DEEP REINFORCEMENT LEARNING FOR ROBOT PATH PLANNING WITH PERIODICALLY MOVING OBSTACLES: CURRENT METHODS AND CHALLENGES

[1]Ziyad Ahmed Mohammed, [2]Siva Rama Krishnan V K, [3]Vayuvegula Naga Venkatesh Sripad, [4]Sakkari Vishal Goud, [5]K Janakiram Sharma, [6]Kesireddy Rohit Reddy, [7]Kesireddy Rajashekar Reddy

[1]Department – EEE, Chaitanya Bharathi Institute of Technology (CBIT), Hyderabad

[2,3,4,5,6]Department – CSE, Chaitanya Bharathi Institute of Technology (CBIT), Hyderabad

Email - ziyadahmed910@gmail.com[1], sram5330335@gmail.com[2], sripadv2000@gmail.com[3], sakkarivishal@gmail.com[4], janakiramsharma18@gmail.com[5], rohitt.officialwork@gmail.com[6], kesireddyshekar@gmail.com[7]

[7]Department – ECE, SR Engineering College,

ABSTRACT

Due to the complexity of huge, dynamic settings, mobile robots' path planning is a difficult issue to solve. This is because the robots must effectively attain their objectives while avoiding possible collisions with other robots or dynamic objects. Re-planning techniques, which recall a planning algorithm to look for an alternate route whenever the robot faces a conflict, are often used in situations with moving barriers in the way of conventional solutions. Nonetheless, route re-routing tactics often result in extra side trips. We present a learning-based method that uses spatial and temporal data from the surrounding environment to solve this problem. In this research, we survey the landscape of DRL techniques and navigational frameworks built on DRL. "Finally, we conduct a comprehensive comparison and analysis of the similarities and differences across four common use cases: local obstacle avoidance, indoor navigation, multi-robot navigation, and social navigation." The evolution of DRL-based navigation is discussed next. At last, we talk about the problems with DRL-based navigation and some potential answers to those problems.

Keywords: *"Mobile Robot Navigation; Obstacle Avoidance; Deep Reinforcement Learning"*

DOI Number: 10.14704/nq.2022.20.13.NQ88258 Neuro Quantology 2022; 20(13):2063-2074

I.INTRODUCTION

One of the first challenges faced by roboticists is the challenge of PATH planning. Specifically, the task is to figure out how to get a robot from point A to point B without running into any obstacles and describe that environment. Planning may be done offline (on the assumption that barriers are static and the environment is completely known) or in real-time (with an emphasis on coping with dynamic impediments and a less-than-perfectly-understood environment) [1]. Since conventional off-line planning algorithms [2] presume that impediments are immovable, they are inapplicable nonhandling online route planning challenges. One tactic is to map out a potential course of action and then request a recalculation if the feasibility of carrying it out is compromised [3]. "Due to the lack of motion information of dynamic impediments, however, re-planning will be inefficient in both time and route. In the event of robot deadlocks, re-planning solutions may also fail. Some approaches use an additional temporal dimension to prevent future conflicts rather than re-planning [4]." However, this method requires knowledge of the future trajectories of dynamic impediments and increases the number of states to be explored. It is possible to try to describe the behavior of dynamic barriers and anticipate their courses [5] if the future motions of obstacles are

uncertain. However, the "frozen robot" issue arises when the navigation task is broken down into independent prediction and planning processes. In such an instance, the robot will be unable to locate a workable course of action since the anticipated pathways may label a large chunk of the environment as inaccessible. Online planning in uncertain situations has been the research focus [6, 7]. The foundational study [8] popularized this approach by using deep neural networks to estimate function in value-based reinforcement learning (RL). Despite RL's impressive track record in many domains, various obstacles still prevent it from making a meaningful contribution to the route planning issue.

To begin with, the learning process becomes inefficient, and training effort increases when the environment is very big since the reward is less abundant[9]. Overfitting is a further difficulty. When exposed to novel conditions, the robot performs poorly compared to its performance in training [10]. The sizes of the robot state, joint action, and joint observation spaces expand exponentially with the number of robots [11], making it challenging for even the most current techniques to scale to arbitrarily large multi-robot systems [7]. “As a result, many applications' needs cannot be met by the efficiency, generalizability, and scalability of current RL-based planners.”

The automobile industry, equipment manufacture, food industry, health care, and agriculture are just some of the many automated settings that use robotic arms with multiple degrees of freedom (Multi-DOF). Multi-degree-of-freedom (DOF) robotic arms have traditionally been used in controlled settings, while humans have often handled activities that need them to adjust to changing circumstances. People and robots work together with HRC to increase manufacturing's adaptability and productivity [12]. However, defining a safe, collision-free HRC system is challenging for typical motion planning algorithms since their parameters are defined based on a given environment. “Some path planning algorithms, like the Probability Road Map (PRM) and the Rapidly-exploring Random Tree (RRT), are unsuitable for dynamic environments because they require higher real-time performance of algorithms to deal with dynamic obstacles. Specifically, these algorithms need to construct a real-time mapping of obstacles in the configuration space to plan a collision-free path, which is a very computationally expensive task.” Potential field (PF) is another popular method that is commonly used instead of PRM and RRT due to its reduced computational requirements and improved real-time control. However, PF often becomes trapped in the local minimum and performs poorly when obstacles are close to the objective. Therefore, it is important to develop an effective, secure, and adaptable motion planning algorithm. As an alternate approach, reinforcement learning (RL) can outperform traditional methods in many high-dimensional tasks and games. Some even argue that this is sufficient for DeepMind to achieve general artificial intelligence [13].

Deep RL, which uses neural networks to approximate functions, has recently been successful in various challenging robotic control tasks. Without training on a big dataset, multiple RGB pictures with double-deep Q-learning may achieve a success rate of over 80% in diverse grasping tasks, as shown by Joshi et al. [14]. “With just a few hours of training time, the robot in Gu et al [15] learns to open doors using DDPG and the Normalized Advantage Function algorithm (NAF).” Similarly, Haarnoja et al. [16] demonstrate that a robot can learn to stack Lego bricks together using soft Q-learning after just 2 hours of policy training. As a result, we put into action a notion using a DRL-based approach to deal with sophisticated trajectory planning in ambiguous settings. However, deep RL is not without its drawbacks: (1) it is not simple to define an acceptable reward function; when dealing with high-dimensional issues, it is common to wind up with the outcome we incentivize rather than the one we planned. (2) An RL agent can often find an optimum policy in a short time for basic tasks, but it may take several million training steps to get the required outcome for complicated jobs. Thirdly, training an RL agent in various contexts is computationally costly, which is necessary to avoid overfitting.

II.REVIEW OF LITERATURE

Pathfinding and obstacle avoidance are problems with current potential answers. The RRT has

seen extensive use in the collision-free trajectory planning of Multi-DOF manipulators due to its probability completeness and exploration efficiency. Adiyatov and Varol [17] When a manipulator encounters an incorrect route due to a dynamic obstruction, the RRT Fixed Nodes Dynamic (RRT*FND) algorithm's Reconnect and Regrow operations may help it fix the path in an average of 300 milliseconds. To construct a smoother route and more steady mobility while avoiding obstacles, Wei and Ren [18] devised an enhanced RRT algorithm they named Smoothly RRT (S-RRT), which has shown greater exploration speed and efficiency than Basic-RRT and Bi-RRT. Kurosu et al. [19] treat one of the robot arms as a moving obstacle in a pick-and-place scenario with two robots, and they use the RRT algorithm to prevent collisions between the two arms successfully. Although RRT-based algorithms have shown their strength in avoiding both dynamic and static obstacles, more study is still needed to develop techniques for avoiding obstacles that are continually and randomly on the move.

The artificial potential field is another method often used for collision-free trajectory planning (APF). This strategy uses the PF force of attraction and repulsion to successfully complete the mission. “To prevent a static/dynamic barrier from colliding with a collaborative robot arm, Xu et al [20] use a method very similar to APF, dubbed potential velocity field (VPF).” They employ the robot's velocity instead of the distance in AFP to avoid the local minima difficulties that arise when attractive and repulsive forces/velocities are juxtaposed along the same line. To predict the velocity of an obstacle and so prevent a collision, Flacco et al. [21] use a simplified form of APF, a repulsive vector, based on the distance between the two. Using the repulsive vector and the distance determined from the point cloud, Melchiorre et al. [22] have shown their ability to avoid collisions with static and moving objects. However, its performance degrades when the PF is confronted with two impediments that are too close together. If the desired location is sandwiched between or hidden by two nearby barriers, the robot will lose interest and go on.

The PRM alternative uses a random sample of the robot's configuration space to find a path that avoids collisions between the starting and ending nodes. Liu et al. [23] proposed a Grid-Local PRM that uses a mapping model, sampling approaches, slow collision detection, and a unified local detection strategy. One practical application of this technology is dynamic route planning for static/dynamic obstacle avoidance. Planners like the Probabilistic Roadmap of Obstacles [24], created by Wittmann et al. (ORSR-PRM). Real-time trajectory planning is handled by PRM, while PF handles obstacle avoidance. Like the PF method, the chance of building nodes between slim corridors prevents a route from being planned via the gap.

III.METHOD

(1) Model Parameters

For experimental purposes, we produce worldwide instructions with the help of the A* algorithm. Robot local FOV size Hl = Wl = 15, input sequence length Nt = 4, and reward parameters r1 = 0.01, r2 = 0.1, and r3 = 0.1 are the default settings. The convolutional block is repeated N times with Nc = 3 iterations and Nb = 32 input samples. These activation procedures use ReLU. In the first convolutional block, we utilize 32 convolution kernels, and in subsequent blocks, we increase the number of kernels by a factor of 2. The feature maps' structure is 4 2 2 128 after the CNN layers are applied. We use 512 units in the LSTM layer and 512 units in each of the two FC levels.

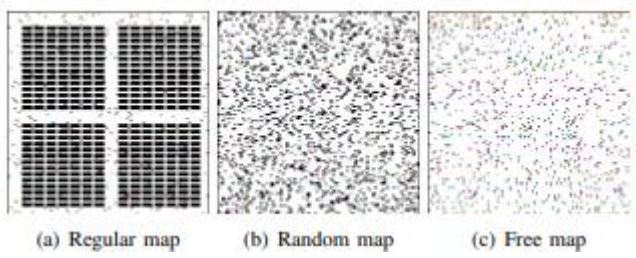


Fig. 1. Examples of maps. Dynamic barriers are represented by colored nodes, whereas black ones represent static ones. Parameters for the map are described in detail in Section V-B.

(2) Environments

In order to verify our method, we take into account a regular map, a random map, and a free map of the environment, as illustrated in Figure 1. The first simulates a warehouse setting by including static and moving obstacles, the former organized in predictable patterns down the

aisles. We randomly place a specified number of static and moving obstacles on the random map. There is a limit on the number of moving obstructions that may be considered in the unrestricted map. All of the default map sizes are 100 by 100. Each map's static obstacle density is set at 0.392, 0.15, and 0; their dynamic obstacle densities are 0.03, 0.05, and 0.1. In this analogy, moving impediments are represented by autonomous robots that can move one cell in any direction at each step. Their starting and ending cells are produced at random, and their intended paths are determined using A* while considering the locations of any additional obstacles in the way. "Each dynamic obstacle advances constantly along its trajectory throughout training and testing, and if a motion conflict arises, it will either: 1) remain in its current cell until the next cell is accessible (with a 90% chance of doing so), or 2) reverse its path and return to its start cell."

(3) Training and Testing

Using Python 3.6 and TensorFlow 1.4 [25], the model is trained on a single NVIDIA GTX 1080ti GPU. The learning rate is 3105, and the optimizer used for training is RMSprop. We use the value of -greed to find a happy medium between exploration and exploitation. After starting with a value of 1, the parameter is steadily lowered throughout 200,000 training steps, reaching a final value of 0.1. During training, we choose a map at random, configure the dynamic obstacles using the settings described in Section V-B, and finally select a cell at random as both the training ground and the destination. "The current training episode is over when one of the following three conditions is met: 1) the number of training steps in the episode has reached the maximum defined as Nm = 50 + 10 Ne; 2) the robot is unable to obtain any global guidance information in its local FOV; or 3) the robot has reached its goal cell cgoal. When the robot reaches 50 episodes, the dynamic barriers' first and last cells are likewise switched places." One robot does preliminary exploration to gather information for the learning environment's replay buffer, a Sumtree structure used to replay experiences in a prioritized order [26]. Gumtree is a binary tree where the value of a parent node is calculated by adding the values of all its children. The DDQN method computes the temporal difference error for each segment. Then the robot uses prioritized experience replay to randomly choose a sequence of transitions from the replay buffer (PER). All procedures are performed on an Intel i7-8750H CPU during testing.

IV.RESULTS

(1) Single-Robot Path Planning Results

Two methodologies, which we refer to as Global Re-planning and Local Re-planning, are used to evaluate our method against dynamic A*-based approaches. When the robot reaches a dead end, the A* algorithm is used to find an alternate route from the current cell to the objective cell while considering the current location of all the dynamic obstacles. When doing Local Re-planning, the robot's FOV is scanned to find the furthest cell from the present one, and then an alternate route is sought. Our reward function is compared to a naïve reward function that just rewards the robot for adhering to global guidance. More specifically, we provide a continuous positive reward R(t) = 0.01 if the robot's next position is on the global guidance. "If the robot moves too far away from the closest global guidance cell, it will get a substantial negative reward of R(t) = r1+Dr r4, where Dr is the distance between the robot's next position and the nearest cell and r4 is 0.03." For each test, we implement a time limit; if the robot cannot complete the task within that time, the test is considered unsuccessful. Every trial has a time limit equal to two Manhattan distances between the robot's initial and final cells.

Each map is compared across three groups, with the Manhattan distance between the starting and finishing cells set at 50, 100, and 150. The mean and standard deviation are computed for each group based on data from a sample of 100 randomly chosen sets of origin and destination coordinates. Testing each approach is conducted under identical conditions to provide a level playing field, using the same intended trajectories of dynamic obstacles.

Table I's comparative findings show that: 1) Our method requires the fewest possible iterations to get the desired outcome compared to Local Re-planning and Global Re-planning. 2) Our method is the most consistent across varied environments, as shown by the fact that it has

the fewest standard deviations in all the examples. 3) The robot's performance is lower using the naïve reward function since it is motivated to adhere rigidly to the global direction. When motion conflicts are present, the naïve reward may lead to deadlocks due to increased detour steps and waiting for time steps (which further lead to failure cases). In contrast, our reward mechanism encourages the robot to explore all possible options to reach the target cell with the fewest steps, promoting convergence of the navigation tasks. Keep in mind that while figuring up the Moving Cost and Detour Percentage, we only consider the successful instances. "Compared to our consistently high success rate of 100%, the naïve incentive based strategy has a range of 68% to 89%. 4) Although our computation time is longer than that of Local Re-planning and Global Re-planning, it is still within a tolerable range. Our RL planner achieves an update frequency of more than 35Hz on the provided CPU platform, meeting the real-time needs of most application situations, with a maximum compute time of roughly 28ms."

Table i: Calculated expenses of course changes and percentages of deviations for a single robot

	Moving Cost				
	Local	Global	Naive	G2RL	Local
Regular-50	1.58(0.50)	1.31(0.29)	1.38(0.35)	**1.18(0.16)**	36.7(46)%
Regular-100	1.57(0.35)	1.23(0.21)	1.42(0.31)	**1.12(0.12)**	36.3(34)%
Regular-150	1.50(0.32)	1.19(0.14)	1.36(0.26)	**1.09(0.08)**	33.3(32)%
Random-50	1.35(0.28)	1.28(0.18)	1.36(0.28)	**1.21(0.13)**	25.1(26)%
Random-100	1.43(0.27)	1.26(0.14)	1.34(0.29)	**1.15(0.10)**	30.0(26)%
Random-150	1.37(0.17)	1.17(0.09)	1.40(0.28)	**1.11(0.08)**	27.0(17)%
Free-50	1.27(0.20)	1.24(0.15)	1.31(0.24)	**1.14(0.09)**	21.2(20)%
Free-100	1.31(0.13)	1.21(0.11)	1.34(0.25)	**1.11(0.07)**	23.6(13)%
Free-150	1.27(0.12)	1.14(0.06)	1.32(0.22)	**1.07(0.05)**	21.2(12)%

Values are listed as "mean (standard deviation)" across 100

We also investigate how changing the field of view (FOV) of the robot and the duration of the input sequence, Nt, affect performance. The starting and ending cells are produced randomly on the random map shown in Figure 3, with Manhattan distances of 50, 100, and 150. One hundred pairs of initial and final values are tried for each value. First, we pick out certain FOV values (Hl Wl) to examine. Table II shows that: 1) The average number of steps required for the robot to reach the target cell decreases as the field of view (FOV) rises. When the field of view (FOV) is more than 13 by 13 pixels, there is no discernible speed boost. Our experiments show that a 1515 FOV size is optimal for achieving a good trade-off between performance and computational cost since a lower FOV size indicates a smaller learning model. We then evaluate several alternatives for the input sequence length Nt. As a matter of convenience, we employ empty observation pictures for Nt = 4 and maintain a constant network input size for all other scenarios. Remember that the robot will only consider the current observation if Nt = 1. Table III demonstrates that 1) including local observational history considerably boosts system performance across the board. 2) The performance gains from raising Nt when Nt is more than three are small. Since a lower Nt results in a less complex learning model, we find that Nt = 4 strikes a good compromise between performance and computational cost for our use cases.

Table ii: Examination of Ablation on Varying Field-of-View Robots

$H_l \times W_l$	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: Research on Ablation for Various Input Sequence Lengths

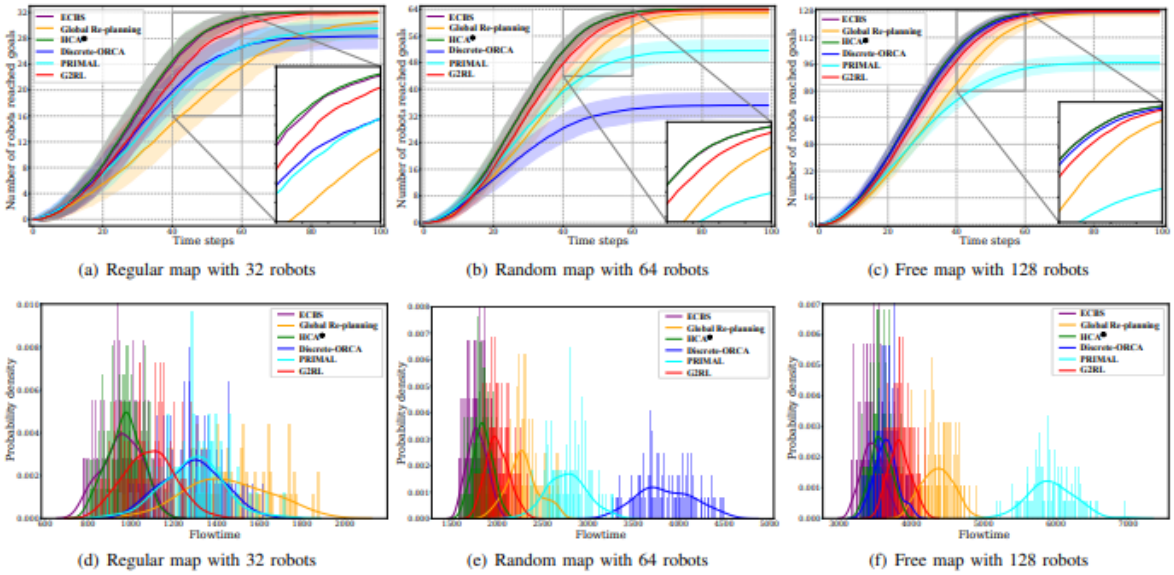
N_t	1	2	3	4
Random-50	1.42	1.28	1.22	1.21
Random-100	1.48	1.34	1.21	1.15
Random-150	1.56	1.31	1.19	1.13

Values show the mean of the Moving Cost index.

(2) Comparison with Multi-Robot Path Planning Methods

We apply our method to the issue 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 [25]; (iii) a centralized optimal path planning method, Conflict Based Search (CBS) [27];" and (iv) a velocity-based method, ORCA [28]. When the number of robots to be controlled is too high for CBS's optimal solution, we resort to its subpar variation, ECBS [27], using a sub-optimality constraint of 1.03. By knowing the velocity output angle, we can transform the continuous-

space ORCA into the discrete version Discrete-ORCA and use it to predict where the robot will go next. However, it must be stressed that neither our approach nor PRIMAL has been trained on the test maps. There is just one trained model for PRIMAL available online. Therefore we use that one. Our method relies only on the model developed for a specific robot scenario. Three smaller copies of the maps in Figure 1 (40x40) are used for comparisons with differing numbers of robots. For the regular, random, and free maps, the static obstacle densities are 0.45, 0.15, and 0, correspondingly, while the robot numbers are 32, 64, and 128. Each map has one hundred possible robot setups, each with a unique set of origin and destination cells. Once a robot reaches its destination cell, it is removed from the environment to prevent collisions and guarantee that the issue can be solved. If a robot cannot complete a test within 100 steps, it is said to have failed. The proportion of robots that have completed their objectives across different time intervals is shown in Figures 2 (a)-(c). "Figures 2(d)–(f) compare the flowtime of the various methods, where flowtime is the total number of traversal steps taken by all of the robots in the instance". You can see how successful they were and how long it took them to compute in Table V. "Since the robot action is not created online at each step in Global Re-planning, ECBS, and HCA, we only compare the computation time of Discrete-ORCA, PRIMAL, and our method. Computing times are scaled according to mean flow rates [29]."



(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. 2. Planned paths for teams of robots. The first row displays the total number of robots reached across all three test maps, while the second row details the percentage of robots that reached each approach level. The dashed lines indicate the median value for 100 measurements, while the shaded regions reflect the standard deviations. Histograms of flowtime values are shown in the bottom row. All possible failure scenarios have had their flowtime increased to a maximum of 100-time steps.

These results show that 1) in most benchmarks, our technique consistently beats Global Re-planning, PRIMAL, and Discrete-ORCA (Discrete-ORCA can not handle the crowded static obstacles, so it is only effective in the free map). Centralized approaches like ECBS and HCA, which provide simultaneous access to the trajectories of all robots, perform very well. HCA works by combining the paths taken by robots with different priorities and using that information to direct the actions of those with lower priorities. Compared to these other approaches, ours stands out as a stark contrast due to the fact that it is decentralized and neutral with regard to any external communication (i.e., requiring no trajectory information of other robots). We have a success rate that is 2) higher than Global Re-planning, Discrete-ORCA, and PRIMAL, and on par with ECBS and HCA. All other distributed methods are shown to be inferior to ours in performance, proving the superiority of our technique in terms of its resilience, scalability, and generalizability. We point out that we employ global guidance rather

than PRIMAL's generic 'direction vector' [6]. As for the static impediments, they are all taken into account by the global guiding system, which gives comprehensive worldwide data. This means that our approach can free PRIMAL from many situations in which it normally would be trapped (such as deadlocks).

V.CURRENT CHALLENGE AND SOLUTION

A. Partial observation

a) Challenge

Because the state space is assumed to be completely observable, RL is frequently used for motion control problems in robotics, such as manipulator control. The mobile robot navigation job is only partly visible due to the restricted field of view and the unknown status of other subjects. "This leads to two issues: (a) the agent can only learn suboptimal methods, and (b) the agent cannot identify its state based on individual current observations."

Because it presumes that the state is fully observable, the MDP cannot capture the intricate details of a given ecosystem. Partially Observable Markov Decision Process (POMDP) is an extension of MDP that allows for additional information gleaned from past trajectories. It is widely used as a modeling tool in the field of research. A POMDP has six tuples and is defined by its structure. (S; A; R; P; Ω; O), where(O t ∈ Ω) is the observation space and (O) is the probability distribution of observations given the current state of the system ($O_t \sim O(S_t)$). The agent performs an observation at each time step t. $_{It}$ carries out a procedure by the policy $\pi(a_t \mid O_t)$, obtains a prize r_t, and relocation to a different country s_{t+1} by the breakdown $P(s_{t+1} \mid s_t, a_t)$.

b) Solution

There are primarily two ways to address the partial observation dilemma.

(1) Increasing the amount of data coming into the network

When an agent's current observations do not allow it to discriminate between states uniquely, the easiest answer is to feed in many frames' worth of historical observations into the network to help it out [30]. Some research has also inputted past rewards and behaviors into the network [31] since they also include state information. "The two-stream Q network, introduced by Wang et al [32], takes the difference between two frames of laser scanning data and uses it as an additional input."

(2) Memory capacity enhancement

Having the agent automatically remember and analyze the prior observation is preferable to manually choosing the previous observation frame as input [33]. In deep learning studies, the Recurrent Neural Network (RNN) is a standard network structure due to its ability to handle sequence data as input and memory features. The RNN's current output is affected by the data that came before it. Researchers have suggested using a recurrent neural network (RNN) inside the DRL architecture to answer the POMDP. To complete DRL-based navigation tasks, Wang et al.[26, 27] used this method with great success. However, RNNs have difficulty with long-term dependencies. Therefore deep-learning researchers suggested the LSTM network as a solution. Several LSTM layers have been added to the network topology in several DRL-based navigation experiments to address POMDP. Gated recurrent units (GRUs) are a simplified version of LSTM that are simpler to train. Using a GRU-based memory neural network, Zeng et al.[34] demonstrated improved performance in challenging navigation tests.

B. Sparse reward

a) Challenge

The navigation challenge for mobile robots includes point-to-point communication and avoiding obstacles. When the agent successfully reaches the goal location, it is rewarded positively, whereas when it collides with an obstacle, it is punished negatively. Since the award is only created once every epoch, it is extremely infrequent. Agents must choose a sequence of corrective actions in order to meet the aim of creating a sparse reward since it is well-known that efficient RL is dependent on gathering valuable reward signals. Finding a sparse reward signal via a random search is quite unlikely. "There is a dramatic increase in the size of the possible set of samples as the complexity and volatility of the external environment increases." Data inefficiency may be exacerbated by sparse rewards, leading to slow convergence during training and extended total training durations.

b) Solution

DRL-based navigation uses the tried-and-true experience replay mechanism, investigation, and utilization approaches from the field of RL study. Furthermore, three methods exist for dealing with the sparse reward issue.

(1) Reward shaping

Intuitively, one way to simplify the problem's learning curve is to construct elaborate incentives manually. Expert knowledge allows the agent to be rewarded at each stage of action execution and environment interaction. Most methods use reward shaping approaches, particularly those that can be empirically tested in real-world settings. When getting closer to a goal, you will get a little positive reward [35] depending on the target distance, but getting closer to a barrier will result in a small negative reward [36]. To motivate the robot to get there quicker, a little penalty is added at each time step. Agents may be incentivized to progress toward a goal by offering incentives tied to the direction they are headed in. Sampedro et al. [36] relied on an APF formulation when developing their incentive structure. Different researchers have come up with different reward functions to help with learning. "However, there are two drawbacks to manually constructed reward functions: (1) they are intimately tied to the task scene, and over-fitting a specific scene leads to its non-universality, and (2) an improper reward occasionally leads to the incorrect direction for learning". The author technique, introduced by Chiang et al. [37] to address these issues, uses automation to search for both the shaped reward and the neural network architecture. Using a matching network, Zhang et al.[38] developed a generic reward function. In addition, the sparse reward issue may be solved using DRL by utilizing the supplementary experience that expert demonstration can give.

(2) Auxiliary task

We may speed up learning by assigning supplementary tasks in the event of a scarce reward when completing the original job is very challenging. Additional dense pseudo-rewards may be provided for RL via auxiliary activities. Since auxiliary jobs are monitored, the auxiliary loss associated with them may be utilized to fine-tune the network. Some of the network topologies are common to both the primary and secondary jobs, which aids in developing the model's representation. In studies of indoor visual navigation, the auxiliary task approach is often used. The Nav A3C framework was expanded by Mirowski et al. [39] to include two more tasks: depth and loop-closure predictions. They used the auxiliary heading prediction task [40] to estimate the deviation from true north from the current heading in an urban navigation scenario. "To hasten convergence, Hsu et al. [41] trained their local region model with an additional task." By adding pixel control, reward prediction, depth-map prediction, and picture segmentation tasks to the batched A2C algorithm, Kulhanek et al. [42] advanced the auxiliary task approach for visual navigation.

(3) Curriculum learning

The term "curriculum learning" describes the process of creating educational programs that cater to students of varying skill levels. Training a neural network may be sped up by substituting a simpler job with a greater chance of reward for the original task. The level of difficulty increases during the duration of the course. DRL-based navigation tasks have made use of this method. The two-stage training technique for curriculum learning was established by Chen et al. [43]. They started by training the policy with eight robots in a random environment, then expanded to 16 robots and trained it in both random and circular environments. The DRL training efficiency may be improved under the sparse reward situation by using multi-stage learning with policy evolution, similar to curriculum learning.

C. Poor generalization

a) Challenge

DRL-based navigation also has issues due to poor generalization. Researchers often first hone the agent's skills in a simulation environment before introducing it to the genuine environment because of the high costs and risks associated with training in the real world. As a rule, there are two categories of generalizations:

- The agent is moved from one virtual world to another. o DRL is an algorithm that teaches a system to respond to its surroundings and take the course of action that will provide the highest cumulative return. Transferring a navigation model trained from one environment to another

is challenging because of the vast differences in data distributions across settings.

- This agent is taken from a simulated world and placed in the actual world. The actual world, in most cases, is more intricate and ever-changing. One of the biggest obstacles to putting a trained model into an actual robot is the disconnect between the two worlds.

b) Solution

To solve the issue of generalization, we may either increase the size of the sample space or decrease the number of states.

(1) Expansion of the sample space

One popular approach to overcoming the generalization issue is to include some degree of randomness in the simulation environment to increase the sample space size. Distributing data from a wider range of situations and making the transition between simulation settings easier, random sensor noise, a random target location, and random impediments are all ways to do this. Lei et al.[44] suggest expanding the state-space distribution of the experience pool by arbitrarily setting the goal location in an empty region to motivate the agent to adjust to the new circumstances. Zhang et al.[45] trained the agent in four distinct labyrinth settings, each with a random starting point. "Using a large number of robots in complicated contexts, Long et al [46] introduced a multi-scenario training framework to discover an ideal strategy." In their study, Zhu et al. [47] demonstrated that a model trained on 16 scenes could be used on additional scenes without further training. After some minor adjustments, the model is ready to be moved from the virtual to the physical world.

(2) Reduction of the state space

It takes a lot of work and subsequent fine-tuning in real-world applications to expand the sample space. It is possible to obtain the same level of generalization with less processing cost by reducing the state-space dimensions. Although the picture may be used as a component of the state space, which has the benefit of providing a wealth of information, there are frequently aspects, such as lighting, texture, and color shifts, that are not useful or even damaging to the learning navigation abilities. If the DRL model is fed sparse laser range data, it may concentrate on how those distances among the learning obstacles transfer to the motion commands. Range finders have been the sensor of choice in almost all DRL-based navigation studies with practical applicability. Tai et al. [48] relied only on the results of 10-dimensional sparse laser range measurements and data about the targets when constructing their state space. Their concept is readily transferable to a practical mobile robot platform with little to no adjustment. It was also found by Shi et al. [49] that sparse laser range results might help narrow the credibility gap. Instead of simply entering the picture, the approach described by Yokoyama and Morioka [50] transforms the depth images calculated from a monocular camera into 2D range data.

VI.CONCLUSION

The mobile robot relies heavily on its navigational abilities. Although DRL policy is limited in its capacity to provide ideal behavior consistently and DRL-based navigation performance is inconsistent, DRL remains the most promising approach to achieve significant advances in navigability. In this study, we proposed G2RL, a hierarchical route planning strategy that allows for end-to-end learning with a fixed-sized model in arbitrarily vast settings. We demonstrated how our method might be used for scalable, distributed, uncoupled route planning amongst a group of robots of any size. Experimental data verified the robustness, scalability, and generalizability of this route-planning method. "To highlight, we showed that its implementation in multi-robot route planning outperforms current distributed techniques and is on par with state-of-the-art centralized systems that assume global dynamic information." We want to apply our method to collaborative multi-robot route planning with further development.

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 International Joint Conference on Artificial Intelligence, 2018, pp. 1427– 1433.

[3] A. Stentz, "Optimal and efficient path planning for partially known environments," in IEEE International Conference on Robotics and Automation, 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 Transactions on Automation Science and Engineering, 2020.

[5] M. Phillips and M. Likhachev, "Sipp: Safe interval path planning for dynamic environments," in IEEE International Conference on Robotics and Automation, 2011, pp. 5628–5635

[6] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. S. Kumar, S. Koenig, and H. Choset, "Primal: Pathfinding via reinforcement and imitation multiagent learning," IEEE Robotics and Automation Letters, 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 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2020.

[8] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, 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 International Conference on Autonomous Agents and Multi-Agent Systems, 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," arXiv preprint arXiv:2003.01040v2, 2020.

[12] Vysocky, A., and Novak, P. (2016). Human-robot collaboration in industry. *MM Sci. J.* 9, 903–906. doi: 10.17973/MMSJ.2016_06_201611

[13] Shanahan, M., Crosby, M., Beyret, B., and Cheke, L. (2020). Artificial intelligence and the common sense of animals. *Trends Cogn. Sci*. 24, 862–872. doi: 10.1016/j.tics.2020.09.002

[14] Joshi, S., Kumra, S., and Sahin, F. (2020). "Robotic grasping using deep reinforcement learning," in *2020 IEEE 16th International Conference on Automation Science and Engineering (CASE)* (Hong Kong), 1461–1466. doi: 10.1109/CASE48305.2020.9216986

[15] Gu, S., Holly, E., Lillicrap, T., and Levine, S. (2017). "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *2017 IEEE International Conference on Robotics and Automation (ICRA)* (Singapore), 3389–3396. doi: 10.1109/ICRA.2017.7989385

[16] Haarnoja, T., Pong, V., Zhou, A., Dalal, M., Abbeel, P., and and15 Levine, S. (2018a). "Composable deep reinforcement learning for robotic manipulation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 6244–6251. doi: 10.1109/ICRA.2018.8460756

[17] Adiyatov, O., and Varol, H. A. (2017). "A novel RRT*-based algorithm for motion planning in dynamic environments," in *2017 IEEE International Conference on Mechatronics and Automation (ICMA)* (Takamatsu), 1416–1421. doi: 10.1109/ICMA.2017.8016024

[18] Wei, K., and Ren, B. (2018). A method on dynamic path planning for robotic manipulator autonomous obstacle avoidance based on an improved RRT algorithm. *Sensors* 18:571. doi: 10.3390/s18020571

[19] Kurosu, J., Yorozu, A., and Takahashi, M. (2017). Simultaneous dual-arm motion planning for minimizing operation time. *Appl. Sci*. 7:1210. doi: 10.3390/app7121210

[20] Xu, X., Hu, Y., Zhai, J., Li, L., and Guo, P. (2018). A novel non-collision trajectory planning algorithm based on velocity potential field for a robotic manipulator. *Int. J. Adv. Robot. Syst*. 15:1729881418787075. doi: 10.1177/172988141878707

[21] Flacco, F., Kröger, T., De Luca, A., and Khatib, O. (2012). "A depth space approach to human-robot collision avoidance," in *2012 IEEE International Conference on Robotics and Automation* (St Paul, MN), 338–345. doi: 10.1109/ICRA.2012.6225245

[22] Melchiorre, M., Scimmi, L. S., Pastorelli, S. P., and Mauro, S. (2019). "Collison avoidance using point cloud data fusion from multiple depth sensors: a practical approach," in *2019 23rd International Conference on Mechatronics Technology (ICMT)* (Salerno), 1–6. doi: 10.1109/ICMECT.2019.8932143

[23] Liu, Y., Chen, B., Zhang, X., and Li, R. (2021). Research the dynamic path planning of manipulators based on a grid-local probability road map method. *IEEE Access* 9, 101186–101196. doi: 10.1109/ACCESS.2021.3098044

[24] Wittmann, J., Jankowski, J., Wahrmann, D., and Rixen, D. J. (2020). "Hierarchical motion planning framework for manipulators in human-centered dynamic environments," in *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)* (Naples), 525–532. doi: 10.1109/RO-MAN47096.2020.9223549

[25]] M. Abadi et al., "TensorFlow: Large-scale machine learning on heterogeneous systems," arXiv preprint arXiv:1603.04467v1, 2015.

[26] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," arXiv preprint arXiv:1511.05952, 2015

[27] D. Silver, "Cooperative pathfinding," in AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 2005, pp. 117–122

[28] M. Barer, G. Sharon, R. Stern, and A. Fellner, "Sub-optimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in European Conference on Artificial Intelligence, 2014, pp. 961–962.

[29] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha, "Reciprocal nbody collision avoidance," in Robotics Research, 2011, pp. 3–19

[30] G. D. Chen, S. Y. Yao, J. Ma, L. F. Pan, Y. A. Chen, P. Xu, J. M. Ji, and X. P. Chen, Distributed noncommunicating multi-robot collision avoidance via map-based deep reinforcement learning, Sensors, vol. 20, no. 17, p. 4836, 2020

[31]] V. J. Hodge, R. Hawkins, and R. Alexander, Deep reinforcement learning for drone navigation using sensor data, Neural Computing and Applications, vol. 33, no. 6, pp. 2015–2033, 2021.

[32] Y. D. Wang, H. B. He, and C. Y. Sun Learning to navigate through the complex dynamic environment with modular deep reinforcement learning, IEEE Transactions on Games, vol. 10, no. 4, pp. 400–412, 2018.

[33] E. Parisotto and R. Salakhutdinov, Neural map: Structured memory for deep reinforcement learning, arXiv preprint arXiv:1702.08360, 2017.

[34] J. J. Zeng, R. S. Ju, L. Qin, Y. Hu, Q. J. Yin, and C. Hu, Navigation in unknown dynamic environments based on deep reinforcement learning, Sensors, vol. 19, no. 18, p. 3837, 2019.

[35] P. X. Long, T. X. Fan, X. Y. Liao, W. X. Liu, H. Zhang, and J. Pan, Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning, arXiv preprint arXiv:1709.10082, 2018.

[36] C. Sampedro, H. Bavel, A. Rodriguez-Ramos, P. de la Puente, and P. Campoy, Laser-based reactive navigation for multirotor aerial robots using deep reinforcement learning, in Proc. 2018 IEEE/RSJ Int. Conf. Intelligent Robots and Systems, Madrid, Spain, 2018, pp. 1024–1031

[37] H. T. L. Chiang, A. Faust, M. Fiser, and A. Francis, Learning navigation behaviors end-to-end with AutoRL, IEEE Robotics and Automation Letters, vol. 4, no. 2, pp. 2007–2014, 2019

[38] Q. C. Zhang, M. Q. Zhu, L. Zou, M. Li, and Y. Zhang, Learning reward function with matching network for mapless navigation, Sensors, vol. 20, no. 13, p. 3664, 2020

[39] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Bonino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu, et al., Learning to navigate in complex environments, arXiv preprint arXiv:1611.03673, 2017

[40] P. Mirowski, M. K. Grimes, M. Malinowski, K. M. Hermann, K. Anderson, D. Teplyashin, K. Simonyan, K. Kavukcuoglu, A. Zisserman, and R. Hadsell, Learning to navigate in cities without a map, arXiv preprint arXiv:1804.00168, 2019

[41] IN PROC, S. H. Hsu, S. H. Chan, P. T. Wu, K. Xiao, and L. C. Fu, Distributed deep reinforcement learning based on indoor visual navigation. 2018 IEEE/RSJ Int. Conf.

Intelligent Robots and Systems, Madrid, Spain, 2018, pp. 2532–2537.

[42] J. Kulhanek, E. Derner, T. de Bruin, and R. Babu' ska, Vision- ˇ based navigation using deep reinforcement learning, arXiv preprint arXiv:1908.03627, 2019.

[43] G. D. Chen, S. Y. Yao, J. Ma, L. F. Pan, Y. A. Chen, P. Xu, J. M. Ji, and X. P. Chen, Distributed noncommunicating multi-robot collision avoidance via map based deep reinforcement learning, Sensors, vol. 20, no. 17, p. 4836, 2020

[44] X. Y. Lei, Z. Zhang, and P. F. Dong, Dynamic path planning of unknown environment based on deep reinforcement learning, Journal of Robotics, vol. 2018, p. 5781591, 2018

[45] IN PROC, J. W. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, Deep reinforcement learning with successor features for navigation across similar environments. 2017 IEEE/RSJ Int. Conf. Intelligent Robots and Systems, Vancouver, Canada, 2017, pp. 2371–2378

[46] P. X. Long, T. X. Fan, X. Y. Liao, W. X. Liu, H. Zhang, and J. Pan, Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning, arXiv preprint arXiv:1709.10082, 2018.

[47] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, Target-driven visual navigation in indoor scenes using deep reinforcement learning, in Proc. 2017 IEEE Int. Conf. Robotics and Automation, Singapore, 2017, pp. 3357–3364.

[48] L. Tai, G. Paolo, and M. Liu, Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation, in Proc. 2017 IEEE/RSJ Int. Conf. Intelligent Robots and Systems, Vancouver, Canada, 2017, pp. 31–36

[49] H. B. Shi, L. Shi, M. Xu, and K. S. Hwang, End-to-end navigation strategy with deep reinforcement learning for mobile robots, IEEE Transactions on Industrial Informatics, vol. 16, no. 4, pp. 2393–2402, 2020

[50] K. Yokoyama and K. Morioka, Autonomous mobile robot with a simple navigation system based on deep reinforcement learning and a monocular camera, in Proc. 2020 IEEE/SICE Int. Symp. System Integration, Honolulu, HI, USA, 2020, pp. 525–530.