CNN-LSTM-based Deep Recurrent Q-Learning for Robotic Gas Source Localization

Iliya Kulbaka, Ayan Dutta, Ladislau Bölöni, O. Patrick Kreidl, Swapnoneel Roy

***Abstract*—Locating the source of harmful, flammable, or polluting gas leaks is an important task in many practical scenarios. A recently proposed localization approach is to use a mobile robot equipped with a chemical sensor. The localization algorithm guides the movement of the robot based on the previous observations, with the objective of reaching the source as quickly as possible. In this paper, we propose an approach where the robot policy is represented by a neural network combining convolutional and LSTM layers. The approach relies on a gas dispersion model that takes into account obstacles, wind direction, and molecular movement. We found that the trained model provides a 47.34% higher success rate in finding the gas source than an existing greedy approach on test cases with unseen gas plumes and random obstacles.**

***Index Terms*—Deep Q-Learning, LSTM, CNN, Gas source localization, Mobile robot**

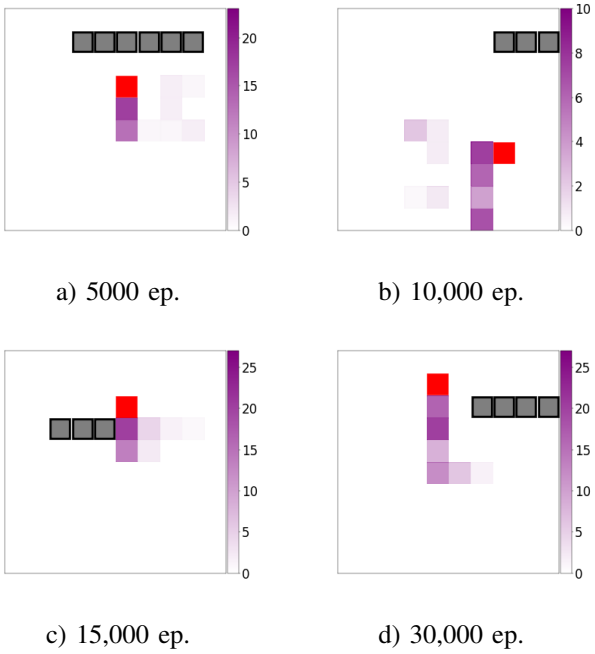
I. Introduction

A mobile robot with a gas measurement sensor can measure gas concentrations at different locations in an unknown environment. The gas might be leaking from a bomb, drugs, or from a sub-sea pipeline [1]–[3]. It might be hazardous to send a human to find the gas source. Therefore, using an autonomous mobile robot that can find the source of the gas leak would be a safer option. Most of the robotic gas source localization studies have relied on unmanned ground vehicles (UGVs) [4], [5]. An array of chemical sensor-enabled E-noses are often placed on these ground robots, which can be used for both indoor and outdoor applications. On the other hand, a few studies have also used unmanned aerial vehicles (UAVs), aka drones (both fixed and rotary-wing), for this application [6].

In this paper, we study the gas source localization problem from a computational perspective. We propose a learning algorithm that relies on the gas concentration measurements by the robot's onboard array of sensors, which makes our presented solution robot-model agnostic. Most of the existing GSL studies rely on environments with no obstacles [4], [5]. Unlike these, the proposed solution in this paper can gracefully handle static obstacles such as walls that might affect the gas flow. To this end, we propose a deep Q-learning technique that takes the state of the environment as input and helps the robot to take the best decision, i.e., moving towards the source with minimum distance covered. Our proposed neural network architecture uses three convolutional layers and two stacked recurrent layers (namely, LSTM [7]) – the CNN layers are responsible for extracting meaningful features from the spatial state information, e.g., measured gas concentrations in the environments so far along with the corresponding visited locations by the robot while the LSTM layers are used to recognize patterns in the robot's trajectory and gas measurements across multiple timesteps. We use an existing gas generation model from [1], which takes the wind and the obstacles in the environment into account.

This work is supported in part by NSF CPS Grants #1932300 and #1931767.

I. Kulbaka, A. Dutta, O.P. Kreidl, and S. Roy are with the University of North Florida, USA. Emails: {n01427009, a.dutta, patrick.kreidl, s.roy}@unf.edu
L. Bölöni is with the University of Central Florida, USA. Email: {ladislau.boloni}@ucf.edu



a) 5000 ep. b) 10,000 ep.

c) 15,000 ep. d) 30,000 ep.

Fig. 1: Gas source localization with a mobile robot using deep Q-learning. Four test samples are shown after 5-, 10-, 15-, and 30,000 episodes with random walls (black cells) present. The red cell indicates the source. Robot trajectories are shown with their corresponding measured gas concentration intensities (purple). One can observe that after only 5000 episodes (top left), the robot takes a longer path to reach the source, whereas it takes the shortest path to reach the target when tested with a 30,000-episode-trained model.

We have trained the presented model for 30,000 episodes with random walls, random gas sources, and/or random robot start positions. Note that the robot start locations might have zero gas concentration levels. Once the model is trained, we deployed the trained model on 1000 randomly generated

test cases with various obstacles, robot start locations, and/or gas source configurations. Results show that our proposed technique achieves a high success rate in finding the gas sources in the test environments. An illustrative example of a few test cases at various checkpoints is presented in Fig. 1. Furthermore, we have compared our approach against a standard existing gradient-ascent technique [8] and also a random walk-based solution. Our learning algorithm has been shown to outperform both of these baselines in terms of the success rate in finding the gas source and the number of steps needed to find the source.

II. Related Work

Gas source localization (GSL) with an autonomous mobile robot is a practical problem as it can help save humans from hazardous exposures. A state-of-the-art review on robotic GSL can be found in [4]. From the algorithmic point of view, some of the earliest yet still relevant algorithms used chemical gradients to find the source [8]. Algorithms for both indoor and outdoor applications have been developed in the literature. In indoor applications, primarily low-cost, small UGVs have been used [9]. Similar to our setting, where the robot (e.g., UAV) can move on a 2D plane in four directions, have been studied in [10]. The authors have proposed an informative path planning approach to solve the GSL problem. Vision sensors have also been used along with chemical sensing for the studied problem [11]. In [5], the authors have used wind and gas concentration information for their GSL solution. Along a similar line, the authors in [12] have proposed a strategy for such a search while taking the mean airflow over a period of time into account. In our paper, we imitate this from a machine learning perspective where the history of wind information and robot trajectories are memorized and leveraged using a couple of stacked LSTM layers. In robotics, LSTM has been used extensively for deep Q-learning applications [13]. Recently, there have been GSL-related studies that used deep learning-based approaches. One of the closest studies to us is due to [1]. We model our gas distribution following their work. However, as suggested in some of the prior work, we incorporate the 'memory' of the sensor measurements unlike [1]. In [14], the authors have proposed a deep Q-learning model that can find multiple gas sources with a mobile robot. The authors achieved this by incorporating domain knowledge into their model. Unlike all of the relevant studies mentioned here, we utilize a combination of spatially as well as temporally deep machine learning architecture that learns from the prior gas concentrations, wind directions, and the robot's path.

III. Model

A. Robot and Environment Model

The environment P is divided into grid cells, C. The source of the gas is in cell $c_s \in C$. We have a mobile robot r (e.g., an unmanned aerial robot) that is equipped with a gas sensing sensor (e.g., CO2Meter's S8 sensor for CO_2 measurement or an E-nose sensor). With the onboard sensor, r can measure the gas concentration in its current cell $c \in C$. Once the gas concentration in the current cell is measured, the robot can move to any of its four neighbor cells unless the cell is blocked by an obstacle (e.g., a wall). Therefore, r has four actions (i.e., up, down, left, right) in its action set $\mathcal{A}$. Once the robot reaches c_s, it stops.

B. POMDP Model

We model the GSL problem as a partially observable Markov Decision Process (POMDP) following [15]. The robot cannot sense the gas concentrations in all the cells of the environment. Therefore, the robot can not observe the global dynamics of the gas dispersion and wind. A POMDP is defined as a tuple $\langle S, A, T, O, R, \Omega \rangle$, where $S, A, T,$ and R have usual meanings from MDP, i.e., state space, set of actions, state-transition function, and reward respectively. Ω denotes the set of observations. Finally, O represents the probability that the observed state is sampled from the state space. The reward function R is designed as follows. For any terminal action, if it leads to success or failure, the robot gets a reward of 20 or -10 respectively. For non-terminal actions, the robot gets -0.5 for being 'alive' as formally defined below.

$$R(a) = \begin{cases} 20, & \text{success} \\ -10, & \text{failure} \\ -0.5, & \text{default} \end{cases} \quad (1)$$

From here, Q-learning estimates how good an action is in a given state $s \in S$. The recursive Q-value update function follows from the Bellman update as the following.

$$Q(s,a) = Q(s,a) + \alpha(R + \gamma \max_{a'} Q(s',a') - Q(s,a)) \quad (2)$$

where s' is the next state to which the robot transitions from s. However, maintaining and updating Q-values for all possible $S \times A$ values might be prohibitive. Instead, the values can be approximated by a deep neural network following [16]. A deep network such as designed by ours is maintained. Let us parameterize it by θ (called a policy network). The network will output the approximated Q-values for all available actions (maximum 4 in our setting) for a given state. The parameter set θ is tuned with experiences that the robot goes through over time by using a loss function (Eq. 3). The experiences (i.e., the transition tuples) are stored in an experience buffer $\mathcal{D}$. In every training iteration t, a batch of these is randomly selected and the loss is calculated.

$$\mathcal{L}_t(s,a,\theta_t) = \mathbb{E}[(y - Q(s,a,\theta_t))^2], \quad y = R(a) + \gamma \max_{a'} Q(s',a',\theta_t^-) \quad (3)$$

y is calculated from a deep network that is a copy of the original network and is called the target network. The target network is parameterized by θ^- and only gets updated (i.e., $\theta^- \leftarrow \theta$) every few episodes for stability.

C. Gas Generation Model

The gas plume generation model follows from [1] and our experimentation is based on these generated gas types. The equations $4-8$ formally represent the gas generation process.

Equations 4 and 5 are responsible for gas concentration coming from x and y directions respectively. Variables A_w and A_d in Eq. 4 represent the plume width and diffusion parameters. On the other hand, variable A_c in Eq. 5 represents the gas particle decay parameter. Variable A_r in Eq. 6 is a random walk parameter responsible for the probability that the measured block is going to have more or less concentration due to the random movement of the gas. λ in Eq.7 is the general concentration measurement before noise is applied for randomization. During this stage, an obstacle may block the flow of the gas – then, only the r_m parameter of the gas distribution will be registered. A_s is the gas release rate from the source. A_b indicator represents the state of the airflow – if an obstacle blocks the airflow, it is set to 0; otherwise 1. A_o is set to 1 if r's current coordinate is in alignment with the obstacle such that the gas would travel around it, otherwise it is set to 0. C is the probability of hits for gas generated by λ and is a sample from a Poisson distribution. The values of these parameters in our experiments are kept the same as [1].

$$r_y = exp\left(-2\left(\frac{y}{A_w + A_d \cdot x}\right)^2\right) \tag{4}$$

$$r_x = \begin{cases} 1 - A_c \cdot x^2 & \text{if } 0 \leq A_c^{-\frac{1}{2}} \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

$$r_m = \frac{1}{A_r(x^2 + y^2 + 1)} \tag{6}$$

$$\lambda = A_s \left(r_x r_y \cdot A_b + \frac{r_x}{2} \cdot A_o + r_m\right) \tag{7}$$

$$C \backsim Poisson(\lambda) \tag{8}$$

IV. Proposed Methodology

A. Neural Network Architecture

The neural network architecture proposed in this paper consists of a series of three convolutional (CNN) layers and then a stacked layer of LSTMs for decision-making. The input to the CNN is $4 \times L \times W$ tensor, where L and W are the length and the width of the environment. The four layers in this tensor together represent a state $s \in S$ and these are shown in Fig. 3. The state input remains identical to the one proposed in [1], with robot trajectory, concentration measured, wind direction, and the map of the environment. These inputs are passed through three convolutional layers and rectified linear unit (ReLu) activation functions. All three convolutional layers have 16 output channels. The filter size differs between the first CNN layer and the following, being 5 and 3 respectively with stride being 1 for all convolutional layers. The output (of size $1 \times trace \times 100$) of the last convolutional layer is passed into two stacked LSTM layers. The hidden cell size of the LSTM layers is set to 128. The output from the LSTM layers is then fed to a fully-connected linear layer with the output size of $|A|$. These outputs are the Q-values of the $|A|$ actions that the robot can take. The proposed architecture is shown in Fig. 2.

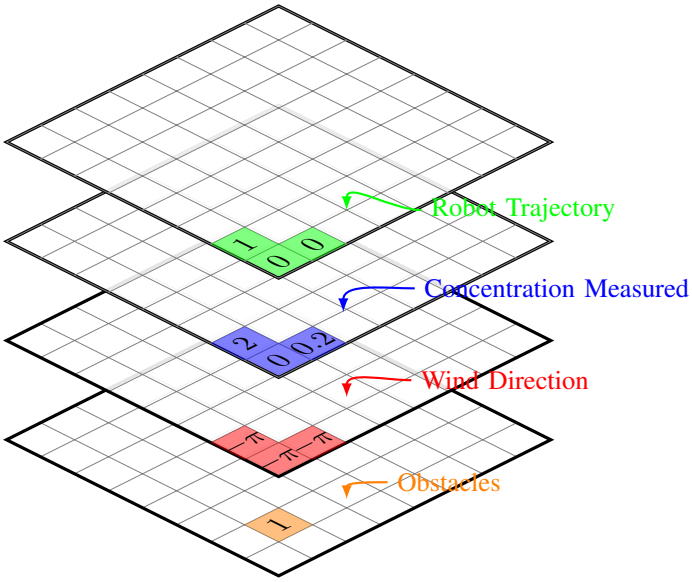


Fig. 3: An illustration of the state input. For brevity, an 8×8 grid environment ($L = W = 8$) is used. This tensor is passed to the neural network in Fig. 2. The 1 and the 0 in the trajectory layer represent the current and the past robot locations. The values in the measured concentration matrix are the gas readings collected by the robot. The obstacle layer contains binary values that represent obstacle/no-obstacle cells in the environment.

B. Training

The overall training loop is similar to Double DQN [17] with the exception of network optimization and state tensor stacking. The pseudocode of our algorithm is presented in 1.

As mentioned earlier, we maintain two copies of the same network as the policy network (Q_θ) and the target network for the purpose of stabilizing the learning process. At the beginning of every episode, the environment is reset in one of three pre-defined configurations. During each episode, r selects an action using the ϵ-greedy strategy, i.e., depending on the current epsilon value robot selects a random action with $(1-\epsilon)$ probability or the best action calculated based on the Q-values outputted by the policy network with ϵ probability.

To promote further exploration in the beginning, the first 500 episodes are purely exploratory. During these episodes, the robot selects actions completely randomly (i.e., uniform selection with each action having a probability of 0.25) from the provided set of four actions, A. After taking the action $a \in A$ in state s, the robot transitions to state s' and moves to a new cell c in the grid. Next, the robot r records its

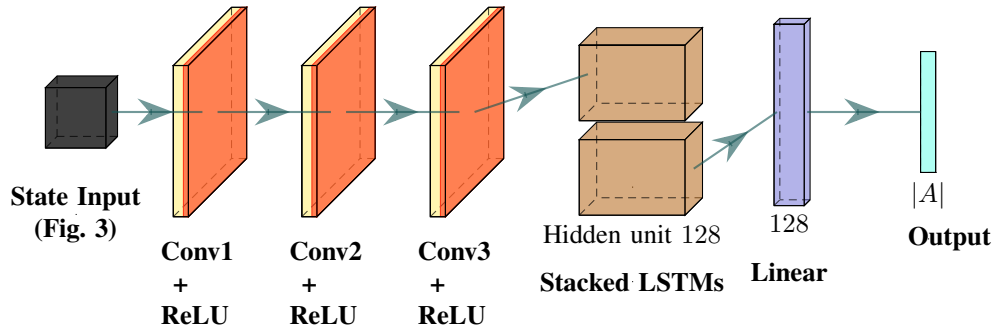


Fig. 2: The proposed CNN-LSTM neural network architecture.

Algorithm 1: CNN-RNN-based robotic gas source localization.

Initialize replay memory $\mathcal{D}$, the policy and target networks Q_θ and $Q_\theta-$ respectively;
while *training not complete* **do**
  Initialize the environment with the robot placed;
  for *episode length* **do**
    $s \leftarrow$ current state (e.g., Fig. 3); Select action $a \in A$ using ϵ-greedy strategy;
    Execute action a, transition to state s', and measure the gas concentration $c(g)$ in the current cell g;
    Check if the source c_s is reached, i.e., set $done$ to $True$ if c_s reached, otherwise $False$;
    Receive reward R;
    Store transition information $\langle s, a, R, s', d, done \rangle$ in $\mathcal{D}$;
    if $done = True$ **then**
      break;
    Sample minibatch of size $m \cdot trace$ experiences from $\mathcal{D}$;
    Calculate the target value y and regress the Q-network toward it;
  Update the target network parameters θ^- every T episode.

trajectory and wind direction in cell c. After which it measures gas concentration $\mathcal{C}(c)$ in that cell and checks if it is located within one block (up, down, left, right) of the gas source c_s. Based on this check, r calculates the reward following Eq. 1. The reward is then added to the running total of the episode score. This score is maintained for tracking the performance of the learning model.

Training the network happens after the robot takes an action, transitions to a new state, makes an observation, and receives a reward for it. We randomly select m transition batches of length $trace$ from $\mathcal{D}$ for training. Note that each such batch is a set of $trace$ number of consecutive experiences from a single episode. We iterate over m batches while parsing each of the $trace$ transitions in it as their own batch and feed them to the network. The target Q-values, y, are calculated using Eq. 3, and the loss for the t-th training iteration is calculated. The policy network parameters are then regressed to minimize the loss value using the Adam optimizer method.

C. Testing

The testing phase consists of the robot conducting 1000 episodes of randomly generated environments in one of three pre-defined different configurations, the configuration is the same as the one that was used for training of the currently tested neural network. During these 1000 episodes, the actions are selected purely by the trained neural network. The testing stage also removed the network training component as it has been trained prior to this.

V. Experiments and Results

A. Settings

The proposed technique is implemented in Python, with the use of the PyTorch library. The main parameters used in the experiments are listed in Table I. The environment size is 10×10 m.^{2}. We have three environment configurations that were used for training and testing: 1) *All random*: robot location, gas source, and the obstacles (walls) in the environment are randomly generated every episode (see Fig. 1), 2) *No walls*: there is no obstacle in the environment and the rest of the setting is the same as 'all random', and 3) *No walls set robot position*: the robot start position is fixed (cell $(3, 3)$) across training and testing but the gas source is randomized.

We have compared our proposed algorithm against two baselines: 1. *Greedy.* This baseline is similar to the algorithm proposed in [8]. At every step, the robot moves to the neighbor cell with the highest concentration. Due to the nature of the gas generation algorithm, at times the greedy algorithm gets stuck in the loop of going back and forth as the generated concentration can be zero. 2. *Random.* This one lets the robot choose a random action at every step. These baselines do not require training, thus only our test results are compared with their performance. The test data is based on 1000 test episodes.

TABLE I: List of parameters used in our experiments.

Parameters	Values
State	$4 \times 10 \times 10$ tensor
Action	Up, Down, Left, Right
Number of training episodes	$30,000$
Episode length	100
Priority replay memory size	$20,000$
Mini-batch size	32
Sample length	20
Discount factor	0.90
Learning rate	0.0001
Target network update frequency	20
Epsilon decay type	Linear
Epsilon decay rate	0.0002
Epsilon start value	1.0
Epsilon end value	0.001
Loss function	Mean Square Error
Optimizer	Adam
Number of testing episodes	$1,000$

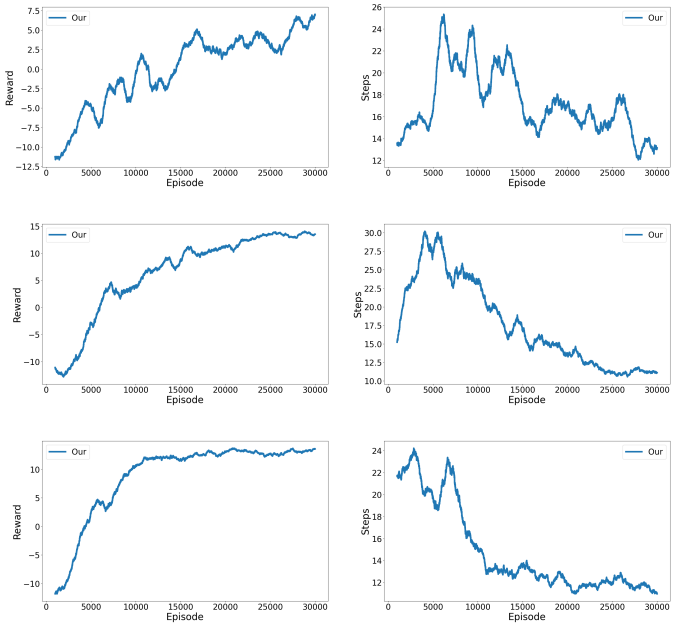


Fig. 4: Training plots (1000-episode rolling average) for reward and steps. Top: All random, Middle: No walls, and Bottom row: No walls set robot position configuration.

B. Results

We are interested in investigating three main metrics to evaluate the performance of the neural network algorithm: reward (a higher reward indicates a shorter path found), steps (the lower value indicates a more efficient trajectory), and success rate (success % in locating the gas source). The training results are presented in Fig. 4. As can be seen, the proposed technique rapidly increases its efficiency with respect to the episodic reward, which steadily goes up before converging and the number of steps gradually decreases before it almost converges. At around episode 5500, ϵ value of 0.001, resulting in all of the actions taken past episode 5500 to primary driven by the neural network proposed. The testing results are presented in Fig. 5, and show improvement provided by our proposed algorithm over the baselines in the three main metrics. The data presented in Tables II, III, and IV show averages for testing performance in three testing environments along with standard deviation. Individually the results can be misleading, for example in Table III, the random baseline and Our proposed algorithm have similar average steps per environment configuration, however, this alone does not draw a meaningful conclusion. When combined with the reward and success rate tables (Tables II and IV), we can understand that the random movements might not have led the robot to the gas source. Overall, across all configurations, our proposed algorithm achieves a success rate of 91.67% in finding the gas source, whereas the random and the greedy baselines yield 22.33% and 44.33% respectively.

TABLE II: Average reward gained during 1000-episode testing on three different configurations

	All Random	No Walls	Set Robot Position
Random	-12 ± 13.31	-12 ± 14	-13 ± 18.57
Greedy	-30 ± 34.48	-28 ± 34.76	-29 ± 35.02
Ours	$\mathbf{6 \pm 16.90}$	$\mathbf{14 \pm 8.40}$	$\mathbf{13 \pm 9.71}$

TABLE III: Average steps during 1000-episode testing on three different configurations

	All Random	No Walls	Set Robot Position
Random	14 ± 16.82	15 ± 17.64	23 ± 20.65
Greedy	64 ± 42.49	63 ± 42.68	64 ± 42.48
Ours	$\mathbf{13 \pm 14.38}$	$\mathbf{14 \pm 8.40}$	$\mathbf{12 \pm 11.72}$

TABLE IV: Average success rate (%) during 1000-episode testing on three different configurations

	All Random	No Walls	Set Robot Position
Random	16	20	31
Greedy	43	46	44
Ours	**78**	**99**	**98**

VI. Conclusion and Future Work

In this paper, we study the problem of localization of a gas source with a mobile robot. The robot is equipped with gas and wind sensors, e.g., an E-nose, to measure gas concentration, wind speed, and direction. We propose a deep recurrent Q-learning technique that learns a policy from 30,000 episodes of training. The employed reward function gives the robot a positive reward if a training episode terminates successfully in finding the gas source, a large penalty if it is an unsuccessful training episode, and a small negative reward for any other non-terminating state. We have tested our proposed learning mechanism on environments with random obstacles, random robot starts locations, and/or random gas sources. Results show that our proposed approach achieves a high success rate in finding the gas source in all types of environments while outperforming a gradient ascent and a random-walk baseline strategy. In the future, we will investigate more efficient learning approaches. Furthermore, we are interested

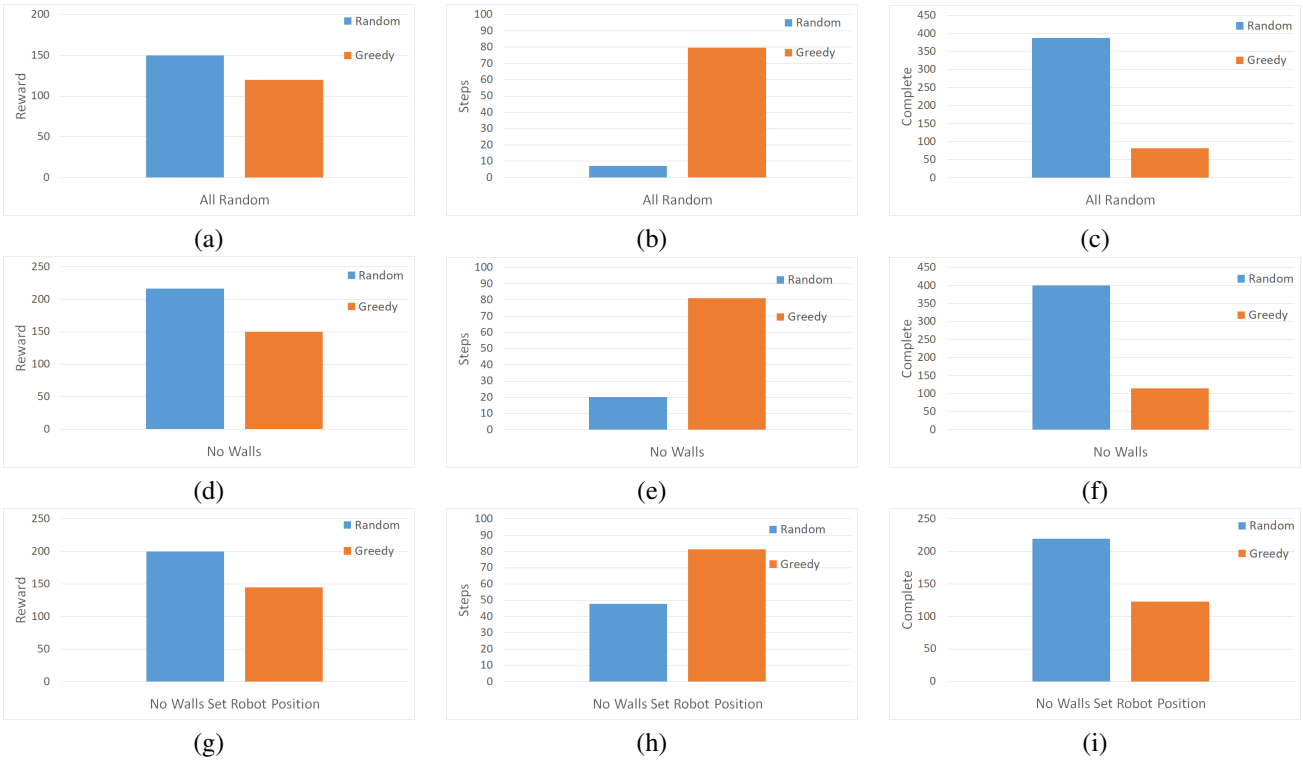


Fig. 5: Improvement of performance metrics (%) by our model on the baselines.

in employing our proposed learning technique on Gaussian plume models.

REFERENCES

[1] X. Chen, C. Fu, and J. Huang, "A deep Q-network for robotic odor/gas source localization: Modeling, measurement and comparative study," *Measurement*, vol. 183, p. 109725, 2021.

[2] M. Mohan, T. Panwar, and M. Singh, "Development of dense gas dispersion model for emergency preparedness," *Atmospheric Environment*, vol. 29, no. 16, pp. 2075–2087, 1995.

[3] C. Liu, Y. Liao, S. Wang, and Y. Li, "Quantifying leakage and dispersion behaviors for sub-sea natural gas pipelines," *Ocean Engineering*, vol. 216, p. 108107, 2020.

[4] A. Francis, S. Li, C. Griffiths, and J. Sienz, "Gas source localization and mapping with mobile robots: A review," *Journal of Field Robotics*, vol. 39, no. 8, pp. 1341–1373, 2022.

[5] P. Ojeda, J. Monroy, and J. Gonzalez-Jimenez, "Information-driven gas source localization exploiting gas and wind local measurements for autonomous mobile robots," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 1320–1326, 2021.

[6] P. P. Neumann, V. Hernandez Bennetts, A. J. Lilienthal, M. Bartholmai, and J. H. Schiller, "Gas source localization with a micro-drone using bio-inspired and particle filter-based algorithms," *Advanced Robotics*, vol. 27, no. 9, pp. 725–738, 2013.

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

[8] R. Rozas, J. Morales, and D. Vega, "Artificial smell detection for robotic navigation," in *Proc. of Fifth Int. Conf. on Advanced Robotics*, pp. 1730–1733, 1991.

[9] M. Hutchinson, P. Ladosz, C. Liu, and W.-H. Chen, "Experimental assessment of plume mapping using point measurements from unmanned vehicles," in *Proc. of Int. Conf. on Robotics and Automation (ICRA-2019)*, pp. 7720–7726, 2019.

[10] F. Rahbar, A. Marjovi, and A. Martinoli, "An algorithm for odor source localization based on source term estimation," in *Proc. of Int. Conf. on Robotics and Automation (ICRA-2019)*, pp. 973–979, 2019.

[11] J. Monroy, J.-R. Ruiz-Sarmiento, F.-A. Moreno, F. Melendez-Fernandez, C. Galindo, and J. Gonzalez-Jimenez, "A semantic-based gas source localization with a mobile robot combining vision and chemical sensing," *Sensors*, vol. 18, no. 12, p. 4174, 2018.

[12] J.-G. Li, M.-L. Cao, and Q.-H. Meng, "Chemical source searching by controlling a wheeled mobile robot to follow an online planned route in outdoor field environments," *Sensors*, vol. 19, no. 2, p. 426, 2019.

[13] J. Orr and A. Dutta, "Multi-agent deep reinforcement learning for multi-robot applications: a survey," *Sensors*, vol. 23, no. 7, p. 3625, 2023.

[14] T. Wiedemann, C. Vlaicu, J. Josifovski, and A. Viseras, "Robotic information gathering with reinforcement learning assisted by domain knowledge: an application to gas source localization," *IEEE Access*, vol. 9, pp. 13159–13172, 2021.

[15] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," in *Proc of the 2015 AAAI Fall Symposium*, 2015.

[16] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[17] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. of AAAI Conf. on Artificial Intelligence*, vol. 30, 2016.