DeepMNavigate: Deep Reinforced Multi-Robot Navigation Unifying Local & Global Collision Avoidance

Qingyang Tan[1], Tingxiang Fan[2], Jia Pan[2], Dinesh Manocha[1]

Video Link: https://youtu.be/LWLBxWuwPeU

***Abstract*— We present a novel algorithm (DeepMNavigate) for global multi-agent navigation in dense scenarios using deep reinforcement learning. Our approach uses local and global information for each robot based on motion information maps. We use a three-layer CNN that uses these maps as input and generate a suitable action to drive each robot to its goal position. Our approach is general, learns an optimal policy using a multi-scenario, multi-state training algorithm, and can directly handle raw sensor measurements for local observations. We demonstrate the performance on complex, dense benchmarks with narrow passages on environments with tens of agents. We highlight the algorithm's benefits over prior learning methods and geometric decentralized algorithms in complex scenarios.**

I. Introduction

Multi-robot systems are increasingly being used for different applications, including surveillance, quality control systems, autonomous guided vehicles, warehouses, cleaning machines, etc. A key challenge is to develop efficient algorithms for navigating such robots in complex scenarios, avoiding collisions with each other and the obstacles in the environment. As large number of robots are used, we need more efficient methods that can handle complex and dense scenarios.

Multi-agent navigation has been studied extensively in robotics, AI, and computer animation. At a broad level, previous approaches can be classified into centralized or decentralized planners. The centralized planners tend to aggregate all the individual robots into one large composite system and leverage known motion planning methods (e.g., sampling-based planners) to compute collision-free trajectories for all the agents [32], [31], [17], [23], [27]. In practice, they are limited to multi-robot systems with only a few robots or agents. The geometric decentralized planners compute a trajectory for each robot separately for a short horizon (e.g., a few timesteps) using local navigation methods [30], [9], [12], [7], [32]. This is followed by some coordination scheme to resolve any possible collisions. One benefit of decentralized methods is that they can scale to a large number of agents, though it is difficult to provide any global guarantees on the resulting trajectories [8] or to handle challenging scenarios with narrow passages (see Figures 1 or 2).

Recently, there is considerable work on developing new, learning-based planning algorithms [6], [20], [4], [5], [14], [15] for navigating one or more robot robots through dense scenarios. Compared to prior geometric decentralized systems, techniques based on reinforcement learning tend to learn an optimal policy using a multi-scenario, multi-stage

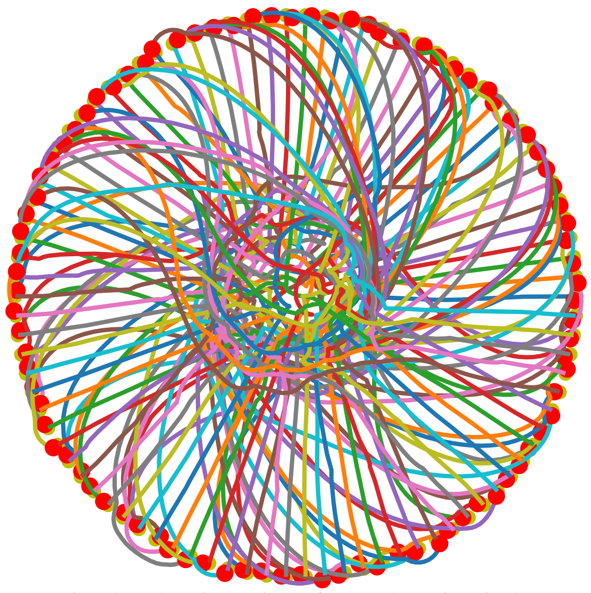


Fig. 1: Simulated trajectories of 90 robots in circle crossing scenarios generated by our algorithm. The yellow points are the initial positions of the robots, and the red points are the goals of the robot that are diametrically opposite. Our DeepMNavigate algorithm can handle such scenarios, there are no collision along the trajectories, and all the robots reach their goals. Prior learning methods that only use local methods [6], [4] cannot handle such scenarios.

training algorithm. Based on extensive training on a large number of robots in complex synthetic environments, these learning-based methods tend to work in practical scenarios and can also handle raw sensor measurements. However, current learning-based local methods are limited to using only the local information and do not exploit any global information that could be available. Therefore, it is difficult to provide any global guarantees on the performance of such planners or to use them in dense environments or narrow passages.

Main Results: We present a novel, multi-robot navigation algorithm (DeepMNavigate) based on re-inforcement learning that exploits local and global information. We use a multi-stage training scheme that uses various multi-robot simulation scenarios with global information. In terms of training, we represent the global information using a *motion information* map that includes the location of each agent or robot. We place the robot information in the corresponding position to generate a bit-map and use the resulting map as an input to a three-layer CNN. Our CNN considers this global information along with local observations in the scene to generate a suitable action to drive each robot to the goal without collisions. We have evaluated our algorithm

[1]Qingyang Tan and Dinesh Manocha are with Department of Computer Science and Electrical & Computer Engineering, University of Maryland at College Park. {qytan,dm@cs.umd.edu} [2]Tingxiang Fan and Jia Pan are with Department of Computer Science, University of Hong Kong. {tingxfan@hku.hk, jpan@cs.hku.hk}

in dense environments with many tens of robots (e.g., 90 robots) navigating in tight scenarios with narrow passages. Our approach scales with the number of robots and is able to compute collision-free and smooth trajectories. We compare the performance of DeepMNavigate with prior learning-based navigation algorithms that only use local information and highlight the benefits in terms of handling challenging or dense scenarios. Our approach can easily handle challenging scenarios like inter-changing positions in a narrow corridor (see Fig. 2), which are rather difficult for geometric decentralized methods.

II. Related Work

A. Geometric Multi-Robot Navigation Algorithms

Most prior algorithms are based on geometric techniques like sampling-based methods, geometric optimization, or complete motion planning algorithms. The centralized methods assume that each robot has access to complete information about the state of the other robots based on some global data structure or communication system. The resulting algorithms [16], [26], [34], [29] can compute a safe, optimal, and complete solution for navigation, though they dont scale to large multi-robot systems with tens of robots. Many geometric decentralized methods for multi-agent systems are based on reciprocal velocity obstacles [30] or its variants. Many extensions have been proposed that account for dynamics constraints (Alonso-Mora et al. [1]) and sensory uncertainty and can be combined with global planners that use sample-based methods to compute a global roadmap. These synthetic methods can be used during the training phase of learning algorithms.

B. Learning-Based Navigation Methods

Learning-based collision avoidance techniques usually try to optimize a parameterized policy using the data collected from different tasks. Many navigation algorithms adopt a supervised learning paradigm to train collision avoidance policies. Muller et al. [18] present a vision-based static obstacle avoidance system using a 6-layer CNN to map input images to steering angles. Zhang et al. [35] describe a successor-feature-based deep reinforcement learning algorithm for robot navigation tasks based on raw sensory data. Barreto et al. [2] apply transfer learning to deploy a policy to new problem instances. Sergeant et al. [25] propose an approach based on multimodal deep autoencoders that enables a robot to learn how to navigate by observing a dataset of sensor inputs and motor commands collected while being tele-operated by a human. Ross et al. [22] adapt an imitation learning technique to train reactive heading policies based on the knowledge of a human pilot. Pfeiffer et al. [21] map the laser scan and goal positions to motion commands using expert demonstrations. To be effective, these methods need to collect training data in different environments, and the performance is limited by the quality of the training sets.

To overcome the limitations of supervised-learning, Tai et al. [28] present a mapless motion planner trained end-to-end without any manually designed features or prior demonstrations. Kahn et al. [13] propose an uncertainty-aware model-based learning algorithm that estimates the probability of collision uses that information to minimize the collisions at training time. To extend learning-based methods to highly dynamic environments, some decentralized techniques have been proposed. Godoy et al. [10] propose a Bayesian inference approach that computes a plan that minimizes the number of collisions while driving the robot to its goal. Chen et al. [4], [3] and Everett et al. [5] present multi-robot collision avoidance policies based on deep reinforcement learning, which require deploying multiple sensors to estimate the state of nearby agents and moving obstacles. Yoon et al. [33] extend the framework of centralized training with decentralized execution to perform additional optimization for inter-agent communication. Fan et al. [6], and Long et al. [14], [15] describe a decentralized multi-robot collision avoidance framework, where each robot makes navigation decisions independently without any communication with other agents. However, all these methods do not utilize global information about the robot or the environment, which could be used to improve the optimality of the resulting paths.

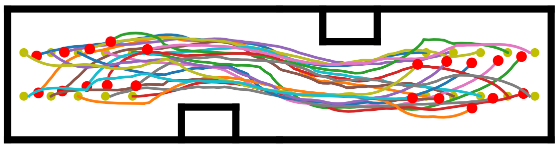


Fig. 2: Computed trajectories of two groups of robots (in total 20) exchanging their positions through the narrow corridor computed using DeepMNavigate. The yellow points are the initial positions, and the red points are the final positions. Prior geometric decentralized methods [30] are unable to handle such complex, dense environments.

III. Multi-Robot Navigation

A. Problem Formulation and Notation

We consider the multi-robot navigation problem for non-holonomic differential drive robots. Our goal is to design a scheme that avoids collisions with obstacles and other robots and works well in dense and general environments. We describe the approach for 2D, but can be extended to 3D workspaces and to robots with other dynamics constraints.

Let the number of robots be N_{rob}. We represent each of them as a disc with radius R. At each timestep t, the i-th robot $(1 \leq i \leq N_{\text{rob}})$ has access to an observation $\mathbf{o}_i^t$ and then computes a collision-free action $\mathbf{a}_i^t$ that drives the i-th robot towards its goal $\mathbf{g}_i^t$ from the current position $\mathbf{p}_i^t$. The observation of each robot includes four parts: $\mathbf{o}^t = [\mathbf{o}_z^t, \mathbf{o}_g^t, \mathbf{o}_v^t, \mathbf{o}_M^t]$, where $\mathbf{o}_z^t$ denotes the sensor measurement (e.g., laser sensor) of its surrounding environment, $\mathbf{o}_g^t$ stands for its relative goal position, $\mathbf{o}_v^t$ refers to its current velocity, and $\mathbf{o}_M^t$ is the robot motion information, which will be discussed in Section IV. The computed action $\mathbf{a}^t$ drives the robot to approach its goal while avoiding collisions with other robots and obstacles $\mathbf{B}_k(1 \leq k \leq N_{\text{obs}})$ within the time horizon Δt until the next observation $\mathbf{o}^{t+1}$ is received.

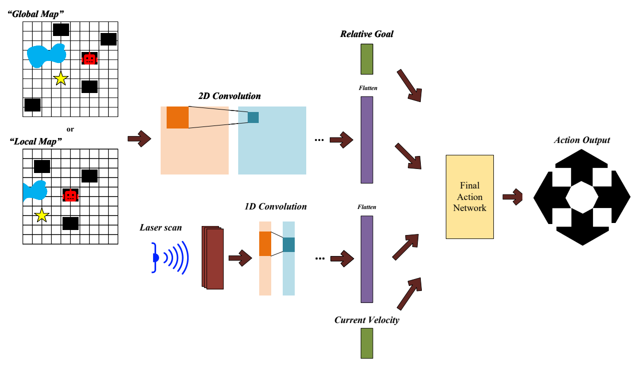


Fig. 3: We highlight the architecture of our policy network (DeepMNavigate). We use local and global information captured using different maps. We use 2D convolutional neural network to handle the additional global information input and compute the action for each robot. We also highlight the two maps (global and local) used by our approach. The resulting CNN computes the action for each robot.

Let $\mathbb{L}$ be the set of trajectories for all robots, subject to the robot's kinematic constraints, i.e.:

$$\mathbb{L} = \{l_i, i = 1, ..., N_{rob} | \mathbf{v}_i^t \sim \pi_\theta(\mathbf{a}_i^t|\mathbf{o}_i^t), \mathbf{p}_i^t = \mathbf{p}_i^{t-1} + \Delta t \cdot \mathbf{v}_i^t,$$
$$\forall j \in [1, N_{\text{rob}}], j \neq i, \left\|\mathbf{p}_i^t - \mathbf{p}_j^t\right\| > 2R \wedge \forall k \in [1, N_{\text{obs}}],$$
$$\forall \mathbf{p} \in \mathbf{B}_k, \left\|\mathbf{p}_i^t - \mathbf{p}\right\| > R \wedge \left\|\mathbf{v}_i^t\right\| \leq v_i^{\max}\}. \quad (1)$$

B. Multi-Agent Navigation Using Reinforcement Learning

Our approach builds on prior reinforcement learning approaches that use local information in terms of various observations. Some of them only utilize three of the four elements mentioned in III-A. The term $\mathbf{o}_z^t$ may include the measurements of the last three consecutive frames from a sensor. The relative goal position $\mathbf{o}_g^t$ is these cases is a 2D vector representing the goal position in polar coordinates with respect to the robots current position. The observed velocity $\mathbf{o}_v^t$ includes the current velocity of the robot. These observations are normalized using the statistics aggregated over training. The action of a differential robot includes the translational and rotational velocity, i.e. $\mathbf{a}^t = [v^t, w^t], v \in (0, 1), w \in (-1, 1)$.

We use the following reward function to guide a team of robots:

$$r_i^t = ({}^g r)_i^t + ({}^c r)_i^t + ({}^w r)_i^t. \quad (2)$$

When the robot reaches its goal, it is rewarded as

$$({}^g r)_i^t = \begin{cases} r_{\text{arrival}} & \text{if } \left\|\mathbf{p}_i^t - \mathbf{g}_i\right\| < 0.1 \\ w_g(\left\|\mathbf{p}_i^{t-1} - \mathbf{g}_i\right\| - \left\|\mathbf{p}_i^t - \mathbf{g}_i\right\|) & \text{otherwise.} \end{cases} \quad (3)$$

When there is a collision, it is penalized using the function

$$({}^c r)_i^t = \begin{cases} r_{\text{collision}} & \text{if } \left\|\mathbf{p}_i^t - \mathbf{p}_j^t\right\| < 2R \\ & \text{or } \left\|\mathbf{p}_i^t - \mathbf{p}\right\| < R, \mathbf{p} \in \mathbf{B}_k \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

In addition to collision avoidance, one of our goals is to generate a smooth path. A simple technique is to impose penalties whenever there are large rotational velocities. This can be expressed as

$$({}^w r)_i^t = w_w |w_i^t| \text{if } |w_i^t| > 0.7. \quad (5)$$

IV. DeepMNavigate: Trajectory Computation Using Global Information

In this section, we present our novel, learning-based, multi-agent navigation algorithm that uses the positional information of other agents. Our formulation is based on motion information maps and uses a 3-layer CNN to generate a suitable action for each agent.

A. Motion Information Maps

Prior rule-based decentralized methods such as [30] use information corresponding to the position and velocity of each agent to compute a locally-optimal and collision-free trajectory. Our goal is to compute similar state information to design better learning-based navigation algorithms. Such state information can either be gathered based on some communication with nearby robots or computed using a deep network that uses raw sensor data. In our formulation, we use maps that consist of each agent's location as input. This approach can handle an arbitrary number of robots and can be easily combined with a convolutional neural network to compute the action for each robot. In particular, we use two different map representations: one corresponds to all the robots based on the world coordinate system and is called the *global-map*; the other map is centered at each robot's current location and uses the relative coordinate system and is called the *local-map*.

We give the formulation for computing and representing the global-map. The local-map is computed in a similar manner. During each timestep t, we specify the i-th robot's position in the world frame as $\mathbf{x}_i^t \in \mathbb{R}^2$. We also use the goal positions $\mathbf{g}_i, \forall 1 \leq i \leq N_{\text{rob}}$, obstacle information $\mathbf{B}_k, \forall 1 \leq k \leq N_{\text{obs}}$, to build the map $M_i^t \in \mathbb{R}^{h \times w}$ for the i-th robot. Assume the size of the simulated robot scenario is $H \times W$, where H represents the height and W represents the width, and the origin of the world frame is located at $(\frac{H}{2}, \frac{W}{2})$. Each pixel of the map $(M_i^t(p, q), \forall 1 \leq p \leq h, 1 \leq q \leq w)$ indicates which kind of object lies in the small area $\mathcal{A}_{pq} = \left(\frac{(p-1)H}{h} - \frac{H}{2}, \frac{pH}{h} - \frac{H}{2}\right] \times \left(\frac{(q-1)W}{w} - \frac{W}{2}, \frac{qW}{w} - \frac{W}{2}\right]$ in the world frame. Assuming each object's radius is r_i, then $M_i^t(p, q)$ corresponds to:

$$M_i^t(p, q) = \begin{cases} 1, & \{y | \|y - \mathbf{x}_i^t\| \leq R\} \bigcup \mathcal{A}_{pq} \neq \emptyset \\ 12 & \exists 1 \leq j \leq N_{\text{rob}},\ j \neq i, \\ & \quad s.t.\ \{y | \|y - \mathbf{x}_j^t\| \leq R\} \bigcup \mathcal{A}_{pq} \neq \emptyset \\ 3, & \{y | \|y - \mathbf{g}_i\| \leq R\} \bigcup \mathcal{A}_{pq} \neq \emptyset \\ 4, & \exists 1 \leq k \leq N_{\text{obs}}, s.t.\ \mathbf{B}_k \bigcup \mathcal{A}_{pq} \neq \emptyset \\ 0, & \text{otherwise,} \end{cases} . \quad (6)$$

where "1" represents the corresponding robot, "2" represents the neighboring robots, "3" represents the robot's goal, "4" represents the obstacles and "0" represents the empty background (i.e. free space).

In some scenarios, there could be a restriction on the robot's movement in terms of static obstacles or regions that are not accessible. Our global-map computation takes this into account in terms of representing $M_i^t(p,q)$. However, these maps may not capture the locations of dynamic obstacles. In those cases, we would use the local-map for each agent instead of considering the whole scenario with size $H \times W$. These maps only account for information in a relatively small neighborhood with fixed size $H_l \times W_l$. Moreover, we use the local frame of each agent to generate the input. The size of the local neighborhood (H_l, W_l) can be tuned to find a better performance for different applications. In addition to the position information, these maps may contain other state information of the robot, including velocity, orientation, or dynamics constraints. This information is used to compute a local collision-free trajectory for each robot and also to ensure that the resulting trajectory is smooth.

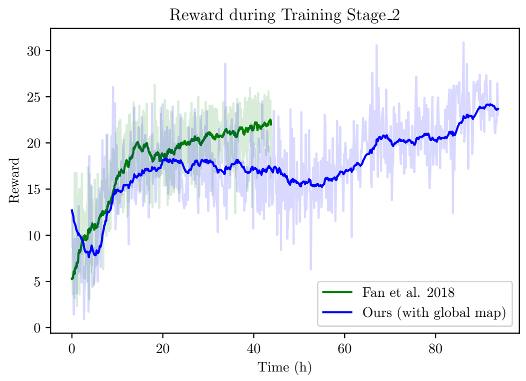


Fig. 4: We highlight the reward as a function of the training time during the second stage of the overall training algorithm. We compare the performance of a reinforcement learning algorithm that only uses local information [6] to our method, which uses local and global information. Our method obtains a higher reward than [6].

B. Proximal Policy Optimization

We use proximal policy optimization [24] to train the overall system. Our training algorithm has the stability and reliability of trust-region methods: it tries to compute an update at each step that minimizes the cost function while ensuring that the deviation from the previous policy is relatively small. The resulting proximal policy algorithm updates the network using all the steps after several continuous simulations (i.e. after the entire system of robots reach the goals or stop running due to collisions), instead of using only one step to ensure the stability of network optimization. In these cases, if we store the robot positions or motion information as dense matrices corresponding to the formulation in Eq. 6, it would require a significant amount of memory and also increase the overall training time. Instead, we use a sparse matrix representation for M_t^i. We compute the non-zero entries ofM_t^i based on the current position of each robot, goal positions and obstacle information using Eq. 6. To feed the input to our neural network, we obtain dense representations from temporary sparse storage. This design choice allows us to train the system using trajectories from 58 agents performing 450 actions using only 2.5GB memory. More details on the training step are given in Sec. V.

C. Network

To analyze a large matrix and produce a low-dimensional feature for input M_i^t, we use a convolutional neural network (CNN) because such network structures are useful for handling an image-like input (or our map representation). Our network has three convolutional layers with architecture, as shown in Fig. 3. Our network extracts the related location of different objects in the environment, and could guide the overall planner to avoid other agents and obstacles to reach the goal.

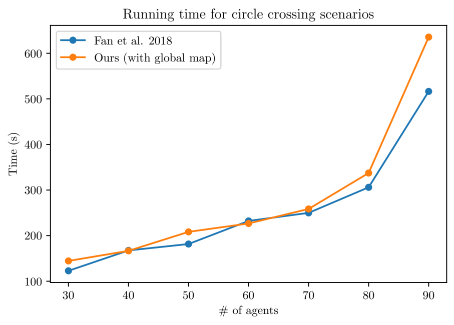
Our approach to handling raw sensor data (e.g., 2D laser scanner data) uses the same structure as local methods[6], i.e. a two-layer, 1D convolutional network. Overall, we use a two-layer, fully-connected network that takes as input the observation features, including the feature generated by the 1D & 2D CNNs, related goal position, and observed velocity. It generates the action output and local path for each robot. We highlight the whole pipeline of the network for local and global information in Fig. 3.

Layer	Convolutional Filter	Stride	Padding	Activation Fuction	Output Size
Input	-	-	-	-	$250 \times 250 \times 1$
Conv 1	$5 \times 5 \times 1 \times 8$	1×1	'SAME'	ReLU	$250 \times 250 \times 8$
Max Pooling 1	3×3	2×2	'SAME'	-	$125 \times 125 \times 8$
Conv 2	$5 \times 5 \times 8 \times 12$	1×1	'SAME'	ReLU	$125 \times 125 \times 12$
Max Pooling 2	3×3	2×2	'SAME'	-	$63 \times 63 \times 12$
Conv 3	$5 \times 5 \times 12 \times 20$	1×1	'SAME'	ReLU	$63 \times 63 \times 20$
Max Pooling 3	3×3	2×2	'SAME'	-	$32 \times 32 \times 20$
Flatten	-	-	-	-	20480
Fully connected	20480×128	-	-	ReLU	128
Fully connected	128×64	-	-	ReLU	64

TABLE I: Architecture and hyper-parameters of a convolutional neural network that considers the global information.

D. Network Training

Our training strategy extends the method used by learning algorithms based on local information [6], [4]. To accelerate the training process, we can divide the overall training computation into two stages. In the first stage, we use k robots (e.g. k=20) with random initial positions and random goals in a fully-free environment. In the second stage, we can include a more challenging environment, such as a narrow passage, random obstacles, etc. We generate a high number of scenes with different kind of narrow passages and random obstacles with up to 70 or 80 robots and many tens of obstacles. These varying training environments and the large number of robots in the system can result in a good final policy. Moreover, the use of global information results in much larger network parameters. We use a 20480×128 FC layer, which is more difficult to train than a relatively simple, small network that only accounts for local information. To accelerate the training process and get accurate results, we do not train the entire network from scratch, but use the multi-stage approach. During the first stage, we retrain



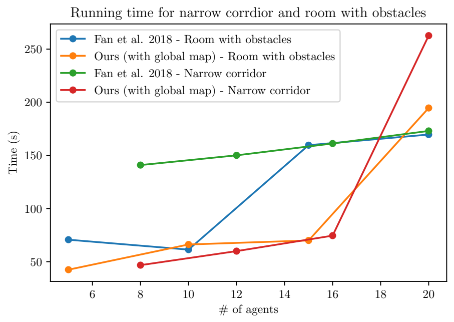


Fig. 5: We show the running time with different numbers of agents in several benchmarks. This graph demonstrates that our approach is practical for tens of robots. We also compare the running time with a learning-based algorithm that only uses local information [6]. The additional overhead in the running time with the global information is rather small.

the network with additional structures corresponding to the global information (i.e. the global-map) using the pretrained local information network part afterward. We highlight the reward as a function of the training time during the second stage of the training and compare it with the local method[6] in Fig. 4. Notice that, since we use a 2D convolutional neural network, our overall training algorithm needs more time during each iteration of training. As a result, we do not perform the same number of training iterations as [6] in Fig. 4.

V. Implementation and Performance

In this section, we highlight the performance of our multi-agent navigation algorithm (DeepMNavigate) on complex scenarios and highlight the benefits over prior reinforcement learning methods that only use local information [6].

A. Parameters

In the current implementation, we set $H = W = 500$ for the global-map and $H_l = W_l = 250$ for each local-map. In both situations, we set $w = h = 250$. Although a larger map (e.g., $w = h = 500$) could include more details from the system, it would significantly increase the network size and the final running time. For parameters in the reward function, we set $r_{\text{arrival}} = 15$, $r_{\text{collision}} = -15$, $w_g = 2.5$, and $w_w = -0.1$.

B. Performance Evaluation

To evaluate the performance of our navigation, we use the following metrics:

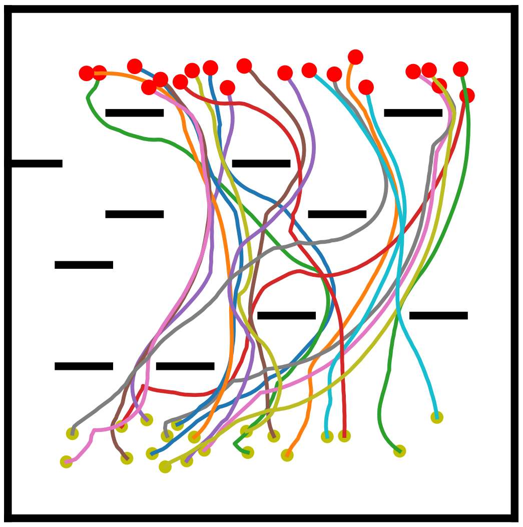


Fig. 6: We highlight the trajectories of 20 robots in a room with obstacles scenarios. We highlight the initial position of agent (yellow) and the final position (red) along with multiple obstacles. Prior learning methods that only use local methods [6], [4] will take more time and may not be able to handle such scenarios when the number of obstacles or the agents increase.

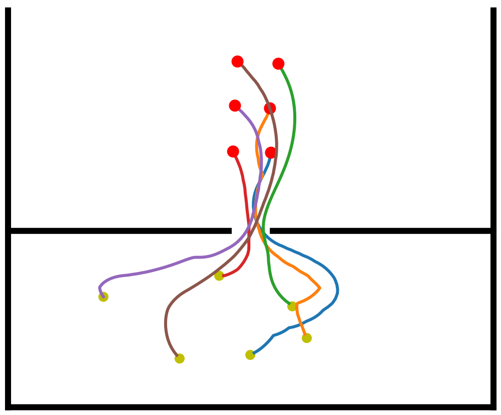


Fig. 7: Simulated trajectories of 6 robots evacuating from a room by our algorithm. The yellow points are the initial positions, and the red points are the final positions.

- *Success rate*: the ratio of the number of robots reaching their goals in a certain time limit without any collisions to the total number of robots in the environment.
- *Extra time*: the difference between the average travel time of all robots and the lower bound of the robots' travel time. The latter is computed as the average travel time in terms of going straight towards the goal at the maximum speed without any collision avoidance.
- *Average speed*: the average speed of all robots during the navigation.

We have evaluated our algorithm in several challenging benchmarks:

- *Circle crossing*: The robots are uniformly placed around a circle, and the goal position of each robot is on the opposite end of the circle based on the diameter. The scenarios are widely used for multi-agent navigation

Metrics	Methods	# of agents (radius)						
		30 (8)	40 (8)	50 (8)	60 (8)	70 (8)	80 (12)	90 (12)
Success Rate	[6]	**1**	0.975	0.96	0.95	0.928571	0.7375	0.722222
	Ours	**1**	**1**	**1**	**1**	0.985714	**1**	**1**
Extra Time	[6]	**4.32333**	**8.20256**	**7.8625**	11.3088	13.54	15.1238	15.3292
	Ours	8.94667	8.73	9.738	**10.1**	**12.4725**	17.5525	34.1256
Average Speed	[6]	**0.787272**	**0.661087**	**0.670508**	0.585892	0.541638	**0.54341**	0.610233
	Ours	0.641368	0.646987	0.621649	**0.613027**	**0.561946**	0.506294	0.412899

Metrics	Methods	# of agents			
		8	12	16	20
Success Rate	[6]	0.375	0.75	0.5625	0.0
	Ours	**1**	**1**	**1**	**1**
Extra Time	[6]	4.8	12.5111	30.7222	-
	Ours	**2.7**	**4.075**	**5.33125**	**8.36**
Average Speed	[6]	0.653784	0.410699	0.230921	-
	Ours	**0.742279**	**0.656969**	**0.594551**	**0.482519**

Metrics	Methods	# of agents			
		5	10	15	20
Success Rate	[6]	**1**	0.7	0.6	0.7
	Ours	**1**	**1**	**1**	**1**
Extra Time	[6]	9.55894	5.13058	6.38476	9.99585
	Ours	**2.19487**	**2.55377**	**3.47301**	**7.80055**
Average Speed	[6]	0.707441	0.734304	0.721682	0.508305
	Ours	**0.858985**	**0.828276**	**0.775802**	**0.582426**

TABLE II: The performance of our proposed method (DeepMNavigate) and the baseline method on different benchmarks (**Up**: circle crossing; **Down left**: narrow corridor; **Down right**: room with obstacles.), measured in aspects of success rate, extra time, and average speed, evaluated under different numbers of agents. The bold entries represent the best performance. Our method can guarantee a higher success rate in dense environments as compared to prior multi-agent navigation algorithms.

algorithms [30], [6].

- *Narrow corridor*: Two groups robots exchange their positions through one narror corridor. This benchmark is hard for geometric decentralized methods [30] which cannot navigate robots through narrow passage.
- *Room with obstacles*: The robots cross one room full of obstacles, from one side to the other. Methods only using local information [6] will spend more time to find the path to the goal, even fail due to limited observation.

We highlight the performance in Table II on page 6 . Our method succeed in all the benchmarks even with high density. We also display the running times for different numbers of robots in Fig. 5.

C. Trajectory Computation

During the second stage, our model is trained over one random circle crossing scenario where the number of robots is 12. When we apply the model onto scenarios where there are more agents, the computed trajectories occur as shwon in Fig. 1. In the cases where $N_{\text{obs}} = 90$, the success rates are 1.0. We also highlight the benefits of our approach in another challenging benchmark, narrow corridor (Fig. 2). This specific scenario was not used during our training process. Prior decentralized methods may not perform well in such scenarios. Another Simulated trajectories of room with obstacles benchmark is shown in Fig. 6.

VI. Conclusion, Limitations, and Future Work

We present a novel, multi-agent navigation algorithm based on deep reinforcement learning. We show how global information about the environment can be used based on the global-map and highlight a novel network architecture to compute the trajectory for each robot. We highlight its benefits over many challenging scenarios and show benefits over geometric decentralized methods or reinforcement learning methods that only use local information. Even with training on simple scenarios, our DeepMNavigation algorithm can handle complex scenarios robustly, which cannot be handled by prior learning-based methods or decentralized multi-agent navigation algorithms. While we present results for 2D, it can be easily extended to 3D workspaces using 3D maps and 3D CNNs.

Our results are promising and there are many ways to improve the performance. We need good techniques to compute the optimal size of global-map and local-map, and we can also include other components of the state information like velocity, orientation, or dynamics constraints. We also need to extend the approach to handle general dynamic scenes where no information is available about the motion of the moving obstacles. The use of global information increases the complexity of the training computation and we use a two-stage algorithm to reduce its running time. One other possibility is to use an auto-encoder to automatically derive the low-dimensional feature representations and then use them as a feature extractor. It may be possible to split the global and local navigation computation to combine our approach with local methods to avoid collisions with other agents or dynamic obstacles [30], [9]. Such combination of local and global methods have been used to simulate large crowds [19] and it may be useful to develop a similar framework for learning-based algorithms.

References

[1] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsley, and R. Siegwart, "Optimal reciprocal collision avoidance for multiple non-holonomic robots," in *Distributed Autonomous Robotic Systems*. Springer, 2013, pp. 203–216.

[2] A. Barreto, W. Dabney, R. Munos, J. J. Hunt, T. Schaul, H. P. van Hasselt, and D. Silver, "Successor features for transfer in reinforcement learning," in *Advances in neural information processing systems*, 2017, pp. 4055–4065.

[3] 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)*. IEEE, 2017, pp. 1343–1350.

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

[5] M. Everett, Y. F. Chen, and J. P. How, "Motion planning among dynamic, decision-making agents with deep reinforcement learning," in *IROS*. IEEE, 2018, pp. 3052–3059.

[6] T. Fan, P. Long, W. Liu, and J. Pan, "Fully distributed multi-robot collision avoidance via deep reinforcement learning for safe and efficient navigation in complex scenarios," *arXiv preprint arXiv:1808.03841*, 2018.

[7] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[8] T. Fraichard and H. Asama, “Inevitable collision statesa step towards safer robots?” *Advanced Robotics*, vol. 18, no. 10, pp. 1001–1024, 2004.
[9] R. Geraerts, A. Kamphuis, I. Karamouzas, and M. Overmars, “Using the corridor map method for path planning for a large number of characters,” in *International Workshop on Motion in Games*. Springer, 2008, pp. 11–22.
[10] J. Godoy, I. Karamouzas, S. J. Guy, and M. L. Gini, “Moving in a crowd: Safe and efficient navigation among heterogeneous agents.” in *IJCAI*, 2016, pp. 294–300.
[11] L. He, J. Pan, and D. Manocha, “Efficient multi-agent global navigation using interpolating bridges,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 4391–4398.
[12] D. Helbing and P. Molnar, “Social force model for pedestrian dynamics,” *Physical review E*, vol. 51, no. 5, p. 4282, 1995.
[13] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine, “Uncertainty-aware reinforcement learning for collision avoidance,” *arXiv preprint arXiv:1702.01182*, 2017.
[14] P. Long, T. Fanl, X. Liao, W. Liu, H. Zhang, and J. Pan, “Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 6252–6259.
[15] P. Long, W. Liu, and J. Pan, “Deep-learned collision avoidance policy for distributed multiagent navigation,” *IEEE Robotics and Automation Letters*, vol. 2, no. 2, pp. 656–663, 2017.
[16] R. Luna and K. E. Bekris, “Efficient and complete centralized multi-robot path planning,” in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 3268–3275.
[17] R. J. Luna and K. E. Bekris, “Push and swap: Fast cooperative path-finding with completeness guarantees,” in *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.
[18] U. Muller, J. Ben, E. Cosatto, B. Flepp, and Y. L. Cun, “Off-road obstacle avoidance through end-to-end learning,” in *Advances in neural information processing systems*, 2006, pp. 739–746.
[19] R. Narain, A. Golas, S. Curtis, and M. C. Lin, “Aggregate dynamics for dense crowd simulation,” in *ACM transactions on graphics (TOG)*, vol. 28, no. 5. ACM, 2009, p. 122.
[20] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena, “From Perception to Decision: A Data-driven Approach to End-to-end Motion Planning for Autonomous Ground Robots,” *arXiv e-prints*, p. arXiv:1609.07910, Sep 2016.
[21] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena, “From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots,” in *2017 IEEE international conference on robotics and automation (icra)*. IEEE, 2017, pp. 1527–1533.
[22] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert, “Learning monocular reactive uav control in cluttered natural environments,” in *2013 IEEE international conference on robotics and automation*. IEEE, 2013, pp. 1765–1772.
[23] G. Sanchez and J.-C. Latombe, “Using a prm planner to compare centralized and decoupled planning for multi-robot systems,” in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, vol. 2. IEEE, 2002, pp. 2112–2119.
[24] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, “Proximal policy optimization algorithms,” *arXiv preprint arXiv:1707.06347*, 2017.
[25] J. Sergeant, N. Sünderhauf, M. Milford, and B. Upcroft, “Multimodal deep autoencoders for control of a mobile robot,” in *Proc. of Australasian Conf. for Robotics and Automation (ACRA)*, 2015.
[26] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, “Conflict-based search for optimal multi-agent pathfinding,” *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.
[27] K. Solovey and D. Halperin, “On the hardness of unlabeled multi-robot motion planning,” *The International Journal of Robotics Research*, vol. 35, no. 14, pp. 1750–1759, 2016.
[28] L. Tai, G. Paolo, and M. Liu, “Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 31–36.
[29] S. Tang, J. Thomas, and V. Kumar, “Hold or take optimal plan (hoop): A quadratic programming approach to multi-robot trajectory generation,” *The International Journal of Robotics Research*, vol. 37, no. 9, pp. 1062–1084, 2018.
[30] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, “Reciprocal n-body collision avoidance,” in *Robotics research*. Springer, 2011, pp. 3–19.
[31] J. van Den Berg, J. Snoeyink, M. C. Lin, and D. Manocha, “Centralized path planning for multiple robots: Optimal decoupling into sequential plans.” in *Robotics: Science and systems*, vol. 2, no. 2.5, 2009, pp. 2–3.
[32] J. P. Van Den Berg and M. H. Overmars, “Prioritized motion planning for multiple robots,” in *2005 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2005, pp. 430–435.
[33] H.-J. Yoon, H. Chen, K. Long, H. Zhang, A. Gahlawat, D. Lee, and N. Hovakimyan, “Learning to communicate: A machine learning framework for heterogeneous multi-agent robotic systems,” in *AIAA Scitech 2019 Forum*, 2019, p. 1456.
[34] J. Yu and S. M. LaValle, “Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics,” *IEEE Transactions on Robotics*, vol. 32, no. 5, pp. 1163–1177, 2016.
[35] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, “Deep reinforcement learning with successor features for navigation across similar environments,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 2371–2378.