2019 International Conference on Robotics and Automation (ICRA)
Palais des congres de Montreal, Montreal, Canada, May 20-24, 2019

Deep Reinforcement Learning of Navigation in a Complex and Crowded Environment with a Limited Field of View

Jinyoung Choi[1], Kyungsik Park[1], Minsu Kim[1] and Sangok Seok[1]

Abstract— **Mobile robots are required to navigate freely in a complex and crowded environment in order to provide services to humans. For this navigation ability, deep reinforcement learning (DRL)-based methods are gaining increasing attentions. However, existing DRL methods require a wide field of view (FOV), which imposes the usage of high-cost lidar devices. In this paper, we explore the possibility of replacing expensive lidar devices with affordable depth cameras which have a limited FOV. First, we analyze the effect of a limited field of view in the DRL agents. Second, we propose a LSTM agent with Local-Map Critic (LSTM-LMC), which is a novel DRL method to learn efficient navigation in a complex environment with a limited FOV. Lastly, we introduce the dynamics randomization technique to improve the robustness of the DRL agents in the real world. We found that our method with a limited FOV can outperform the methods having a wide FOV but limited memory. We provide the empirical evidence that our method learns to implicitly model the surrounding environment and dynamics of other agents. We also show that a robot with a single depth camera can navigate through a complex real-world environment using our method.**

I. INTRODUCTION

Recently, increasing number of mobile robots are being deployed in living spaces. Mobile robots provide services such as delivery, surveillance, and guidance. To provide these services, safe navigation in a complex and crowded environment is essential.

Most of methods for mobile robot navigation consist of global planner and local planner/control policy. The global planner creates a trajectory or waypoints using a global structure of the whole environment. Then, the local planner or control policy follows the global plan while avoiding collisions with unexpected, dynamic obstacles such as pedestrians.

For the local planners (or control policies), approaches such as artificial potential fields [1], dynamic window approaches [2] are widely used. However, most of these rule-based algorithms are known to suffer problems such as being stuck in local minima [3], heavy dependency on an accurate map, and a lack of generalization across different environments.

To overcome these problems, deep reinforcement learning (DRL)-based control approaches were proposed [4]–[7]. These approaches learn the optimal parameters that directly map the sensor input to the velocity of the robot by interacting with the environment.

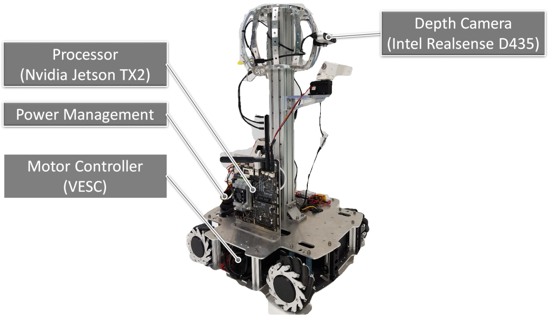

Fig. 1. Mobile robot platform that was used in our hardware experiments. It has only one Intel Realsense D435 depth camera with a FOV of 90° and one NVIDIA Jetson TX2 as a processor.

Although the deep reinforcement learning approaches showed promising results, existing methods consider only static, simulated environments [8] or require a wide FOV that imposes the usage of expensive lidar devices [4], [5].

In this work, we explore the possibility of replacing high-cost lidars with inexpensive depth cameras with a limited FOV in DRL agents. Our experiments show that the limited FOV severely degrades the performance of the DRL models. We also propose a novel deep reinforcement learning method to overcome the limited FOV. Our approach uses recurrent neural network and novel Local-Map Critic (LMC) architecture to cope with the limited FOV. In the simulation experiments, our method with a limited FOV outperformed the memoryless methods with wide FOVs by implicitly modeling the surrounding areas and dynamics of other agents. Then, we introduce the dynamics randomization technique to overcome the recurrent agent's sensitivity to the control frequency and wheel drift in the real world. Lastly, we show that a mobile robot with a single depth camera (Figure 1) can navigate safely and efficiently in challenging real world scenarios using our method.

The main contributions of our work are the followings:

- Empirical analysis of the effect of the limited FOV in the DRL agents,
- Novel DRL method for mobile robot navigation with a limited FOV, and
- Dynamics randomization technique to improve DRL agent's robustness in the real world.

First, related works will be introduced. Then, our methods will be discussed in detail. After that, we will present the simulation experiment results. Lastly, the results of the

[1] NAVER LABS, Gyeonggi-do, 13494, South Korea

[1] jy-choi, kay.pak, minsu17.kim, sangok.seok @navelabs.com

*High resolution version of the supplementary video can be found in https://youtu.be/432jivlvx-o

978-1-5386-6026-3/19/$31.00 ©2019 IEEE

hardware experiments will follow.

II. Related Works

A. DRL methods for mobile robot navigation

As classic approaches for the mobile robot navigation [1], [2], [9] depend on the human-engineered hyper parameters and rules, they often fail in a complex and dynamic environment due to the problems such as sensitivity to the hyper parameters or local minima.

Deep reinforcement learning (DRL)-based approaches are widely studied to overcome these problems. In DRL approaches, the agent learns to map the sensor inputs directly to the velocity of the robot from the data collected by interacting with the environment.

Recently, works such as [6] and [8] proposed the DRL agents that can navigate through a complex indoor environment using the RGB-D images. Although these methods showed remarkable results in the simulation experiments, they are difficult to be deployed in the real world due to the large difference between RGB-D scenes across different environments and a lack of ability to avoid dynamic obstacles.

Other works proposed more practical real-world solutions. [7] proposed a socially aware collision avoidance method. Although [7] showed a robust performance in the real world, it requires explicit measurement of positions and velocities of other agents (or pedestrians). Works such as [5], [10], and [4] proposed DRL agents that use raw lidar data. [5] combined probabilistic road map [11] and DRL to enable the long-range navigation across the complex environments, but they considered only static obstacles, making their usage in a crowded real-world environment difficult. [10] and [4] proposed DRL agents that can learn to navigate in a crowded environment. Although these agents were successfully deployed in the real world, they require a expensive lidar equipment to retain a wide FOV (220°in [5], 180°in [4], [10]).

Our work follows a similar strategy to [10] and [4]. In contrast to these works, we explore the effect of a limited FOV in a DRL agent since it is often more desirable to use a low-cost depth camera with a limited FOV instead of a expensive lidar device.

B. Multi-agent DRL

DRL methods for multi-agent setting have gained increasing attention recently. [12]–[14] proposed neural network architectures that can learn implicit communication protocol between multiple agents. Although the agents with these methods showed better performance than agents without communications or a centralized controller, they require direct messaging to one another, which is impossible in human-robot interaction scenarios. [15] proposed Multi-Agent Deep Deterministic Policy Gradient (MADDPG) method that gives other agents' information only to the critic. This algorithm showed that a cooperative behavior can emerge without explicit message exchange in test time, opening the possibility to be used in a human-robot interaction situation such as a navigation in a crowded environment. We extend the approach of [15] by giving information about the environment as well as other agents to the critic.

C. Direct deployment of DRL agents in the real world using dynamics randomization

Although DRL methods achieved great success in game domains [16]–[18], deploying DRL agents in the real world robotics tasks is considered more difficult because of difference between the real world and simulators. This difference severely degrades the performance of agents when they are deployed without fine tuning after being trained in a simulator. To overcome this problem, dynamics randomization in a simulator has been used [19]–[21]. These works showed that dynamics randomization improves robustness of the agents in challenging real-world robotics tasks such as locomotion of quadruped robots [19] or object manipulation using robotic arms [20], [21]. In this work, we explore the effects of dynamics randomization in the mobile robot navigation task by randomizing sensor noise, wheel drift, and control frequency in simulation.

III. APPROACH

In this section, we briefly cover the deep reinforcement learning framework that we use in our method and problem setting. Then, we introduce our LSTM-LMC architecture. Lastly, we provide details about our training environment and dynamics randomization technique.

A. Deep Reinforcement Learning

Due to the partial observability caused by a limited FOV and uncertainty about other agents' states, we model our environment as partially observed Markov decision process (POMDP) [22].

POMDP consists of 6 tuples (S, A, P, R, Ω O) where S is the state space, A is the action space, P is the transition probability, R is the reward function, Ω is observation space, and O is observation probability.

The goal of reinforcement learning [23] is to learn the policy of the agent $\pi(a,o) = p(a|o)$ that maximizes discounted return

$$G = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}[r(s_t, a_t)] \tag{1}$$

where $\gamma \in [0,1]$ is the discount factor for future rewards.

Recently, deep neural networks are widely used to learn the policy parameters or value functions of the reinforcement learning agents [16], [17], [24], [25]. In this work, we use the Soft Actor-Critic (SAC) algorithm [26] which jointly maximizes the entropy of stochastic policy with return :

$$G = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}[r(s_t, a_t) + \alpha H(\pi(\cdot|s_t))] \tag{2}$$

We choose the SAC algorithm for its robustness to the hyper parameters, sample efficient learning in continuous action space, and desirable exploration property. We refer [26] for details of SAC algorithm.

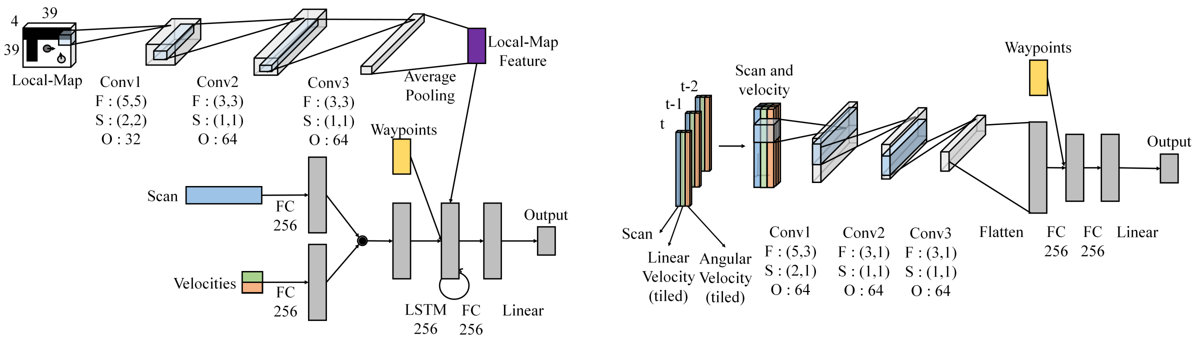

Fig. 2. Neural network architectures that were used in experiments: Proposed LSTM-LMC architecture (Left) and CNN architecture (Right). In the convolutional layers, 'F' is the filter size, 'S' is the stride, and 'O' is the output channel. We use the same architecture for the actor, the Q network, and the V network. Note that the local-map branch was not used in the actor network.

B. Problem Setting

1) Observation space: For the observation o of the agent, we use sliced point clouds, which are similar to the lidar data, with various horizontal FOVs (90°, 120°, and 180°). We first calculate the point cloud from the depth image and truncate the point cloud horizontally to remove floors and ceilings. Then we uniformly slice the truncated point cloud vertically with a 5°interval and select the distances from the nearest point in each segment, resulting in ($\mathbb{R}^{18}$,$\mathbb{R}^{24}$,$\mathbb{R}^{36}$) vectors. We will refer these vectors as 'depth scan' hereinafter.

We also use the $\mathbb{R}^2$ vectors that consist of the agent's current linear and angular velocities. These velocities are normalized into range [-1,1].

In addition, we use $\mathbb{R}^{15}$ vectors that represent the relative distances and angles of next 5 waypoints in the form of $[r_1, sin(\theta_1), cos(\theta_1), r_2, ..., cos(\theta_5)]$ where r_i is the distance to the i'th waypoint, and θ_i is the angle in a polar coordinate.

2) Action space: For the action a of the agent, we use $\mathbb{R}^2$ vectors for linear and angular velocities. Linear velocity of the agent is in range [0,1]m/s and angular velocity is in range [-90,90]°/s. We use normalized velocities which are in range [-1,1] as outputs of the neural networks.

3) Reward function: The reward r consists of four terms as follows :

$$r = r_{base} + r_{collision} + r_{waypoint} + r_{rotation} + r_{safety}$$

$r_{base} = -0.05$ is the small negative base reward that is given in every timestep to encourage the agents to follow the shortest path.

$r_{collision} = -20$ is the collision reward that penalizes the agents when they collide to a wall or other agents.

$r_{waypoint} = 3$ is given to the agent when the distance between the agent and the next waypoint is less than 1 meter. For the final waypoint (goal), we set the threshold to 0.6 meter.

$r_{rotation}$ is the penalty for a large angular velocity which is also used in [10]. It is defined as

$$r_{rotation} = \begin{cases} -0.15 * |w| & \text{if } \pi/4 \leq |w| \\ 0 & \text{else} \end{cases},$$

where w is the angular velocity of the agent in radian.

r_{safety} is the small penalty to encourage the agents to avoid the obstacles as proactive as possible and is defined as

$$r_{safety} = min_{o_i \in Obs}(-0.15 * (score_x(o_i) + score_y(o_i)))$$

where Obs is a set of all obstacles in the environment including other agents. $score_x$ and $score_y$ are defined as

$$score_x(o_i) = \begin{cases} max(0, 1 - d_x/3) & \text{if } 0 \leq d_x \\ max(0, 1 + d_x/0.3) & \text{if } d_x < 0 \end{cases},$$

$$score_y(o_i) = max(0, 1 - |d_y|),$$

where d_x and d_y are relative displacements between the agent and o_i in x-axis and y-axis.

C. LSTM Agent with Local-Map Critic (LSTM-LMC)

A limited FOV imposes heavy partial observability to the DRL agents. Partial observavility makes the estimation of accurate state-action value [23] difficult and may lead to sub-optimal decision making. To overcome this partial observability, we propose the LSTM agent with Local-Map Critic (LSTM-LMC) (Figure 2, Left).

1) LSTM agent: We use LSTM [27] to give the agents memory ability. As we will analyze in Experiments section, memory plays an important role in collision avoidance by implicitly building the representation of the surrounding environment and dynamics of moving obstacles. LSTM alone significantly improved the performance of the agent with a limited FOV in our experiments. We train LSTM (and LSTM-LMC) agents by sampling the 200-step trajectories from the experience replay [28]. Trajectories are sampled from random points of the episodes and LSTM's states are zeroed at the beginning of each trajectory.

2) Local-Map Critic: [15] showed that incorporating additional information such as other agents' actions in the critic improves the performance in multi-agent DRL. Since the actor does not require additional information and the critic is usually not used after the training is complete, we can still deploy the agents that are trained with this approach without expensive additional information. We extend this method by giving the 2D local-map of the surrounding area, instead of just other agents' actions, to the critic. The local-map M covers ($10m \times 10m$) area around the agent. It is a tensor that has a size of ($39 \times 39 \times 4$) and the values of $M_{i,j,k}$ are defined as follows.

$$M_{i,j,0} = \begin{cases} 1 & \text{if } M_{i,j} \text{ is movable} \\ 0 & \text{if } M_{i,j} \text{ is obstacle} \\ 0.33 & \text{if } M_{i,j} \text{ is other agent} \\ 0.66 & \text{if } M_{i,j} \text{ is self} \end{cases},$$

and $M_{i,j,1:3}$ encodes the normalized heading, linear velocity, and angular velocity if $M_{i,j}$ represents an agent (0 otherwise).

3) Network architecture: The network architecture of LSTM-LMC model is depicted in Figure 2 (Left). We first project the depth scan and the velocities to the vector of the same size using the fully connected layers and apply element-wise product to these two vectors to obtain observation feature. In the critics (Q network and V network of [26]), local-map tensor passes through 3 convolutional layers and global average pooling is applied, resulting in local-map feature. Then, the concatenation of the observation feature, the local-map feature, and the waypoints is used as the input for LSTM. The output of LSTM layer passes through a fully connected layer, followed by the policy output layer or the value output layer. Note that the local-map feature is not used in the actor and LSTM of the Q network [26] has an additional action input. For the policy output, we used Gaussian policy with tanh squashing function [26].

We also implemented the CNN based memoryless models (similar to [4] and [10]) with FOVs of (90°, 120°, 180°) for comparative experiments (Figure 2, Right). For the CNN models, velocity vectors ($\mathbb{R}^2$) are tiled to match the shape of depth scan vectors ($\mathbb{R}^{d_{scan} \times 2}$) where d_{scan} is the size of the depth scan vector. Then, this tiled vector is concatenated to the depth scan vector ($\mathbb{R}^{d_{scan} \times 1}$), resulting in a matrix that has a size of $\mathbb{R}^{d_{scan} \times 3}$. We stacked these matrices from 3 recent timesteps to obtain a network input tensor ($\mathbb{R}^{d_{scan} \times 3 \times 3}$). This tensor is passed through 3 convolutional layers and is flattened to obtain an observation feature. The observation feature is then concatenated to the waypoints and passes through 2 fully connected layers, followed by the output layer.

D. SUNCG 2D Simulator and Dynamics Randomization

1) SUNCG 2D environment: A 2D multi-agent navigation simulator is implemented for our experiments (Figure 3). 1,000 random floor plans from the SUNCG [29] dataset are extracted. Then 75 maps are handpicked as our learning environments.

2) Training scenarios: In each training episode, a random environment among the 75 maps in the dataset is sampled. In early experiments, we found that avoiding moving obstacles is more difficult than avoiding static obstacles. Therefore, a small empty map that has only moving obstacles (Figure 3, Right) is picked with probability 0.33 to strengthen the ability to avoid moving obstacles. Once the map is chosen, up to 20 agents are placed in random positions and random goal positions are assigned to the agents. Then, the environment is represented as grid of ($1m \times 1m$) cells and waypoints for each agent are extracted using dijkstra algorithm [30]. For each agent, the episode ends when it collides to an obstacle or 1,000 timestep passes. When an agent reaches its goal, a new random goal and waypoints are assigned to the agent.

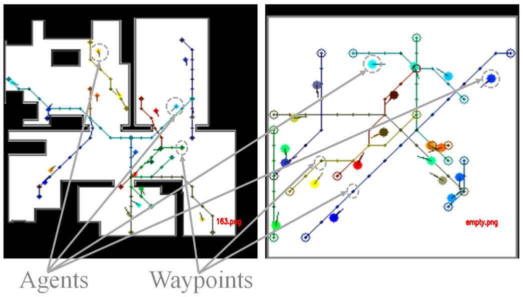

Fig. 3. SUNCG 2D simulator. Black areas are obstacles,colored circles represent agents (robots), and colored lines are plans from the global planner. We started episode in the empty map with probability 0.33 (Right).

3) Dynamics randomization: Dynamics and observations in the real world are different from those in simulators. Also, real-world dynamics and observations are highly noisy. These difference and noise make agents that are trained in simulators only often fail to perform well in the real world. To overcome this problem, recent studies randomized the observation and the dynamics of simulators to improve the robustness of the learned policy [19]–[21].

We found that mobile robot navigation also benefits from this randomization technique. We apply following randomization in our simulator. Note that we resample the noises in every timestep since the real world noises that mobile robots may encounter are usually not consistent within an episode.

- Scan noise : Real world scan data are often noisier than those from the simulator and depth images are known to even noisier than lidar data. Therefore, we add $\mathcal{N}(0, 0.1)$ meters to every depth scan values. The same randomization was also applied in [5].
- Velocity randomization : In the real world, the robot does not move with the same velocity as the input because of wheel drift, an error of motor controller, frictions, ETC. To cope with this, we multiply the input velocity by $\mathcal{N}(1, 0.1)$ before it is applied to the robot. Additionally, since the real world motors cannot change the velocity instantly, we set the velocity of the agent at timestep t to $\overline{v_t} = 0.8v_t + 0.2\overline{v_{t-1}}$ where v_t is the command from the agent multiplied by the noise and $\overline{v_t}$ is the actual velocity that is applied to the robot.
- Timescale randomization : In the simulator, we set 1 timestep to 0.15 seconds. However, in the real world hardware, we cannot expect an exact control frequency. This may be harmful to the mobile robot navigation because the timescale noise causes a wrong estimation of the moving objects' dynamics including the robot itself. To overcome this, we add $\mathcal{N}(0, 0.05)$ seconds to every timestep in the simulator.

We hypothesize that observation and dynamics noises of the real world affect the LSTM-LMC agent more heavily than the CNN agents because the LSTM-LMC agent considers a longer history so that errors from noises are accumulated. We will discuss the effect of the randomization in detail in Experiments section.

IV. EXPERIMENTS

We trained 5 types of agents (CNN agents with FOVs of 90°, 120°, and 180°, LSTM agent with a FOV of 90°, and LSTM-LMC agent with a FOV of 90°) with the hyper parameters listed in Table I.

TABLE I
HYPER PARAMETERS

Hyper parameter	Value
α in Equation 2	0.02
Mini-batch size	256
Trajectory sample length	200
Size of experience replay	5,000,000
Training iteration	3,000,000
Discount factor γ	0.99
Learning rate	3e-4
LSTM unroll	20
Target network update ratio	0.01
Activation functions	LeakyReLU [31]

We trained each agent for 3 million environment steps.

A. Performances

We evaluated the trained agents in 100 evaluation episodes. We fixed the random seeds for the evaluation sessions so all the agents were evaluated in the same maps with the same starting positions and initial goal positions. The results of evaluations are summarized in Table II.

As seen in Table II, the performances of the CNN (memoryless) agents rapidly dropped as FOV decreased.

On the other hand, proposed LSTM-LMC agent with a FOV of 90° outperformed all the other agents, even the CNN agent with a FOV of 180°, in terms of the number of passed waypoints/goals.

The LSTM agent outperformed the 120° CNN agent but failed to outperform the 180° agent. However, it showed the highest survival rate until episode ends.

B. Analysis

We hypothesize that the proposed method shows a superior performance over other methods because it builds robust and accurate model of the surrounding environment and dynamics of other agents implicitly. We analyze the behavior of the trained agents in the following controlled scenarios to verify our hypothesis. For the visual materials of the analysis experiments, see the supplementary video.

1) Blocked path scenario: To identify whether the proposed agent memorizes the structure of the environment, we designed the 'Blocked path scenario' (Figure 4, Top). In this scenario, path from the global planner is blocked by a wall. The wall has a randomly positioned slit on the top side or the bottom side so that the agent has to explore to find which side is open while remembering that the original path is blocked. For 50 episodes, our agent achieved the highest success rate (Table III). Qualitatively, our agent efficiently explored both side of the wall and did not return to the original path when the blocked original path went outside of its FOV. On the other hand, the CNN agents tried to return to their original path as soon as the blocked original path went outside of their FOVs. The LSTM agent was able to pass the blocked path but could not outperform the best CNN agent.

TABLE II
PERFORMANCE OF AGENTS WITH VARIOUS FOV AND ARCHITECTURES

Architecture	CNN	CNN	CNN	LSTM	LSTM-LMC (proposed)
FOV	90°	120°	180°	90°	90°
Mean number of passed waypoints	29.73	40.62	51.45	42.96	**53.64**
Mean number of passed goals	1.42	2.22	3.47	3.06	**3.63**
Survival rate until episode ends	13.53%	19.90%	27.67%	**35.80%**	26.90%

TABLE III
RESULTS OF EXPERIMENTS IN ANALYSIS SCENARIOS

		CNN FOV 90°	CNN FOV 120°	CNN FOV 180°	LSTM FOV 90°	LSTM-LMC FOV 90°
Blocked path	Success rate	0%	16%	82%	78%	92%
Crossing	Success rate	78%	92%	96%	100%	100%
Passing	Success rate	100%	98%	100%	100%	100%

2) Crossing & passing scenario: To see the effect of memory and Local-Map Critic in modeling the dynamics of moving obstacles, we conducted 'Crossing' (Figure 4, Middle), and 'Passing' (Figure 4, Bottom) experiments. Two agents pursue the orthogonal paths (the blue agent was positioned randomly on the top side or the bottom side) in Crossing scenario and the agents follow the same paths but in opposite directions in Passing scenario. The agents have to model the future path of the other agent to break the symmetry in both scenarios. We conducted each scenario 50 times for each agent and the results are summarized in Table III. LSTM-LMC and LSTM agents achieved the highest success rate in Crossing scenario and all the agents performed well in Passing scenario in terms of success rate. However, in qualitatively, the CNN agents often failed to break the symmetry in both scenarios (Figure 4). In contrast, our method showed stable symmetry breaking in all the episodes.

C. Hardware experiments

To verify the performance of our method in the real world, we conducted hardware experiments.

1) Hardware setup: We built a mobile platform with 4 wheels (Figure 1). The platform has NVIDIA Jetson TX-2 as its main processor and is equipped with one Intel Realsense

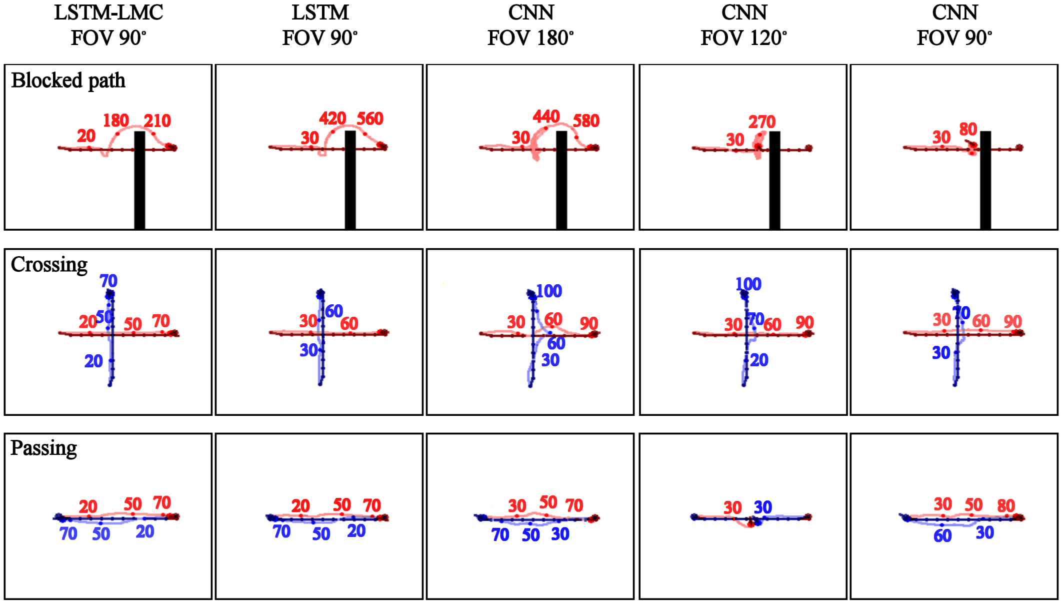

Fig. 4. Analysis scenarios. Blocked path (Top), Crossing (Middle), Passing (Bottom). Red or blue circles depict the agents. Dark lines are paths from global planner, bright lines are trajectories of the agents, and numbers are timesteps. Our method shows superior performance in bypassing wall and symmetry breaking between agents.

TABLE IV
Real world performance with/without randomization

	CNN, FOV 90° No randomization			CNN, FOV 90° Randomization			LSTM-LMC, FOV 90° No randomization			LSTM-LMC, FOV 90° Randomization		
Trial	1	2	3	1	2	3	1	2	3	1	2	3
Passed Waypoints	20	51	55 (all)	24	23	55 (all)	55 (all)	44	24	55 (all)	55 (all)	51
Elapsed Time	-	-	63.969	-	-	59.711	64.093	-	-	49.493	50.391	-

D435 RGB-D camera which has a FOV of 90°. In our experiments, Apriltag [32] and wheel odometry were used for localization. However, other methods such as GPS, ultra-wideband [33], or visual localization [34] may be used.

2) Effects of dynamics randomization in simulator: We deployed the CNN agents and the LSTM-LMC agents with and without randomized training in a real world indoor environment. The environment is quite challenging for the agents with a limited FOV since it has narrow corridors, many curves, and complex obstacles such as stairs or thin pillars. Also, noisy localization obstructs stable navigation. We conducted 3 experiments for each agent and the results are summarized in Table IV. Both of the CNN agents with and without randomization performed poorly, colliding to obstacles in the early stage of episodes. Also, the CNN agents did not show meaningful benefits from dynamics randomization. As we expected, the LSTM-LMC agent without dynamics randomization suffered more from the real world noises: it showed unstable movement, leading to collision or slow navigation. The LSTM-LMC agent with dynamics randomization was the only agent that showed stable performance. See the supplementary video for more details.

3) Navigation in a crowded real world environment: To see overall performance of our method in the real world, we deployed the LSTM-LMC agent with dynamics randomization in a crowded environment. The robot repeated a straight route of $7m$, and 2 participants crossed, passed, or interrupted the robot's path. The robot was able to finish 12 successive routes (roughly $84m$), even under the abusive situations. See the supplementary video for more details.

V. Conclusion

In this paper, we explored the possibility of replacing expensive lidar devices with affordable depth cameras, which have a limited FOV, in DRL navigation methods. We showed that a limited FOV severely degrades the performance of an agent. We also proposed novel LSTM-LMC architecture and dynamics randomization technique. Our method outperforms memoryless agents with a twice larger FOV and shows robust performance in the real world with a single depth camera. Future works will consider more sophisticated concepts such as learning to integrate information from multiple sensors [35], or learning implicit localization [36].

REFERENCES

[1] F. A. Cosío and M. P. Castañeda, "Autonomous robot navigation using adaptive potential fields," *Mathematical and computer modelling*, vol. 40, no. 9-10, pp. 1141–1156, 2004.

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

[3] S. Campbell, W. Naeem, and G. W. Irwin, "A review on improving the autonomy of unmanned surface vehicles through intelligent collision avoidance manoeuvres," *Annual Reviews in Control*, vol. 36, no. 2, pp. 267–283, 2012.

[4] T. Fan, X. Cheng, J. Pan, D. Monacha, and R. Yang, "Crowdmove: Autonomous mapless navigation in crowded scenarios," *arXiv preprint arXiv:1807.07870*, 2018.

[5] A. Faust, O. Ramirez, M. Fiser, K. Oslund, A. Francis, J. Davidson, and L. Tapia, "Prm-rl: Long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning," *arXiv preprint arXiv:1710.03937*, 2017.

[6] 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 *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, IEEE, 2017, pp. 3357–3364.

[7] Y. F. Chen, M. Everett, M. Liu, and J. P. How, "Socially aware motion planning with deep reinforcement learning," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, IEEE, 2017, pp. 1343–1350.

[8] Y. Wu, Y. Wu, G. Gkioxari, and Y. Tian, "Building generalizable agents with a realistic and rich 3d environment," *arXiv preprint arXiv:1801.02209*, 2018.

[9] T. M. Howard, C. J. Green, A. Kelly, and D. Ferguson, "State space sampling of feasible motions for high-performance mobile robot navigation in complex environments," *J. Field Robotics*, vol. 25, pp. 325–345, 2008.

[10] P. Long, T. Fan, X. Liao, W. Liu, H. Zhang, and J. Pan, "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning," *arXiv preprint arXiv:1709.10082*, 2017.

[11] K. Hauser, T. Bretl, J.-C. Latombe, and B. Wilcox, "Motion planning for a sixlegged lunar robot," in *The Seventh International Workshop on the Algorithmic Foundations of Robotics*, Citeseer, 2006.

[12] S. Sukhbaatar, R. Fergus, *et al.*, "Learning multiagent communication with backpropagation," in *Advances in Neural Information Processing Systems*, 2016, pp. 2244–2252.

[13] J. Choi, B.-J. Lee, and B.-T. Zhang, "Multi-focus attention network for efficient deep reinforcement learning," *arXiv preprint arXiv:1712.04603*, 2017.

[14] J. Foerster, I. A. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," in *Advances in Neural Information Processing Systems*, 2016, pp. 2137–2145.

[15] R. Lowe, Y. Wu, A. Tamar, J. Harb, O. P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments," in *Advances in Neural Information Processing Systems*, 2017, pp. 6379–6390.

[16] 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.

[17] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International Conference on Machine Learning*, 2016, pp. 1928–1937.

[18] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, p. 484, 2016.

[19] J. Tan, T. Zhang, E. Coumans, A. Iscen, Y. Bai, D. Hafner, S. Bohez, and V. Vanhoucke, "Sim-to-real: Learning agile locomotion for quadruped robots," *arXiv preprint arXiv:1804.10332*, 2018.

[20] X. B. Peng, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Sim-to-real transfer of robotic control with dynamics randomization," *arXiv preprint arXiv:1710.06537*, 2017.

[21] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *Intelligent Robots and Systems (IROS), 2017 IEEE/RSJ International Conference on*, IEEE, 2017, pp. 23–30.

[22] M. T. Spaan, "Partially observable markov decision processes," in *Reinforcement Learning*, Springer, 2012, pp. 387–414.

[23] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*. MIT press Cambridge, 1998, vol. 135.

[24] 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.

[25] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[26] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep

reinforcement learning with a stochastic actor," *arXiv preprint arXiv:1801.01290*, 2018.

[27] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[28] L.-J. Lin, "Reinforcement learning for robots using neural networks," Carnegie-Mellon Univ Pittsburgh PA School of Computer Science, Tech. Rep., 1993.

[29] S. Song, F. Yu, A. Zeng, A. X. Chang, M. Savva, and T. Funkhouser, "Semantic scene completion from a single depth image," in *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, IEEE, 2017, pp. 190–198.

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

[31] B. Xu, N. Wang, T. Chen, and M. Li, "Empirical evaluation of rectified activations in convolutional network," *arXiv preprint arXiv:1505.00853*, 2015.

[32] J. Wang and E. Olson, "Apriltag 2: Efficient and robust fiducial detection.," in *IROS*, 2016, pp. 4193–4198.

[33] R. Ye, S. Redfield, and H. Liu, "High-precision indoor uwb localization: Technical challenges and method," in *Ultra-Wideband (ICUWB), 2010 IEEE International Conference on*, IEEE, vol. 2, 2010, pp. 1–4.

[34] S. Brahmbhatt, J. Gu, K. Kim, J. Hays, and J. Kautz, "Geometry-aware learning of maps for camera localization," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2616–2625.

[35] G.-H. Liu, A. Siravuru, S. Prabhakar, M. Veloso, and G. Kantor, "Learning end-to-end multimodal sensor policies for autonomous navigation," *arXiv preprint arXiv:1705.10422*, 2017.

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