Enhanced Spatial Attention Graph for Motion Planning in Crowded, Partially Observable Environments

Weixian Shi Yanying Zhou Xiangyu Zeng Shijie Li Maren Bennewitz

***Abstract*— Collision-free navigation while moving amongst static and dynamic obstacles with a limited sensor range is still a great challenge for modern mobile robots. Therefore, the ability to avoid collisions with obstacles in crowded, partially observable environments is one of the most important indicators to measure the navigation performance of a mobile robot. In this paper, we propose a novel deep reinforcement learning architecture that combines a spatial graph and attention reasoning to tackle this problem. We take the relative positions and velocities of observed humans as nodes of the spatial graph and robot-human pairs as nodes of the attention graph to capture the spatial relations between the robot and the humans. In this way, our approach enhances the modeling of the relationship between the moving robot, static obstacles, and the people in the surrounding. As a result, our proposed navigation framework significantly outperforms state-of-the-art approaches [1], [2] in crowded scenarios when the robot has only a limited sensor range in terms of a reduced collision rate. Furthermore, we realize a seriously decreased training time by applying parallel Double Deep Q-Learning.**

I. Introduction

With the rapid improvement of artificial intelligence, mobile robots are not only required to move safely in a static environment but also navigate smoothly in human crowds, which is a challenging problem [3]. Recently, approaches based on deep reinforcement learning (DRL) have been proposed and achieved success in simulated scenes [1], [2], [4], [5]. However, most of the methods assume that all the humans are observable during the entire navigation process. To deal with realistic conditions, we aim at creating a action-learning-based model for robot navigation in only *partially* observable human crowds.

Robot navigation is a hard task mainly for the following reasons. First, navigation in human crowds is not a centralized problem, which means that the agent is not able to know the other agents' policies and goals. Instead, the agent can only estimate the information by prediction based on their observable states such as position, speed, etc. Second, an environment usually contains both static and dynamic agents. Each of them might interact with some of the others implicitly during the whole navigation process, meaning their goals might change at any time. Finally, since the environment is generally partially observable, the robot cannot obtain all humans' states, resulting in higher uncertainty on the environment modeling. Although robot crowd navigation is challenging, previous approaches have already achieved success in some aspects. Initial work [6]–[10] used hand-designed techniques to avoid collisions. These methods usually apply the one-step-look-ahead strategy to determine the next best action. However, they are too shortsighted and, thus, often yield sub-optimal solutions in complex scenarios. With the help of DRL, more recent work has lead to systems that can successfully navigate to target locations without any heuristics. These approaches can be roughly divided into two main branches: methods based on trajectory [11]–[13] and methods based on action learning [1], [2], [4], [5]. The former first predict the other agents' future trajectories, then make decisions based on the predictions. However, they are usually too computationally expensive for real-time applications. Methods based on action learning instead use collected observations to decide on the next action directly.

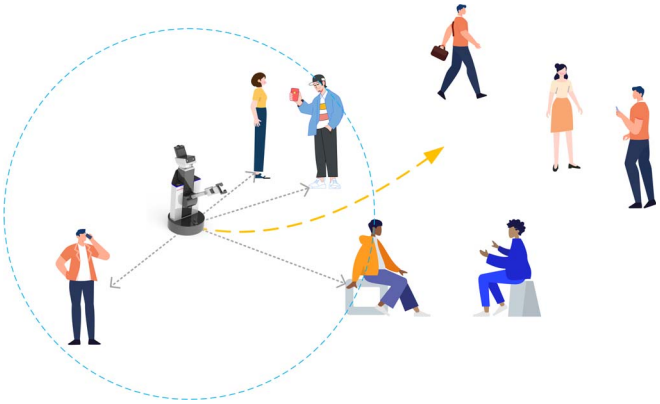


Fig. 1: Motivation for our work. During navigation, the robot can only observe the humans in the indicated sensor range. We process the relation between the robot and observed humans into two graphs: First, we treat the positions and velocities of the humans relative to the robot as nodes of the spatial graph. Second, we regard each robot-human pair as a node in the attention graph. By combining the two graphs in a deep reinforcement learning architecture, the robot's navigation behavior is significantly improved wrt. the state of art.

In this paper, we present a new method, called Enhanced Spatial Attention (ESA) graph, based on action learning for robot navigation in crowded, partially observable scenes. We model the high-level relationship with both spatial Long Short-Term Memory (LSTM) [14] and attention techniques [15], and train the network with Double Deep Q-learning (DDQN) [16], [17]. We first model the crowd navigation scenario as a decentralized spatial graph to encode the spatial relations between the robot and the other agents. Furthermore, we capture the importance of each robot-human interaction to form an attention graph. Finally, we combine both graphs to generate the robot's next action.

All authors are with the University of Bonn, Germany. This work has partially been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy, EXC-2070 – 390732324 – Phenorob and BE 4420/2-2 (FOR 2535 Anticipating Human Behavior).

Our main contributions are the following: (1) A novel DRL-based algorithm that uses a spatial graph as a parallel branch of the modified attention graph [1], resulting in a significantly lower collision rate and a higher success rate while keeping a similar navigation time compared to state-of-the-art methods [1], [2]. (2) By removing the imitation learning using ORCA [7] as the expert, and replacing deep V-Learning with parallel DDQN [16], [17], we eliminate the bad effect caused by the imperfect demonstration and seriously decrease the training time compared to [1]. The source code of our framework is available at: `https://github.com/weixians/esa`.

II. Related Works

The mobile robot's crowd navigation problem is complex since the environment is partially observable and the robot cannot directly acquire each human's implicit goal and behavior policy directly. Recent works for this problem can be roughly divided into three categories: methods based on reaction, methods based on trajectory, and methods based on action learning.

Reaction-based methods. Methods based on reaction for dynamic environments started many years ago. Reciprocal Velocity Obstacle [6] and ORCA [7], [8] design patterns under the assumptions of knowing each other's information and avoid collisions reciprocally. Social Force [9] model the crowd interaction relationships with attractive and repulsive forces. Also, Interacting Gaussian Process [10] models each agent's trajectory as an individual Gaussian process and then couples them. Yet these methods rely heavily on heuristics and they are often shortsighted and face the freezing robot problem, especially in more complex scenes.

In contrast, based on deep reinforcement learning, our approach does not need to design any heuristics and the agent is able to formulate the humans' implicit relations, encouraging the robot to consider long-term values, thus, it learns a better collision avoidance behavior and reduces the freezing robot problem.

Trajectory-based methods. Methods based on trajectory first try to predict the other agents' future trajectories, thus allow the planner to look into the future for decision making [11]–[13]. For example, by combining Gaussian process and RRT-Reach, RR-GP [11] learns a motion pattern model to identify probabilistically feasible paths and enables the vehicle to navigate more safely in a complex environment. Based on the predicted trajectories, agents can have a long-sight view to make better decisions. Chen *et al.* [18] presented a relational graph learning on the robot-crowd interactions using Graph Convolutional Network (GCN) [19]. Furthermore, Kretzschmar *et al.* [12] modeled humans' behavior in accordance with a mixture distribution to capture their discrete navigation decisions for exploring available trajectories to the goal position. However, this approach is computationally expensive especially when there are many other agents, as is needs to predict each individual's trajectory. Additionally, since the agent can only have partial observations of the environment the resulting strategies might be too conservative. In contrast to that, in our work we do not rely on a prediction about the other agents' trajectories but decides on actions directly based on the observations.

Methods based on action learning. Methods based on action learning are recently prevalent ways to achieve good performance in crowd navigation. CADRL [4] and SA-CADRL [5] were the earliest works replacing manual design techniques with DRL and achieve success in two-agent scenarios. However, They are still based on reciprocal assumptions and achieve limited results in complex decentralized scenarios. LSTM-RL [2] uses LSTM cells to encode other agents and combine the agent's state for planning. Chen *et al.* [1] proposed an attention-based mechanism to compute attention weights for all robot-human interactions in the environment and their approach, called SARL, makes decisions based on the attention scores. Taking supervised imitation learning as a warm start, Chen *et al.* also incorporate deep V-Learning for training the network which generates next best action with the help of one-step-look-ahead to calculate values for all possible actions. These methods, however, assume that the agent can observe all the agents in the surrounding, which is not reasonable for real-world environments. Furthermore, Deep V-Learning is much more time-consuming than Deep Q-Learning [16], [20].

To tackle the problems above, we propose a new policy network trained by DDQN [16] to navigate in partial observable environments. Further, we show that incorporating a spatial graph encoding spatial relations between the robot and humans and an attention graph in the network improves the performance in challenging crowded navigation scenarios compared to previous methods.

III. Our approach

In this section, we first describe the assumptions, the observations, and how we formulate the decision-making on the robot's actions for navigation in crowded scenes as a DRL problem. Then, we introduce our approach to model the crowd navigation scenario as a spatial graph and an attention graph. Finally, we explain how our architecture combines these two graphs.

A. Problem Formulation

Assumptions. Consider a holonomic robot navigating in a partially observable environment with other dynamic and/or static humans. We suppose all the agents moving in a 2D Euclidean space and assume that the robot's action command in terms of a velocity pair (v_x, v_y) can be executed instantly and accurately, meaning that the agent's velocity will change immediately when the action is executed. Moreover, the robot is assumed to be invisible to the humans since we assume the humans are moving according to the rule of ORCA [7], [8] and, thus, would react to the robot to avoid collisions. As a result the robot would only learn to move straight towards the goal position. Compared to previous work [1], [2], [4],

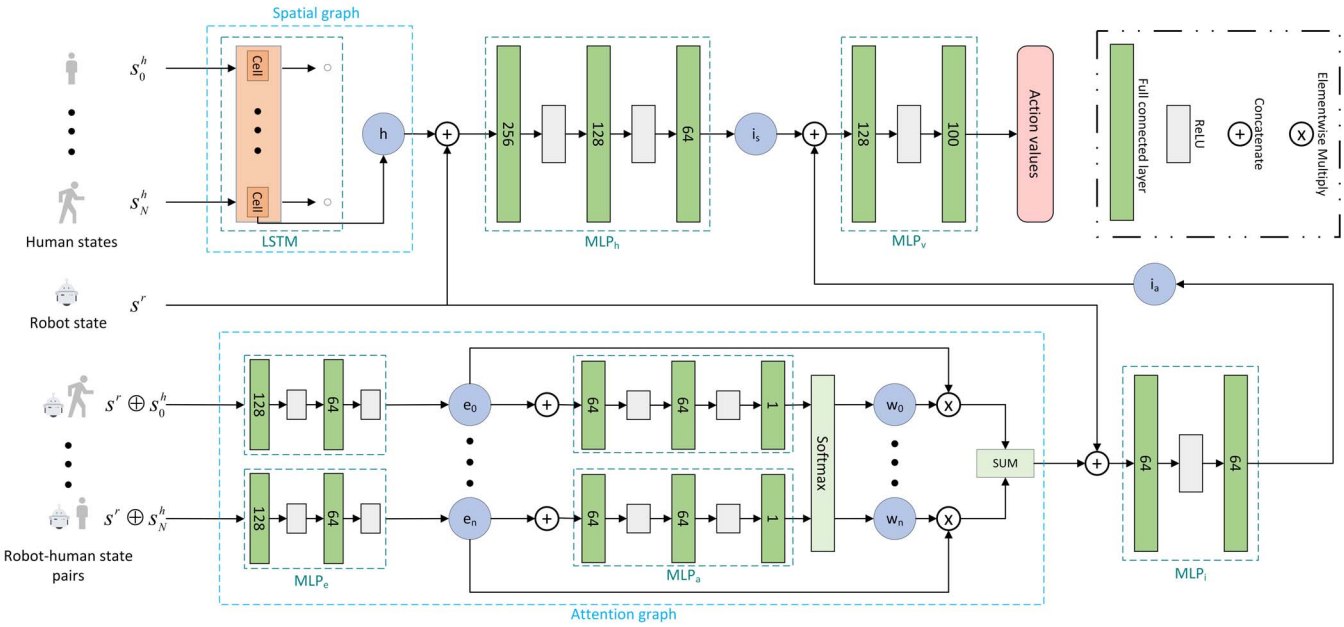


Fig. 2: Illustration of our neural network architecture. The spatial graph part of the network shows how we use an LSTM [14] to encode the spatial relations between the robot and other humans. We first sort each observed human by descending order of the weighted sum of current distance and possible future distance, then take their states (positions and velocities) as cell inputs into the Recurrent neural network. While in the attention graph part, we concatenate robot state and i-th human's state as inputs and formulate the interactions by the embedding network MLP$_e$ and attention network MLP$_a$. Finally, the network takes the two modules' output as inputs and offers action values as the action-making indicator.

[5], we do not assume that the robot can observe all other agents in the navigation process. Instead, we assume that the robot can only acquire states of humans located within a short sensor rage.

Observation. For training, we first transform the positions of all agents and velocities to robot-centric coordinates, where the x-axis points from robot's current position to its goal position. We consider the robot's current velocity (v_x^r, v_y^r), maximum speed v_{max}, and radius r as well as the current position (p_x^i, p_y^i), current velocity (v_x^i, v_y^i), and radius r^i of the i-th human. Let s_t^r be the robot state and $s_{i,t}^h$ be the state of the i-th human at timestep t as defined in the following, for simplicity, we omit the timestep t:

$$\mathbf{s}^r = [d_g, v_x^r, v_y^r, v_{max}, r] \tag{1}$$

$$\mathbf{s}_i^h = [p_x^i, p_y^i, v_x^i, v_y^i, d^i, r^i, r + r^i] \tag{2}$$

where d_g is the distance from robot position to its goal position, and d^i denotes the distance from the robot to the i-th human's position.

Formulation. The robot navigation problem can be formulated as a sequential decision making problem [1], [2], [4], [5]. We model the relationships between the robot and humans as a Markov Decision Process (MDP), defined by the tuple $\langle S, A, P, R, \gamma, S' \rangle$, where S is the state space, A is the action space, P is the probability transition function, R is the reward function, and γ is the discount factor which we take it 0.9 in our work. The state $\mathbf{s}_t = [s^r, s_0^h, ..., s_N^h] \in S$ represents the robot state and the N human states scanned by the robot range scanner at timestep t, where $N(\geq 0)$ might change at any timestep. When navigating in an episode, the robot starts at an initial state $s_0 \in S_0$. At each timestep t, the robot takes an action sampled from the learned policy $\pi(a_t|s_t)$ and transits to the next state on the basis of an unknown state transition function $P(s_{t+1}|s_t, a_t)$. At the same time, all other agents take actions and move to the next state according to their own policies. After an episode ends, we use Monte Carlo Value Estimation [21] to estimate the value of the state at each timestep in the episode. Finally, the optimal policy π^* can be formulated as below [1]:

$$\pi^* = \underset{\mathbf{a}_t}{argmax}[R(\mathbf{s}_t, \mathbf{a}_t) + \gamma \underset{\mathbf{a}'}{max}\, \mathbf{Q}^*(\mathbf{s}_{t+\Delta t}, \mathbf{a}')] \tag{3}$$

where $\mathbf{Q}^*$ and Δt denote the related optimal action-state-value function from DDQN [16] and the length of each timestep, respectively.

Reward Function. Following previous work [1], [2], [4], [5], we give a large award for successfully navigating to the goal position and punish the collisions or too close to other agents:

$$R_t(\mathbf{s}_t, \mathbf{a}_t) = \begin{cases} 1 & \text{if reach goal} \\ -0.25 & \text{else if } d_t < 0 \\ -0.1 + d_t/2 & \text{else if } d_t < 0.2 \\ 0 & \text{otherwise} \end{cases}$$

where d_t is the distance between the robot and the nearest human at timestep t.

Action space. To reduce the complexity and enable the robot moves smoothly, we divide the rotation range of 2π into

8 parts and its translational speed into 4 steps, and include (0,0) for stopping, so there are 33 choices for the robot's velocity (v_x, v_y).

B. Network Architecture

The proposed network architecture for our approach is depicted in Fig. 2. It consists of two main pipelines processing three different kinds of inputs: (i) states of the observed static and dynamic humans who are located in the robot sensor range, (ii) the robot's own state, and (iii) the concatenation of the robot state and each observed human state. The two pipelines are the spatial graph and attention graph on the observed humans.

Spatial Graph on Observed Humans. To process the spatial features of humans, we regard each pair of robot and human as nodes in the graph, and the edges represent the spatial relations, as shown in Fig. 3. Many methods based on action learning for local obstacle avoidance use feed-forward neural networks to process these spatial relations [4], [5]. However, they need to fix the size of the input or convert spatial relations as an occupancy grid map, and then use CNNs [22] and pooling. But grid maps can be coarse or computationally expensive if too precise. Here we use an LSTM [14] to handle the varying number of observed humans and make each human state $\mathbf{s}^h$ as the input of the LSTM cells in the reverse order of the distance to the robot:

$$\mathbf{h}_i = \mathit{LSTM}(\mathbf{h}_{i-1}, \mathbf{s}_i^h) \tag{4}$$

where $\mathbf{h}_i$ is the i-th hidden state of the LSTM network.

We take a balance between the current and estimated next distances based on the humans' current velocities as the criterion to sort the inputs, which is computed as

$$\begin{aligned} dist =& w_c({p'_x}^2 + {p'_y}^2)^{1/2} + \\ & (1-w_c)((p'_x + v'_x\Delta t)^2 + (p'_y + v'_y\Delta t)^2)^{1/2} \end{aligned} \tag{5}$$

where $w_c \in [0, 1]$ is a hyper-parameter to adjudge the importance between current distance to the robot and possible future distance. In our experiments, We set w_c to 0.8. In addition, we let the LSTM network remember all human states by keeping the output hidden state having a large dimension without spending much extra time. As shown in the upper part of Fig. 2, we first input the human states into the LSTM units by a reverse order based on the distance, then concatenate the LSTM output hidden state and the robot's self-state into a multi-layer perceptron (MLP) [23]. Finally, we get encoded spatial relationships $\mathbf{i}_s$ between the robot and observed humans.

Attention Graph on Observed Humans. Inspired by the work of Chen *et al.* et al. [1], we additionally build an attention graph on the observable humans to encode each pair of the robot-human relationship.

To handle varying number of scanned humans on each observation, we first apply a ReLU [24] activated non-linear transformation on every robot-human interaction pairs and get the rough embedding features:

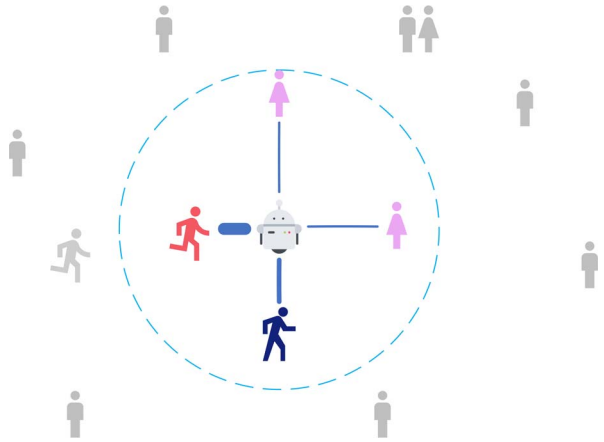


Fig. 3: Illustration of the spatial graph. The light-blue dash-line circle indicates the robot's sensor range. The robot is able to observe the humans' positions and velocities and forms spatial connections with the humans who are inside the sensor range. We take both current positions and future positions computed by observed velocities into account to determine the importance of each human's state to the robot, as indicated by the line width. We input the states into the recurrent neural network with descending order of the distance.

$$\mathbf{e}_i = \mathit{MLP}_e(\mathbf{s}^r, \mathbf{s}_i^h) \tag{6}$$

By feeding each embedding vector e_i into another MLP, the embedding features are transformed into attention weights:

$$\mathbf{w}_i = \mathit{MLP}_a(\mathbf{e}_i) \tag{7}$$

Note that the last layer of MLP_a does not contain a ReLU [24] activation layer, and both MLP_e and MLP_a share parameters when computing each interaction. Compared to SARL [1], we simplify the network structure by removing deeper embedding and the mean pooling module, so that we spend less time but still get similar performance.

We then get the final interaction representation by the product sum of each pairwise embedding vector e_i and attention weight w_i:

$$\mathbf{a} = \sum_i^N w_i e_i \tag{8}$$

Final Output. After processing information along the spatial graph branch and the attention branch separately, we build another non-linear transformation containing ReLU [24] activations taking the spatial vector $\mathbf{i}_s$ and attention vector $\mathbf{i}_a$ as inputs, and output action values as the current learned path planning policy indicator:

$$\mathbf{q} = \mathit{MLP}_v(\mathbf{i}_s, \mathbf{i}_a) \tag{9}$$

IV. Experiments

In the experimental evaluation, we will demonstrate that our approach has the highest success rate and lowest collision rate while still keeping similar navigation time in comparison to state-of-art methods [1], [2].

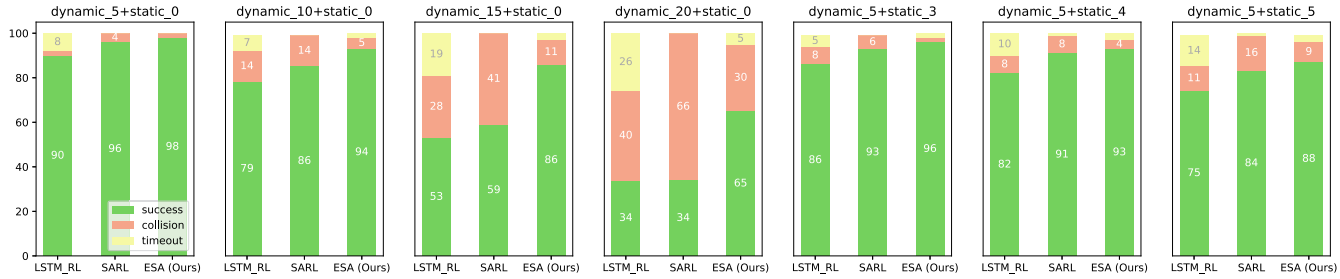


Fig. 4: Quantitative evaluation on scenarios with a various number of humans. Among the three methods, our ESA always has the highest success rate and lowest collision rate. In comparison to the other two, LSTM-RL [2] has the highest timeout rate and more collisions than our ESA. SARL [1] never reaches timeouts, however, the collision rate increases dramatically with an increasing number of humans.

A. Environment Setup

We adapt our simulation environments from the work by Chen *et al.* [1]. Holonomic kinematics is used for all the agents. All agents except the robot behave according to ORCA policy [7]. Invisible setting is kept for the robot in all experiments to avoid the humans reacting to the robot too much so that the robot only learns an aggressive policy that moves straight to the goal. We use circle crossing scenarios with radius 5 m for training and testing, and all humans will move to the opposite perturbed positions on the circle. To realistically simulate real-world conditions, we limit the sensor range of the robot to 2.5 m, i.e., the robot can only observe the humans within this radius, rather than observing all humans in the environment all the time.

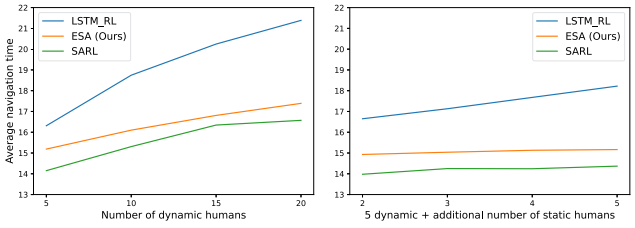
B. Training and Testing

We use PyTorch [25] and PFRL [17] to implement the neural network, 32 as the batch size, and Adam [26] as the optimizer. For fair comparison, we set the same reward function, training batch size, optimizer, learning rate for all methods. In contrast to Chen *et al.* [1], we abandon imitation learning since ORCA [7] performs poorly in such a partially observable environments and cannot be a qualified expert for demonstration. Note that We were able to reduce the training time from around 10 hours to around 2 hours on Mac Mini with M1 chip compared to Chen *et al.* [1].

We trained all three approaches with 10,000 episodes in the environment containing 5 dynamic humans and 2 static humans and learn on each step. For testing, we set up scenarios containing different numbers of dynamic humans and static humans. We set 60 sec as the timeout limit for training so that the agent can get enough exploration, but set the limit to 24 sec for testing. We set different random seeds for various episodes. Consequently, in a specific test episode, the scenarios for all approaches have the same robot start position, end position, maximum velocity, and humans' start and end positions. However, the humans' start and end positions differ in the episodes by re-randomization.

C. Comparative Evaluation

Baselines. We compare the performances between our work and LSTM-RL [2] and SARL [1].

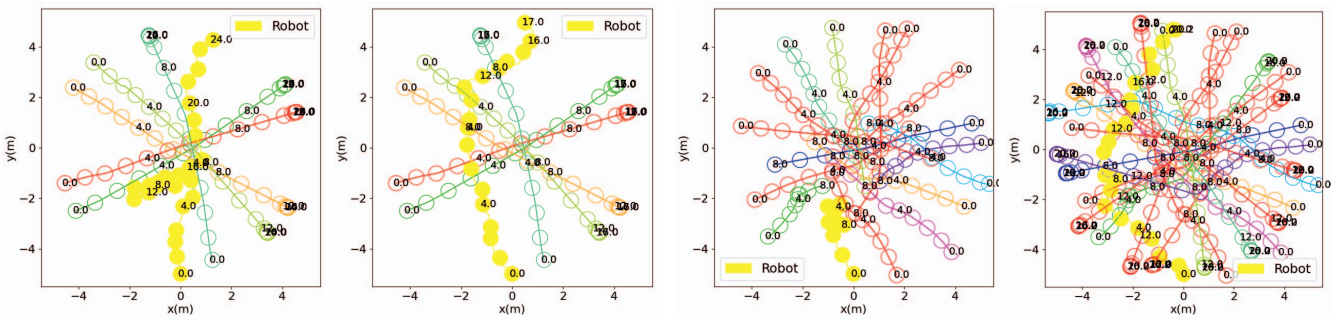


(a) (b)

Fig. 5: Average navigation time for the episodes where all methods succeed in scenarios with different numbers of humans. Fig. (a) depicts the time on scenarios with different numbers of dynamic humans, while Fig. (b) is based on the scenarios with 5 dynamic humans plus different numbers of static humans as indicated. LSTM-RL [2] always takes the longest navigation time showing that it behaves rather conservative, while our ESA is only around 1 sec slower than SARL [1].

Quantitative Evaluation. We performed a comparative evaluation based on success rate, collision rate, timeout rate, average navigation time of episodes that were successfully solved by all three approaches, and average reward among all tested episodes. Fig. 4 and Fig. 5 depict the results of the three methods in scenes with different numbers of dynamic and static humans. Compared with the other two methods, SARL [1] behaves more aggressively and takes the shortest navigation time. However, it is more likely to collide with humans, and the collision rate grows dramatically with an increasing number of obstacles, while LSTM-RL [2] behaves quite conservative, resulting in many timeouts and taking the longest time even in the successful episodes. In contrast, our approach shows the power of obstacle avoidance with the increasing number of humans while keeping a similar navigation time as SARL [1]. The collision rate and navigation time of our method increase much slower, in both scenarios, in the pure dynamic environment and the environment containing both dynamic and static humans. We also collected the average time on each inference, SARL [1] takes steady time on different numbers of humans, while LSTM-RL [2] and our approach slightly increase with the growing number. Although our inference time is slightly increased, the maximum difference is only 0.0003 ms.

In addition, Tab. I and Tab. II show the average episode rewards on the different test scenarios. As can be seen,



(a) LSTM-RL [2] (b) ESA (ours) (c) SARL [1] (d) ESA (ours)

Fig. 6: Illustration of the resulting trajectories. Fig. (a) and (b) show the navigation behavior over time in the scenario with five dynamic humans. We omit the result of SARL [1] since it performs similarly with our method. Fig. (c) and (d) show how the robot navigates in the environment with 15 dynamic humans. Here, the result of LSTM-RL [2] is omitted since it behaves as in Fig. (a) but collides with humans because it basically waits there.

our method outperforms the other two especially with an increasing number of humans. According to a paired t-tests, the difference on success rate, collision rate, and average rewards are statistically significant. This again shows our method is more robust than others as the collision rate is the most important metric in crowd navigation.

	Number of dynamic humans			
	5	10	15	20
SARL [1]	0.949	0.795	0.425	0.1
LSTM-RL [2]	0.891	0.737	0.438	0.204
ESA (Ours)	**0.972**	**0.909**	**0.8**	**0.531**

TABLE I: Average rewards on 1,000 test episodes in the scenarios containing the different numbers of *dynamic* humans and no static humans. Our ESA always gets the highest mean rewards in all different test scenarios. SARLčitesarl performs better than LSTM-RL [2] on the first two scenarios but worse in the scenarios containing 15 and 20 humans, which is due to the much higher collision rate.

	5 dynamic + additional number of static humans				
	1	2	3	4	5
SARL [1]	0.953	0.944	0.902	0.876	0.77
LSTM-RL [2]	0.93	0.889	0.828	0.781	0.696
ESA (Ours)	**0.981**	**0.956**	**0.945**	**0.912**	**0.837**

TABLE II: Average rewards on 1,000 test episodes in the scenarios containing 5 dynamic humans plus the different indicated numbers of *static* humans. Again, our ESA outperforms the other two.

Qualitative Evaluation. We further investigated the improved performance of our model by qualitative results. All three RL-based methods can successfully navigate to the goal in scenarios containing just a few humans. Still, LSTM-RL [2] tends to keep conservative and waits for humans move away. Fig. 6 shows two representative scenarios of obstacle avoidance when navigating in human crowds. With five dynamic people in the environment, the robot controlled by LSTM-RL [2] moves at first, but later it slows down dramatically and moves from right to left and then left to right from 5.0s to 16.0s. In contrast, our ESA-driven robot hesitates at first but then recognizes a better path to the goal through the center, resulting in a shorter navigation time. When the number of dynamic humans increases to 15, SARL [1] is more likely to move into crowds directly, resulting in collisions, while our ESA slows down first and then chooses a short path towards the goal.

D. Ablation Study

To evaluate which part contributes more to the final performance, we separate the spatial and attention branches into two parts, and train and test with the same conditions as before. The ablation study in Fig. 7 shows our ESA combines both the advantages of both and results in a best performance.

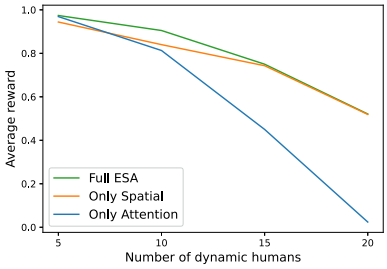


Fig. 7: Average rewards on 1,000 test episodes with different numbers of humans. We can clearly see that the attention graph performs better than the spatial graph when there are just a few people, but the performance drops dramatically with the growing people number. By combining both graphs for reasoning, our model acts best in almost all the test scenarios.

V. Conclusions

In this work, we propose a novel network called Enhanced Spatial and Attention Graph (ESA) that incorporates both, spatial and attention reasoning on robot-human relations to tackle robot navigation in crowded, only partially observable environments. We present a new deep reinforcement learning architecture and train our network with parallel Double DQN [16]. The experiments show that our approach significantly outperforms state-of-the-art baselines [1], [2] in challenging simulation environments with different numbers of dynamic and static humans. Even in scenarios with a high number of humans, our system still keeps a low collision rate.

REFERENCES

[1] 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)*. IEEE, 2019, pp. 6015–6022.

[2] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 3052–3059.

[3] R. Möller, A. Furnari, S. Battiato, A. Härmä, and G. M. Farinella, "A survey on human-aware robot navigation," *Robotics and Autonomous Systems*, vol. 145, p. 103837, 2021.

[4] Y. F. Chen, M. Liu, M. Everett, and J. P. How, "Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 2017, pp. 285–292.

[5] 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, pp. 1343–1350.

[6] 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.

[7] J. van den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics Research*, C. Pradalier, R. Siegwart, and G. Hirzinger, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 3–19.

[8] 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.

[9] D. Helbing and P. Molnár, "Social force model for pedestrian dynamics," *Phys. Rev. E*, vol. 51, pp. 4282–4286, May 1995. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevE.51.4282

[10] 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.

[11] G. S. Aoude, B. D. Luders, J. M. Joseph, N. Roy, and J. P. How, "Probabilistically safe motion planning to avoid dynamic obstacles with uncertain motion patterns," *Autonomous Robots*, vol. 35, no. 1, pp. 51–76, Jul 2013. [Online]. Available: https://doi.org/10.1007/s10514-013-9334-3

[12] H. Kretzschmar, M. Spies, C. Sprunk, and W. Burgard, "Socially compliant mobile robot navigation via inverse reinforcement learning," *The International Journal of Robotics Research*, vol. 35, no. 11, pp. 1289–1307, 2016. [Online]. Available: https://doi.org/10.1177/0278364915619772

[13] M. Kuderer, H. Kretzschmar, C. Sprunk, and W. Burgard, "Feature-based prediction of trajectories for socially compliant navigation." in *Robotics: science and systems*, 2012.

[14] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[15] A. Vemula, K. Muelling, and J. Oh, "Social attention: Modeling attention in human crowds," in *2018 IEEE international Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 4601–4607.

[16] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[17] Y. Fujita, P. Nagarajan, T. Kataoka, and T. Ishikawa, "Chainerrl: A deep reinforcement learning library," *Journal of Machine Learning Research*, vol. 22, no. 77, pp. 1–14, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-376.html

[18] C. Chen, S. Hu, P. Nikdel, G. Mori, and M. Savva, "Relational graph learning for crowd navigation," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 10 007–10 013.

[19] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[20] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[21] D. P. Kroese, T. Taimre, and Z. I. Botev, *Handbook of monte carlo methods*. John Wiley & Sons, 2013, vol. 706.

[22] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," in *Proceedings of 2010 IEEE international symposium on circuits and systems*. IEEE, 2010, pp. 253–256.

[23] L. Noriega, "Multilayer perceptron tutorial," *School of Computing. Staffordshire University*, 2005.

[24] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[25] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.