Mobile Robot Path Planning in Dynamic Environments Through Globally Guided Reinforcement Learning

Binyu Wang, Zhe Liu, *Member, IEEE*, Qingbiao Li, and Amanda Prorok, *Member, IEEE*

***Abstract*—Path planning for mobile robots in large dynamic environments is a challenging problem, as the robots are required to efficiently reach their given goals while simultaneously avoiding potential conflicts with other robots or dynamic objects. In the presence of dynamic obstacles, traditional solutions usually employ re-planning strategies, which re-call a planning algorithm to search for an alternative path whenever the robot encounters a conflict. However, such re-planning strategies often cause unnecessary detours. To address this issue, we propose a learning-based technique that exploits environmental spatio-temporal information. Different from existing learning-based methods, we introduce a *globally guided reinforcement learning* approach (G2RL), which incorporates a novel reward structure that generalizes to arbitrary environments. We apply G2RL to solve the multi-robot path planning problem in a fully distributed reactive manner. We evaluate our method across different map types, obstacle densities, and the number of robots. Experimental results show that G2RL generalizes well, outperforming existing distributed methods, and performing very similarly to fully centralized state-of-the-art benchmarks.**

***Index Terms*—Hierarchical path planning, mobile robots, reinforcement learning, scalability.**

I. Introduction

PATH planning is one of the fundamental problems in robotics. It can be formulated as: given a robot and a description of the environment, plan a conflict-free path between the specified start and goal locations. Traditionally, there are two different versions: off-line planning, which assumes static obstacles and perfectly known environments, and on-line planning, which focuses on dealing with dynamic obstacles and partially known environments [1]. Traditional off-line planning algorithms [2] cannot be directly utilized for solving on-line path planning tasks as they assume that the obstacles are static. One strategy is to plan an initial path and invoke re-planning whenever its execution becomes infeasible [3]. However, re-planning will suffer from time inefficiency (frequent re-planning) and path inefficiency (oscillating movements and detours) due to the absence of motion information of dynamic obstacles. Furthermore, re-planning strategies may fail in the presence of robot deadlocks. Instead of re-planning, some methods include an extra time dimension to avoid potential conflicts [4]. However, this approach increases the number of states to be searched, and additionally requires the knowledge of the future trajectories of dynamic obstacles. If the future movements of obstacles are unknown, one may attempt to model the behavior of dynamic obstacles and predict their paths [5]. Yet, separating the navigation problem into disjoint prediction and planning steps can lead to the 'freezing robot' problem. In that case, the robot will fail to find any feasible action as the predicted paths could mark a large portion of the space as untraversable.

Recently, learning-based approaches have been studied to address on-line planning in dynamic environments [6], [7]. This is popularized by the seminal work [8], which utilized deep neural networks for the function estimation of value-based reinforcement learning (RL). Although RL has demonstrated outstanding performance in many applications, several challenges still impede its contribution to the path planning problem. First of all, when the environment is extremely large, the reward becomes sparse, inducing an increased training effort and making the overall learning process inefficient [9]. Another challenge is the over-fitting issue. The robot is often limited to training environments and shows poor generalizability to unseen environments [10]. Most recent approaches still show difficulties in scaling to arbitrarily large multi-robot systems [7], as the sizes of the robot state, joint action, and joint observation spaces grow exponentially with the number of robots [11]. Thus, the efficiency, generalizability, and scalability of existing RL-based planners can still not fulfill the requirements of many applications.

In order to overcome the above challenges, we develop a hierarchical path-planning algorithm that combines a *global guidance* and a *local RL-based planner*. Concretely, we first utilize a global path planning algorithm (for example, A*) to obtain a globally optimal path, which we refer to as the *global guidance*. During robot motion, the *local RL-based planner*

Manuscript received May 11, 2020; accepted September 3, 2020. Date of publication September 24, 2020; date of current version October 6, 2020. This letter was recommended for publication by Associate Editor N. Amato and M. Morales upon evaluation of the Reviewers' comments. This work was supported by the Engineering and Physical Sciences Research Council (grant EP/S015493/1), and ARL DCIST CRA W911NF- 17-2-0181. *(Corresponding author: Zhe Liu.)*

Binyu Wang is with the Department of Mechanical and Automation Engineering, The Chinese University of Hong Kong, Hong Kong (e-mail: zbwby819@sina.com).

Zhe Liu, Qingbiao Li, and Amanda Prorok are with the Department of Computer Science and Technology, University of Cambridge, Cambridge CB3 0FD, U.K. (e-mail: zl457@cam.ac.uk; ql295@cam.ac.uk; asp45@cam.ac.uk).

Digital Object Identifier 10.1109/LRA.2020.3026638

generates robot actions by exploiting surrounding environmental information to avoid conflicts with static and dynamic obstacles, while simultaneously attempting to follow the fixed global guidance. Our main contributions include:

- We present a hierarchical framework that combines global guidance and local RL-based planning to enable end-to-end learning in dynamic environments. The local RL planner exploits both spatial and temporal information within a local area (e.g., a field of view) to avoid potential collisions and unnecessary detours. Introducing global guidance allows the robot to learn to navigate towards its destination through a *fixed-sized* learning model, even in large-scale environments, thus ensuring scalability of the approach.
- We present a novel reward structure that provides dense rewards, while not requiring the robot to strictly follow the global guidance at every step, thus encouraging the robot to explore all potential solutions. In addition, our reward function is independent of the environment, thus enabling scalability as well as generalizability across environments.
- We provide an application of our approach to multi-robot path planning, whereby robot control is fully distributed and can be scaled to an arbitrary number of robots.
- Experimental results show that our single-robot path planning approach outperforms local and global re-planning methods, and that it maintains consistent performance across numerous scenarios, which vary in map types and number of obstacles. In particular, we show that our application to multi-robot path planning outperforms current state-of-the-art distributed planners. Notably, the performance of our approach is shown to be comparable to that of centralized approaches, which, in contrast to our approach, assume global knowledge (i.e., trajectories of all dynamic objects).

II. Background and Related Work

Traditional Path Planning Approaches: Path planning can be divided into two categories: global path planning and local path planning [12]. The former approach includes graph-based approaches (for example, Dijkstra and A* [13]) and sampling-based approaches (for example, RRT and its variant [14]), in which all the environmental information is known to the robot before it moves. For local path planning, at least a part or almost all the information on the environment is unknown. Compared to global path planners, local navigation methods can be very effective in dynamic environments. However, since they are essentially based on the fastest descent optimization, they can easily get trapped in a local minimum [15]. A promising solution is to combine the local planning with global planning, where the local path planner is responsible for amending or optimizing the trajectory proposed by the global planner. For instance, [16] proposed a global dynamic window approach that combines path planning and real-time obstacle avoidance, allowing robots to perform high-velocity, goal-directed, and reactive motion in unknown and dynamic environments. Yet their approach can result in highly sub-optimal paths. The authors in [17] adopt multi-policy decision making to realize autonomous navigation in dynamic social environments. However, in their work, the robots trajectory was selected from closed-loop behaviors whose utility can be predicted rather than explicitly planned.

Learning Based Approaches: Benefiting from recent advances in deep learning techniques, learning-based approaches have been considered as a promising direction to address path planning tasks. Reinforcement Learning (RL) has been implemented to solve the path planning problem successfully, where the robot learns to complete the task by trial-and-error. Traditionally, the robot receives the reward after it reaches the target location [9]. As the environment grows, however, the robot needs to explore more states to receive rewards. Consequently, interactions become more complex, and the learning process becomes more difficult. Other approaches apply Imitation Learning (IL) to provide the robot dense rewards to relieve this issue [6], [7], [18]. However, basing the learning procedure on potentially biased expert data may lead to sub-optimal solutions [19], [20]. Compounding this issue, the robot only receives rewards by strictly following the behavior of expert demonstrations, limiting exploration to other potential solutions. Also, over-fitting remains a problem. This is clearly exemplified in [10], where the robot follows the previously learned path, even when all obstacles have been removed from the environment.

III. Problem Description

Environment Representation: Consider a 2-dimensional discrete environment $\mathcal{W} \subseteq \mathbb{R}^2$ with size $H \times W$ and a set of N_s static obstacles $\mathcal{C}_s = \{s_1, \ldots, s_{N_s}\}$, where $s_i \subset \mathcal{W}$ denotes the i^{th} static obstacle. The free space $\mathcal{W} \setminus \mathcal{C}_s$ is represented by a roadmap $G = \langle \mathcal{C}_f, \mathcal{E} \rangle$, where $\mathcal{C}_f = \{c_1, \ldots, c_{N_f}\} = \mathcal{W} \setminus \mathcal{C}_s$ represents the set of free cells and $e_{ij} = (c_i, c_j) \subset \mathcal{E}$ represents the traversable space between free cells c_i and c_j that does not cross any other cell (the minimum road segment). The set of dynamic obstacles $\mathcal{C}_d(t) = \{d_1(t), \ldots, d_{N_d}(t)\}$ denotes the position of N_d dynamic obstacles at time t, where $\forall i, j, t, d_i(t) \subset \mathcal{C}_f$, $(d_i(t), d_i(t+1)) \subset \mathcal{E}$ or $d_i(t) = d_i(t+1)$, and $d_i(t) \neq d_j(t)$. In addition, if $d_i(t+1) = d_j(t)$, then $d_j(t+1) \neq d_i(t)$, i.e., any motion conflict should be avoided.

Global Guidance: A *traversable path* $\mathcal{G} = \{g(t)\}$ is defined by the following rules: 1) the initial location $g(0) = c_{start}$ and there is a time step t_f that $\forall t \geq t_f$, $g(t) = c_{goal}$. Note that, $c_{start}, c_{goal} \in \mathcal{C}_f$; 2) $\forall t$, $g(t) \in \mathcal{C}_f$, $(g(t), g(t+1)) \in \mathcal{E}$. The *global guidance* $\mathcal{G}^*$ is the shortest traversable path, defined as $\mathcal{G}^* = \arg\min_{t_f} \mathcal{G}$, between the initial location c_{start} and the goal c_{goal}. Note that the global guidance may not be unique—in this letter, we randomly choose one instance.

Assumptions: We assume that the robot knows the information of all the static obstacles and calculates the global guidance at the start of each run. Note that the global guidance is only calculated once and remains the same. During robot motion, we assume that the robot can obtain its global location in the environment and acquire global guidance information. However, we do not assume that the trajectories of dynamic obstacles are known to the robot. The robot can only obtain the current location of dynamic obstacles when they are within its local field of view.

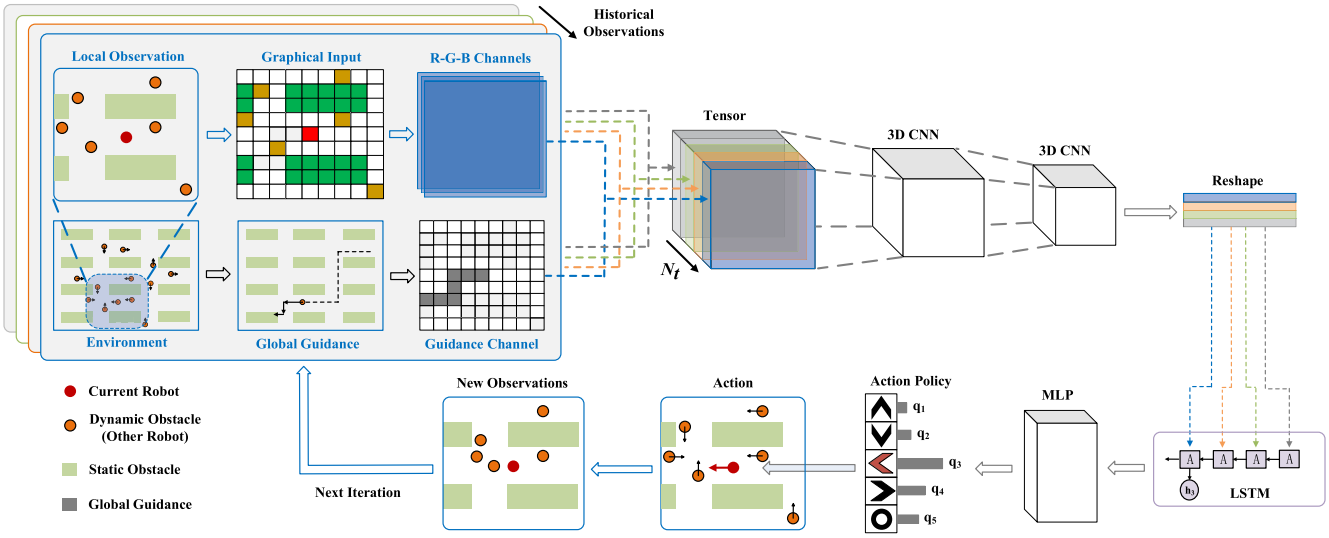


Fig. 1. The overall structure of G2RL. The input of each step is the concatenation of the transformed local observation and the global guidance information. A sequence of historical inputs is combined to build the input tensor of the deep neural network, which outputs a proper action for the robot.

Local Observation: The robot has a local field of view (FOV) within which it observes the environment. More specifically, at each time step t, the robot collects the local observation $O_t = \{o_t^f, o_t^s, o_t^d\}$ which is a collection of the location of free cells o_t^f, static obstacles o_t^s and dynamic obstacles o_t^d, within the local FOV with the size of $H_l \times W_l$. In addition, we also define a local segment of the global guidance $\mathcal{G}^*$ as the local path segment $\mathcal{G}_t^*$, which is located within the robot's local FOV. Both O_t and $\mathcal{G}_t^*$ are considered as input information at each time step t.

Robot Action: The robot action set is defined as $\mathcal{A} = \{\text{Up}, \text{Down}, \text{Left}, \text{Right}, \text{Idle}\}$, i.e., at each time step, the robot can only move to its neighboring locations or remain in its current location.

Objective: Given as input the local segment $\mathcal{G}_t^*$ of the global guidance, the current local observation $\mathcal{O}_t$, and a history of local observations $\mathcal{O}_{t-1}$,..., $\mathcal{O}_{t-(N_t-1)}$, output an action $a_t \subset \mathcal{A}$ at each time step t that enables the robot to move from the start cell c_{start} to the goal cell c_{goal} with the minimum number of steps while avoiding conflicts with static and dynamic obstacles.

IV. RL-Enhanced Hierarchical Path Planning

In this section, we first describe the overall system structure and then present details of our approach.

A. System Structure

Fig. 1 illustrates the overall system structure, which shows that our local RL planner contains four main modules:

- *Composition of Network Input:* Firstly, we transform the local observation O_t into a graphic and use its three channels (RGB data) in combination with a guidance channel to compose the current input. Then a sequence of historical inputs is used to build the final input tensor;
- *Spatial Processing:* Secondly, we utilize a series of 3D CNN layers to extract features from the input tensor, then reshape the feature into N_t one-dimensional vectors;
- *Temporal Processing:* Thirdly, we use an LSTM layer to further extract the temporal information by aggregating the N_t vectors;
- *Action Policy:* Finally, we use two fully connected (FC) layers to estimate the quality q_i of each state-action pair and choose the action $a_i \subset \mathcal{A}$ with $\max_i q_i$.

B. Global Guidance and Reward Function

The main role of global guidance is to provide long-term global information. By introducing this information, the robot receives frequent feedback signals in arbitrary scenarios, no matter how large the environment and how complex the local environments are. We achieve this by proposing a novel reward function that provides dense rewards while simultaneously not requiring the robot to follow the global guidance strictly. In this manner, we encourage the robot to explore all the potential solutions while also promoting convergence of the learning-based navigation.

More specifically, at each step, the reward function offers: 1) A small negative reward $r_1 < 0$ when the robot reaches a free cell which is not located on the global guidance; 2) A large negative reward $r_1 + r_2$ when the robot conflicts with a static obstacle or a dynamic obstacle, where $r_2 < r_1 < 0$; 3) A large positive reward $r_1 + N_e \times r_3$ when the robot reaches one of the cells on the global guidance path, where $r_3 > |r_1| > 0$ and N_e is the number of cells removed from the global guidance path, between the point where the robot first left that path, to the point where it rejoins it. The reward function can be defined formally as:

$$R(t) = \begin{cases} r_1 & \text{if } c_r(t+1) \in \mathcal{C}_f \setminus \mathcal{G}^* \\ r_1 + r_2 & \text{if } c_r(t+1) \in \mathcal{C}_s \cup \mathcal{C}_d(t+1) \\ r_1 + N_e \times r_3 & \text{if } c_r(t+1) \in \mathcal{G}^* \setminus \mathcal{C}_d(t+1) \end{cases} \quad (1)$$

where $c_r(t+1)$ is the robot location after the robot takes the action a_t at time t, $R(t)$ is the reward value of action a_t.

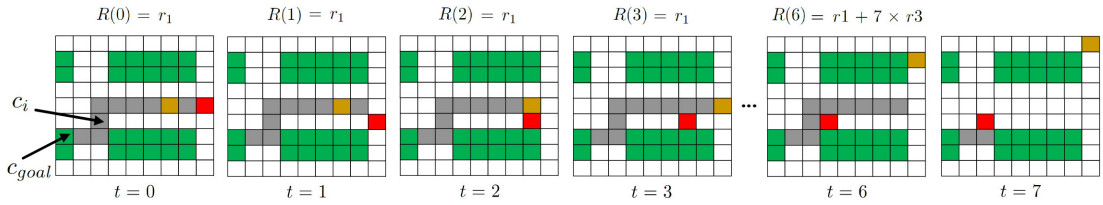


Fig. 2. An illustration of our reward function. The green, black, yellow and red cells represent the static obstacle, the global guidance, the dynamic obstacle and the robot, respectively. At $t = 7$, the robot reaches a global guidance cell c_i and receives the reward based on the number of eliminated guidance cells.

Fig. 2 shows an example of our reward function. At $t = 0$, since there is a dynamic obstacle in front, our RL planner moves the robot to the lower cell to avoid conflict. From $t = 1$ to $t = 6$, the robot does not need to return to the global guidance path immediately, but can continue to move left until it reaches one cell c_i located on the global guidance path at $t = 7$. Here $R(0) = R(1) = R(2) = R(3) = R(4) = R(5) = r_1$ and $R(6) = r_1 + 7 \times r_3$, since at $t = 7$, $N_e = 7$ cells have been removed in the global guidance. We remove the path segment up to c_i as soon as the robot obtains the reward $R(6)$ to ensure that the reward of each cell on the global guidance path can only be collected once. The removed guidance cells will be marked as normal free cells (white cells) in the graphic images inputted in the following iterations. In contrast to IL-based methods, we do not require the robot to strictly follow the global guidance at each step, since the robot receives the same cumulative reward as long as it reaches the same guidance cell given from the same start cell. As a result, our model can also be trained from scratch without IL, which circumvents potentially biased solutions [19]. In the training process, we stop the current episode once the robot deviates from the global guidance too much, namely, if no global guidance cell can be found in the FOV of the robot.

C. Local RL Planner

As shown in Fig. 1, we transform a robot's local observation into an *observation image* defined as: 1) The center pixel of the image corresponds to the current robot position, the image size is the same as the robot's FOV $H_l \times W_l$, i.e., one pixel in the image corresponds to one cell in the local environment; 2) All the static and dynamic obstacles observed are marked in the image, where we use one color to represent static obstacles and another color to denote dynamic ones. In addition, a guidance channel that contains the local path segment $\mathcal{G}_t^*$ of the global guidance is introduced to combine with the three channels (RGB data) of the observation image, to compose the current input $\mathcal{I}_t$. Our RL planner takes the sequence $\mathcal{S}_t = \{\mathcal{I}_t, \mathcal{I}_{t-1} \ldots \mathcal{I}_{t-(N_t-1)}\}$ as the inputs at step t.

We use Double Deep Q-Learning (DDQN) [21] for our RL planner. At time step t, the target value $Y_t = R(t)$, if $c_{t+1} = c_{goal}$, otherwise,

$$Y_t = R(t) + \gamma Q(\mathcal{S}_{t+1}, \arg\max_{a_t} Q(\mathcal{S}_{t+1}, a_t; \theta); \theta^-) \quad (2)$$

where $Q(.)$ is the quality function, θ and θ^- are the current and target network parameters respectively, γ is the discount value and $R(t)$ is defined in (1). To update the parameters θ, we sample N_b transitions from the replay buffer, and define the loss $\mathcal{L}$ as:

$$\mathcal{L} = \frac{1}{N_b} \sum_{j=1}^{N_b} (Y_t^j - Q(\mathcal{S}_t^j, a_t^j; \theta))^2 \quad (3)$$

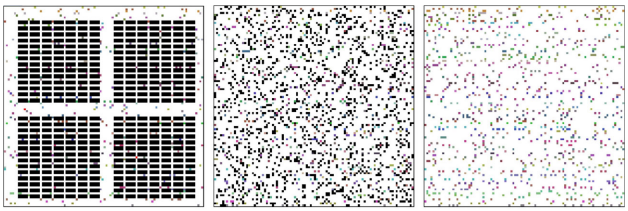
As shown in Fig. 1, our model is comprised of 3D CNN layers followed by LSTM and FC layers. The input is a five-dimensional tensor with size $N_t \times H_l \times W_l \times 4$, where N_b is the batch size sampled from the replay buffer. We first use 3D CNN layers to extract the spatial information. The size of the convolutional kernel is $1 \times 3 \times 3$. We stack one CNN layer with kernel stride $1 \times 1 \times 1$ and another with stride $1 \times 2 \times 2$ as a convolutional block, which repeats N_c times for downsampling in the spatial dimension. The embeddings extracted by the CNN layers are reshaped into N_t one-dimensional vectors and fed into the LSTM layer to extract temporal information. Two FC layers are attached to the LSTM layer, where the first layer (followed by a `ReLU` activation function) reasons about the extracted information and the second layer directly outputs the value q_i of each state-action pair $(\mathcal{S}_t, a_i)$ with a `Linear` activation function.

D. Application to Reactive Multi-Robot Path Planning

The main idea of our local RL planner is to encourage the robot to learn how to avoid surrounding obstacles while following global guidance. Since robots only consider local observations and global guidance, and do not need to explicitly know any trajectory information nor motion intentions of other dynamic obstacles or robots, the resulting policy can be copied onto any number of robots and, hence, scales to arbitrary numbers of robots. Based on the aforementioned rationale, our approach is easily extended to resolving the multi-robot path planning problem in a fully distributed manner, whereby dynamic obstacles are modeled as independent mobile robots. Each robot considers its own global guidance and local observations to generate actions. Since we do not require communication among robots, this is equivalent to an uncoordinated reactive approach. Note that the above extension is fully distributed, can be trained for a single agent (i.e., robot) and directly used by any number of other agents.

V. Implementation

In this section, we introduce the network parameters and describe our training and testing strategies.



(a) Regular map (b) Random map (c) Free map

Fig. 3. Map examples. The black and colored nodes represent the static and dynamic obstacles respectively. Map parameters can be found in Section V-B.

A. Model Parameters

In the experiments, we use the A* algorithm to generate the global guidance. The default parameters are set as follows: robot local FOV size $H_l = W_l = 15$, the length of input sequence $N_t = 4$, the reward parameters $r_1 = -0.01$, $r_2 = -0.1$, and $r_3 = 0.1$. The convolutional block is repeated $N_c = 3$ times with input batch size $N_b = 32$. The activation functions are `ReLU`. We use 32 convolution kernels in the first convolutional block and double the number of kernels after each block. After the CNN layers, the shape of feature maps is $4 \times 2 \times 2 \times 128$. In the LSTM layer and the two FC layers, we use 512, 512, and 5 units, respectively.

B. Environments

As shown in Fig. 3, we consider three different environment maps to validate our approach, i.e., a regular map, a random map, and a free map. The first one imitates warehouse environments and contains both static and dynamic obstacles, where the static obstacles are regularly arranged and the dynamic ones move within the aisles. In the random map, we randomly set up a certain density of static obstacles and dynamic obstacles. In the free map, we only consider a certain density of dynamic obstacles. The default size of all the maps is 100×100. The static obstacle density in each map is set to 0.392, 0.15, and 0, respectively, and the dynamic obstacle density is set to 0.03, 0.05, and 0.1, respectively. Dynamic obstacles are modeled as un-controllable robots that are able to move one cell at each step in any direction. Their start/goal cells are randomly generated, and their desired trajectories are calculated through A* by considering the current position of any other obstacles. During training and testing, each dynamic obstacle continuously moves along its trajectory, and when motion conflict occurs, it will: 1) with a probability of 0.9, stay in its current cell until the next cell is available; 2) otherwise, reverse its direction and move back to its start cell.

C. Training and Testing

We train our model with one NVIDIA GTX 1080ti GPU in Python 3.6 with TensorFlow 1.4 [22]. The learning rate is fixed to 3×10^{-5}, and the training optimizer is `RMSprop`. We use ε-greedy to balance exploration and exploitation. The initial value is set to be $\varepsilon = 1$ and decreases to 0.1 linearly when the total training steps reach 200,000. In training, we randomly choose one of the three maps as the training map and configure the dynamic obstacles by following the settings in Section V-B. Then we randomly select two free cells as the start and goal cells. During the training, we end the current episode and start a new one if one of the following conditions is satisfied: 1) the number of training steps in the current episode reaches a maximum defined as $N_m = 50 + 10 \times N_e$; 2) the robot can not obtain any global guidance information in its local FOV; 3) the robot reaches its goal cell c_{goal}. In addition, after the robot completes 50 episodes, the start and goal cells of all the dynamic obstacles are re-randomized.

Before learning starts, one robot explores the environment to fill up the replay buffer, which is comprised of a `Sumtree` structure to perform prioritized experience replay [23]. Note that `Sumtree` is a binary tree, which computes the sum of the values of its children as the value of a parent node. In each episode, the robot samples a batch of transitions from the replay buffer with prioritized experience replay (PER) based on the calculated temporal difference error by the DDQN algorithm. During testing, all methods are executed on an Intel i7-8750H CPU.

D. Performance Metrics

The following metrics are used for performance evaluation:

- *Moving Cost:*

$$\text{Moving Cost} = \frac{N_s}{||c_{goal} - c_{start}||_{L1}} \tag{4}$$

  where N_s is the number of steps taken and $||c_{goal} - c_{start}||_{L1}$ is the Manhattan distance between the start cell and the goal cell. This metric is used to indicate the ratio of actual moving steps w.r.t the ideal number of moving steps without considering any obstacles.

- *Detour Percentage:*

$$\text{Detour Percentage} = \frac{N_s - L_{A^*}(c_{start}, c_{goal})}{L_{A^*}(c_{start}, c_{goal})} \times 100\% \tag{5}$$

  where $L_{A^*}(c_{start}, c_{goal})$ is the length of the shortest path between the start cell and the goal cell, which is calculated with A* algorithm by only considering the static obstacles. This metric indicates the percentage of detour w.r.t the shortest path length.

- *Computing Time:* This measure corresponds to the average computing time at each step during the testing.

VI. Results

In this section, we present comparative results for both the single-robot and multi-robot path planning tasks. More details of our results can be found in the video https://youtu.be/KbAp38QYU9o.

A. Single-Robot Path Planning Results

We compare our approach with dynamic A* based methods with global re-planning and local re-planning strategies, and we call these two methods `Global Re-planning` and `Local Re-planning` respectively. For `Global Re-planning`,

TABLE I
SINGLE ROBOT PATH PLANNING RESULTS: MOVING COSTS AND DETOUR PERCENTAGES OF DIFFERENT APPROACHES

	Moving Cost				Detour Percentage				Computing Time (s)		
	Local	Global	Naive	G2RL	Local	Global	Naive	G2RL	Local	Global	G2RL
Regular-50	1.58(0.50)	1.31(0.29)	1.38(0.35)	**1.18(0.16)**	36.7(46)%	23.7(27)%	31.5(34)%	**15.2(15)**%	0.004	**0.003**	0.011
Regular-100	1.57(0.35)	1.23(0.21)	1.42(0.31)	**1.12(0.12)**	36.3(34)%	18.7(20)%	39.5(30)%	**10.7(12)**%	0.005	**0.004**	0.012
Regular-150	1.50(0.32)	1.19(0.14)	1.36(0.26)	**1.09(0.08)**	33.3(32)%	16.0(14)%	35.0(36)%	**8.2(8)**%	0.007	**0.004**	0.015
Random-50	1.35(0.28)	1.28(0.18)	1.36(0.28)	**1.21(0.13)**	25.1(26)%	21.1(17)%	30.1(35)%	**16.7(12)**%	0.005	**0.004**	0.013
Random-100	1.43(0.27)	1.26(0.14)	1.34(0.29)	**1.15(0.10)**	30.0(26)%	20.5(14)%	32.3(34)%	**13.0(10)**%	0.006	**0.006**	0.015
Random-150	1.37(0.17)	1.17(0.09)	1.40(0.28)	**1.11(0.08)**	27.0(17)%	14.5(9)%	39.4(40)%	**9.1(8)**%	0.010	**0.006**	0.018
Free-50	1.27(0.20)	1.24(0.15)	1.31(0.24)	**1.14(0.09)**	21.2(20)%	19.4(15)%	28.9(23)%	**12.3(9)**%	0.008	**0.007**	0.018
Free-100	1.31(0.13)	1.21(0.11)	1.34(0.25)	**1.11(0.07)**	23.6(13)%	17.3(11)%	33.6(25)%	**9.1(7)**%	0.015	**0.011**	0.022
Free-150	1.27(0.12)	1.14(0.06)	1.32(0.22)	**1.07(0.05)**	21.2(12)%	12.3(6)%	31.5(22)%	**6.5(5)**%	0.017	**0.013**	0.028

* Values are listed as "mean (standard deviation)" across 100 instances. The lowest (best) values are highlighted.

TABLE II
ABLATION STUDY ON DIFFERENT ROBOT FOV SIZES

$H_l \times W_l$	7×7	9×9	11×11	13×13	15×15
Random-50	1.49	1.35	1.32	1.24	1.21
Random-100	1.33	1.25	1.23	1.17	1.15
Random-150	1.27	1.21	1.18	1.14	1.13

* Values show the mean of the Moving Cost index.

TABLE III
ABLATION STUDY ON DIFFERENT INPUT SEQUENCE LENGTHS

N_t	1	2	3	4
Random-50	1.42	1.28	1.22	1.21
Random-100	1.48	1.34	1.21	1.15
Random-150	1.56	1.31	1.19	1.13

* Values show the mean of the Moving Cost index.

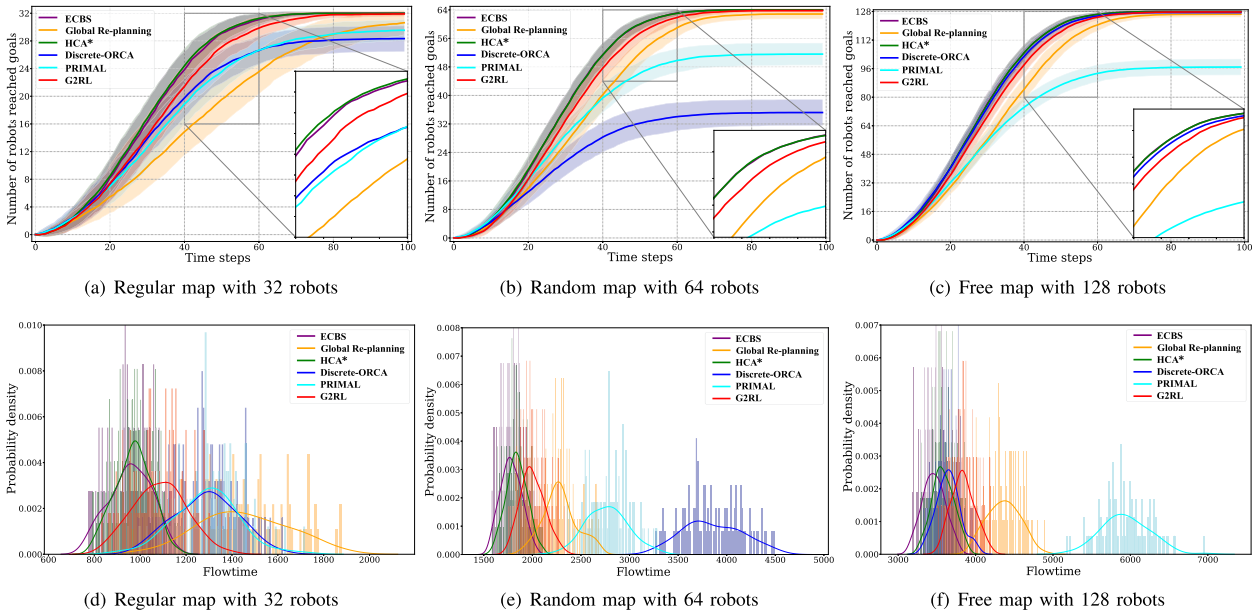
each time the robot encounters a conflict, an alternative path is searched for from the current cell to the goal cell by using the A* method, considering the current position of all the dynamic obstacles. For `Local Re-planning`, an alternative path is searched for from the current cell to the farthest cell within the robot's FOV. We also compare our reward function with a naive reward function, which strictly encourages the robot to follow the global guidance. Concretely, if the robots next location is on the global guidance, we give a constant positive reward $R(t) = 0.01$. Otherwise, we give a large negative reward $R(t) = r_1 + D_r \times r_4$, where $r_4 = -0.03$ and D_r is computed by calculating the distance between the robot s next location with the nearest global guidance cell. We set a time-out for all the tests, within which time if the robot can not reach its goal, this test is defined as a failure case. In each test, the time-out value is set as the double of the Manhattan distance between the start cell and the goal cell of the robot. We train our model and the naive reward based model by using both the three environments shown in Fig. 3, and then compare our testing results with `Global Re-planning` and `Local Re-planning` in the three environments separately. For each map, we separate the comparison into three groups with different Manhattan distances between the start cell and the goal cell, which are set to 50, 100, and 150, respectively. For each group, 100 pairs of start and goal locations are randomly selected and the mean value and its standard deviation are calculated. The desired trajectories of dynamic obstacles are consistent among the testing of each method for a fair comparison.

The comparison results in Table I validate that: 1) Compared with `Local Re-planning` and `Global Re-planning`, our approach uses the smallest number of moving steps in all the cases. 2) Our approach has the smallest standard deviations in all the cases, which demonstrates our consistency across different settings. 3) Since under the naive reward function, the robot is encouraged to follow the global guidance strictly, its performance is worse than when utilizing our reward function. The naive reward introduces more detour steps and waiting time steps in the presence of motion conflicts, and may even cause deadlocks (which further lead to failure cases). In contrast, our reward function incites convergence of the navigation tasks while simultaneously encouraging the robot to explore all the potential solutions to reach the goal cell with the minimum number of steps. Note that the `Moving Cost` and `Detour Percentage` are calculated by only considering the successful cases. The range of success rates for the naive reward based approach lies between $68\% - 89\%$, whereas for ours it is consistently 100%. 4) Our computing time is slightly higher than `Local Re-planning` and `Global Re-planning`, but still remains within an acceptable interval. Our maximum computing time is about 28 ms, which means that our RL planner achieves an update frequency of more than 35 Hz on the given CPU platform, fulfilling the real-time requirements of most application scenarios.

TABLE IV
EXPERIMENTAL RESULTS IN UNSEEN ENVIRONMENTS

	Local	Global	G2RL
Moving Cost	1.46(0.19)	1.21(0.08)	**1.12(0.06)**
Detour Percentage	42.8(19)%	20.5(8)%	**9.6(6)**%

We further test the effect of different robot FOV size and input sequence length N_t on the performance. The random map in Fig. 3 is used, and the start and goal cells are generated with fixed Manhattan distances of 50, 100, and 150. For each value, 100 pairs of start and goal cells are tested. We first choose different values of the FOV size of $H_l \times W_l$ for comparison. The results in Table II show that: 1) As the FOV size increases, the robot reaches the goal cell in less average steps. 2) The



(a) Regular map with 32 robots (b) Random map with 64 robots (c) Free map with 128 robots

(d) Regular map with 32 robots (e) Random map with 64 robots (f) Free map with 128 robots

Fig. 4. Multi-robot path planning results. The upper row shows the number of reached robots at different steps of different approaches in three testing maps. The solid lines show the average number across 100 tests, and the shadow areas represent the standard deviations. The lower row plots the corresponding histograms of flowtime values. The flowtime of all the failure cases is set to the maximum time step 100.

TABLE V
MULTI-ROBOT PATH PLANNING RESULTS: SUCCESS RATE AND COMPUTATIONAL TIME OF DIFFERENT APPROACHES

	Success Rate						Computing Time (ms)		
	ECBS	HCA*	Global Re-planning	Discrete-ORCA	PRIMAL	G2RL	Discrete-ORCA	PRIMAL	G2RL
Regular	**100%**	**100%**	95.7%	88.7%	92.3%	99.7%	**2.064(0.107)**	5.892(0.104)	6.367(0.337)
Random	**100%**	**100%**	98.2%	55.0%	80.6%	99.7%	**1.233(0.044)**	6.620(0.280)	6.206(0.260)
Free	**100%**	**100%**	98.8%	99.5%	75.7%	99.8%	**1.480(0.050)**	7.319(0.300)	6.246(0.277)

performance improvement is not significant when the FOV size is large than 13×13. Since a smaller FOV size implies a smaller learning model, a FOV size of 15×15 is large enough for our cases to balance the performance and computation cost. We then compare different values of the input sequence length N_t for comparison. For ease of implementation, we keep the same network input size in all the cases and use empty observation images for the cases with $N_t < 4$. Note that if $N_t = 1$, the robot loses all the temporal information of the dynamic obstacles and only considers the current observation. The results in Table III show that: 1) Introducing the historical local observation information improves the system performance in all the cases significantly. 2) When $N_t > 3$, increasing N_t only marginally improves the performance. Since a smaller N_t implies a smaller learning model, $N_t = 4$ is large enough for our cases to balance the performance and computation cost.

Next, we test our approach in an unseen environment to validate its generalizability. The environment is an enlarged version of the random map in Fig. 3 with a size 200×200. The densities of static and dynamic obstacles are set to 0.15 and 0.05, respectively. We randomly generate 100 pairs of start and goal cells with a constant Manhattan distance of 200. Note that we directly use the model trained in the small maps for testing. The results in Table I and Table IV show that our approach performs consistently well, under different environments, which validates the scalability and generalizability of our approach.

B. Comparison With Multi-Robot Path Planning Methods

We apply our method to the problem of multi-robot path planning, and compare it to five state-of-the-art benchmarks: *(i)* a decoupled dynamic A* based method, `Global Re-planning`, *(ii)* a decoupled path planning method, `HCA*`[24], *(iii)* a centralized optimal path planning method, Conflict-Based Search (`CBS`) [25], *(iv)* a velocity-based method `ORCA` [26] and, *(v)*, the RL based approach `PRIMAL` [6]. For `HCA*`, the priorities of the robots are randomly chosen. For `CBS`, since computing an optimal solution when the robot number is larger than 50 is often intractable, we use its sub-optimal version `ECBS` [25] with a sub-optimality bound set to 1.03. For `ORCA`, we calculate the next position of the robot by only considering the angle of the velocity output and thus transform the continuous-space `ORCA` into a discrete version `Discrete-ORCA`. It should be noted that neither our approach nor `PRIMAL` has been trained in the testing maps. For `PRIMAL`, we directly use the trained model provided online by the authors.[1] In our approach, we directly use the model trained in a single robot case.

[1]https://github.com/gsartoretti/distributedRL_MAPF

Comparisons are performed in three different maps with size 40×40 and varying numbers of robots, which are smaller versions of the maps in Fig. 3. The static obstacle densities are set to 0.45, 0.15, and 0 in the regular, random, and free maps, respectively, and the robot numbers are set to 32, 64, and 128. In each map, we generate 100 random configurations of robots with different start and goal cells. To ensure the solvability of the problem, once each robot arrives at its goal cell, we remove the robot from the environment to avoid conflicts. We set a time-out of 100 time-steps for all the tests, within which time, if any robot can not reach its goal, this test is defined as a failure case. In Figs 4 (a)–(c), we compare the number of robots that have reached their goals as a function of time. In Figs 4 (d)–(f), we compare the flowtime of the different approaches, which is defined as the sum of traversal time steps across all the robots involved in the instance. In Table V, we compare their success rates and computing times. Since in `Global Re-planning`, `ECBS` and `HCA*`, the robot action is not generated online at each step, we only compare the computing time of `Discrete-ORCA`, `PRIMAL` and our approach. The computing time is normalized by the average flowtime.

These results show that: 1) Our approach maintains consistent performance across different environments, outperforming `Global Re-planning` and `PRIMAL`, also outperforming `Discrete-ORCA` in most cases (`Discrete-ORCA` can not handle the crowded static obstacles, so it is only effective in the free map). `ECBS` and `HCA*` achieve the best performance overall, since they are both centralized approaches that have access to all robots' trajectory information. In `HCA*`, the trajectory information of all robots with higher priorities is shared and utilized in the planning of the robots with lower priorities. Our approach is in stark contrast to these approaches, since it is fully distributed and non-communicative (i.e., requiring no trajectory information of other robots). 2) Our success rate is similar to that of `ECBS` and `HCA*`, and higher than `Global Re-planning`, `Discrete-ORCA` and `PRIMAL`. The results show that our approach outperforms all the distributed methods, which validates its robustness, scalability, and generalizability. We note that in contrast to `PRIMAL`, which uses a general 'direction vector'[6], we utilize the global guidance instead. The global guidance provides dense global information which considers all the static obstacles. Thus, our method is able to overcome many of the scenarios (e.g., deadlocks) where `PRIMAL` gets stuck.

VII. Conclusion

In this letter, we introduced G2RL, a hierarchical path planning approach that enables end-to-end learning with a fixed-sized model in arbitrarily large environments. We provided an application of our approach to distributed uncoupled multi-robot path planning that is naturally scaled to an arbitrary number of robots. Experimental results validated the robustness, scalability, and generalizability of this path planning approach. Notably, we demonstrated that its application to multi-robot path planning outperforms existing distributed methods and that it performs similarly to state-of-the-art centralized approaches that assume global dynamic knowledge. In future work, we plan to extend our approach to cooperative multi-robot path planning.

References

[1] R. Smierzchalski and Z. Michalewicz, *Path Planning in Dynamic Environments*. Springer Berlin Heidelberg, 2005.

[2] L. Cohen, T. Uras, S. Jahangiri, A. Arunasalam, S. Koenig, and T. K. S. Kumar, "The fastmap algorithm for shortest path computations," in *Int. Joint Conf. Artif. Intell.*, 2018, pp. 1427–1433.

[3] A. Stentz, "Optimal and efficient path planning for partially known environments," in *Proc. IEEE Int. Conf. Robot. Automat.*, 1994, pp. 3310–3317.

[4] Z. Liu, H. Wang, H. Wei, M. Liu, and Y.-H. Liu, "Prediction, planning and coordination of thousand-warehousing-robot networks with motion and communication uncertainties," *IEEE Trans. Automat. Sci. Eng.*, 2020.

[5] M. Phillips and M. Likhachev, "Sipp: Safe interval path planning for dynamic environments," in *Proc. IEEE Int. Conf. Robot. Automa.*, 2011, pp. 5628–5635.

[6] G. Sartoretti *et al.*, "Primal: Pathfinding via reinforcement and imitation multi-agent learning," *IEEE Robot. Automat. Lett.*, vol. 4, no. 3, pp. 2378–2385, 2019.

[7] Q. Li, F. Gama, A. Ribeiro, and A. Prorok, "Graph neural networks for decentralized multi-robot path planning," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020.

[8] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[9] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*. MIT Press, 2018.

[10] X. Pan, W. Wang, X. Zhang, B. Li, J. Yi, and D. Song, "How you act tells a lot: Privacy-leaking attack on deep reinforcement learning," in *Proc. Int. Conf. Auton. Agents Multi-Agent Syst.*, 2019, pp. 368–376.

[11] C. Sun, M. Shen, and J. P. How, "Scaling up multiagent reinforcement learning for robotic systems: Learn an adaptive sparse communication graph," 2020, *arXiv:2003.01040v2*.

[12] P. Li, X. Huang, and M. Wang, "A novel hybrid method for mobile robot path planning in unknown dynamic environment based on hybrid dsm model grid map," *J. Exp. Theor. Artif. Intell.*, vol. 23, no. 1, pp. 5–22, 2011.

[13] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[14] B. Ichter, J. Harrison, and M. Pavone, "Learning sampling distributions for robot motion planning," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 7087–7094.

[15] L. C. Wang, L. S. Yong, and M. H. A. Jr, "Hybrid of global path planning and local navigation implemented on a mobile robot in indoor environment," in *Proc. IEEE Int. Symp. Intell. Control*, 2002, pp. 821–826.

[16] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," in *Proc. IEEE Int. Conf. Robot. Automat.*, 1999, pp. 341–346.

[17] D. Mehta, G. Ferrer, and E. Olson, "Autonomous navigation in dynamic social environments using multi-policy decision making," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 1190–1197.

[18] B. Rivière, W. Hoenig, Y. Yue, and S.-J. Chung, "GLAS: Global-to-local safe autonomy synthesis for multi-robot motion planning with end-to-end learning," *IEEE Robot. Automat. Lett.*, vol. 5, no. 3, pp. 4249–4256, 2020.

[19] D. Silver and *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, pp. 354–359, 2017.

[20] M. Bhardwaj, S. Choudhury, and S. Scherer, "Learning heuristic search via imitation," in *Conf. Robot Learn.*, 2017, pp. 271–280.

[21] H. V. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proc. AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.

[22] M. Abadi *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, *arXiv:1603.04467v2*.

[23] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *Proc. 4th Int. Conf. Learn. Representations*, 2016. [Online]. Available: http://arxiv.org/abs/1511.05952

[24] D. Silver, "Cooperative path-finding," in *Proc. AAAI Conf. Artif. Intell. Interactive Digit. Entertainment*, 2005, pp. 117–122.

[25] M. Barer, G. Sharon, R. Stern, and A. Felner, "Sub-optimal variants of the conflict-based search algorithm for the multi-agent path-finding problem," in *Proc. Eur. Conf. Artif. Intell.*, 2014, pp. 961–962.

[26] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal n-body collision avoidance," in *Proc. Robot. Res.*, 2011, pp. 3–19.