MARS: Multi-Agent Deep Reinforcement Learning for Complex Environment Exploration

MARS: Multi-Agent Deep Reinforcement Learning for Complex Environment Exploration

Francesco Gervino*, Andrea Eirale†, Marcello Chiaberge†, Alessio Sacco*, Guido Marchetto*, Claudio Casetti*
* *Department of Control and Computer Engineering*, Politecnico di Torino, Italy
† *Department of Electronics and Telecommunications*, Politecnico di Torino, Italy

***Abstract*—Autonomous exploration of complex, unknown environments is a cutting-edge task not entirely solved by the scientific community. When an agent needs to explore a maze without any a priori information about the environment, the lack of proper destinations and explicit task objectives make traditional navigation policies inappropriate. While the literature presents some sporadic deterministic systems able to face the tasks, learning approaches still need an adequate investigation which could prove them to be more suitable and versatile for this purpose. In this paper, we present MARS, a path planner that exploits swarms of robots to optimize the exploration of complex unknown environments, such as mazes. To make the solution scalable, the proposed method exploits two cooperating modules: local and global planners. The local planner is modeled as a Markov Decision Process (MDP) and trained as a Reinforcement Learning (RL) multi-agent system. Each agent has access to image representations of a section of the global map, always centered in the robot reference frame, and decides the next navigation goal to complete the local exploration. The global planner is a deterministic system that recovers the navigation when a local solution is unavailable. The robots share the explored section with peers when they meet in a rendez-vous. We compared our approach to a single deterministic agent, a single RL agent and a close-to-optimal deterministic approach which deploys five greedy agents. The simulation results demonstrate MARS' efficiency, reaching near-optimal levels in significantly less time.**

***Index Terms*—multi-agent reinforcement learning, robots swarm, environment exploration, machine learning**

I. Introduction

Deep Reinforcement Learning (DRL) leverages deep neural networks to approximate functions in high-dimensional reinforcement learning tasks, achieving leading performance in numerous research domains [1]. DRL has been explored extensively for developing navigation policies in mobile robots. Traditional robotic navigation approaches typically involve a combination of procedures like Simultaneous Localization and Mapping (SLAM) [2], path planning, and motion control using a predefined map. However, with the advanced representation learning capabilities of deep neural networks, DRL offers a way to learn control policies directly from raw sensory data, eliminating the need for these intermediate steps.

By removing the dependence on localization, mapping, and path planning, DRL-based methods have been developed to learn effective navigation strategies solely from sensor inputs. These approaches include target-driven navigation [3], successor feature reinforcement learning for policy transfer [4], and using auxiliary tasks to improve DRL training efficiency [5]–[8]. Further research has expanded on these ideas, incorporating SLAM-like structures into DRL models and applying DRL for multi-robot collision avoidance [9].

This paper concentrates on mapless exploration, where a multi-agent system must navigate a maze without prior knowledge of the environment's layout. We assume that the agents can easily determine their relative position when they are sufficiently close to each other using cost-effective localization methods such as visible light [10] or WiFi-based localization [11]. When two agents receive corresponding signals, a rendez-vous occurs and the agents merge their global maps sharing information about the environment they are exploring.

In this paper, we propose Multi-Agent Robot Swarm (MARS) that trains an autonomous multi-agent policy able to optimize the exploration of complex unknown environments, such as mazes. The proposed solution exploits two cooperating modules: a local and a global planner. The problem is modeled as a Markov Decision Process (MDP). A Reinforcement Learning (RL) agent, trained by a Double DQN algorithm, solves the planning problem locally. This agent has access to image representations of a maze section, always centered in the robot reference frame, and decides the next navigation goal to complete the local exploration. The global planner uses a deterministic greedy algorithm that recovers the navigation when a local solution is unavailable. The trained network was deployed on every agent of our multi-agent system, which was then compared with a deterministic system of 5 agents. The results obtained demonstrate our method's efficiency, which reaches near-optimal levels in significantly less time, marking a promising achievement since diminishing the computation time is really important, especially for real scenarios.

The main challenges addressed in this paper consist in:

- Define a suitable Markov Decision Process to model the problem, guaranteeing stability and efficiency during the training.
- Develop a generator able to produce fully-explorable, complex environments and a general agent capable of navigating and mapping them.
- Design a suitable policy able to drive autonomous agents to explore the unknown environment efficiently.

We evaluate the effectiveness of MARS against a greedy approach and a single agent using RL to lean. The results show how MARS can achieve a reward, which depends on the agent's path and what it has explored during an action, 25% higher than the greedy agent (on average) and 45%

than the single agent while requiring only 15% of the time in our simulation environments. MARS reaches near-optimal performance if we compare it to a 5 deterministic agents system while demanding 50% of the time.

The rest of the paper is structured as follows. Section II overviews the state-of-the-art of autonomous exploration, while Section III describes the system and agents' decision making. In Section IV we report the learning method and in Section V we detail the final parameters of the training along with the results obtained comparing a deterministic method with our trained agents. Finally, Section VI concludes the paper and proposes possible future directions.

II. Related Work

Traditionally, the literature faces the exploration task with deterministic methods. E.g. Yamauchi [12] proposed a widely adopted approach for selecting exploration points. This technique identifies the boundaries between known free space and unexplored areas using occupancy grid maps. After detecting these edges, the method determines the center of each boundary as a potential next exploration point. The robot then utilizes a depth-first search algorithm to navigate toward the nearest point.

Another common traditional approach is rooted in information theory. Bourgault et al. [13] utilized Shannon entropy to assess the uncertainty in both the robot's position and the occupancy grid maps, using this information metric as a basis for robot control. The control strategy was derived from solving a multi-objective problem centered on this utility. The utility function for localization originated from filter-based SLAM algorithms [14]. Numerous studies have extensively discussed the importance of the graph-based SLAM algorithm [15].

More recent works utilize DRL methods as Tai et al. [16] proposed a perception network with RGB-D image inputs and a control network trained using the DQN algorithm. However, it is important to note that this method only guarantees that the robot can navigate without collisions, rather than achieving specific goals.

Complex Environment Exploration is a cutting-edge problem, which has never been really addressed when large environments are involved. Li et al. [17] have successfully trained a robot with a 2D LiDAR sensor to explore unknown house environments utilizing a Fully Convolutional Network [18] applied on a Deep Q Learning agent with a pixel-wise policy, whilst Mirowski et. al. [5] have addressed the problem of exploring small 3D mazes to reach some objects scattered around a labyrinth training an Asynchronous Advantage Actor-Critic(A3C) agent. A similar work on swarm robot exploration was discussed by Latifet al. [19] where they trained a decentralized system to explore labyrinths using Reinforcement Learning; the MDP they proposed has the positions of the robots and the walls of the maze as states and the movements of the robots (i.e. up, right, down, left) as actions.

In this paper, we developed a multi-agent system able to explore more complex environments, firstly training an agent with a RL method and then deploying the trained network on every agent of the swarm which has to explore the unknown environment.

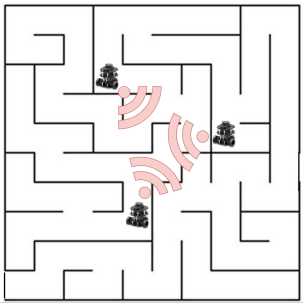


Fig. 1: System Overview. A robot swarm, in this case Turtlebot3, explores an unknown maze sharing discovered information.

III. Overall System

Our proposed system consists of a randomly generated, fully-explorable labyrinth and a swarm of autonomous agents equipped with LiDAR and communication capabilities. It is designed to achieve efficient exploration through coordinated multi-agent navigation and map merging techniques. The environment in which the agents operate is a randomly generated maze, ensuring full explorability. This means that the environment contains no closed or inaccessible rooms, and every area within the maze can be reached. The maze is represented as a grid map, where each cell can either be an empty space (indicating an explorable square) or a wall, creating distinct pathways and boundaries for the agents to navigate.

The agent swarm is composed of multiple autonomous robots, each equipped with a 2D LiDAR sensor. This LiDAR system enables the robots to scan their immediate surroundings, producing partial local observations of the environment. Additionally, each agent is capable of emitting and receiving WiFi signals within a certain range, facilitating communication between agents. This WiFi-based communication allows agents to perform a rendezvous, a critical operation in which two agents meet to share information. During a rendezvous, the robots exchange their individual global maps and merge them, resulting in a more comprehensive map of the environment that benefits the entire swarm.

Each agent in the swarm is equipped with three planners to guide its navigation decisions: a global greedy planner, a trained local planner, and a greedy local planner. The inclusion of multiple planners ensures both practical exploration and robustness in decision-making.

The Greedy Global Planner is responsible for high-level navigation across the environment. It makes decisions based on the global map, prioritizing actions that maximize the exploration of unvisited areas. The Trained Local Planner is trained using reinforcement learning techniques to make decisions at a local level, taking into account the agent's immediate surroundings. The local planner enables more efficient navigation through complex areas that require fine-

grained decision-making. The Greedy Local Planner serves as a fallback mechanism to enhance system robustness. It is invoked when the trained local planner selects a wrong action, such as going into a wall or an unknown space, or when the agent risks becoming trapped in a navigational loop (i.e., circling around the same locations). The greedy local planner ensures that the agent is able to recover from errors made by the trained planner and continue effective exploration.

The decision-making process of each agent is designed to guarantee task completion while preventing inefficiencies such as repetitive navigation or stagnation in exploration. Specifically, when the trained local planner fails to navigate the agent effectively, the greedy local planner is activated to ensure that the agent does not become stuck in a loop. The global greedy planner then takes over for long-range navigation, coordinating the agent's overall movement towards unvisited areas of the environment.

Algorithm 1 provides a formal description of the decision-making process, highlighting the transitions between the different planners to ensure robust and efficient exploration. The ability of the agents to share information during rendezvous events plays a pivotal role in enhancing the swarm's performance. When two agents meet within communication range, they exchange and merge their global maps. This merging process allows agents to update their knowledge of the environment, integrating information about areas they may not have individually explored. As a result, the agents benefit from a collective intelligence, improving their overall efficiency in completing the exploration task.

The combination of these planning mechanisms, along with the agents' ability to merge maps during rendezvous, ensures that the system is both scalable and robust, capable of efficiently exploring large and complex environments while minimizing redundancies in navigation.

In details, each agent starts checking if it has completed the task (i.e. exploring a certain percentage of the map), in that case the exploration ends, otherwise it controls if it is in a dead end Figure 3, when it occurs the greedy global planner recovers the navigation selecting a place of the agent's global map. If there is something left to explore in the robot's local map, the trained local planner picks an action and it checks if it is available and it does not force the robot to be stuck in a loop, otherwise a greedy global planner chooses the next move. After the action is performed, if the agent is sufficiently close to another robot they perform a rendez-vous.

Algorithm 1: MARS Decision making

```
Data: glob_map, loc_map, pos, new_squares
positions = [];
finished = False;
while not finished do
    for Agent i=1..k do
        if map_explored(glob_map) then
            finished = True;
            break;
        end
        else if dead_end(loc_map) then
            action =
             greedy_global(glob_map);
        end
        else
            action = trained_local(loc_map);
            if action is wrong then
                action =
                 greedy_local(loc_map);
            end
            else if new_squares is 0 then
                if pos in positions then
                    action =
                     greedy_local(loc_map);
                end
                else
                    positions.append(pos);
                end
            end
            else
                positions = [];
            end
        end
        move(action);
        if Rendez-vous with j then
            send explored map to agent j
        end
    end
end
```

IV. Methodology

In this section we overview the RL model running on each agent and how the global planner interacts with the learning capabilities.

A. Background

A typical Reinforcement Learning (RL) framework can be formulated as an episodic Markov Decision Process (MDP)

An episodic MDP M is defined as a 6-tuple (S, A, p, r, S_0, S_f), where S represents the set of states, A is the set of actions, $p(s, a, s')$ is the transition function that provides the probability of transitioning to state s' after taking action a in state s, and $r(s, a)$ is the reward function that returns the immediate reward for taking action a in state s. S_0 is the initial state distribution, and S_f is the set of terminal states.

At each time step t, the agent observes its current state and selects an action based on its policy $\pi : S \to A$. The objective of the agent is to learn an optimal policy π^*, which maximizes the expected return G_t until the episode terminates at time step T.

$$G_t = \sum_{T-t}^{i=1} R_{t+i}$$

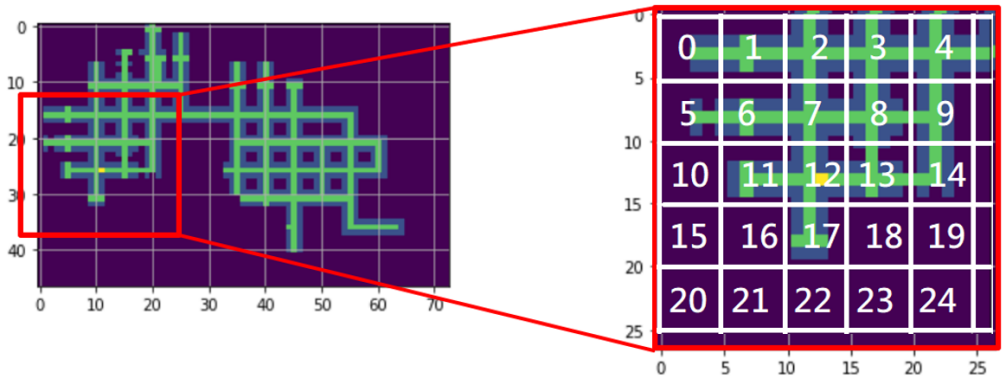
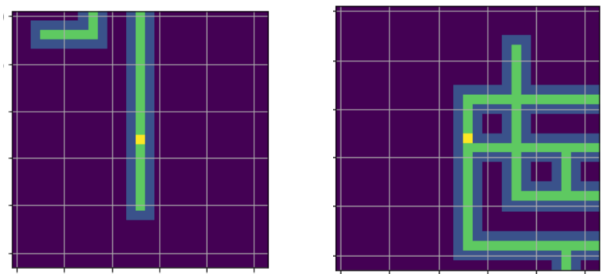


Fig. 2: Global and local map. In this representation, the yellow square indicates the agent's location, green squares represent the explorable space on the local map, blue squares depict walls, and purple squares denote unknown areas. The global map shows the area the agent has already explored, while the local map is a section of the global map centered on the agent. The local map is divided into clusters to reduce the action space, as each cluster corresponds to a possible action.



(a) (b)

Fig. 3: Example of dead ends. A dead end occurs when the agent encounters walls with no remaining explorable spaces in the local map.

To achieve this, one common approach involves learning the optimal action-value function $Q^*(s, a)$, which represents the expected return when taking action a in state s and then following the optimal policy π^*. Algorithms like SARSA or Q-learning [20] are typically used to learn this function. Once $Q^*(s, a)$ is known, the optimal policy can be derived by selecting the action that maximizes $Q^*(s, a)$ in each state, i.e., $\text{argmax}_a\, Q^*(s, a)$.

For environments with high-dimensional or continuous state spaces, deep learning approaches are often employed. Deep Q-Learning is a significant type of reinforcement learning algorithm that employs a deep neural network to estimate the Q-function, aiding in decision-making across various domains like gaming, robotics, and autonomous vehicles. This function helps determine the best action to take in a given state, based on expected cumulative rewards. DQN, as a model-free reinforcement learning approach, approximates the Q-values for each action using a neural network. Like standard Q-Learning, it employs an off-policy learning strategy, where the agent can choose actions that maximize the Q-value without strictly following the learned policy. This is enabled by the use of the ϵ-greedy exploration technique, which balances exploration and exploitation during training.

Taking the maximum overestimated values implies estimating the maximum value itself, leading to systematic overestimation and introducing maximization bias in Q-learning. As Q-learning involves bootstrapping, where estimates are derived from existing estimates, this overestimation poses significant challenges.

To address maximization bias in Q-learning, using two separate Q-value estimators is suggested. Each estimator updates the other, ensuring unbiased Q-value estimates of actions chosen using the alternate estimator.

This technique called Double Q-Learning had different versions, the one that will be shown in this paper is made by Hasselt et al. 2010 [21], where there are two independents model of Q. One used for action selection Q', called target network, and the other for action evaluation Q, called primary network.

Hence, the equation to evaluate the Q-value is now defined as:

$$Q^*(s_t, a_t) \approx r_t + \gamma Q(s_{t+1}, \underset{a'}{\text{argmax}}\, Q'(s_t, a_t))$$

The target network slowly copies the parameters of Q and it is usually updated through Polyak averaging:

$$\theta' \leftarrow \tau\theta + (1 - \tau)\theta'$$

where θ' is the target network parameter, θ is the primary network parameter, and τ is called rate of averaging.

B. Markov Decision Process model

The exploration problem of robots is modeled as a Markov Decision Process (MDP), composed of the following elements:

Environment. A generator produces random, fully-explorable mazes. The generator starts creating the maze from the center and continues to add explorable spaces next to the ones generated before, guaranteeing the condition of full explorability. Another module creates an agent able to move around the map, scanning the area with a simulated 2D LiDAR. The portions of the maze scanned by the agent in such a way are summed up to form the explored, global grid map. An example of such an environment can be seen in Figure 2.

State space. At each time instant, the agent has access to a portion of the explored global map, namely a local map centered on the agent. The local map is a square of 5×5 cells in order to reduce the state space and make training and deployment of the network more efficient. This size is a good trade-off for a good view of the close regions and a treatable input space. Each cell of both the global and local maps is associated with a value indicating accessible, unknown, and occupied spaces, in addition to the agent's location.

Action space. The policy has to choose the next navigation location from a cluster of pixels on the local map. By choosing a cell of the local map, we can also limit the possible sets of actions.

Reward function. The reward function represents the objective of each robot of the system, and is defined as:

$$R_i = \begin{cases} -10 & \text{If } action \textbf{ is not } \text{feasible} \\ new\_squares - steps & \text{otherwise} \end{cases} \tag{1}$$

This function incentives exploration based on the number of $new\_squares$, namely spaces previously unknown, i.e., no agent of the swarm has previously explored it, and the number of $steps$ it took. It also penalizes the agent when an unfeasible action is taken (e.g., selecting a location containing only unknown pixels or walls).

Initial state. Initially, the agent is spawned at a randomly accessible location in the maze. The whole environment is unknown to the agent except the area reached by the simulated 2D LiDAR. The policy has to choose a space out of the accessible, known locations on this local grid map that constitutes the next navigation goal. When a feasible action is selected, an A-star algorithm finds the optimal path to reach the selected goal. In such a way, the agents explore the environment while creating the global, explored map.

Terminal state. The agent ends its run if it encounters a dead end (Figure 3). This situation occurs when there is nothing left to explore on the local map, which means that none of the accessible spaces has any unknown space nearby. In this case, the global planner is invoked. The global planner is a deterministic system that recovers the navigation by selecting the nearest accessible square with at least one unknown pixel nearby on the global explored map. From there, the local planner can start exploring again. Another terminal state, which terminates both the local and the global planner, arises when a fixed percentage of the maze has been explored.

V. Experiments and Results

In this section, we evaluate the performance of MARS in exploring diversified environments and compare the obtained results to other two benchmarks.

A. Training of the agent

The Reinforcement Learning agent is a Convolutional Neural Network (CNN) trained with a Double DQN algorithm since it is perfectly suitable with discrete state and action spaces. The agent is trained for 500.000 iterations. The training is performed on a 200×200 pixels maze, and the agent's local map has a size of 45×45 pixels. An episode ends when the agent explores 80% of the environment. The network features two convolutional layers with 64 filters, each of dimension 3×3 and stride respectively 1 and 2. Other training hyperparameters are gathered in Table I.

TABLE I: Hyperparameters' values used during the training.

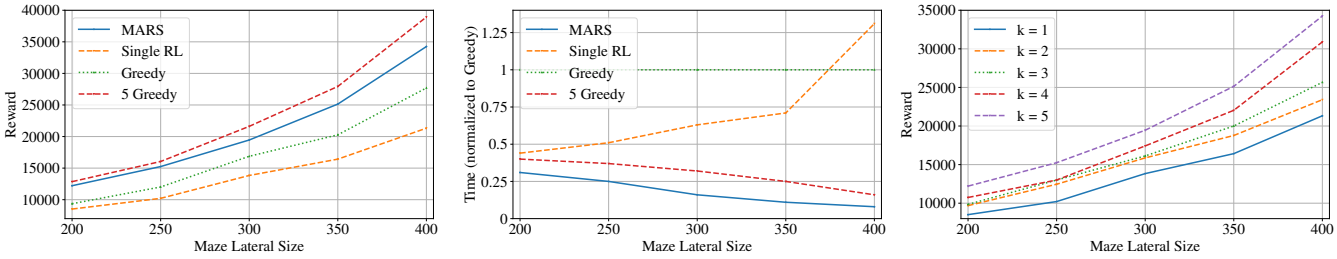
Hyperparameters	Value
Learning rate	10^{-4}
Initial ϵ	1
Final ϵ	0
ϵ decay steps	$2.5 \cdot 10^5$
τ	1
γ	1
Steps per iteration	5

The parameter τ, a temperature value determining the update rate of the target network, is set to 1 to ensure rapid learning. If τ were lower, the network would take too long to learn how to avoid unfeasible actions. The discount rate γ must be set to 1, as values below this would cause the agent to remain stationary, since the step penalty decreases with each step, eventually approaching zero. Additionally, the agent solely exploits its learned knowledge starting from halfway through the training process. The training takes approximately 12 hours on a machine featuring an Intel Core i7-9700K CPU and an Nvidia GeForce RTX 2080 Ti.

B. Evaluation and Results

In order to validate the proposed methodology, we train the RL model in a single agent and use it on the multiple agents. We compare the performance against (i) a single deterministic agent, i.e., an agent with both greedy local and global planners, (ii) a single RL agent and (iii) five deterministic agents. All solutions encounter the same testing conditions and we repeat the experiments for all solutions 10 times.

We report the results obtained in Figure 4. The tests are conducted on mazes featuring different sizes; in the x axis is reported the lateral dimension of the square containing the maze, in pixels. Figure 4a reports the rewards obtained by the single trained agent, the single greedy agent, 5 greedy agents and the MARS solution composed of 5 agents. MARS's and 5 greedy agents system's reward is the sum of the rewards of each agent of the swarm. It can be noted that, as per Equation (1), a solution that takes more steps to explore the entire maze obtains a lower reward, and if an agent explores a cell already visited by others, penalizes the global reward. As it can be seen, the MARS can reach higher reward values than the single greedy agent in less steps, but it still gathers a lower reward (90% on average) than the multiple agent deterministic system. However, we also measure the computational time and report the normalized value to the time taken by the greedy agent in Figure 4b and it is worth noting that MARS requires half the time that the 5 greedy agents system would need to explore a maze. We can observe how the single agent performance worsens with increasing maze size. This is due to the RL



(a) (b) (c)

Fig. 4: Performance comparison between a single traditional greedy algorithm, a single RL agent, 5 traditional greedy agents and MARS at varying the lateral dimension of the squared maze. The plots show respectively (a) the obtained reward, (b) the time consumption normalized to the greedy agent and (c) the reward gathered by swarms of different numbers of agents (k). In every plot, the MARS' reward is the sum of the rewards of each agent of the swarm

agent and the greedy one sharing the same greedy global planner, which needs more time when the map is larger. This is not the case for MARS, since being multiple agents it is less probable for an agent to invoke the global planner. This is why MARS consistently maintains low computational time performance for all maze sizes. Finally, Figure 4c shows the MARS's reward when varying the number of agents in the swarm. We can observe how the more agents help to explore the maze in a shorter time and that the more, the merrier.

VI. Conclusion and Future Works

In this paper we presented MARS, a multi-agent method to explore unknown and complex environments. The results obtained validated how MARS is more effective compared to a traditional solution by leveraging learning capabilities and swarm abilities in approximating the solution while achieving faster and more efficient performances. In the future, we plan to advance MARS training by employing more advanced techniques, e.g., curriculum learning can help generalize over more difficult and diverse mazes. In addition, fully convolutional networks (FCN) and attention-based structures could be implemented to increase the agent's generalization capabilities.

Acknowledgment

The work presented in this paper has born from the collaboration with the PIC4SeR Center for Service Robotics at Politecnico di Torino.

References

[1] 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 *Proceedings of The 33rd International Conference on Machine Learning (ICML)*. PMLR, 2016, pp. 1928–1937.

[2] S. Thrun, "Probabilistic robotics," *Communications of the ACM*, vol. 45, no. 3, pp. 52–57, 2002.

[3] 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 *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2017, pp. 3357–3364.

[4] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, "Deep reinforcement learning with successor features for navigation across similar environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 2371–2378.

[5] P. Mirowski, R. Pascanu, F. Viola, H. Soyer, A. J. Ballard, A. Banino, M. Denil, R. Goroshin, L. Sifre, K. Kavukcuoglu *et al.*, "Learning to navigate in complex environments," *arXiv preprint arXiv:1611.03673*, 2016.

[6] A. Sacco, M. Flocco, F. Esposito, and G. Marchetto, "Supporting Sustainable Virtual Network Mutations with Mystique," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 2714–2727, 2021.

[7] ——, "Owl: Congestion Control with Partially Invisible Networks via Reinforcement Learning," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021, pp. 1–10.

[8] ——, "Partially Oblivious Congestion Control for the Internet via Reinforcement Learning," *IEEE Transactions on Network and Service Management*, vol. 20, no. 2, pp. 1644–1659, 2022.

[9] J. Zhang, L. Tai, M. Liu, J. Boedecker, and W. Burgard, "Neural slam: Learning to explore with external memory," *arXiv preprint arXiv:1706.09520*, 2017.

[10] Q. Liang and M. Liu, "Plugo: a vlc systematic perspective of large-scale indoor localization," *arXiv preprint arXiv:1709.06926*, 2017.

[11] Y. Sun, M. Liu, and M. Q.-H. Meng, "Wifi signal strength-based robot indoor localization," in *IEEE International Conference on Information and Automation (ICIA)*. IEEE, 2014, pp. 250–256.

[12] B. Yamauchi, "A frontier-based approach for autonomous exploration," in *Proceedings 1997 IEEE International Symposium on Computational Intelligence in Robotics and Automation CIRA'97.'Towards New Computational Principles for Robotics and Automation'*. IEEE, 1997, pp. 146–151.

[13] F. Bourgault, A. A. Makarenko, S. B. Williams, B. Grocholsky, and H. F. Durrant-Whyte, "Information based adaptive robotic exploration," in *IEEE/RSJ international conference on intelligent robots and systems*, vol. 1. IEEE, 2002, pp. 540–545.

[14] M. Montemerlo, "Fastslam: A factored solution to the simultaneous localization and mapping problem," *Proc. of AAAI02*, 2002.

[15] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[16] L. Tai and M. Liu, "Mobile robots exploration through cnn-based reinforcement learning," *Robotics and biomimetics*, vol. 3, pp. 1–8, 2016.

[17] H. Li, Q. Zhang, and D. Zhao, "Deep reinforcement learning-based automatic exploration for navigation in unknown environment," *IEEE transactions on neural networks and learning systems*, vol. 31, no. 6, pp. 2064–2076, 2019.

[18] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 3431–3440.

[19] E. Latif, W. Song, and R. Parasuraman, "Communication-efficient reinforcement learning in swarm robotic networks for maze exploration," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2023, pp. 1–6.

[20] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[21] H. Hasselt, "Double q-learning," *Advances in neural information processing systems*, vol. 23, 2010.