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 (DRL). Our approach uses local and global information for each robot from motion information maps. We use a three-layer CNN that takes these maps as input to 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 dense, complex benchmarks with narrow passages and 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 while avoiding collisions with each other and the obstacles in the environment. As larger numbers of robots are used, more efficient methods are needed that can handle dense and complex 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 [1], [2], [3], [4], [5], [6] or decentralized planners [7], [8], [9], [10]. 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 [11] or to handle challenging scenarios with narrow passages (see Figures 1 or 2).

Recently, there has been considerable work on developing new, learning-based planning algorithms [12], [13], [14], [15], [16], [17] for navigating one or more robots through dense scenarios. Most of these learning methods tend to learn an optimal policy using a multi-scenario, multi-stage training algorithm. However, current learning-based methods are limited to using only the local information and do not exploit any global information. Therefore, it is difficult to use them in dense environments or narrow passages.

Main Results: We present a novel, multi-robot navigation algorithm (DeepMNavigate) based on reinforcement learning that exploits *a combination of 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 in dense environments with many tens of robots (e.g., 90 robots) navigating in tight scenarios with narrow passages. As compared to prior multi-robot methods, our approach offers the following benefits:

1) We use global knowledge in terms of motion information maps in our network to improve the performance of DRL. This also results in higher reward value.
2) Our approach scales with the number of robots and is able to compute collision-free and smooth trajectories. Running our trained system takes many tens of seconds on a PC with a 32-core CPU and one NVIDIA RTX 2080 Ti on multi-robot systems with $10 - 90$ robots.
3) We can easily handle challenging multi-robot scenarios like inter-changing robot positions or multiple narrow corridors, which are difficult for prior geometric decentralized or local learning methods. In particular, we highlight the performance on five difficult environments that are very different from our training

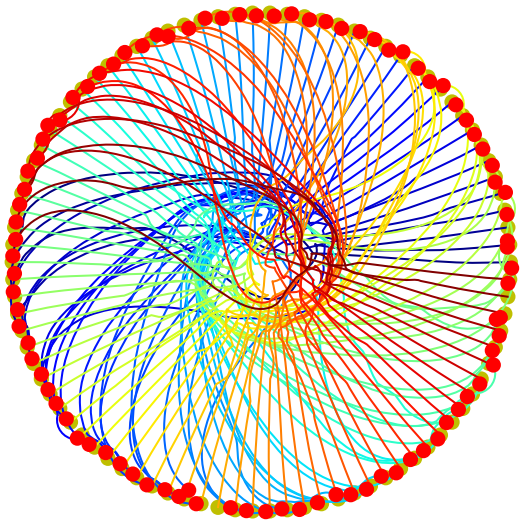


Fig. 1: **Circle Crossing:** Simulated trajectories of 90 robots in circle crossing scenarios generated by our algorithm that uses global information. The yellow points are the initial positions of the robots and the red points are the diametrically opposite goals of the robot. Our DeepMNavigate algorithm can handle such scenarios without collisions along the trajectories, and all the robots reach their goals. Prior learning methods that only use local methods [16], [12], [14] cannot handle such scenarios, as the robots tend to get stuck.

[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}

scenarios and have more agents. This demonstrates the generalizability of our method.

II. Related Work

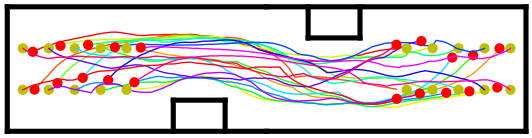
A. Geometric Multi-Robot Navigation Algorithms

Most prior algorithms are based on geometric techniques such as 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 [18], [19], [20], [21] and compute a safe, optimal, and complete solution for navigation. However, they do not scale to large multi-robot systems with tens of robots. Many pratical geometric decentralized methods for multi-agent systems are based on reciprocal velocity obstacles [7] or its variants [22]. These synthetic methods can be used during the training phase of learning algorithms.

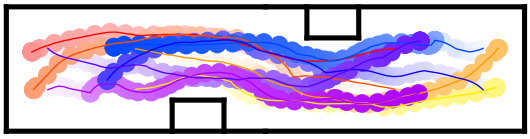
B. Learning-Based Navigation Methods

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. [23] present a vision-based static obstacle avoidance system using a 6-layer CNN to map input images to steering angles. Zhang et al. [24] describe a successor-feature-based deep reinforcement learning algorithm for robot navigation tasks based on raw sensory data. Barreto et al. [25] apply transfer learning to deploy a policy for new problem instances. Sergeant et al. [26] 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. [27] adapt an imitation learning technique to train reactive heading policies based on the knowledge of a human pilot. Pfeiffer et al. [28] 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.

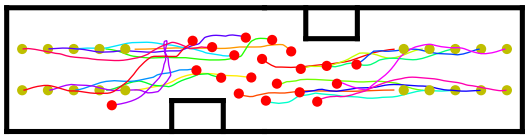
To overcome the limitations of supervised-learning, Tai et al. [29] present a mapless motion planner trained end-to-end without any manually designed features or prior demonstrations. Kahn et al. [30] propose an uncertainty-aware model-based learning algorithm that estimates the probability of collision, then 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. [31] 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. [14], [32] and Everett et al. [15] present multi-robot collision avoidance policies based on deep reinforcement learning, requiring the deployment of 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. [12] and Long et al. [16], [17] describe a decentralized multi-robot collision avoidance framework where each robot makes navigation decisions independently without any communication with other agents. It has been extended in terms of multiple sensors and explicit pedestrian motion prediction [34]. Other methods account for social constraints [15]. 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 or handle challenging narrow scenarios.



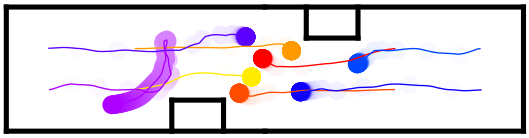
(a) Our algorithm is able to compute collision-free trajectories for 20 robots.



(b) We highlight some of the computed trajectories with temporal information computed using our algorithm.



(c) Trajectories computed using local learning method [16]. All agents do not reach the goal position.



(d) Selected trajectories computed by [16] with temporal information. The agents get stuck.

Fig. 2: **Narrow Corridor:** We compute the trajectories computed by DeepMNavigate and [16] for two groups of robots (20 total) exchanging their positions through narrow corridors. In (a) and (c), the yellow points correspond to the initial positions and the red points correspond the final positions. (b) and (d), highlight the temporal information along the trajectories using color and transparency. Prior local planning methods [16] can only handle these scenarios with up to 12 agents and the geometric decentralized methods [7] cannot handle such cases. This benchmark is quite different from training datasets.

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 it can be extended to 3D workspaces and to robots with other dynamics constraints.

Let the number of robots be N_{rob}. We represent each robot 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 an 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 includes the global state of the system, discussed in Section IV. In this paper, we focus on analyzing and incorporating motion information in the navigation system. Meanwhile, there are N_{obs} static obstacles in the environment. We use $\mathbf{B}_k$ to denote the area occupied by a static k-th obstacle. The computed action $\mathbf{a}^t$ drives the robot to its goal while avoiding collisions with other robots and obstacles within the timestep Δt until the next observation $\mathbf{o}^{t+1}$ is received.

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

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

where p_i^0 is the initial position of the robot and p_i^t are the positions at timestep t. v_i^t is the current linear velocity on the 2D plane (i.e. v_x, v_y) as a result of the action a_i^t. The agent we are simulating is non-holonomic and can only control the linear velocity on the X axis and the angular velocity on the Z axis (which is used to describe the rotation in the 2D plane).

B. Multi-Agent Navigation Using Reinforcement Learning

Our approach builds on prior reinforcement learning approaches that use local information comprised 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 $\mathbf{o}_g^t$ in these cases is a 2D vector representing the goal position in polar coordinates with respect to the robot's current position. The observed velocity $\mathbf{o}_v^t$ includes the current velocity of the robot. These observations are normalized using the statistics aggregated during training[35], [36]. This normalization can make RL training more stable and improve its performance. The action of a differential robot includes the translational and rotational velocity, i.e. $\mathbf{a}^t = [v^t, \omega^t], v \in (0, 1), \omega \in (-1, 1)$. We use the following reward function to guide a team of robots:

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

When the robot gets closer or reaches its goal, it is rewarded as

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

The $\|\mathbf{p}_i - \mathbf{g}_i\|$ item denotes the distance between the robot and its goal. When there is a collision, it is penalized using the function

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

In addition to collision avoidance, one of our goals is generating a smooth path. A simple technique is to impose penalties whenever there are large rotational velocities. Although it is not a standard way to obtain smooth path, we found this technique can achieve a smooth trajectory empirically. This can be expressed as

$$({}^\omega r)_i^t = \begin{cases} r_{\text{smooth}} |\omega_i^t| & \text{if } |\omega_i^t| > 0.7. \\ 0 & \text{otherwise,} \end{cases} \tag{5}$$

where r_{arrival}, $r_{\text{approaching}}$, $r_{\text{collision}}$ and r_{smooth} are parameters used to control the reward. These parameters provide reward feedback for the agents, which makes the training process more stable [37]. In practice, the reward parameters can be tuned to obtain desirable behaviors (e.g., learn more conservative behaviors without a larger collision penalty). We do not change the reward function when we use our approach for different environments.

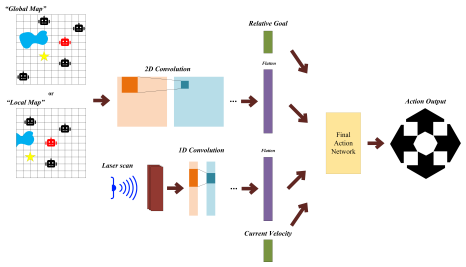


Fig. 3: We highlight the architecture of our policy network (DeepMNavigate), including (*global* and *local*) maps used by our approach. The *global map* is based on the world coordinate system and each *local map* is centered at the corresponding robot's current location. The red robot represents the map's corresponding robot, black robots represent the neighboring robots, the yellow star represents the goal, and the blue area is an obstacle. In our implementation, the map is discretized and assigned different values. We use a 2D convolutional neural network to handle the additional global information input from the map and fully-connected network to compute the action for each robot.

IV. DeepMNavigate: Trajectory Computation Using Global Information

In this section, we present our novel, learning-based, multi-agent navigation algorithm that uses 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 [7] 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 be either 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. 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 use the following method to compute the global-map and the local-map. 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, 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:

$$M_i^t(p,q) = \begin{cases} 1, & \{y | \|y - \mathbf{x}_i^t\| \leq R\} \bigcup \mathcal{A}_{pq} \neq \emptyset \\ 2 & \exists 1 \leq j \leq N_{\text{rob}},\ j \neq i, \\ & \quad s.t.\ \{y | \|y - \mathbf{x}_j^t\| \leq R\} \bigcup \mathcal{A}_{pq} \neq \emptyset \\ 3, & \{y | \|y - \mathbf{g}_i\| \leq R\} \bigcup \mathcal{A}_{pq} \neq \emptyset \\ 4, & \exists 1 \leq k \leq N_{\text{obs}}, s.t.\ \mathbf{B}_k \bigcup \mathcal{A}_{pq} \neq \emptyset \\ 0, & \text{otherwise,} \end{cases} . \quad (6)$$

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). This is highlighted in Fig. 3.

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 scenes with no clear boundaries or very large environments spread over a large area. If we use a *global-map* with the world coordinate representation for all the agents, the resulting map would be extremely large and would involve a high computational cost and memory overhead. In these cases, we use the *local-map* for each agent, instead of considering the whole scenario, which whold have a size $H \times W$ These local maps only account for information in a relatively small neighborhood with a fixed size $H_l \times W_l$. 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 as additional channels.

B. Proximal Policy Optimization

We use proximal policy optimization [38] to optimize the overall system. This training algorithm has the stability and reliability of trust-region methods: i.e., 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 each robot in the entire system reaches its goal or stops 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, the goal positions and obstacle information using Eq. 6. To feed the input to our neural network, we generate dense representations using 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 locations 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[16], 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. This 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 and Algorithm 1.

Algorithm 1 Policy Making for DeepMNavigate

for timestep $t = 1, 2, ...$ **do**
  // Each robot runs individually in parallel
  for robot $i = 1, 2, ...N$ **do**
    Collect observation $\mathbf{o}_i^t = [\mathbf{o}_z^t, \mathbf{o}_g^t, \mathbf{o}_v^t, \mathbf{o}_M^t]$
    Run policy π_θ represented by the network, get action $\mathbf{a}_i^t$
    Update the robot position $\mathbf{p}_i^t$ according to action $\mathbf{a}_i^t$
  end for
end for

D. Network Training

Our training strategy extends the method used by learning algorithms based on local information [16], [14]. To accelerate the training process, we 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 include a more challenging environments, such as a narrow passage,

Layer	Convolutional Filter	Stride	Padding	Activation Fuction	Output Size
Input	-	-	-	-	$250 \times 250 \times 1$
Conv 1	$7 \times 7 \times 1 \times 8$	1×1	'SAME'	ReLU	$250 \times 250 \times 8$
Max Pooling 1	3×3	2×2	'SAME'	-	$125 \times 125 \times 8$
Conv 2	$7 \times 7 \times 8 \times 12$	1×1	'SAME'	ReLU	$125 \times 125 \times 12$
Max Pooling 2	3×3	2×2	'SAME'	-	$63 \times 63 \times 12$
Conv 3	$7 \times 7 \times 12 \times 20$	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	384
Fully connected	128×64	-	-	ReLU	256

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

Fig. 4: We highlight the reward as a function of the number of iterations during the second stage of the overall training algorithm. We compare the performance of a reinforcement learning algorithm that only uses local information [16] to our method, which uses local and global information. Our method obtains a higher reward than [16] due to the global information.

random obstacles, etc. We show the training scenarios in Fig. 5. These varying training environments and the large number of robots in the system can result in a good overall 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 small, simple network that only accounts for local information. To accelerate the training process and generate accurate results, we do not train the entire network from scratch and instead include pre-training. During the first stage, we retrain the network with additional structures corresponding to the global information (i.e. the global-map) using the pre-trained local information network part. This pre-training stage uses the parameters proposed in [16]. We highlight the reward as a function of the iterations during the second stage of the training and compare the overall reward computation with that described in the local method [16] in Fig. 4. Notice that, since we use a 2D convolutional neural network, our overall training algorithm needs more time during each iteration of the training (ours around 1200s vs [16] around 400s). As a result, we do not perform the same number of training iterations as[16], as shown in Fig. 4. The total training time is around 40 hours.

V. Implementation and Performance

In this section, we discuss 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 [16].

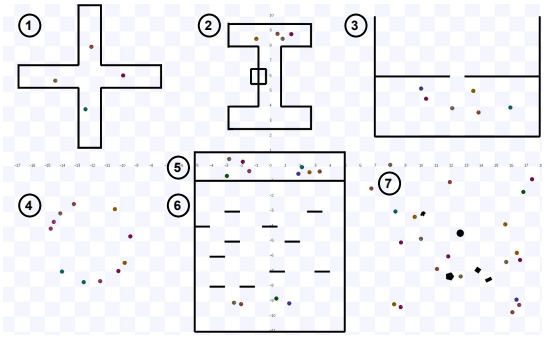


Fig. 5: Scenarios corresponding to challenging environments used in the second stage of training. This two-stage training improves the performance.

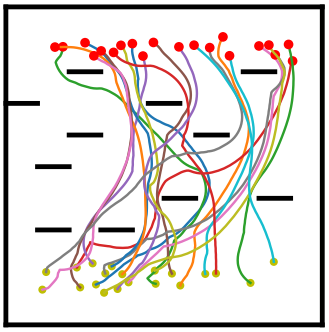


Fig. 6: **Room with Obstacles**: We highlight the trajectories of 20 robots in a room with multiple obstacles. We highlight the initial position of the agent (yellow) and the final position (red) along with multiple obstacles. Prior learning methods that only use local methods [16], [14] will take more time and may not be able to handle such scenarios when the number of obstacles or the number of agents increases.

A. Parameters

During simulation, the radius is set as $R = 0.12m$. In the current implementation, we set $H = W = 500m$ for the global-map and $H_l = W_l = 250m$ 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 instance, the memory requirement of CNN increases quadratically with the input size. In current implementation, we use a PC with 32-core CPU, 32GB memory and one NVIDIA RTX 2080 Ti. To include the additional global information, the algorithm consumes 1.63GB CPU memory, 970MB GPU memory and requires 0.25s for computing one time step, compared to [16] as 1.57 GB CPU memory, 340MB GPU memory and 0.2s. The overhead is not significant. For parameters in the reward function, we set $r_{\text{arrival}} = 15$, $r_{\text{collision}} = -15$, $r_{\text{approaching}} = 2.5$, and $r_{\text{smooth}} = -0.1$. We choose r_{arrival} and $r_{\text{collision}}$ of the same magnitude to obtain an effective, safe behavior, which is a key metric to evaluate robots' trajectories; $r_{\text{approaching}}$ is chosen to encourage robots to approach their goal as fast as they can, which provides a

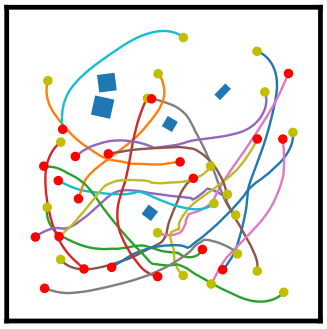


Fig. 7: **Random Start and Goal Positions:** Simulated trajectories of 20 robots from and to random positions in a scene with obstacles. The yellow points are the initial positions and the red points are the final positions. The blue areas are obstacles with random locations and orientations.

dense feedback to make convergence relatively faster; a small r_{smooth} value should regularize the trajectory and make it smoother.

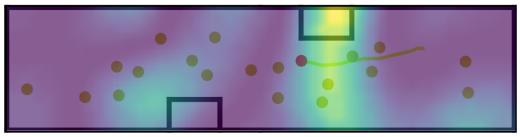


Fig. 8: Perturbation saliency generated by method [39] in the narrow corridor benchmark. Red point represents the current agent and yellow points represent the surrounding agents. Yellow area is saliency for the action policy.

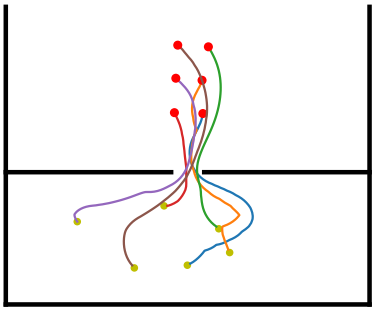


Fig. 9: **Room Evacuation:** Simulated trajectories of 6 robots evacuating a room with our algorithm. The yellow points are the initial positions and the red points are the final positions. Even with a narrow passage, our DeepMNavigate algorithm works well. This benchmark is quite different from training data.

B. Evaluation Metrics and Benchmarks

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

- *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.
- *Collision or stuck rate*: if the robot cannot reach the destination within a limited time or collides, they are considered as getting stuck or in-collisions, respectively.
- *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 when going straight towards the goal at the maximum speed without checking for any collisions.
- *Average speed*: the average speed of all robots during the navigation.

We have evaluated our algorithm in five challenging and representative 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 in prior multi-agent navigation algorithms [7], [16] (Fig. 1).
- *Narrow Corridor*: Two groups of robots exchange their positions through a narrow corridor. This benchmark is hard for geometric decentralized methods [7], which cannot navigate robots through narrow passages (Fig. 2).
- *Room with Obstacles*: The robots cross across a room full of obstacles, from one side to the other. Methods only using local information [16] will spend more time finding the path to the goal and may even fail due to lack of global information (Fig. 6).
- *Random Starts and Goals*: The robots start from random initial positions and moves to random goal positions. Also, there are obstacles with random locations and orientation. Global information will help find safer and faster paths (Fig. 7).
- *Room Evacuation*: The robots start from random initial positions and evacuate to the outside of the room. They need to cross one small door while avoiding collisions (Fig. 9).

C. New and Different Benchmarks

We have evaluated the performance of our method on benchmarks that are quite different from the training data in terms of layout and the inclusion of narrow passages. They also use different numbers of agents. In the Circle Crossing benchmark, we only train with 12 agents, but evaluate in a similar scene with 90 agents. Furthermore, some of the benchmarks like Narrow Corridor and Room Evacuation are quite different from the training dataset. DeepMNavigate is still able to compute collision-free and smooth trajectories for all the agents with no collisions and each agent arrives at its goal position. As shown in Fig. 2, the local learning method [16] does not consider the global map information and fails in such scenarios. In contrast, our approach enable robots to learn a reciprocal navigation behavior according to the global map information without any communication on action decision.

D. Quantitative Evaluation

We have evaluated the performance of our algorithm in terms of different evaluation metrics described above. In the circle crossing scenario, the failure rate of [16] rises with the increasing number or the density of robots. However, our approach always results in a stable performance and can

Metrics	Methods	# of agents (cirle radius (unit:m))						
		30 (8)	40 (8)	50 (8)	60 (8)	70 (8)	80 (12)	90 (12)
Success Rate	[16]	**1**	0.975	0.96	0.95	0.929	0.7375	0.722
	Ours	**1**	**1**	**1**	**1**	0.986	**1**	**1**
Stuck/Collision Rate	[16]	**0/0**	0/0.025	0/0.04	0/0.05	0/0.071	0.175/0.0875	0.233/0.044
	Ours	**0/0**	**0/0**	**0/0**	**0/0**	**0/0.014**	**0/0**	**0/0**
Extra Time	[16]	**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	[16]	**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	[16]	0.875	0.75	0.5625	0.0
	Ours	**1**	**1**	**1**	**1**
Stuck/Collision Rate	[16]	0/0.125	0/0.25	0.3125/0.125	1/0
	Ours	**0/0**	**0/0**	**0/0**	**0/0**
Extra Time	[16]	4.8	12.5111	30.7222	-
	Ours	**2.7**	**4.075**	**5.33125**	**8.36**
Average Speed	[16]	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	[16]	**1**	0.7	0.6	0.7
	Ours	**1**	**1**	**1**	**1**
Stuck/Collision Rate	[16]	**0/0**	0.1/0.2	0.133/0.267	0.15/0.15
	Ours	**0/0**	**0/0**	**0/0**	**0/0**
Extra Time	[16]	9.55894	5.13058	6.38476	9.99585
	Ours	**2.19487**	**2.55377**	**3.47301**	**7.80055**
Average Speed	[16]	0.707441	0.734304	0.721682	0.508305
	Ours	**0.858985**	**0.828276**	**0.775802**	**0.582426**

TABLE II: The performance of our method (DeepMNavigate) and prior methods based on local information on different benchmarks (**Top**: *Circle Crossing*; **Bottom left**: *Narrow Corridor*; **Bottom right**: *Room with Obstacles*.), measured in terms of various metrics using 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.

Metrics	Methods	# of agents			
		20	30	40	50
Success Rate	[16]	**1**	0.867	0.825	0.76
	Ours	**1**	**1**	**1**	**1**
Stuck/Collision Rate	[16]	**0/0**	0.033/0.1	0.05/0.125	0.2/0.04
	Ours	**0/0**	**0/0**	**0/0**	**0/0**
Extra Time	[16]	4.5974	9.38872	13.8348	12.6948
	Ours	**3.59674**	**4.06355**	**10.4823**	**11.2948**
Average Speed	[16]	0.601266	0.432238	0.354548	0.332425
	Ours	**0.674891**	**0.63543**	**0.404211**	**0.383049**

TABLE III: The performance of our proposed method and prior learning algorithms on the *Random Starts and Goals* benchmark. Our method demonstrates better results (bold face) than prior multi-agent navigation algorithms.

avoid the deadlock situation. At times, some of the robots may need to take a longer path to avoid congestion and this could reduce the robot's efficiency. As a result, we obtain better performance as compared to [16] or the decentralized collision avoidance methods in high density benchmarks like Circle Crossing.

Different from the circle crossing scenario, the other benchmarks incorporate some static obstacles in the environment. In this case, our method integrates the map information into the policy network and utilizes that information to handle such static obstacles and narrow passages. As the experimental results shown, our approach outperforms [16] both in terms of success rate and efficiency metric. In addition, our method behaves robustly even with the increasing density of robots.

Another important criteria to evaluate the performance of multi-robot systems is the stuck or collision rate, which is a measure of the number of robots cannot reach the goals or collide on the road. As shown in Tables II and III, our collision rate is zero for all these benchmarks. On the other hand, techniques based on local navigation information result in some number of failures on different benchmarks. Furthermore, the failure increases as the number of agents or the density increases.

To better understand how the global information help the navigation system, we compute pertubation saliency over global map using method from [39] in the narrow corridor benchmark. We show the result in Fig. 8). The most important areas in the global map to help the decision making include: 1) agents in the front, which are blocked by other agents in local laser scan; 2) agents from the back, which could not be covered by local laser scan; 3) nearby obstacle. Thus, the global information can help the agent plan in advance, and be more alert to nearby obstacles.

E. Scalability

The running times for different numbers of robots are shown in Fig. 10. We can observe linear time performance with number of agents. That is because the decision process of our method is independent, each agent can compute their action by itself, based on the information it receives. Compared with most traditional geometric-based methods, a step analyzing the global environment may have super-linear time requirements, especially for the congested or challenging benchmarks used in this paper. For example, some methods compute K-nearest neighbors or the roadmap of the environment using the Voronoi diagram, which can have super-linear complexity. In contrast, our approach does not perform any such global computations and only leverages the power of the neural network. Moreover, the computation

of the global map takes $O(n)$ time.

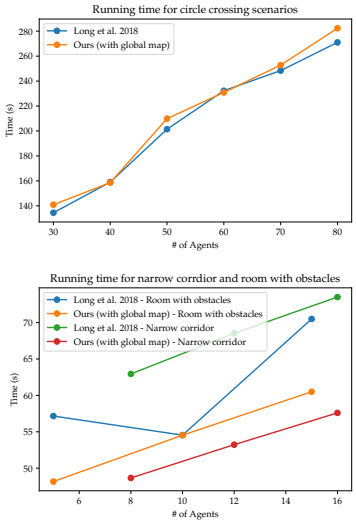


Fig. 10: We show the running time with different numbers of agents in several benchmarks. We use a CPU with 32 cores and NVIDIA RTX 2080 Ti to generate these performance graphs. This timing graph demonstrates that our approach is practical for many tens of robots. We also compare the running time with a learning-based algorithm that only uses local information [16]. The additional overhead in the running time with the use of the global information is rather small.

VI. Conclusion, Limitations, and Future Work

We present a novel, multi-agent navigation algorithm based on deep reinforcement learning. We show that global information about the environment can be used based on the global-map and present a novel network architecture to compute a collision-free 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. Moreover, our experimental results show that our DeepMNavigation algorithm can offer improved performance in dense and narrow scenarios, as compared to prior approaches. Furthermore, we demonstrate the performance on new, different benchmarks that are different from training scenarios. Overall, our approach demonstrates the value of global information in terms of discretized maps for DRL-based ethod.

Our results are promising and there are many ways to improve the performance. Our idea can be extended to 3D environments, by replacing the global map with a 3D version and using a 3D convolutional neural network to process it. Current training scenarios do not include dynamic and dense obstacles, so we may include them in the future work. Also, we need better techniques to compute the optimal size of the global-map and the 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. Our approach assumes that the global information is available and that, in many scenarios, obtaining such information could be expensive. 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 [7], [8]. Such a combination of local and global methods has been used to simulate large crowds [40] and it may be useful to develop a similar framework for learning-based algorithms. Furthermore, DRL-based methods that use local or global information can also be combined with global navigation data structures (e.g., roadmaps) to further improve navigation performance.

We have only demonstrated the application of our DRL-method on challenging, synthetic environments. A good area for future work is extending to real-world scenes, where we need to use other techniques to generate the motion information maps. It would be useful to combine our learning methods with SLAM techniques to improve the navigation capabilities.

References

[1] J. P. Van Den Berg and M. H. Overmars, "Prioritized motion planning for multiple robots," in *IROS*. IEEE, 2005, pp. 430–435.

[2] 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, 2009.

[3] R. J. Luna and K. E. Bekris, "Push and swap: Fast cooperative path-finding with completeness guarantees," in *IJCAI*, 2011.

[4] G. Sanchez and J.-C. Latombe, "Using a prm planner to compare centralized and decoupled planning for multi-robot systems," in *ICRA*, vol. 2. IEEE, 2002, pp. 2112–2119.

[5] 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.

[6] J. Yu and D. Rus, "An effective algorithmic framework for near optimal multi-robot path planning," in *Robotics Research*. Springer, 2018, pp. 495–511.

[7] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Robotics research*. Springer, 2011.

[8] 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*, 2008.

[9] D. Helbing and P. Molnar, "Social force model for pedestrian dynamics," *Physical review E*, vol. 51, no. 5, p. 4282, 1995.

[10] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, 1997.

[11] T. Fraichard and H. Asama, "Inevitable collision states—a step towards safer robots?" *Advanced Robotics*, 2004.

[12] 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*, 2018.

[13] 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.

[14] 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.

[15] 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.

[16] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," in *ICRA*. IEEE, 2018, pp. 6252–6259.

[17] 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.

[18] R. Luna and K. E. Bekris, "Efficient and complete centralized multi-robot path planning," in *IROS*. IEEE, 2011, pp. 3268–3275.

[19] 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.

[20] 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.

[21] S. Tang, J. Thomas, and V. Kumar, "Hold or take optimal plan (hoop): A quadratic programming approach to multi-robot trajectory generation," *IJRR*, vol. 37, no. 9, pp. 1062–1084, 2018.

[22] 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.

[23] 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.

[24] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, "Deep reinforcement learning with successor features for navigation across similar environments," in *IROS*. IEEE, 2017, pp. 2371–2378.

[25] 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 *NIPS*, 2017, pp. 4055–4065.

[26] 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.

[27] 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 *ICRA*. IEEE, 2013.

[28] 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 *ICRA*, 2017.

[29] L. Tai, G. Paolo, and M. Liu, "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation," in *IROS*. IEEE, 2017, pp. 31–36.

[30] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine, "Uncertainty-aware reinforcement learning for collision avoidance," *arXiv*, 2017.

[31] 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.

[32] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *IROS*, 2017.

[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] A. Jagan Sathyamoorthy, J. Liang, U. Patel, T. Guan, R. Chandra, and D. Manocha, "Densecavoid: Real-time navigation in dense crowds using anticipatory behaviors," *arXiv*, pp. arXiv–2002, 2020.

[35] Wikipedia contributors, "Algorithms for calculating variance — Wikipedia, the free encyclopedia," 2020, [Online; accessed 31-May-2020]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Algorithms_for_calculating_variance&oldid=959752885

[36] L. Engstrom, A. Ilyas, S. Santurkar, D. Tsipras, F. Janoos, L. Rudolph, and A. Madry, "Implementation matters in deep rl: A case study on ppo and trpo," in *ICLR*, 2020.

[37] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming exploration in reinforcement learning with demonstrations," in *ICRA*. IEEE, 2018, pp. 6292–6299.

[38] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv7*, 2017.

[39] S. Greydanus, A. Koul, J. Dodge, and A. Fern, "Visualizing and understanding atari agents," *arXiv preprint arXiv:1711.00138*, 2017.

[40] R. Narain, A. Golas, S. Curtis, and M. C. Lin, "Aggregate dynamics for dense crowd simulation," in *TOG*. ACM, 2009.