST, the mask $\mathbf{M_S}$ for the edges is calculated with Gumbel-Softmax (cf. Fig. 4 (Right)):

$$s^k_{i,j} = Softmax_j \frac{MLP(a^k_{i,j}) + g}{\tau}$$
$$m_{i,j} = \frac{1}{N_{head}} \sum_{k=0}^{N_{head}} s^k_{i,j} \quad (3)$$

where N_{head} is the number of heads, g is sampled from the Gumbel distribution, MLP is a linear layer, τ is the temperature parameter used in Gumbel-Softmax trick and $m_{i,j}$ are the coefficients of the adjacency matrix $\mathbf{M_S}$ produced as a mask on the edges. Thus, in the final sparse graph $G_S = (\mathbf{E}_S, \mathbf{M}_S)$:

- the feature of each node in $\mathbf{E}_S$ is the attention scores for agent i.
- there is an edge between the i^{th} and j^{th} nodes if $m_{i,j} \neq 0$. Moreover, **each node i has at most N_{head} edges**.

The Edge-Selector realizes a discrete clustering on the edges and allows to introduce the parameter N_{head} as a constraint of density on the dynamical graph. Greater N_{head} is, denser the graph will be.

3) Crowd Coordinator: Now that a sparse graph with entities in the FoV of the agent of interest has been calculated, we can propagate information between neighboring nodes to compute node features influenced by their neighbors.

Various approaches are used in the literature for this. DICG [27] , like MAGE-X [10], applied a GCN on the graph while AttnGraph [7] used a simple attention mechanism analogous to a GAT on a star graph. It has to be noted that an ablation study on MAGE-X proved that GCN could be replaced by attention mechanism without dramatic loss in accuracy (7% less success with this replacement [10]). We decide to use a GAT [31] as attention mechanism to analyse the sparse graph. This choice applies naturally on the directed sparse graph and allows us to remain consistent with AttnGraph, which we aim to expand for multi-agent social navigation.

Considering the sparse graph G_S produced by the Edge-Selector (cf. Fig. 4), with h_i being the features of the i^{th} node and $\mathbf{H}$ the matrix concatenating all h_i, the attention is defined by:

$$Attention(\mathbf{H}) = Softmax(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}) \quad (4)$$

where $\mathbf{Q} = \mathbf{H}\mathbf{W}_Q$, $\mathbf{K} = \mathbf{H}\mathbf{W}_K$, $\mathbf{W}_Q$ and $\mathbf{W}_K$ being learned weight matrices, d_k is the dimension of the node and

$$Softmax_j(d_{i,j}) = \frac{e^{d_{i,j}}}{\sum_{k \in N_i} e^{d_{i,k}}} \quad (5)$$

where $d_{i,j}$ is the (i,j) coefficient of $\frac{QK^T}{\sqrt{d_k}}$ and N_i is the neighborhood of the node i in the sparse graph (deduced from $\mathbf{M_S}$).

Each attention head k produces $\alpha^k_{i,j} = Attention^k(\mathbf{H})_{i,j}$.

Finally, the nodes are weighted summed together according to their neighborhood in the sparse graph. The crowd coordinator then produces a graph G_C with features c^i for each node i (cf. Fig. 3):

$$c^i = Concat_k(\sum_{j \in N_i} \alpha^k_{i,j} h_j \mathbf{W}_V) \tag{6}$$

where $Concat_k$ is the function concatenating all the vectors over the heads (and then merges the vectors produced by each head in an unique vector) and $\mathbf{W}_V$ is a learned weight matrix. It is possible to add an activation function as $\sigma = ReLu$ before concatenating over the heads and more importantly, it is also possible to add new layers. We do not add these supplements because the graphs are small enough in our problem (small field of view producing relatively small graphs).

4) Constraints Coordinator:

a) Extract and Concat: The crowd coordinator produces a graph representing the influence of each node on the others. In line with this idea, only the node of interest has to be kept. We can observe that, when we use a GAT with one layer, as we do, only the neighborhood of the node of interest has an influence on it. We can then keep only the edges between the agent and its neighborhood in the previous GAT, as it is done in AttnGraph. However, to be complete, we keep the general formulation and do not truncate any edges before the operation, keeping so possible generalisation.
Thus, we extract from G_C the node representing the agent of interest and concatenate it with the output of the intrinsic coordinator, which is simply an MLP. By doing so, we obtain both the constraints of the other entities on the agent of interest and the constraint of reaching the goal in one vector v_{const}.

b) RNN: We use a gated recurrent unit (GRU) to keep consistency with the previous action: $h^t = GRU(h^{t-1}, v_{const})$ where h^{t-1} is the hidden state produced at last timestep and v_{const} is the vector of constraints. The value and the action for RL are then produced by passing h^t through 2 different MLPs.

C. Reinforcement Learning

We model the scenario as a Multi-Agent Markov Decision Process. MultiSoc follows CTDE paradigm. The centralization part lies in the fact that all the agents are trained with the same MultiSoc model. But execution is decentralized as at each time-step, each agent passes through the MultiSoc model its intrinsic information and positions of entities in its FoV. This produces the action the agent will take. Thus, each agent receives a reward and the simulation transits to a next state according to an unknown state transition, taking into account humans and other agents actions.

1) MAPPO: Concerning the RL algorithm, AttnGraph [7] uses Proximal Policy Optimization (PPO) [32], a model-free policy gradient algorithm widely used.

The most direct heir of PPO in multi-agent paradigm is Multi-Agent PPO (MAPPO) [24] used by MAGE-X [10]. Even if adaptations to the multi-agent are observed in MAPPO (e.g. value normalization), it differs mainly from PPO by the parameters fine-tuning. Indeed, in multi-agent, it is observed [24] that neural network training is really sensitive to the number of epochs. To stay aligned with AttnGraph and MAGE-X, we also use MAPPO. The parameters will be close to the ones used by AttnGraph, the main difference will remain in the number of epochs, multi-agent being empirically better trained with fewer epochs.

2) Reward: We adopt the reward function used in AttnGraph [7] with some modifications to consider multi-agents. The penalty for being in the predicted path of an entity (human or agent) is:

$$\begin{aligned} r^{j,i}_{pred}(s_t) &= min_{k \in [1,5]}((\not\Vdash^{t+k}_j, i)\frac{r_c}{2^k}) \\ r^{j}_{pred}(s_t) &= min_{i \in [1,N]}(r^{j,i}_{pred}(s_t)) \end{aligned} \tag{7}$$

where s_t is the state of the agent of interest j, r_c the penalty for collision and $(\not\Vdash^{t+k}_j, i)$ indicates whether the agent j collided with the k^{th} predicted position of the entity i.
The potential based reward guides the agent j to approach the goal: $r^j_{pot} = -d^{j,t}_{goal} + d^{j,t-1}_{goal}$ with $d^{j,t}_{goal}$ the distance of j to its goal at timestep t.
The complete reward for agent j doing action a_t in state s_t is then:

$$R^j(s_t, a_t) = \begin{cases} r_c & \text{if j collides any entity} \\ r^j_{pot} + r^j_{pred} & \text{otherwise} \end{cases} \tag{8}$$

We emphasize that there is no reward when reaching the goal. Indeed an episode is not completed until all agents have either collided or reached their goals. Thus each agent that reaches its goal must wait the others to finish. If there was a reward for reaching a goal, one agent would wait the other with infinite reward.

IV. Experimentations

A. Simulation Environment

a) Simulator: We extend the CrowdNav[1] mono-agent simulator used in AttnGraph [7] for a multi-agent version. First, we improve the use of matrices especially concerning the visibility, which is central in our work. Second, we implemented our multi-agent simulator **MultiCrowdNav**[2] based on multi particle environments (MPE) [33], in which small particle agents must navigate and communicate. It facilitates greatly the migration from PPO to MAPPO, as MAPPO already supports MPE environments.

b) Crowd Simulation: The simulation of human interactions is essential for agent learning in a context of social navigation. A complete model requires a huge amount of information to be taken into account, and becomes computationally intractable. Despite their lack of realism, methods such as ORCA [12] or social force (SF) [11] are often used to simulate humans (for learning and testing). It's also

[1] https://github.com/Shuijing725/CrowdNav_Prediction_AttnGraph
[2] Code will be available online if the paper is accepted.

worth noting, according to [1] (§4.3.1), that ORCA is mostly used for testing, while social force brings more adversity to the agent. The combination of the two would seem to be a prerequisite if we are to entertain the idea of a real-life implementation. Thus in our simulator each human can be controlled by ORCA or SF and some experiments will be done with heterogeneous human policies.

c) Scenario: The scenarios are initialized with H humans arranged on a circle with some noise on their positions. Human goals are chosen so that they must cross the circle to reach an opposite point. Humans react only to other humans but not to robots (adversarial crowd). A new random goal is assigned to a human as soon as it reaches its goal. At initialisation, R agents are also laid out randomly with their own goals (positioned and assigned randomly[3]). When an agent collides with another entity (human or agent) or reaches its goal, it keeps moving but no longer counts in the metric. Therefore, it is still considered as a moving obstacle for the remaining agents (considered in their penalty reward (cf. eq. 7)). An episode is over when all agents have either collided (collision) or reached their goal (success). For more details concerning the simulation (kinematics, action space, sensor range, ...) the reader can refer to [7].

d) Metrics: Our metrics include navigation and social metrics traditionally used in multi-robot and social robot navigation. The **success rate** is the number of agents that reached their goals to the total number of agents, evaluated on all test episodes.

Safety is mainly summarized by the **collision rate**, i.e. the number of agents colliding with other entity (humans or not). Once an agent has reached its goal, if at least one other agent has not yet reach its goal, collisions are no longer counted in the score for a succeeded agent. The proximity of the agent to humans can also be considered to evaluate the social awareness of the agents. We used the **intrusion ratio** as the percentage of time the agent was "too close" to a human averaged over all episodes ("too close" is defined as in [7] with a distance defining the space "close" to an individual). To compare relative performance between algorithms without comparing to a theoretical optimal solution that is not available, several criteria are defined. **Travel time** and **travel length** are mean duration, resp. length, of the trajectories between the initial position to the goal (or ending position if not reached) for all agents. The **reward** is the mean reward obtained by each agent at the end of episode.

B. Results

We now present and analyze results obtained with MultiSoc in a set of various scenarios (number of agents/humans) and compare it with baseline (AttnGraph). Models are trained during N_{train} timesteps. Training hyperparameters can be found in the Appendix. Evaluations are conducted on N_{test} random unseen episodes, each consisting of 150 timesteps.

[3]Unlike MAGE-X, agents can not exchange their goals with each other at the beginning of the episode.

TABLE I

BASELINE COMPARISON. $N_{train} = 20M$ OF TIMESTEPS AND $N_{test} = 1000$ EPISODES (150 TIMESTEPS) WITH $seed = 1000$. DURING TRAINING AND TEST ORCA POLICY IS USED FOR HUMANS NAVIGATION. REWARD METRIC IS NOT GIVEN FOR ATTNGRAPH BECAUSE IT IS NOT COMPARABLE TO MULTISOC (ATTNGRAPH USES A REWARD FOR REACHING THE GOAL).

Models	Success ↑	Collision ↓	Intrusion ↓ Ratio	Travel ↓ Time	Travel ↓ Length	Reward ↑	Train R	Train H	Tests R	Tests H
Attngraph[4]	0.92	0.05	6.51	15.47	**13.99**	-	1	20	1	20
MultiSoc	**0.96**	**0.03**	**4.33**	**14.70**	14.79	**19.10**				
Attngraph[4]	0.85	0.13	5.84	15.81	**14.38**	-	1	20	5	20
MultiSoc[5]	**0.94**	**0.04**	**3.42**	**15.35**	15.87	17.24	1	20		
AttnGraph[4]	0.68	0.31	11.92	16.39	14.49	-	1	20		
MultiSoc	0.81	**0.02**	**3.98**	16.62	20.02	**5.91**	1	20	6	6
MultiSoc	**0.85**	0.14	13.61	**12.8**	**12.28**	-1.85	5	20		

Videos and simulation screenshots showing the behavior learned by the agents with MultiSoc in different scenarios are available in supplementary materials.

1) Baseline comparison: We first compare our MultiSoc model to Attngraph [7] that, as far as we know, actually overcomes other deep RL models in mono-robot navigation in crowd environment. First, the modifications made to the simulator and the paradigm shift from mono-agent to multi-agent RL brings about a preliminary improvement in terms of learning time. Training Attngraph on 20 Millions of timesteps with CrowdNav simulator actually takes 40h (average) in our training condition (4 Xeon E5-2640v3 CPUs, with 32GB of memory and one NVIDIA GK210 GPU), whereas training Attngraph with our new simulator MultiCrowdNav costs no more than 20h (average) in the same conditions.

Second, as shown in Table I, our **MultiSoc model overcomes Attngraph especially when several robots are involved**.

In a first test, both are compared during a learning and training phase with 1 robot and 20 humans. The results confirm the similarity between MultiSoc and AttnGraph in mono-robot conditions, as they are similar under the same conditions. The better performance of MultiSoc does not allow us to decide on any superiority. AttnGraph is very hard to train, while MultiSoc suffers from the same single-agent limitations. Nevertheless, the two models remain close in their results.

Several robots are introduced during test phase only in a second experiment (5 robots and 20 humans). Each robot executes individual trained Attngraph with other robots con-

[4]The Attngraph used is a pre-trained version of the model provided by authors.

[5]This model was trained over 30 Millions of timesteps.

[6]More results are given in the Appendix.

TABLE II

MULTISOC MODEL QUALIFICATION. GREY CELLS ARE VARIABLES VARIABILITY, BETTER RESULTS ARE IN BOLD, R AND H STAND FOR NUMBER OF ROBOTS AND HUMANS, $H-Policy$ IS THE HUMAN NAVIGATION POLICY DURING TESTS (ORCA IS STILL USED DURING TRAINING). TESTS ARE DONE ON N_{test} EPISODES OF 150 TIMESTEPS WITH $seed = 1000$.($60k$ TEST TIMESTEPS MEANS $N_{test} = 400$ AND $150k$ MEANS $N_{test} = 1000$).

	Metrics							Training Conditions				Test Conditions			
	Success ↑	Collision ↓	Intrusion ↓ Ratio	Travel ↓ Time	Travel ↓ Length	Reward ↑		N_{head}	N_{train} timesteps	R	H	Test timesteps	R	H	H-Policy
1. Pure Multi-agent scalability	**0.91**	**0.03**	**1.1**	**12.27**	**13.81**	**18.75**		4	20M	3	0	150k	3	0	ORCA
	0.81	0.12	4.06	15.17	16.86	8.55				3	0		10	0	
	0.59	0.07	7.26	17.78	19.46	-4.54				3	0		20	0	
	0.92	**0.00**	0.75	**11.83**	**12.29**	**20.47**				10	0		3	0	
	0.94	**0.00**	**1.92**	13.54	13.49	18.29				10	0		10	0	
	0.89	0.01	4.31	15.76	16.20	13.2				10	0		20	0	
2. Human Policy Robustness	0.92	0.07	7.60	13.59	13.18	14.96		4	20M	3	17	150k	3	17	ORCA
	0.94	**0.06**	**7.59**	13.45	13.12	**15.35**									ORCA+SF
	0.66	0.33	8.27	**12.32**	**10.66**	5.47									ORCA+FoV
3. Scalability	**0.96**	**0.03**	**4.33**	14.70	14.79	**19.07**		4	20M	1	20	150k	1	20	ORCA
	0.95	**0.03**	8.94	14.33	13.82	17.67				5	20		1	20	
	0.89	0.08	6.79	**14.22**	**13.68**	14.0				5	20		5	20	
	0.89	0.08	6.95	14.85	14.40	12.68				5	20		10	20	
4. Density Factor	0.88	0.05	3.39	18.10	18.93	15.78		2							
	0.91	**0.02**	**2.39**	**17.36**	**17.30**	**17.95**		4	20M	5	15	60k	5	15	ORCA
	0.86	**0.02**	2.83	18.38	17.86	17.46		8							

sidered as humans. In such condition, the success rate of Attngraph decreases to 0.85% showing the model's difficulty in handling a heterogeneous crowd mixed of robots and humans. Despite the fact that our MultiSoc was trained with only one robot, its still remains resilient (0.94% of success) when several robots are involved. This result shows **a better generalization of our model when dealing with a crowd composed of robots and humans**.

Finally, in a third test, a crowd composed of 6 robots and 6 humans is used during testing. In such condition, the performance of Attngraph drops down to 0.68% of success. This is mainly due to the difficulty of Attngraph to predict behavior of other robots. This leads to "panic" situations and weird behavior when a robot encounters other robots. Regarding MultiSoc, coordinating multiple robots remains challenging in a more balanced crowd, but it maintains a high success rate whether it learns with 1 or 5 robots. More importantly, MultiSoc can properly transfer mono-agent learning to multi-agent problem, meaning that it can use its own behavior to predict similar entities, thanks to the labels on the node.

2) Multi-agent without humans: In Table VI.1, Multi-Soc has been trained on respectively 3 and 10 robots **without humans**, arranging agents and their goals randomly on the map. The multi-agent's great adaptability is particularly noteworthy for tests with 10 robots when model was trained with 3 robots (performance of 81% success rate for 12% collision rate). This is even more striking for a model trained with 10 robots and tested with 20 (low performance loss). We can hypothesise that this **scalability** is due to the management of trajectories by the neural networks. As observed in [1], deep-learning-based navigation methods suffer from short-range. This allows the presented network to focus only on immediate trajectories (i.e. fewer configurations than for longer-term trajectories). As a result, transfer is more efficient, since it's the immediate environment that impacts the strategies and not the overall number of agents, at the cost of a loss of "centrality".

3) Social (Multi-agent with humans): The results presented in Table VI.2 illustrate **heterogeneous human policies management of MultiSoc**[6], which is a flaw of current works on social robot navigation according to [1] (§4.3.1). MultiSoc is trained with several agents, several humans and tested under various social conditions. **The architecture handles the mix of human policies very well** even though training was done with homogeneous human policies (ORCA). Indeed tests mixing ORCA and Social Force (ORCA+SF) do not bring any additional difficulties to the model (94% of success rate). However, having a reduced field of view for humans (ORCA+FoV) remains a clear limitation. In fact this situation creates a very complex crowd, far removed from the crowd on which the models were trained and brings more unpredictable human behaviors.

The results presented in Table VI.3 further demonstrate the **scalability capacities of MultiSoc**. Trainings with different numbers of robots delivered stable performance, but it's worth noting that a given training can also adapt to situations with more robots. The results demonstrate that training with 5 robots achieves very good results in single-robot (95% success rate), but also good results with 10 robots (89% success rate). **The scalability observed in the pure multi-agent case transfers well to the social case.**
These results also open up new possibilities for the single agent case. Indeed, AttnGraph suffered from the difficulty of reproducing the model's learning process (according to the authors of AttnGraph and us). MultiSoc suffered from the

same issue in single-agent mode, but multi-agent training is empirically more stable and faster. Models converge faster at the cost of a slightly longer computation time, since the MultiSoc model is fed N times more than in single-agent (where N is the number of agents) and with more "balanced" data concerning positions, since the agents are far from each other. This results in more stable, less biased, faster training for robust single-agent transfers.

Finally, the results in Table VI.4 illustrate the **usefulness of the density factor** (N_{head}) retrieved from the Edge Selector of GST [3]. As noted previously, AttnGraph has not proven the benefit of its use of GST. We hypothesize that the complete graph used throughout AttnGraph lost the essential information carried by GST, that is the human link management. By extracting only a short-term forecast, we assume that AttnGraph lost the deeper analysis, leaving only a too-compact summary. Using a simplified version of the Edge Selector allows us to better integrate it into the architecture. **Varying the density factor N_{head} allows to obtain different results for the same crowd**. The model performs best for N_{head}=4 (0.91% success rate) for the tested scenario, which is fairly consistent with our observations of the crowd, since agents will most often see fewer than 5 humans in their vicinity.

V. Conclusion

In this paper we propose MultiSoc, a new method for learning multi-robot socially aware navigation strategies. MultiSoc leverages graph-based representations combined with attention mechanisms to capture heterogeneous interactions in the crowd and various influences of each entity on the others. Especially the Edge Selector we integrate in our model allows to take into account the crowd density. The experiments show that our MultiSoc outperforms the main baseline and deals with heterogeneous human policies. It also demonstrate MultiSoc scalability capacities and the usefulness of the density factor we introduced.

Future work needs to investigate how our model deals with a complex environments including obstacles such as sparse environment, corridor, room, ... Moreover, interacting with a crowd with different behaviors (e.g cooperative, adversarial, social groups) may affect the performance of our model and should be studied. Finally, testing our social navigation model on real robot fleet in a real crowd situation remains essential to validate our model.

Acknowledgement — This work has been partially funded by ANR project DeLiCio (ANR-19-CE23-0006-DA).

References

[1] C. Mavrogiannis, F. Baldini, A. Wang, D. Zhao, P. Trautman, A. Steinfeld, and J. Oh, "Core Challenges of Social Robot Navigation: A Survey," ACM Transactions on Human-Robot Interaction, vol. 12, no. 3, pp. 1–39, 2023.

[2] Y. Han, R. Tse, and M. E. Campbell, "Pedestrian motion model using non-parametric trajectory clustering and discrete transition points," CoRR, 2020.

[3] Z. Huang, R. Li, K. Shin, and K. Driggs-Campbell, "Learning sparse interaction graphs of partially detected pedestrians for trajectory prediction," IEEE Robotics and Automation Letters, vol. 7, no. 2, pp. 1198–1205, 2022.

[4] B. Zhou, X. Tang, and X. Wang, "Learning collective crowd behaviors with dynamic pedestrian-agents," International Journal of Computer Vision, 2014.

[5] Y. Chen, C. Liu, B. E. Shi, and M. Liu, "Robot navigation in crowds by graph convolutional networks with attention learned from human gaze," IEEE Robotics and Automation Letters, vol. 5, no. 2, pp. 2754–2761, 2020.

[6] S. Liu, P. Chang, W. Liang, N. Chakraborty, and K. Driggs-Campbell, "Decentralized structural-rnn for robot crowd navigation with deep reinforcement learning," in 2021 IEEE International Conference on Robotics and Automation (ICRA), 2021, p. 3517–3524.

[7] S. Liu, P. Chang, Z. Huang, N. Chakraborty, K. Hong, W. Liang, D. McPherson, J. Geng, and K. Driggs-Campbell, "Intention aware robot crowd navigation with attention-based interaction graph," in IEEE International Conference on Robotics and Automation (ICRA), 2023, pp. 12 015–12 021.

[8] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," p. 6000–6010, 2017.

[9] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The Graph Neural Network Model," IEEE Transactions on Neural Networks (TNN), vol. 20, no. 1, pp. 61–80, 2009. [Online]. Available: http://ieeexplore.ieee.org/document/4700287/

[10] X. Yang, S. Huang, Y. Sun, Y. Yang, C. Yu, W.-W. Tu, H. Yang, and Y. Wang, "Learning graph-enhanced commander-executor for multi-agent navigation," in Proceedings of AAMAS, 2023, p. 1652–1660.

[11] G. Ferrer, A. Garrell, and A. Sanfeliu, "Robot companion: A social-force based approach with human awareness-navigation in crowded environments," 11 2013, pp. 1688–1694.

[12] J. van den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in Robotics Research, 2011, pp. 3–19.

[13] J. van den Berg, M. Lin, and D. Manocha, "Reciprocal velocity obstacles for real-time multi-agent navigation," in 2008 IEEE International Conference on Robotics and Automation, 2008, pp. 1928–1935.

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

[15] 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), 2017, p. 1343–1350.

[16] M. Everett, Y. F. Chen, and J. How, "Collision avoidance in pedestrian-rich environments with deep reinforcement learning," IEEE Access, vol. PP, pp. 1–1, 01 2021.

[17] A. Vemula, K. Muelling, and J. Oh, "Social attention: Modeling attention in human crowds," 2018.

[18] C. Chen, S. Hu, P. Nikdel, G. Mori, and M. Savva, "Relational graph learning for crowd navigation," in IROS, 2020.

[19] Y. Huang, H. Bi, Z. Li, T. Mao, and Z. Wang, "Stgat: Modeling spatial-temporal interactions for human trajectory prediction," in 2019 IEEE/CVF International Conference on Computer Vision (ICCV), 2019, pp. 6271–6280.

[20] W. Du and S. Ding, "A survey on multi-agent deep reinforcement learning: From the perspective of challenges and applications," Artif. Intell. Rev., vol. 54, no. 5, p. 3215–3238, jun 2021.

[21] S. Gronauer and K. Diepold, "Multi-agent deep reinforcement learning: A survey," Artif. Intell. Rev., vol. 55, no. 2, p. 895–943, feb 2022.

[22] P. Sunehag, G. Lever, A. Gruslys, W. M. Czarnecki, V. Zambaldi, M. Jaderberg, M. Lanctot, N. Sonnerat, J. Z. Leibo, K. Tuyls, and T. Graepel, "Value-decomposition networks for cooperative multi-agent learning based on team reward," in Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems, ser. AAMAS '18. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2018, p. 2085–2087.

[23] T. Rashid, M. Samvelyan, C. S. De Witt, G. Farquhar, J. Foerster, and S. Whiteson, "Monotonic value function factorisation for deep multi-agent reinforcement learning," J. Mach. Learn. Res., vol. 21, no. 1, jan 2020.

[24] C. Yu, A. Velu, E. Vinitsky, Y. Wang, A. M. Bayen, and Y. Wu, "The surprising effectiveness of ppo in cooperative multi-agent games," in Neural Information Processing Systems, 2021.

[25] W. Böhmer, V. Kurin, and S. Whiteson, "Deep coordination graphs,"

in Proceedings of the 37th International Conference on Machine Learning, ser. ICML'20. JMLR.org, 2020.
[26] J. Jiang, C. Dun, T. Huang, and Z. Lu, "Graph Convolutional Reinforcement Learning," arXiv e-prints, p. arXiv:1810.09202, Oct. 2018.
[27] S. Li, J. K. Gupta, P. Morales, R. Allen, and M. J. Kochenderfer, "Deep implicit coordination graphs for multi-agent reinforcement learning," in Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems, ser. AAMAS '21. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2021, p. 764–772.
[28] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with gumbel-softmax," 2017. [Online]. Available: https://arxiv.org/abs/1611.01144
[29] D. B. Acharya and H. Zhang, "Weighted graph nodes clustering via gumbel softmax," CoRR, vol. abs/2102.10775, 2021. [Online]. Available: https://arxiv.org/abs/2102.10775
[30] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in 5th International Conference on Learning Representations, ICLR 2017, 2017.
[31] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph Attention Networks," International Conference on Learning Representations, 2018. [Online]. Available: https://openreview.net/forum?id=rJXMpikCZ
[32] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," CoRR, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347
[33] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments," arXiv e-prints, 2017.

APPENDIX

Training hyperparameters

In Table III are given the common training hyperparameters we used for MAPPO [24]. Other hyperparameters specific to MultiSoc can be found in Table V. The detailed architecture of MultiSoc is shown in Table IV.

common hyperparameters	value
nrolloutthread	16
numminibatch	2
episode length	50
data chunck length	50
num env steps	20 000 000
ppo epoch	5
gain	0
lr	4e-5
critic_lr	4e-5

TABLE III
MAPPO HYPERPARAMETERS

architecture hyperparameters	value
human_node_rnn_size	128
human_node_output_size	256
edge_selector_embedding_size	32
agent_embedding_size	64
human_node_embedding_size	64
human_human_edge_embedding_size	32
attention_size	64
human_node_input_size	3
human_human_edge_input_size	2
human_human_edge_rnn_size	256
edge_selector_emb_size	512
edge_selector_num_head	4
mha_emb_size	256
mha_num_head	8

TABLE IV
MULTISOC ARCHITECTURE HYPERPARAMETERS

MultiCrowdNav simulator

In Figure 5, screenshots of our simulator (in chronological order) are given for a scenario with 3 robots traveling among 20 humans. From left to right :

1- The initial situation with robots in green and their field of view (FoV) in a dashed circle is shown. The dashed blue line connects each robot to its goal (red star). Humans in the FoV of robots are in orange; others are in black. Predicted poses on 5 time steps are represented with 5 circles for each entity in the FoV of a robot. The trajectories followed by each entity are given with a solid line that fades over time.

2-3- Given that human goals are chosen so that they must cross the circle to reach an opposite point, we can observe here that all humans have converged in the center of the scene. In order to reach their goals, the robots will also have to traverse the central space by default, leading all entities to cross paths in the center of the environment. The robots will then have to manage challenging crowd navigation and implicit coordination with other entities to avoid collisions and intrusion in human safe space. We can observe that robots 0 and 1 decided to navigate around the high-density area. Especially robot 0 chose to navigate around on the side where it predicts humans won't be. As for robot 2, it traverses the crowd while locally managing interactions.

4- All three robots have successfully reached their objectives without collisions.

Additional Results

In Table VI are given additional results concerning **the heterogeneous human policies management of MultiSoc**.

Other hyperparameters	value
temperature at beginning	5
base temperature	0.05
min temperature	0.03
collision penalty r_c	-20

TABLE V
OTHER HYPERPARAMETERS

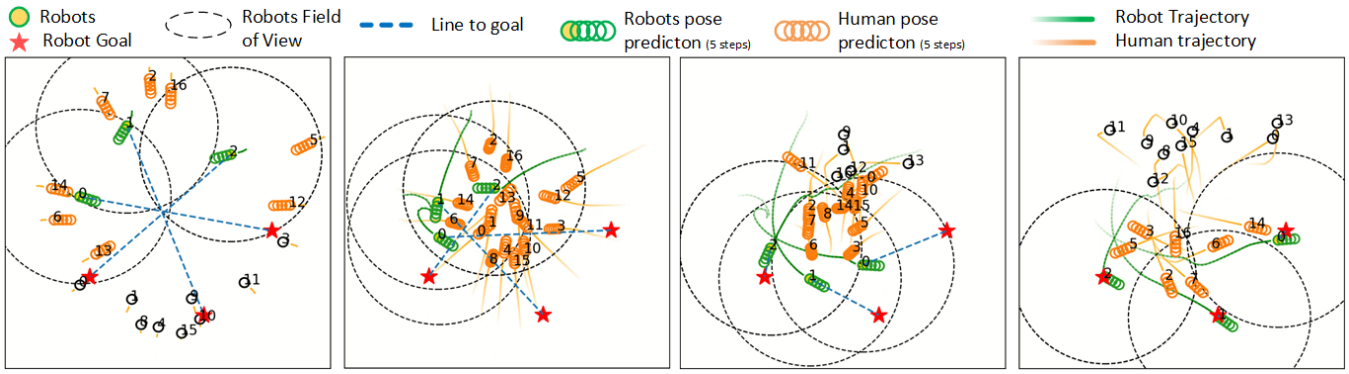

Fig. 5. Screenshots of a scenario (in chronological order from left to right) with 3 robots traveling among 20 humans in the MultiCrowdNav Simulator ($N_{head} = 4$).

TABLE VI

MULTISOC MODEL EXPERIMENTS FOR HUMAN POLICY ROBUSTNESS. GREY CELLS ARE VARIABLES VARIABILITY, BETTER RESULTS ARE HIGHLIGHTED (IN BOLD), R AND H STAND RESP. FOR NUMBER OF ROBOTS AND HUMANS, $H - Policy$ IS THE HUMAN NAVIGATION POLICY DURING TESTS (ORCA IS STILL USED DURING TRAINING). TESTS ARE DONE ON N_{test} EPISODES OF 150 TIMESTEPS EACH WITH RANDOM SEEDS. THUS $150k$ TEST TIMESTEPS MEANS $N_{test} = 1000$.

Metrics						Training Conditions				Test Conditions			
Success ↑	Collision ↓	Intrusion Ratio ↓	Travel Time ↓	Travel Length ↓	Reward ↑	N_{head}	N_{train} timesteps	R	H	Test timesteps	R	H	H-Policy
0.96	0.03	4.33	14.70	14.79	19.07								ORCA
0.98	**0.01**	**3.66**	15.31	15.41	**19.47**	4	20M	1	20	150k	1	20	ORCA+SF
0.69	0.31	9.17	**13.12**	**11.28**	13.71								ORCA+FoV
0.89	0.08	**6.79**	14.22	13.68	69.98								ORCA
0.92	**0.06**	7.05	14.58	14.11	**74.17**	4	20M	5	20	150k	5	20	ORCA+SF
0.67	0.31	6.8	**13.41**	**11.29**	27.27								ORCA+FoV