Context-Aware Deep Reinforcement Learning for Autonomous Robotic Navigation in Unknown Area

Anonymous Author(s)
Affiliation
Address
email

Abstract: Mapless navigation refers to the process of guiding a mobile robot to reach a predefined target in an unknown environment. Inspired by the recent developments in deep reinforcement learning (DRL), we propose a learning-based framework which employs a context-aware policy network to obtain non-myopic and efficient path planning for mapless navigation. Our context-aware policy network is capable of extracting and utilizing the spatial information to generate an effective path planning policy. Additionally, a graph rarefaction algorithm is proposed to remove irrelevant or redundant information in the navigation environment. Our approach allows the agent to learn an efficient path planning policy, which can re-plan at low latency while receiving new environmental observations, especially in large-scale and complex environments. We empirically demonstrate that our approach reduces average travel time (i.e., makespan) up to 61.4% and average planning time up to 88.2% compared to benchmark planners (D*lite and BIT) on hundreds of test scenarios. We also validate our approach both in high-fidelity Gazebo simulation as well as on ground robot, highlighting its promising applicability to real-life environments without further tuning.

Keywords: deep reinforcement learning, mapless navigation, context-aware decision-making

1 Introduction

Autonomous navigation is an essential and fundamental capability for mobile robots. Despite mobile robots with the prior global map information (i.e., map-based navigation) are widely acknowledged as well-performed in autonomous navigation, mapless navigation remains a challenging problem - as the unknown environment is gradually observed during the navigation, the planner is required to re-plan promptly and efficiently, aiming to generate a theoretically near-optimal path. Although reactive planners can achieve real-time and safe navigation, they often suffer from *myopic* decisions and get stuck in dead-ends [1, 2], due to the neglect of long-term path planning within the global environment [3]. While planners with long-term path planning ability have been designed to optimize the complete trajectories towards the target [4, 5], these planners often suffer from high computational burden. These planners lack the ability to promptly adjust the planned

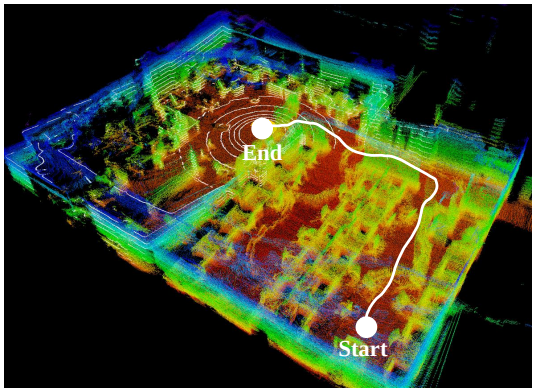


Figure 1: **Illustration of navigation through an unknown environment.** The ground vehicle generates a feasible trajectory to reach the target relies on the sensory inputs and target position.

Submitted to the 7th Conference on Robot Learning (CoRL 2023). Do not distribute.

trajectory at low latency. DRL-based approaches have demonstrated effectiveness in small-scale mapless navigation [2, 3], but these approaches degrade in large-scale and complex environments which could be explained by the agent receives limited feedback of its surrounding environment and struggles to learn an effective policy.

In this paper, we investigate the key issues mentioned above and propose a context-aware DRL framework for mapless navigation. With the incorporation of a graph rarefaction algorithm, we filter out redundant nodes and corresponding edges from the observation graph to get a refined graph as the input of the policy network in our model. The refined graph enhances the efficiency to construct and update the observations received, and also provides spatial information more succinctly, which contributes to a better waypoint policy. Additionally, the attention mechanism in the policy network aggregates the inter-dependencies of the current informative graph, which further helps output an efficient policy for navigation towards the target. Our model not only learns the near-optimal policy to sequence waypoints toward the target, but provides time-efficient re-planning ability even when facing any dead corners or dense obstacles. To assess the performance and generalizability of our approach, we evaluate it with representative baselines in hundreds of simulated maps with three complexity levels, where our model achieves improvements up to 61% for average travel time and 88% for average planning time over the state-of-the-art baselines. We also validate the effectiveness of our approach in large-scale Gazebo simulations, and implement our trained model on hardware in a real-world scenario.

2 Related Works

Search-based approaches: These algorithms are primarily based on discrete grids with node priority evaluation, such as Dijkstra [6], A* [7], Lifelong planning A* (LPA*) [4], and D* lite [5]. Given a global map, these algorithms are able to find near-optimal paths. To achieve efficient re-planning in dynamic environments, environmental incremental information was introduced for LPA* and D* lite [8]. While considering navigation in unknown environments instead of dynamic environments, the unknown area is treated as free space. The agent may enter the unknown area which is not free space, and re-plan accordingly with environmental changes. Nonetheless, numerous limitations arise while attempting to implement these algorithms in a partially-observed environment. For instance, frequent re-plannings always cause significant computation costs and long planning time. Furthermore, these algorithms are highly dependent on heuristic function, which results in them being neither cognitive nor robust [9].

Sampling-based approaches: The rapidly-exploring Random Tree (RRT) [10] family, which includes RRT-Connect [11], Informed RRT* [12], and Batch Informed Tree (BIT) [13], has been leveraged for path planning due to its low computation costs. However, these algorithms usually require global map information to check for collisions and expand tree nodes. Ferguson et al. [14] proposed Dynamic-RRT which can fast re-plan in dynamic environments by regrowing tree nodes from changeable area to the configuration space. However, these algorithms suffer from a long planning time due to large amounts of sampling in both free and unknown areas. Owing to the drawback of random sampling, the generated paths are typically suboptimal and unstable [15], and are prone to being ensnared by local minima in environments characterized by numerous narrow corridors, which necessitates a substantial memory allocation for preserving the tree nodes from previous iterations.

Learning-based approaches: Learning-based mapless navigation approaches have aroused interest in recent years. Tai et al. [16] proposed an end-to-end planner trained with deep deterministic policy gradient (DDPG) [2]. Pfeiffer et al. [3] performed end-to-end navigation based on convolutional neural network (CNN). However, these models are typically based on neural networks with fixed planning horizon, which generate trajectories with their short-term policy. These planners primarily generate collision-free trajectories which may be inefficient for the target oriented navigation. Moreover, these models are typically trained and validated in small-scale and simple environments, which bring the concerns of deploying them into large-scale and complex environments due to their poor generalizability.

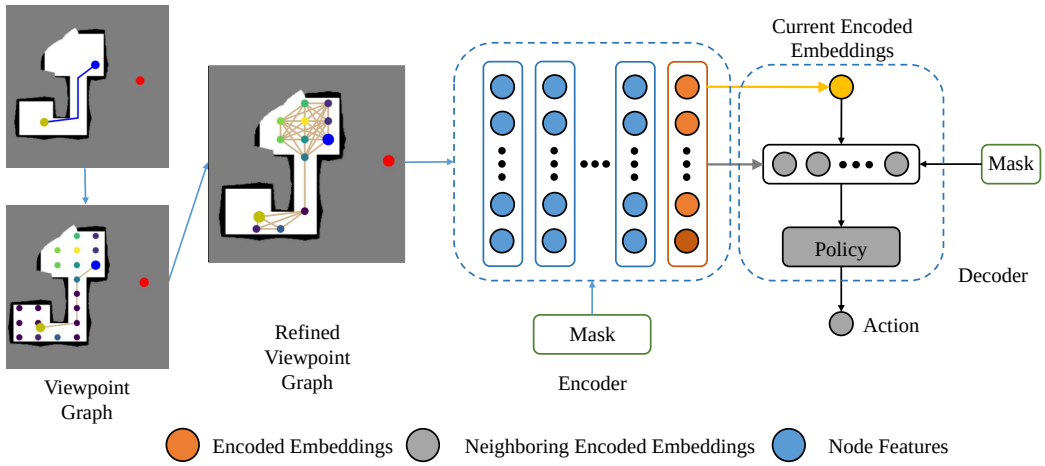


Figure 2: **Framework of our context-aware DRL policy network.**

3 Approach

In this section, we consider mapless navigation as a sequential decision-making RL problem and detail our context-aware policy network, as well as our graph rarefaction algorithm to further improve longer-horizon and larger-scale planning.

3.1 Mapless navigation as a RL Problem

In mapless navigation, the unknown area is updated based on the robot's movements. The robot selects the next waypoint based on its partial observation and iteratively constructs a trajectory of waypoints. We consider mapless navigation as a Partially Observed Markov Decision Process (POMDP), which can be expressed as a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O}, \gamma)$ with the state space $\mathcal{S}$, the action space $\mathcal{A}$, the state transition function $\mathcal{T}$, the reward function $\mathcal{R}$, the observation space Ω, the observation function given the true state s' of the environment $\mathcal{O}(o_t \in \Omega | s', a)$ and the action state a, and the discount factor γ. To promote efficient navigation towards the target, the RL objective is aimed to find an optimal policy π^* that maximizes the expected discounted reward: $\mathbb{E}_{a_t \sim \pi(\cdot|o_t)}\left[\sum_{t=1}^{T} \gamma^{t-1} r_t\right]$, where the policy π can be considered as a mapping function from o_t to the next action a_t.

Observation Ω: At each decision step t, our observation is represented as $o_t = (G_t, S_t)$, which consists of the viewpoint graph G_t and the planning attributes S_t. The robot gets the updated observation in the limited sensor range d_s (80 in practice training). The agent first obtains the viewpoint set V_t, which is generated uniformly in the known area $\mathcal{D}$. To construct a traversable graph, every node in V_t constructs up to k nearest edges with each other, where the edges in E_t can only connect between collision free nodes, i.e., nodes that are *line of sight* with each other. The construction of $G_t = (V_t, E_t)$ eliminates the concerns of collision with obstacles. Moreover, the planning attributes S_t provide additional information about the observed environment and the target for each node in V_t. Inspired by [17], every node has a direction vector $\vec{v}$ which acts as a signpost for the target, which consists of a unit vector $\hat{v}$ indicating the direction towards the target and the Euclidean distance $|\vec{v}|$ from the node to the target. S_t further include an indicator δ_i which records whether the node has been visited before. With the knowledge of the former trajectory, the agent can produce a more informed policy. S_t also includes the utility u_i which is termed as the number of observable frontiers. These frontiers represent the areas with the potential to the target, which are generated at the boundary of the observed and unknown area. S_t of each node in V_t are finally formulated as $\{\vec{v_i}, \delta_i, u_i\}$.

It is worth noting that in large-scale environments, V_t would be populated densely. Meanwhile, the information contained within them is sparse. Therefore, we implement a graph rarefaction algorithm, shown as Algorithm 1 to prune irrelevant nodes and extract key edges of the viewpoints.

Specifically, graph rarefaction first clusters the non-zero utility node set U into multiple groups according to the threshold radius d_{th} (30 in practice). Then, the algorithm uses A* to search for the shortest path ζ, from the robot's current position p_t to each group. After that, the refined node set $V^r \subseteq U$ is constructed by waypoints on ζ, which is either out of d_{th} or out of *line of sight*. After the graph rarefaction process, the refined viewpoint graph $G^r = (V^r, E^r)$ would represent the complete information in the current robot belief. Finally, the G^r along with the corresponding planning states S^r are concatenated as the input of policy network, i.e., the refined observation $o_s = (G^r, S^r)$.

Algorithm 1: Graph Rarefaction Algorithm

Input: non-zero utility node set U, map $\mathcal{M}$, robot position p_t, threshold radius d_{th}
Initialize refined node set $V^r \leftarrow U$, covered node set $\overline{U} \leftarrow \emptyset$;
for $v \in U$ **do**
  if $v \in \overline{U}$ **then** continue;
  Find nearby node set N in d_{th};
  for $v' \in N$ **do**
    if $line(v, v')$ *is collision free* **then** $\bar{U} \leftarrow v'$;
  end
  Find path ζ from p_t to v, set ref node $v_{ref} = v$;
  for $i \in |\zeta|$ **do**
    if $line(v_{ref}, \zeta_i)$ *is not collision free or* $L(v_{ref}, \zeta_i) \geq d_{th}$ **then**
      $v_{ref} = \zeta_{i-1}, V^r \leftarrow v_{ref}$
  end
end
Output: collision free edge set E^r based on V^r and $\mathcal{M}$.

Action $\mathcal{A}$: The collision-free graph extends incrementally along with the update of the agent's observations o_t for every decision step. Our context-aware policy network outputs the stochastic policy π for the agent to select the next waypoint among the neighboring nodes. Then, the agent updates its partial map while moving to the next waypoint.

Reward $\mathcal{R}$: To promote efficient navigation, the agent receives a reward consisting of three parts. The first part r_s is a constant time step penalty r_s (-0.5 in practice). The second part $r_b = d(s_{t-1}) - d(s_t)$ provides continuous feedback on the robot's proximity to the target, where $d(s)$ is the distance between the agent current position p_t and the target computed by A*. r_b is used to encourage the agent reach target as fast as possible. The last part r_f is a fixed finishing reward, set to 20 while reaching the target and 0 otherwise. To sum up, the overall reward is $r_t(o_t, a_t) = r_s + c_b \cdot r_b + r_f$, where c_b is a scaling parameter ($c_b = 1/64$ in practice).

3.2 Policy Network

We design an attention-based neural network (shown in Fig. 2) to generate efficient path planning towards the target. In our policy network, the encoder embeds the overall information of the current partial map, and the decoder reasons the local information of the current node and its neighboring nodes to finally output the action policy.

Encoder: The refined observation is first normalized and projected into d-dimensional (128 in practice), termed as h^n. The node features h^n are then passed into multiple attention layers (6 in practice) to aggregate the spatial representation of current observation. The input of each attention layer consists of a query vector h^q and a key-value vector $h^{k,v}$ of the same dimension. The attention layer updates the query vector with the weighted sum of the value, where the attention weight depends on the similarity between query and key. In each attention layer, the query q_i, key k_i and value v_i are first calculated as $q_i = W^Q h_i^q$, $k_i = W^K h_i^{k,v}$ and $v_i = W^V h_i^{k,v}$ respectively, where W^Q, W^K, $W^V \in \mathbb{R}^{d \times d}$ are learnable matrices. Next, the similarity between the query q_i and the key k_j is computed with a scaled dot product as $u_{ij} = \frac{q_i^T \cdot k_j}{\sqrt{d}}$. The attention weights a_{ij} are then obtained using a softmax function: $a_{ij} = \frac{e^u ij}{\sum_{j=1}^{n} e^u ij}$. Finally, the output embedding from an attention layer, denoted as h'_i, is calculated as the weighted sum of value vectors as v_j: $h'_i = \sum_{j=1}^{n} a_{ij} v_j$. Addition-

ally, an encoder edge mask M is applied to restrict each node to access its neighboring features. The output of the encoder, which we term it as the *encoded embeddings* $\hat{h^e}$, provides condensed spatial information to the decoder.

Decoder: In the decoder, the encoded embeddings of the current node (termed *current encoded embeddings*, $\hat{h^c}$) are first extracted from the encoder as well as its neighboring encoded embeddings (termed *neighboring encoded embeddings*, $\hat{h^n}$). Then the current encoded embeddings and encoded embeddings are applied to an attention layer with $h^q = \hat{h^c}, h^{k,v} = \hat{h^n}$. This attention layer calculates the output attention weights, which represents the relevance of each neighboring node to the current node. The attention weights are concatenated with h^c and projected back to the d-dimensional feature together, termed as the *current decoded embeddings*, $\tilde{h^c}$. The current decoded embeddings incorporate the information from neighboring nodes into the representation of the current node. Finally, the current decoded embeddings and the encoded embeddings of neighboring nodes are input into a pointer layer [18], an attention layer that utilizes normalized attention weights θ as the output.

3.3 Training Settings

Inspired by Chen et al. [19], we implemented a random dungeon map dataset for training, where there are totally 800 maps (Each map is 1000×1000 $pixels$) like figure 3. To build the updated collision-free graph, the agent treats the points in the known free area as candidate viewpoints from 1600 points which are uniformly distributed to cover each dungeon map. Our model is trained using the Soft Actor-Critic (SAC) algorithm [20], where the maximum episode threshold is set to 128, and the size of the replay buffer is set to 10000. The target entropy is set to $0.01 \cdot \log k$, where k represents the number of neighbouring nodes. The Adam optimizers are employed to optimize policy and critic networks with learning rates of 1×10^{-5} and 2×10^{-5} , respectively. Our model is trained on a workstation with one i9-10980XE CPU and one NVIDIA GeForce RTX 3090 GPU, and the training starts after collecting 2,000 steps in the replay buffer. We utilize Ray [21], a distributed framework for machine learning, with eight compute instances in practice to parallelize and expedite data collection. The training process typically requires around 12 hours to converge.

4 Experiments

To implement a precise evaluation, we define the evaluative criteria as the average travel distance $D(p)$ and average travel time $T(s)$ (lower is better for both) for mapless navigation. The $T(s)$ is the sum up of algorithm step planning time $T_p(s)$ and the motion execution time $T_e(s)$. We first compare our approach with state-of-the-art conventional baselines in numerous dungeon environments (like Fig. 3). Then, we compare our model, some variants of our model, and FAR planner [22] in a large-scale Gazebo environment (Fig. 5). Lastly, we implement our trained model on hardware in a real-world scenario. (Fig. 6).

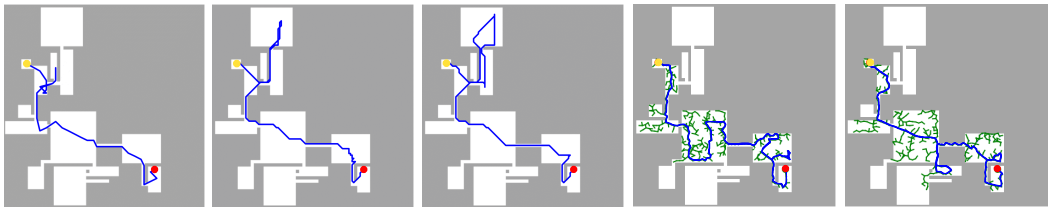
Figure 3: **Examples scenarios with different complexity,** showing the occupied area (grey cells), free space (white cells), start points (yellow block), and target points (red block).

4.1 Comparisons in dungeon environments

To ensure a convincing comparsion, we built extensive testing environments through random dungeon map generator [19], which were never seen by our trained model. Fig. 3 shows the diverse complexity of testing environments, which can be categorized into three types (50 scenarios each), noted as simple, medium, and complex. We define the quantitative criteria of the scenario complexity as follows: (i) The Euclidean distance between start and target positions in each scenario. (ii) The overall number of connecting corridors in all rooms. (iii) The number of intersections that will be encountered along the visually-optimal path.

Table 1: **Evaluations of our model and baselines in simple, medium, and complex scenarios (50 scenarios for each test set).** We suppose the environment is $200 \times 200\ m^2$, the radius of LiDAR scanning range is set as $30\ m$, the robot executes motion constantly at $1.0\ m/s$, and the neighboring nodes parameter k is set as 20 in our model.

	Criteria	Ours	D* lite	LPA*	RRT	BIT
Simple	$D(p)$	1121(±362)	**1072**(±417)	1463(±580)	1988(±1043)	1375(±833)
	$T(s)$	**285**(±90)	383(±152)	414(±206)	662(±381)	404(±237)
	$T_p(s)$	**0.20**(±0.08)	1.24(±0.56)	0.74(±0.47)	0.67(±0.54)	0.55(±0.46)
Medium	$D(p)$	1299(±414)	**1187**(±167)	1917(±650)	2485(±1108)	1802(±758)
	$T(s)$	**329**(±105)	484(±87)	594(±298)	852(±565)	563(±285)
	$T_p(s)$	**0.18**(±0.12)	1.52(±0.73)	0.79(±0.55)	0.68(±0.48)	0.53(±0.44)
Complex	$D(p)$	1877(±595)	**1746**(±472)	2329(±817)	2722(±1110)	2463(±1073)
	$T(s)$	**477**(±151)	680(±221)	754(±265)	1063(±782)	806(±407)
	$T_p(s)$	**0.22**(±0.13)	1.66(±0.95)	0.83(±0.60)	0.62(±0.42)	0.55(±0.45)



Ours **D* lite** **LPA*** **RRT (partial)** **BIT (partial)**

Figure 4: **Visualization of our approach and baselines trajectories in a complex scenario.** The blue lines are robot trajectories starting at the yellow dot and ending at the red dot. The green lines in RRT and BIT are sampling trees in known areas, which are not included in $D(p)$.

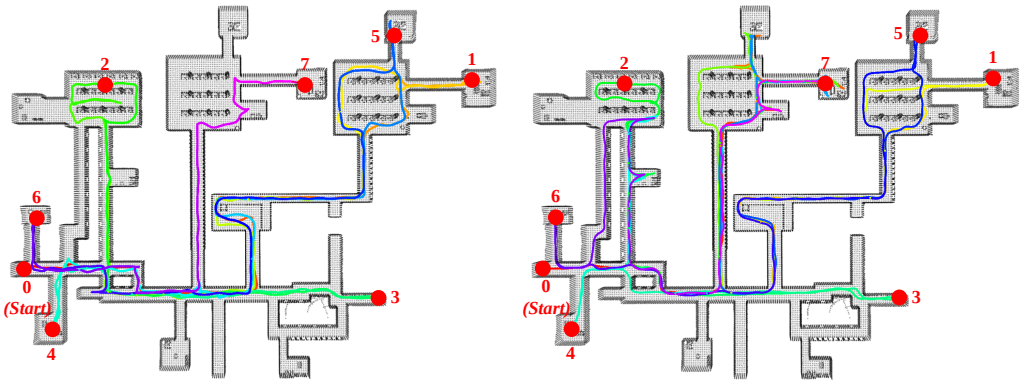
We benchmark two search-based approaches: LPA* [4], D* lite [5], as well as two sampling-based approaches: RRT [10], and BIT [13]. Here, BIT is considered as the state-of-the-art in the sampling-based algorithms and D* lite is considered as the state-of-the-art in the search-based algorithms. To conduct comparative and rigorous analyses, each baseline should fulfill the setting of mapless navigation. Both LPA* and D* lite consider the unknown area as free space and reset each waypoint as their new start point. RRT and BIT are constrained from sampling and verifying collisions in unknown areas, meaning the target can only be sampled when a vertex node of tree/graph is nearby. Otherwise, a sub-goal, the nearest point to the target within the explored area, is provided to offer a rough direction.

The evaluation results are reported in Table 1. For search-based approaches, our model outperforms D* lite in terms of $T(s)$ and $T_p(s)$. Despite D* lite showing a lower $D(p)$, our model still surpasses it in terms of $T(s)$ by up to 15-32%, primarily due to the significantly shorter $T_p(s)$, which is reduced by 83-88%. D* lite is renowned for finding optimal paths through searching and assessing node priorities iteratively. However, the results illustrate that D* lite always incurs intolerable (the worst $T_p(s)$ among all), which is not capable of real-time planning. Moreover, our model outperforms LPA* in three criteria by a substantial margin of approximately 23%, 45%, and 73%. For sampling-based approaches, our model surpasses RRT and BIT in all criteria. Our model finishes the task more than 30-41% faster than BIT, and more than 56-61% faster than RRT. Additionally, our model demonstrates superior performance in terms of $D(p)$, with improvements ranging from 18-28% to BIT and 31-49% to RRT. The results illustrate that both RRT and BIT are prone to generate inefficient trajectories, which consequently increases $D(p)$ and $T(s)$.

Fig. 4 shows an example that our planner generates a more efficient trajectory, while other baselines suffer from the misleading of target position. Our model always exhibits more interest in the unknown area with high potential towards the target, which substantially reduces the likelihood of aimless exploration and consistently aids in generating superior navigation trajectories.

Table 2: **Evaluations of our model and baselines in large-scale, complex Gazebo environment** **(**$130\ m \times 100\ m$**).** the robot executes motion constantly at 2.0 m/s, the radius of LiDAR scanning range is set as 30 m, the $D(p)$ and $T(s)$ are the overall values after traveling 7 goals. The variant without graph rarefaction is trained with $k = 20$, and there is no extra training on $k = 10$ and $k = 5$.

	Ours	**FAR**	**Ours** (no graph rarefaction)	**Ours** ($k = 10$)	**Ours** ($k = 5$)
$D(p)$	**1367.2**	2060.1	1483.5	1849.2	2103.6
$T(s)$	**850.7**	1359.4	1131.5	1112.1	1382.0
$T_p(s)$	0.83($\pm$0.37)	2.42($\pm$0.35)	1.96($\pm$0.29)	0.38($\pm$0.11)	**0.36**($\pm$0.12)



(a) Our planner (b) FAR planner

Figure 5: **Trajectories comparison of two planners in large-scale indoor Gazebo environment.**

4.2 Comparisons in Gazebo environment

We evaluate our planner and some variants (see Table 2) in a highly convoluted environment characterized by narrow corridors and various obstacles. The robot is required to navigate through a series of predefined target points successfully, similar to the setting presented in FAR planner [22]. To ensure fairness and consistency in the evaluations, we employ a practice of resetting the planners after reaching each target point, which ensures each navigation task begins in a fully-unknown environment.

As shown in Fig. 5(a) and 5(b), our model is capable of doing more informed exploration and generating efficient trajectories towards the target. The experimental results (see Table 2) demonstrate that the significant reduction achieved by our model in terms of $D(p)$ and $T(s)$, by up to 692.9m and 508.7s, when compared to FAR planner. It provides evidence that our model excels in generating more optimal paths than FAR planner in unknown environments. Additionally, our model outperforms FAR planner in terms of $T_p(s)$, consuming only 0.83s.

In addition, we conduct further comparisons among several variants of our model. Specifically, different values for k (20, 10, 5 respectively) represent the number of neighboring nodes. The experimental results indicate that $k = 10$ surpasses the FAR planner in all criteria, while there is no extra training on it. Furthermore, the $T_p(s)$ of the $k = 5$ is less than that of the $k = 10$ only by 0.02s, while the $D_p(s)$ of $k = 5$ is larger than $k = 10$ up to 254.4m, indicating much less efficient trajectories of $k = 5$. Thus, it's not wise to set k as a minor value due to the lack of informative dependencies from neighboring nodes. To evaluate the significance of graph rarefaction, the no sparse variant is tested which costs more than 136% $T_p(s)$ and 8% $D(p)$ compared to the ($k = 20$). Therefore, we believe that graph rarefaction significantly benefits our model.

Our model utilizes the context-aware policy network to generate the next waypoint, which takes into account the dependencies between the current node and its neighboring nodes, which plays a crucial role in helping the agent to generate efficient trajectories. Furthermore, the utilization of graph

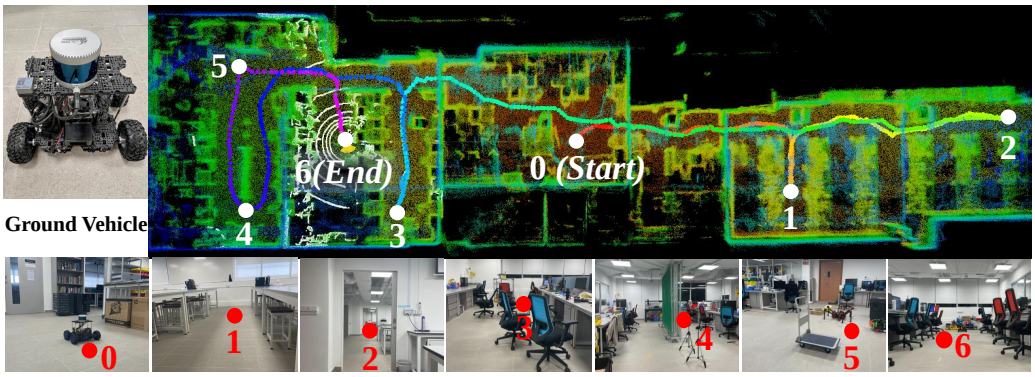


Figure 6: **Validations in real-world environment.** The ground vehicle, provided with partially observed point-cloud data, starts at point 0 and subsequently traverses a series of consecutive points.

rarefaction not only refines the dependencies between the observed partial map and the unknown areas, but also prompts the policy to generate more efficient trajectories towards the target.

4.3 Hardware Validation in a Real-World Scenario

Our ground vehicle utilizes a Velodyne Puck (VLP-16) LiDAR and a HIF-A9 IMU for state estimation (shown in Fig. 6). The real-world scenario is a 60×15 m^2 laboratory with randomly placed obstacles (e.g., chairs, boxes, and camera tripod). The vehicle should travel through a series of predefined target points by following the waypoints that were published from our model. After receiving the waypoint, the local planner [23] generates real-time and feasible motion commands for the ground vehicle. The experimental trajectory depicts a near-optimal solution which validates the effectiveness of our model in a real-world environment.

5 Limitations

We believe that the limitations of our approach can be divided into two aspects: adaptive sampling and smooth motion control.

- We currently sample the observations uniformly, which may not precisely capture the spatial representation of the environment and leads to suboptimal navigation strategies. To tackle this, we will look into the development of an online adaptive sampling strategy.
- Although our model can re-plan the next waypoint, it is better to consider the practical constraints imposed by real-life robot motion. The incorporation of a local motion planner can facilitate the robot navigation in unknown environments with dynamic obstacles.

6 Conclusion

In this paper, we propose a context-aware DRL framework which is capable of efficiently handling navigation in unknown environments. The attention mechanism allows our model to extract and employ the spatial information to generate an effective path planning policy. A graph rarefaction algorithm is utilized to filter out redundant information from the agent observation, thereby significantly pruning the path planning time. We benchmark the performance of the sampling-based methods (RRT and BIT), as well as search-based methods (LPA* and D* Lite), in hundreds of dungeon environments. Furthermore, we validate our trained model in high-fidelity Gazebo simulations and real-world environments with a ground vehicle. Extensive experimental results demonstrate that our model is capable of generating a more real-time and efficient policy towards the target in unknown environments compared to the state-of-the-art baselines.

Future work will first focus on the construction of local planner with more considerations on kinematic/dynamics constraints. Then, we will extend our framework to multi-agent mapless navigation, where robots need to reason about each other and plan efficient paths cooperatively, by leveraging synergies and avoiding redundant work.

References

[1] N. Bucki, J. Lee, and M. W. Mueller. Rectangular pyramid partitioning using integrated depth sensors (rappids): A fast planner for multicopter navigation. *IEEE Robotics and Automation Letters*, 5(3):4626–4633, 2020.

[2] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[3] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena. From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots. In *IEEE International Conference on Robotics and Automation (ICRA)*, page 1527–1533. IEEE, 2017.

[4] S. Koenig, M. Likhachev, and D. Furcy. Lifelong planning a*. *Artificial Intelligence*, 155(1-2): 93–146, 2004.

[5] S. Koenig and M. Likhachev. Dˆ* lite. *Aaai/iaai*, 15:476–483, 2002.

[6] E. W. Dijkstra. A note on two problems in connexion with graphs. In *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, pages 287–290. 2022.

[7] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968. doi:10.1109/TSSC.1968.300136.

[8] K. Xie, J. Qiang, and H. Yang. Research and optimization of d-start lite algorithm in track planning. *IEEE Access*, 8:161920–161928, 2020.

[9] D. Ferguson, M. Likhachev, and A. Stentz. A guide to heuristic-based path planning. In *Proceedings of the international workshop on planning under uncertainty for autonomous systems, international conference on automated planning and scheduling (ICAPS)*, pages 9–18, 2005.

[10] S. M. LaValle et al. Rapidly-exploring random trees: A new tool for path planning. 1998.

[11] J. J. Kuffner and S. M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE, 2000.

[12] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Informed rrt: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2997–3004. IEEE, 2014.

[13] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Batch informed trees (bit): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 3067–3074. IEEE, 2015.

[14] D. Ferguson, N. Kalra, and A. Stentz. Replanning with rrts. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 1243–1248. IEEE, 2006.

[15] A. Perez, R. Platt, G. Konidaris, L. Kaelbling, and T. Lozano-Perez. Lqr-rrt*: Optimal sampling-based motion planning with automatically derived extension heuristics. In *2012 IEEE International Conference on Robotics and Automation*, pages 2537–2542. IEEE, 2012.

[16] L. Tai, G. Paolo, and M. Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 31–36. IEEE, 2017.

[17] G. Sartoretti, J. Kerr, Y. Shi, G. Wagner, T. S. Kumar, S. Koenig, and H. Choset. Primal: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters*, 4(3):2378–2385, 2019.

[18] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. *Advances in neural information processing systems*, 28, 2015.

[19] F. Chen, S. Bai, T. Shan, and B. Englot. Self-learning exploration and mapping for mobile robots via deep reinforcement learning. In *AIAA SciTech Forum*, page 0396, 2019.

[20] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pages 1861–1870. PMLR, 2018.

[21] P. Moritz, R. Nishihara, S. Wang, A. Tumanov, R. Liaw, E. Liang, M. Elibol, Z. Yang, W. Paul, M. I. Jordan, et al. Ray: A distributed framework for emerging ai applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pages 561–577, 2018.

[22] F. Yang, C. Cao, H. Zhu, J. Oh, and J. Zhang. Far planner: Fast, attemptable route planner using dynamic visibility update. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 9–16, 2022. doi:10.1109/IROS47612.2022.9981574.

[23] J. Zhang, C. Hu, R. G. Chadha, and S. Singh. Falco: Fast likelihood-based collision avoidance with extension to human-guided navigation. *Journal of Field Robotics*, 37(8):1300–1313, 2020.