2023 IEEE 33rd International Workshop on Machine Learning for Signal Processing (MLSP) | 979-8-3503-2411-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/MLSP55844.2023.10285997

DEEP REINFORCEMENT LEARNING WITH ACTION MASKING FOR DIFFERENTIAL-DRIVE ROBOT NAVIGATION USING LOW-COST SENSORS

Konstantinos Tsampazis, Manos Kirtas, Pavlos Tosidis, Nikolaos Passalis, Anastasios Tefas

Computational Intelligence and Deep Learning (CIDL) Group, AIIA Lab.,
Department of Informatics,
Aristotle University of Thessaloniki, Thessaloniki, Greece
E-mails: {tsampaka, eakirtas, ptosidis, passalis, tefas}@csd.auth.gr

ABSTRACT

Driving a wheeled differential-drive robot to a target can be a complicated matter when trying to also avoid obstacles. Usually, such robots employ a variety of sensors, such as LiDAR, depth cameras, and others, that can be quite expensive. To this end, in this paper, we focus on a simple differential-drive wheeled robot that uses only inexpensive ultrasonic distance sensors and touch sensors. We propose a method for training a Reinforcement Learning (RL) agent to perform robot navigation to a target while avoiding obstacles. In order to increase the efficiency of the proposed approach we design appropriate action masks that can significantly increase the learning speed and effectiveness of the learned policy. As we experimentally demonstrated, the proposed agent can robustly navigate to a given target even in unknown procedurally generated environments, or even when denying part of its sensor input. Finally, we show a practical use-case using object detection to dynamically search for, and move to objects within unknown environments. The code used for conducted experiments is available online on Github.

***Index Terms*—** Robot Navigation, Low-Cost Robot Sensors, Deep Reinforcement Learning, Action Masking

1. INTRODUCTION

Autonomous navigation of mobile robots has been a popular research topic in the field of robotics for decades. The ability to navigate in complex and dynamic environments is essential for many applications, such as search and rescue, logistics, and home care. One of the most common and versatile types of mobile robots is the differential-drive wheeled robot. These robots are easy to build and control, and their differential-drive system allows them to turn on the spot and move in any direction. At the same time, Deep Reinforcement Learning (DRL) [1] has also emerged as a promising technique for training autonomous agents to perform complex tasks, leading to several robotics applications [2], including navigation, manipulation, and control.

Recent research mainly targets robots that use multiple sensors and relatively expensive configurations with LiDAR, depth cameras and others. One such example is in [3], where the authors used a Jetson Nano for obstacle avoidance using a monocular camera. In another work [4], the authors used the Turtlebot 3 Waffle Pi, and trained a Double DQN [5] agent to navigate to a target. In a more recent work using the same robot [6], a mapless local path planning approach was presented, that used variants of Deep Q-Network [7] to increase success rates, proposing the n-Step Dueling Double DQN with Reward-Based ϵ-Greedy (RND3QN) algorithm. In [8], the authors successfully used Deep Deterministic Policy Gradient [9] to train an agent to drive a differential-drive robot to a target, improving on this paper's authors' earlier work in [10]. They used curriculum learning [11] to gradually train the agent on a small set of increasingly difficult maps. Another more generic example is the established Robot Operating System (ROS)[1] navigation stack, which while it can work with inexpensive ultrasonic distance sensors, it is more well-suited to work with LiDAR and depth sensors. As a result, many DRL stacks are designed exclusively for high-end robotic hardware, which limits their potential impact in a wide range of applications, from education to low-cost mass production robots. At the same time, training DRL agents with low-fidelity and noisy sensors can worsen sample efficiency. This necessitates the use of more sample-efficient paradigms in such applications.

In this paper, we propose a method for training an agent to drive a low-cost differential-drive wheeled robot navigating to a target while avoiding obstacles, using the well-established Proximal Policy Optimization (PPO) [12] RL algorithm. To improve training efficiency we employ invalid action masking [13], also known as Maskable PPO, after appropriately designing two masks that can lead to increased performance. This work a) introduces a more systematic and effective approach to perform action masking, as well as b) paves the way for introducing a state-of-the-art DRL approach using

[1] `https://www.ros.org/`

low-cost sensors in the robotic navigation domain. The agent used, apart from its sensor values, takes only the relative angle and distance to its target and not its own or the target's absolute position, and thus can act as a local path planner and navigate dynamically in unknown environments. We developed a randomized procedural map generation method within the Webots robotics simulator [14], to be able to train and realistically evaluate the agent in complex environments with obstacles of various challenging shapes, using realistic noisy sensors. The code used for conducted experiments is available online[2].

The rest of the paper is structured as follows. The proposed method is provided in Section 2, while the experimental evaluation is provided in Section 3. Finally, Section 4 concludes the paper.

2. PROPOSED METHOD

2.1. Background and Setup

In this work, we employ a custom robot, as shown in Fig. 1a. This robot is created in Webots and consists of two motors connected to wheels providing differential drive, a forward-placed bumper that is split between two touch sensors, one left and one right, and 13 forward-facing ultrasonic distance sensors that are placed in equal-spaced angles between $[-\pi, \pi]$. The ultrasonic distance sensors have a range of $1m$ and return valid values when their ray angle of incidence to obstacles is close to vertical. Moreover, the values returned are noisy, simulating real ultrasonic distance sensors.

We used a discrete action space with a set of five actions that cumulatively control the motor speeds of the robot. The first one increases both motor speeds by a fixed amount up to a limit, the second decreases them, the third and fourth increase one motor speed but decrease the other and the fifth action does not cause any changes to the motor speeds. We refer to these actions as "forward", "backward", "left", "right" and "no action".

The observation space of the agent is primarily described by the following vectors:

$$\boldsymbol{a}_t = [d_t, a_t, m_{l,t}, m_{r,t}, ts_{l,t}, ts_{r,t}], \tag{1}$$

where d_t is the current Euclidean distance to the target, a_t is the current angle to the target in regards to the facing angle of the robot, $m_{l,t}$ and $m_{r,t}$ are the current left and right motor speeds, and finally $ts_{l,t}$ and $ts_{r,t}$ are the left and right touch sensor values, for timestep t and

$$\boldsymbol{b}_t = [ac_{0,t}, ..., ac_{4,t}, ds_{0,t}, ..., ds_{12,t}], \tag{2}$$

where $ac_{i,t}$ for $i \in [0, 4]$ is the one-hot vector representing the previous action, and $ds_{j,t}$ for $j \in [0, 12]$ represents the latest distance sensors values.

[2] https://github.com/aidudezzz/deepworlds/tree/dev/examples/find_and_avoid_v2

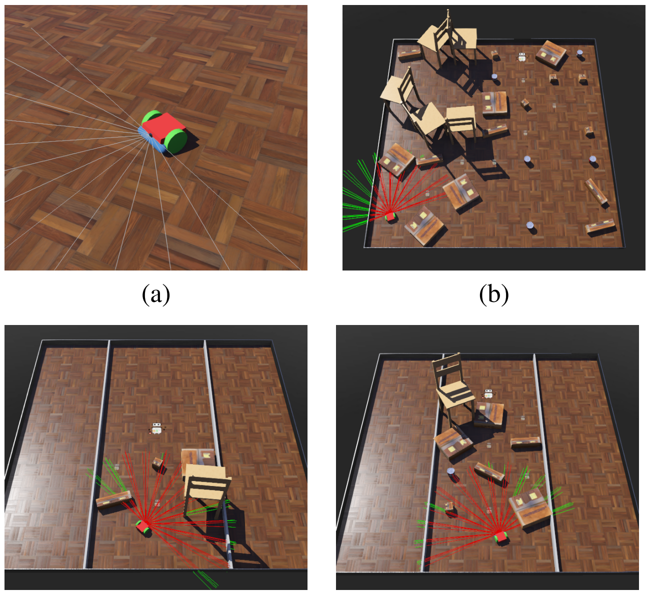
(a) (b) (c) (d)

Fig. 1. (a) the robot used, (b) random map with all 25 obstacles, (c) corridor map with 2 rows of obstacles, (d) corridor map with 4 rows of obstacles.

To construct the full observation $\boldsymbol{o}_t$ at each timestep t, we concatenate $\boldsymbol{a}_t$ and $\boldsymbol{b}_t$ with $\boldsymbol{a}_{t-w}$ and $\boldsymbol{b}_{t-w}$, where $\boldsymbol{a}_{t-w}$ and $\boldsymbol{b}_{t-w}$ are the vectors containing the values of timestep $t - w$ approximately one second before timestep t. Therefore, the full observation is defined as:

$$\boldsymbol{o}_t = [d_t, a_t, ..., ds_{12,t}, d_{t-w}, a_{t-w}, ..., ds_{12,t-w}], \tag{3}$$

where $w = \lceil \frac{1000}{s} \rceil$ and s is a single timestep time defined in milliseconds. This way, even though the agent uses a simple feed forward neural network, it takes a time window as observation giving it insight into how the observation values are changing over time, which experimentally provided the best results. All values are normalized appropriately in the $[-1, 1]$ range.

The reward R is defined as $R = w_{dr}dr + w_{ar}ar + w_{dsr}dsr + w_{rtr}rtr + w_{cr}cr$, where dr is the distance to target reward, ar is the angle to target reward, dsr is the distance sensors reward, rtr is the reach target reward and cr is the collision reward, who all lie within or are normalized in the range $[-1, 1]$. Then every sub-reward is multiplied by their corresponding weights (experimentally) set as $w_{dr} = 1.0, w_{ar} = 1.0, w_{dsr} = 10.0, w_{rtr} = 1000.0, w_{cr} = 100.0$. The distance to target reward is comprised of two components itself, one continuous that penalizes the agent the farther away it is from the target, and one discrete that rewards the agent every time it achieves a new minimum distance to the target. The agent is rewarded when facing the target or when it turns towards the target, and penalized when it turns away from the target. The angle reward is zeroed out when there are obstacles detected nearby, to enable the agent to turn away from

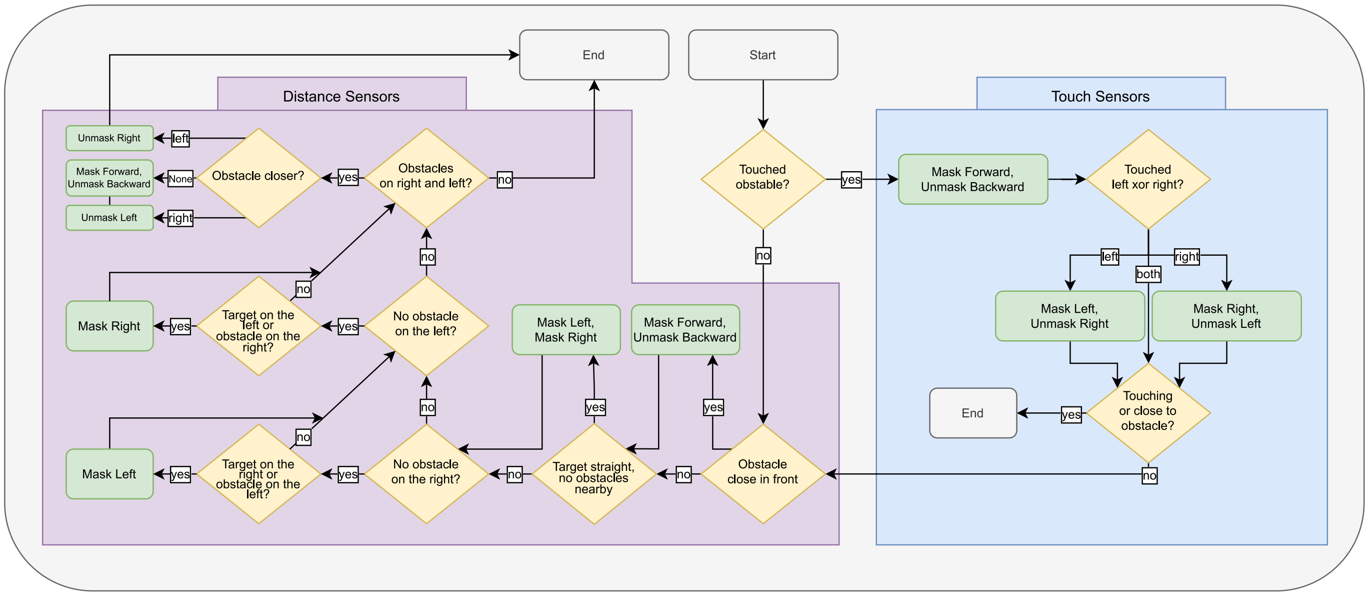
End
Start
Distance Sensors
Touch Sensors
Unmask Right
left
Mask Forward, Unmask Backward
None
Unmask Left
right
Obstacle closer?
yes
Obstacles on right and left?
no
Touched obstable?
yes
Mask Forward, Unmask Backward
Touched left xor right?
left
both
right
Mask Left, Unmask Right
Mask Right, Unmask Left
Mask Right
yes
Target on the left or obstacle on the right?
no
No obstacle on the left?
Mask Left, Mask Right
Mask Forward, Unmask Backward
End
yes
Touching or close to obstacle?
Mask Left
Target on the right or obstacle on the left?
No obstacle on the right?
Target straight, no obstacles nearby
Obstacle close in front

Fig. 2. The proposed mask flowchart.

the target to avoid them. The distance sensors reward penalizes the agent when the sensors are reading below a threshold value and rewards it otherwise. When the agent reaches the target it is rewarded, but the reward is scaled based on the episode time elapsed, i.e. when the agent instantaneously reaches the target the reward is 1.0 and when it reaches the target just before the episode ends it is rewarded with 0.5. Finally, the agent is penalized when it collides with obstacles as detected by its touch sensors. An episode is terminated when the agent either reaches the target or it collides with obstacles for $4,096$ steps.

2.2. Proposed Action Masking

Invalid action masking [13] is a technique used with PPO to handle tasks involving invalid or forbidden actions. It prevents the agent from selecting invalid actions during training and evaluation by setting their probability to zero. This leads to more efficient policies, especially in tasks with common invalid actions or severe consequences. It can be applied to both large and small action spaces. The technique has been most notably, successfully used in Dota 2 [1].

In this work we propose two different masking approaches, a “simple” baseline action mask and a more sophisticated “advanced” mask. Typically, the masks provide a vector of truth values, one for each action, that are either true, enabling or unmasking the action, or are false, disabling or masking the action. These vectors are calculated for every simulation step.

The *simple mask* developed initially, masks the forward action when close to obstacles or masks the backward action when no obstacles are detected. This prevents unnecessary collisions and unwarranted backwards driving. The mask also prevents turning into obstacles by masking left or right actions when respective sensors detect low values.

The *advanced mask* flowchart is provided in Fig. 2. This method runs in every simulation step starting with all actions unmasked and returns the final mask for the next step. As can be seen in the flowchart, the mask method is split into two components, one using the distance sensors and one using the touch sensors. As long as nothing is detected with the touch sensors, the robot’s actions are masked via the relative current target angle and distance sensors. When the distance sensors component fails and a collision is detected, the touch sensors component takes over until the robot is clear of obstacles. Thus, the robust touch sensors act as a fallback for the ultrasonic distance sensors which are quite unreliable, as the incidence angle of their rays to the obstacles must be close to vertical to return a valid value. However, obstacles that can be detected via distance sensors can be effectively avoided.

3. EXPERIMENTAL EVALUATION

3.1. Training

For all the conducted experiments we used a simple feed forward neural network that has three hidden layers with $[1024, 512, 256]$ neurons for the actor and $[2048, 1024, 512]$ for the critic. To this end, we utilized the RL implementation provided by stable-baselines3[3]. All other network architecture values and training parameters are left at their default values apart from γ set to 0.999, entropy coefficient set to

[3]`https://github.com/DLR-RM/stable-baselines3`

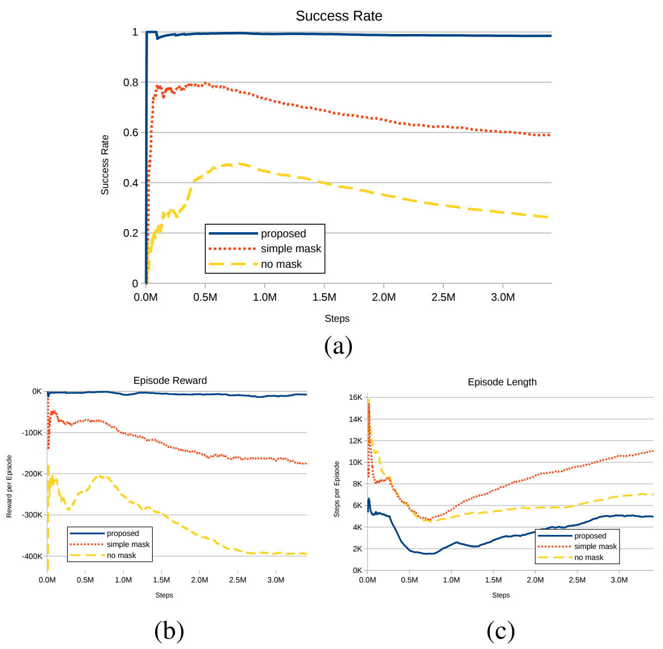

Fig. 3. The training results for the three masks used over training steps: (a) the average success percentage, (b) the average episode reward, (c) the average episode length. Each masking method was trained for 6 different seeds and the average results are presented.

0.001 and base learning rate set to 3e-4 and decaying with a linear schedule.

We leveraged the open-source robotics simulator Webots [14] to create a randomizable environment for training and evaluating the agents. The stable-baselines3 agent implementation was integrated with the simulator through the deepbots [10] framework, which enables the creation of custom OpenAI gym-compliant[4] RL environments in Webots. The training and evaluation maps consist of walled arenas with various shapes of obstacles, known as “random maps”. The number of obstacles can be customized, and the robot and target are placed randomly in free spaces with varying distances between them. An example map can be seen in Fig. 1b. Moreover, the map can be randomly generated as a corridor with adjustable number of rows of obstacles between the robot and the target, referred to as “corridor map”, as can be seen in Fig. 1c, d.

The agents undergo a total of $3,407,872$ training steps, with each new map serving as an episode lasting $16,384$ steps. We employed curriculum learning [11] during training, gradually increasing the difficulty. Initially, for $262,144$ steps, we used the “random map” configuration with 10 random obstacles. The target was placed at a Manhattan distance of 10 from the robot. This setup provided open maps with few obstacles and wide paths, making it the easiest starting point for the agents. Subsequently, we introduced a series of “corridor maps” of increasing difficulty, with the agents trained for $524,288$ steps on each difficulty level. They start out with 1 row of 2 obstacles and end up with 5 rows of 10 obstacles between the target and the robot. Fig. 1d illustrates the challenging nature of corridor maps, which feature dead ends and intricate paths to navigate. Moreover, even in its easiest configuration the “corridor map” puts obstacles between the robot and the target, forcing the agents to learn to traverse around the obstacles. Finally, the agents are trained for another $1,048,576$ steps on a “random map” with all 25 available obstacles and the target Manhattan distance set to a random value between 10 and 12. This configuration provides the agents with a more realistic environment with many obstacles spread around.

The training results can be seen in Fig. 3. The average success percentage over the timesteps seen in Fig. 3a is the ratio of episodes that the robot reached the target over the total number of episodes. The agent without any mask only ever achieves $\approx 47.0\%$ success percentage before decaying to under $\approx 30\%$ at the end of its training with its average reward collapsing. The episodes terminate early due to collisions, thus achieving a lower average episode length than the simple-mask agent. The agent that uses the simple mask has much better performance, as it begins with a success percentage of $\approx 80\%$ and drops to just under 60% before the training ends. The proposed method with the advanced mask quickly achieves a success percentage of close to 100% in the initial easy maps and decays over time to 98.5%, as the curriculum gets harder. As expected, all methods have sharp drops in their initial average episode length as the agents get better at reaching the target efficiently, and increase over time as the maps get more and more complicated.

3.2. Evaluation

	Baseline (no mask)	Proposed (simple mask)	Proposed (advanced mask)
Success (%)	43.7 ± 20.9	64.1 ± 23.1	$\mathbf{98.8 \pm 0.2}$
Reward ($\times 10^2$)	-14.6 ± 6.3	-2.1 ± 3.0	$\mathbf{8.8 \pm 0.1}$
Ep. Len. ($\times 10^3$)	8.6 ± 10.8	10.8 ± 8.6	$\mathbf{2.5 \pm 1.9}$

Table 1. The evaluation results for the different masks used, averaged across 6 runs with different seeds and their corresponding standard deviation.

To evaluate the trained agents in the various mask configurations, we used a set of 600 previously unseen maps of 6 difficulty setups with 100 maps each, using the same seed to get exactly the same robot/target and obstacles randomization for all agents to provide a fair evaluation and comparison. Starting out, the first 100 maps are simple “corridor maps” without any obstacles, that show whether the agent can efficiently move straight to the target that is placed in various

[4]`https://github.com/openai/gym`

distances along the corridor. Earlier in development, unsuccessful agents had trouble reaching the target even in empty corridors and would overshoot or hug the walls. The next 4 sets of 100 maps each are "corridor maps" of increasing difficulty in terms of how many rows of obstacles are placed between the robot and the target, similar to the training setup described previously. The last set of 100 maps uses the "random map" configuration that includes all 25 available obstacles. Note that the seed used for evaluation is different from the training seeds, thus the maps used in evaluation are not previously seen by the agents.

The reward function weights were set as $w_{dr} = 0.0, w_{ar} = 0.0, w_{dsr} = 0.0, w_{rtr} = 1000.0, w_{cr} = 1.0$ to provide a better constrained reward score, where the agent gets rewarded only by reaching the target or punished for colliding. The average evaluation results can be seen in Table 1, along with their standard deviation across the 6 seeds. Similar to the training results, the no mask setup ends episodes prematurely due to collisions without reaching the target, which is reflected in the poor success rate and more decisively in the low average episode reward, and as a consequence gets a lower average episode length than the simple mask. The proposed method converges to a near optimal policy that manages to solve $\approx 98.8\%$ of the evaluation maps with a high average reward of ≈ 880 and low average episode length of ≈ 2500 steps, outperforming the other two methods by far. Finally, the proposed advanced masking method shows much better consistency represented by the very low standard deviation values across the seeds.

3.3. Component ablation and sensor denial

	No DS	50% DS	No TS	Only mask	**Full**
Success (%)	93.8	95.5	84.5	69.3	**98.6**
Reward ($\times 10^2$)	4.9	7.4	8.0	-6.8	**8.7**
Length ($\times 10^3$)	**2.2**	3.4	8.0	15.2	2.3

Table 2. The results of the various component ablation and sensor denial experiments. "DS" refers to distance sensors and "TS" refers to touch sensors.

We evaluated one of the trained agents with the advanced mask in the evaluation maps using the same seed and procedure, but in three different sensor configurations: disabling all distance sensors, randomly disabling 50% of the distance sensors in each episode, and removing touch sensors entirely setting their values to zero. Removing the distance sensors effectively disables the part of the proposed mask depends on them, and similarly, removing the touch sensors disables the touch sensor part of the mask. Disabling sensors also means that the agent has to work with an incomplete observation. Moreover, we evaluated a random policy that practically only uses the advanced mask to navigate, picking random unmasked actions with equal probability. All the results can be seen in Table 2, where the four ablation configurations are compared to the full method. Denying half or all of the distance sensors decreases performance slightly in terms of success percentage. Without any distance sensors the agent moves blindly colliding with obstacles a lot, thus decreasing its average reward, but also less cautiously making it reach the target faster. The agent without touch sensors, is hampered in its ability to reach the target, resulting in a decrease of approximately $\approx 14\%$ in success percentage and prolonged navigation times due to the limitations of realistic ultrasonic distance sensors in detecting obstacles effectively. The high reward is a byproduct of the touch sensors not detecting any collisions. Using the mask alone, that includes the human expert knowledge, coupled with a random policy yields the worse results in all of the metrics, demonstrating that the learned behaviour of the agent via the reward function is a crucial component. These results show that the method is resilient to distance sensor denial, and has an adequate success percentage even without the touch sensors. On a real robot the distance sensors used in simulation can be emulated by a single distance sensor rotating on a servo motor in predetermined positions, which will affect the proposed method's performance marginally due to its resiliency to the distance sensor denial shown.

3.4. Example Use-Case

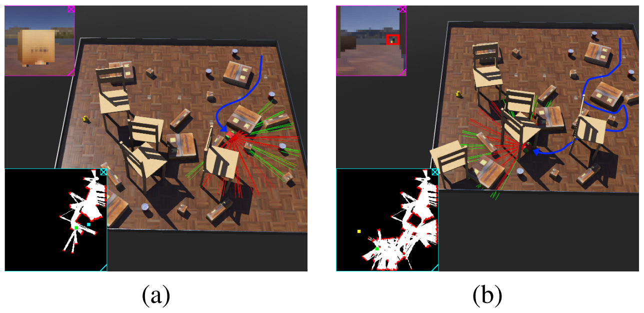
(a) (b)

Fig. 4. Screenshots of the robot (green box on the map) looking for the rubber duck in a random map: (a) robot is moving towards a random exploration target (light blue box), (b) robot has acquired the target (yellow box) and moving towards it. The path the robot has followed is roughly sketched in blue. On top-left the perspective of the camera can be seen, as well as the bounding box of the detected object on screenshot (b). The current map can be seen on the bottom-left.

We used the trained agent with the advanced mask on a more practical use-case, combined with object detection provided by the simulator and a simple custom mapping functionality, where the agent searches the map for a rubber duck as can be seen in Fig. 4. The mapping functionality relies only on the distance sensors, starting out with a black map image that, as the robot moves around slowly gets filled with white pixels for empty space and red pixels for obstacles detected by the distance sensors, as can be seen on the lower-left parts of the screenshots on Fig.4. The mapping functionality provides exploration targets to the agent that lie in black patches, i.e. unexplored parts of the current map. As the agent explores the environment with the help of these targets, it can detect the rubber duck via the camera's object detection module. When this happens the agent is provided with a target that corresponds to the rubber duck's position as approximated by the bounding box on the camera image. Consequently, using the trained agent with the proposed method as a local path planner and a rudimentary exploration and mapping functionality, the robot can find targets within the environment and navigate successfully and efficiently to them, using only simple distance and touch sensors as well as a camera for target object acquisition.

4. CONCLUSIONS

In this paper, we demonstrated that by crafting an appropriate masking function we can effectively combine RL with expert human knowledge to provide an efficient policy for differential-drive robot navigation without using expensive robot configurations and sensors. We trained the three methods on multiple seeds and evaluated the trained agents on the same random maps to provide a fair comparison, while also conducting sensor ablation and denial experiments. The experimental results suggest that the proposed masking method yields significantly better results than the baseline and converges to a near-perfect policy. The proposed method was also able to successfully navigate to the target in nearly all the evaluation maps, even though the random map generation method used produces very complicated maps with a variety of obstacles both in terms of size and shape, that the realistic noisy ultrasonic distance sensors have a great difficulty detecting. The nature of the proposed method makes it suitable to be used as a local path planning algorithm of low cost once trained, which is shown via an example use-case that uses a rudimentary custom mapping functionality and object detection via a camera. In the few failure cases, the agent gets stuck in narrow passages between obstacles it cannot detect or gets trapped in long dead-end paths, that are expected shortcomings of the nature of the method and sensor setup. Future work includes incorporating RGB camera input, as well as adding end-to-end mapping capabilities to the agent to address these challenges.

Acknowledgments This work was supported by the European Union's Horizon 2020 Research and Innovation Program (OpenDR) under Grant 871449. This publication reflects the authors' views only. The European Commission is not responsible for any use that may be made of the information it contains.

5. REFERENCES

[1] Christopher Berner et al., "Dota 2 with large scale deep reinforcement learning," *CoRR*, vol. abs/1912.06680, 2019.

[2] Nikolaos Passalis et al., "OpenDR: An open toolkit for enabling high performance, low footprint deep learning for robotics," in *Proc. IEEE/RSJ Intl. Conf. Intelligent Robots and Systems*, 2022.

[3] Thai-Viet Dang and Ngoc-Tam Bui, "Obstacle avoidance strategy for mobile robot based on monocular camera," *Electronics*, vol. 12, no. 8, 2023.

[4] Hamza Aydemir, Mehmet Gök, and Mehmet Tekerek, "Reinforcement learning based local path planning for mobile robot," in *Interdisciplinary Conf. Mechanics, Computers and Electrics*, 11 2021.

[5] Hado van Hasselt et al., "Deep reinforcement learning with double q-learning," *CoRR*, vol. abs/1509.06461, 2015.

[6] Yan Yin, Zhiyu Chen, Gang Liu, and Jianwei Guo, "A mapless local path planning approach using deep reinforcement learning framework," *Sensors*, vol. 23, no. 4, 2023.

[7] Volodymyr Mnih et al., "Playing atari with deep reinforcement learning," *CoRR*, vol. abs/1312.5602, 2013.

[8] Ali Salimi Sadr et al., "An efficient planning method for autonomous navigation of a wheeled-robot based on deep reinforcement learning," in *12th Intl. Conf. Computer and Knowledge Engineering*, 2022, pp. 136–141.

[9] Timothy P. Lillicrap et al., "Continuous control with deep reinforcement learning," *arXiv 1509.02971*, 2015.

[10] M. Kirtas et al, "Deepbots: A webots-based deep reinforcement learning framework for robotics," in *Artificial Intelligence Applications and Innovations*, Cham, 2020, pp. 64–75, Springer Intl. Publishing.

[11] Sanmit Narvekar et al., "Curriculum learning for reinforcement learning domains: A framework and survey," *arXiv 2003.04960*, 2020.

[12] John Schulman et al., "Proximal policy optimization algorithms," *arXiv 1707.06347*, 2017.

[13] Shengyi Huang and Santiago Ontañ ón, "A closer look at invalid action masking in policy gradient algorithms," *The Intl. FLAIRS Conf. Proceedings*, vol. 35, may 2022.

[14] Olivier Michel, "Webotstm: Professional mobile robot simulation," *Int. Journal of Advanced Robotic Systems*, vol. 1, 03 2004.