BALKAN JOURNAL OF ELECTRICAL & COMPUTER ENGINEERING, Vol. 9, No. 2, April 2021

Autonomous Navigation in Search and Rescue Simulated Environment using Deep Reinforcement Learning

Mohammed Abdeh, Fatih Abut, M. Fatih Akay

Abstract— **Human assisted search and rescue (SAR) robots are increasingly being used in zones of natural disasters, industrial accidents, and civil wars. Due to complex terrains, obstacles, and uncertainties in time availability, there is a need for these robots to have a certain level of autonomy to act independently for approaching certain SAR tasks. One of these tasks is autonomous navigation. Previous approaches to develop autonomous or semi-autonomous SAR navigating robots use heuristics-based methods. These algorithms, however, require environment-related prior knowledge and enough sensing capabilities, which are hard to maintain due to restrictions of size and weight in highly unstructured environments such as collapsed buildings. This study approaches the problem of autonomous navigation using a modified version of the Deep Q-Network algorithm. Unlike the classical usage of the entire game screen images to train the agent, our approach uses only the images captured by the agent's low-resolution camera to train the agent for navigating through an arena avoiding obstacles and to reach a victim. This approach is a much more relevant way of decision making in complex, uncertain contexts; since in real-world SAR scenarios, it is almost impossible to have the area's full information to be used by SAR teams. We simulated a SAR scenario, which consists of an arena full of randomly generated obstacles, a victim, and an autonomous SAR robot. The simulation results show that the agent was able to reach the victim in 56% of the evaluation episodes after 400 episodes of training.**

Index Terms— **Deep Reinforcement Learning, Autonomous Navigation, Autonomous Search and Rescue, Simulation.**

MOHAMMED ABDEH, is with the Department of Computer Science and Engineering, Sabancı University, Istabul, Turkey, (e-mail: mohammadabdeh@sabanciuniv.edu)

https://orcid.org/0000-0002-5450-1316

FATIH ABUT, is with the Department of Computer Engineering, Çukurova University, Adana, Turkey, (e-mail: fabut@cu.edu.tr).

https://orcid.org/0000-0001-5876-4116

M. FATIH AKAY, is with the Department of Computer Engineering, Çukurova University, Adana, Turkey, (e-mail: mfakay@cu.edu.tr).

https://orcid.org/ 0000-0003-0780-0679

Manuscript received August 16, 2020; accepted Feb 05, 2021.
DOI: 10.17694/bajece.781162

I. INTRODUCTION

EVERY YEAR, around 60.000 people die worldwide in natural disasters. The majority of the deaths are caused by building collapse in earthquakes [1]. Only a small fraction of the victims of urban disasters might survive. Consider from [2, 3] that 80% of survivors of urban disasters are surface victims, i.e., the people lying on the surface of the rubble or readily visible. However, only 20% of survivors of urban disasters come from the interior of the rubble, yet the interior is often where the majority of victims are located.

The complexity of the collapses and the need for a fast response after the disaster requires multiple tasks to be done simultaneously. One approach to solve this problem is the use of robots that have a variety of capabilities such as unmanned ground vehicles (UGVs) and unmanned aerial vehicles (UAVs) to assess human search and rescue experts. In other words, the number of human experts can be limited and does not serve the need for controlling the robots along with fulfilling the other critic tasks that require human interference and moral understanding. Hence, there is a need for supplying these robots with a certain level of autonomy to be able to navigate through a SAR area and thus provide some tasks without the need for human control. In [4], Murphy et al. describe the tasks that can be addressed. Those tasks include (a) reaching small and dangerous places (voids) that humans and dogs cannot reach, (b) exploring the interior and exterior of the hot zone and extending wireless communication ranges.

Navigation is one of the fundamental tasks of mobile robots. The algorithms used to approach this task are divided into two sets [5]:

- Greedy heuristics-based algorithms that operate using a map, which needs to be provided or extracted from the environment. After generating such a map, the central system performs the task of path planning, which is finding the optimal or suboptimal path to reach the target. Then the robots perform the navigation through the specified path. Autonomous mobile robots that use these techniques need to localize themselves within the generated map at the start of the planned trajectory, and then monitor their motion along the path using sensors (e.g., wheel encoders and inertial sensors) which are hard to be maintained on board of a small mobile robot navigating through small voids within rubbles.

- Algorithms that operate without the use of maps (e.g., DistBug); These algorithms are often used if a path has to be traveled only once and therefore does not necessarily have to be optimal. Although the Bug family algorithms use limited sensors and computation capabilities, they tend to rely on perfect position estimations. In [6], McGuire et al. concluded that the Bug algorithms performance is sensitive to sensor-related noise, which makes them hard to implement in real-world uncertain noisy cotexts.

In [7], the authors studied different search methods based on greedy heuristics, and partially observable Markov Decision Process (POMDP) to design the autonomous control of UAVs to act in search and rescue contexts. Their evaluation results showed the great potential of POMDP in SAR scenarios; they also acknowledged that using these methods requires a high computation power as the cost grows exponentially (i.e., the curse of dimensionality) and that existed algorithms were not sufficient yet to tackle real-world problems. With the rise of deep learning and the expanding computational power, researchers have been able to come up with algorithms that can be used to tackle the curse of dimensionality more efficiently and solve problems that were not intractable in the past, such as playing Atari games at a superhuman level [8] and defeating a 9-dan Go player [9].

The main purpose of this study is to approach the problem of autonomous navigation using a modified version of DQN by relying only on the agent's camera without the use of any environment-related prior knowledge or any info gathered from another source (e.g., human experts, other robots, maps). This decentralized approach using only the agent's limited vision-based capabilities is a relevant strategy in deep voids where wireless connections might be hard to maintain, and also in any other situation where no enough outside information is available in the process of decision making. The DQN algorithm modified and used in this work is a valid method of solving real-world problems that can be interpreted as a POMDP.

The rest of the paper has the following structure. Section II gives an overview of related works. Section III describes the theoretical foundations of the utilized learning algorithms. Section IV presents the simulation setup and evaluation methodology. Section V presents the results and discussion. Finally, Section VI outlines the paper along with future directions.

II. Related Works

One of the applications of autonomous navigation is creating 3D visualizations of buildings and large-scale construction sites using scanning technologies. The scanning process has been often carried out using scanners handled by people [10] or scanners on board of vehicles controlled by people [11]. In [12], Kim et al. assisted the scanning process with UAVs that generated an initial 3D map of the area, then based on this map, the optimal scanning locations are estimated, and the optimal navigation path is determined. After that, a robot equipped with a laser scanning system is used to follow the path and obtain high-resolution scans in the estimated locations. Although such techniques are powerful in navigating through large construction sites, they cannot be used to assess small UGVs navigating through small-scale indoor environments. These robots must perform navigation without any outside assessments due to the limitation of wireless sensing technologies.

The problem of autonomous navigation in search and rescue contexts has been approached previously using heuristics-based methods. A heuristic function ranks a set of options available at one point and decides which one to choose according to the ranking. These approaches require situation related prior knowledge and a wide range of sensing abilities. In the EU funded project ICARUS [13], several robots with different structures and capabilities have been designed to assess SAR experts. Small Unmanned Ground Vehicles (UGVs) used to navigate through narrow voids were supplemented with a very limited level of autonomy. This is caused by the use of heuristic algorithms that require a massive amount of sensors and power, which cannot be added to the vehicles due to the restrictions of size and weight in highly unstructured environments, such as collapsed buildings. A similar approach has been carried out in the SHERPA project [14] in which the goal was to develop a mix of ground and aerial robots to support SAR teams in real-world hostile environments.

DQN was implemented in [15] to support a UAV with the ability to navigate independently towards a specified goal in a simulated environment. They evaluated their work by comparing the success rate, and the average steps undertook by the trained agent and a random decision-making agent. This research resulted in an 8% success rate for the trained agent compared to a 5% success rate for the random-action taking agent. Although the algorithm showed some success as the trained agent outperformed the random agent in terms of success rate, it is clear that it can be further developed.

This study approaches autonomous navigation using a modified version of the DQN algorithm relying only on the agent's camera. Unlike the previous methods, this approach does not require a map to be generated or any extra sensors to monitor the agent's motion. While the recent implementation of DQN in autonomous navigation [15] used the same proposed CNN, the CNN (convolutional neural network) being used in this work for training the agent to make decisions based on the captured images has been designed with simplicity in mind to suit the simulated scenario.

III. Theoretical Background

A. *Reinforcement Learning*

RL is the machine learning paradigm concerned with training agents to act autonomously to reach certain goals in an environment. A typical reinforcement learning scenario consists of the following components: an agent (the decision-maker), an environment in which its state changes according to the agent's actions. As the agent explores the environment, it receives a reward for each action it makes. Hence by trial and error, it learns to maximize this reward.

Formally an RL scenario is modeled as a Markov Decision Process (MDP) which consists of a tuple (S, A, R, T, γ) where S is the set of states, A is the set of actions, (alternatively $A(s)$: the set of actions that can be taken at state s); R(s) is the reward function, it returns the immediate reward; $T(s, a, s') \sim P(s'|s, a)$ is the probability that taking action a will change the environment state from s to s'; and finally, $\gamma \in [0,1]$ is the discount factor, which is how much we value future reward over the immediate one.

The main problem of an MDP is to find the optimal policy π^*. A policy $\pi(s)$ is a function that specifies the action that should be taken at a certain state. Thus the optimal policy π^* is the policy that maximizes the expected long term reward (i.e., the reward from all states):

$$\pi^* = argmax_\pi \, E[\sum_{t=0}^{\infty} \gamma^t R(s_t)|\pi]. \quad (1)$$

Partially Observable Markov Decision Process (POMDP) is a generalization of Markov Decision Process (MDP). POMDP consists of a tuple$(S, A, O, \Omega, R, T, \gamma)$, where S is the set of states; A is the set of actions (or alternatively $A(s)$ is the set of actions that can be taken at state s); O is the set of observations; $\Omega(o, s', a) \sim P(o|s', a)$ is the probability of observing o at state s' after taking action a; $R(s)$ is the reward function, it returns the immediate reward; $T(s, a, s') \sim P(s'|s, a)$ is the probability that taking action a will change the environment state from s to s'; and finally, $\gamma \in [0,1]$ is the discount factor, which is how much we value future reward over the immediate one.

A POMDP scenario can be described as follows: at each time frame the environment is in some state $s \in S$, the agent takes action $a \in A$ which changes the environment state from s to s' according to the transaction probability T, after a step time, the agent receives an observation o from the environment state s'. The observation depends on the current state s' and the previous action a according to the probability $\Omega(o, s', a)$. The goal of the agent is to find the optimal policy π^* that maximizes the expected long-term reward. As the states in POMDP are not fully observable, the agents only receive partial information about the environment. This generalization is more realistic than MDP, and it can describe a variety of real-world problems. Hence it is an appropriate framework to describe the simulated scenario.

B. *Q-Learning*

Reinforcement learning methods fall into two categories; the policy-based and value-based methods. The policy-based methods aim at finding the optimal policy π^* by starting with a parameterized policy and updating its parameters using gradient-based or gradient-free methods, on the other hand, the value-based methods use a value function that determines the expected long term reward of executing an action at a particular state, and hence providing which action to undertake. One of the most well-known value-based methods is Q-learning. It can find the optimal policy in finite stationary and fully- observable environments with discrete actions. The algorithm starts by initializing a table called the Q-table with arbitrary Q-values. A q-value is a value that corresponds to how well an action is at a specific state. Then at each state, an action a is being selected, and its corresponding Q-value is being calculated using the Q-function:

$$Q^{new}(s, a) = Q(s, a) + \alpha[r + \gamma \, max_{a'} Q(s', a')) - Q(s, a)] \quad (2)$$

where Q(s, a) is the Q-value of executing an action a at state s, $\alpha \in (0,1]$ is the learning rate, and r is the immediate reward. Although Q-learning can find the optimal policy in small finite environments, it performs poorly on non-stationary partially observable complex environments.

C. *Deep Q-Network*

With the rise of deep learning and particularly in [8], Mnih et al. introduced the Deep Q-Network algorithm (DQN), which gave birth to the field of deep reinforcement learning. DQN was the first algorithm to reach a superhuman performance in playing Atari games from pixels. Instead of using a table to track the Q-values, they used a convolutional neural network as a function approximator to predict Q-values from states. One of the problems the RL algorithms have when using artificial neural networks as function approximators is the problem of instability. The DQN algorithm uses two separate models to stabilize the learning; namely, the main model and the target model, the main model's weights get copied to the target model every c steps. The DQN also keeps a memory buffer termed Experience Reply to store the transactions. At every training step, the DQN uses the main model to predict the action to take, and then the agent executes the action and observes a new state together with the immediate reward. After adding the new transaction to the experience replay, it takes random samples from the experience reply and uses them as data to fit the main model by minimizing the mean-square error (MSE) between the main and target models. The training process is visualized in Fig. 1.

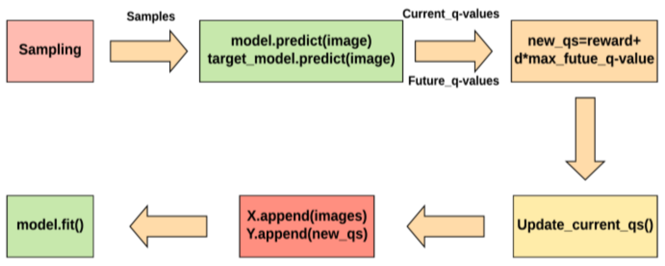


Figure 1. Visualization of one training step

D. *The ε-Greedy Policy*

The ε-greedy policy is a method used to balance the exploration and exploitation done by the agent; that is deciding at a certain point whether it should take random actions to explore unexplored space, or whether it should use the trained model's action-value estimations to predict the best action. It keeps track of a parameter ε, which represents the probability of choosing a random action. Similar to the original DQN, in this work, ε starts at 1 and decreases linearly throughout the training until it reaches a fixed point.

IV. Simulation Setup and Methodology

The scenario was simulated using Webots [16], an open-source robot simulator widely used for educational and professional purposes. It includes a complete toolbox to design environments, robots, and simulation experiments. It can execute controllers written in compiled C/C++ and Java or interpreted python and Matlab languages. In this study, the agent controller was written in python to take advantage of the required libraries (i.e., Numby, Tensorflow, and Keras).

An e-puck robot was used for its motion simplicity. It has two motors, a left wheel motor and a right wheel one. It also has a color camera with a maximum resolution of 640x480, eight light sensors, and eight distance sensors. For this research, only the camera and the motors have been used.

At the beginning of the simulation, a square arena is created. The center of the arena is at the center of the coordination, and the objects can be placed within [-1, 1] and [-1, 1] at X and Z dimensions, respectively, whereas the Y dimension remains the same. At every episode, the agents and obstacles are placed randomly. The victim robot is randomly placed according to the uniform distribution at a position within the upper left part of the arena between [-1, -0.5] and [-1, -0.5] at X and Z dimensions, respectively. Similarly, the agent is placed randomly according to the uniform distribution at the downright part of the arena between [0.5, 1] and [0.5, 1] at X and Z dimensions, respectively. The obstacles (i.e., 20 boxes) are positioned randomly according to the normal distribution of 0 mean and standard deviation of 0.5. The objects are positioned in this manner to ensure that the obstacles will get in the agent's way. Fig. 2 shows a view of the arena at the beginning of an episode.

In this work, DQN has been modified to ensure more success in the simulated scenario. These modifications are as follows:

- The CNN that has been used in the original DQN has performed poorly on the simulated scenario. The agent at an early level of the training starts to memorize one action regardless of the image fed into the network, and hence it rarely reaches the victim robot. Instead, in this work, a simpler CNN has been used. It showed much better results. Fig. 3 shows the architecture of the modified CNN, which is used to predict the best action to be executed according to the image fed.

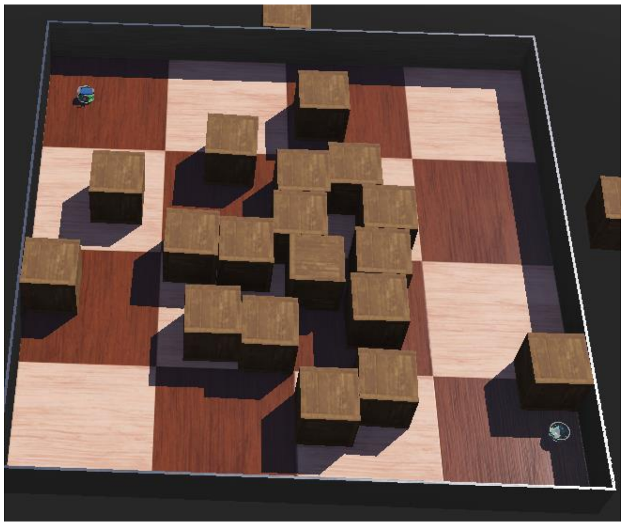


Figure 2. A bird view of the arena at the beginning of an episode. The trained SAR agent is at the downright, and the victim is located up at the left most of the image.

- The agent was trained for 400 training episodes executed in 8 identical parts; within each part, 50 episodes were executed. At each training part, the ε-greedy policy's ε starts from 1 and decays until it reaches 0.1 after 25 episodes, which is half of a training part, as illustrated in Fig. 4.
- While in the original DQN, a stack of 4 frames is being fed into the CNN as one observation, for simplicity, we use only one image. Fig. 5 shows an image captured by the agent during training.
- The use of a simple distance-dependent reward function:

$$reward = 100(last_d - current_d) \quad (3)$$

where $last_d$ is the distance between the agent and the victim at the previous step, and $current_d$ is the distance between the agent and the victim at the current step.

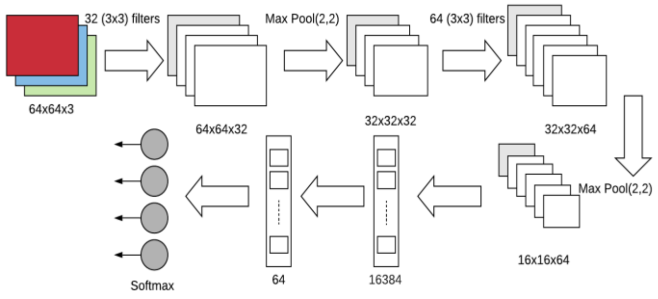


Figure 3. The architecture of the modified CNN used to train the agent to predict actions.

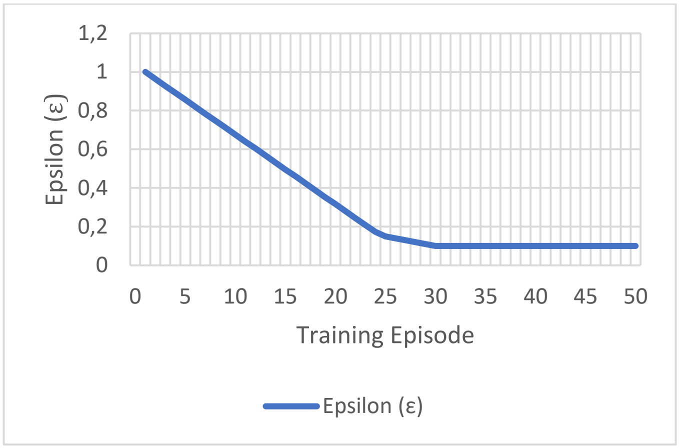


Figure 4. Linear change of ε throughout a single training part

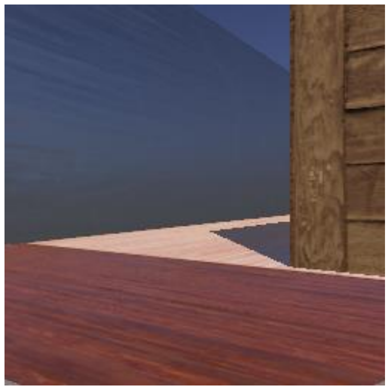


Figure 5. An image captured using the agent's camera. Images of this type were used in training and testing.

V. Results and Discussion

Several evaluation methods were used to test the suitability of the original CNN architecture proposed in [8]. The original CNN was designed to learn generalizing the learning process over many distinct Atari games, while in our scenario, the agent has to learn to interact in one environment. In other words, due to the complexity of the model, the agent starts to memorize behaviors at a very early level, and thus at one point, it starts to take only one action regardless of the image captured. This resulted in a bad performance in terms of navigating towards the target.

Fig. 6 illustrates the number of steps taken by the agent to reach the victim in a single episode by using the original CNN. If the agent does not reach the victim during the episode, the finishing step will be equal to the maximum number of steps used (i.e., 1000). This is due to the low ε value 0.1 which limits the agent exploration. The line shows that using the original approach the agent was only able to reach the victim before the episode 25 (i.e., the episode at which the randomness is minimized), which demonstrates clearly that the agent is not able to reach the victim when relying on the original complex CNN and hence performs poorly in the simulated environment.

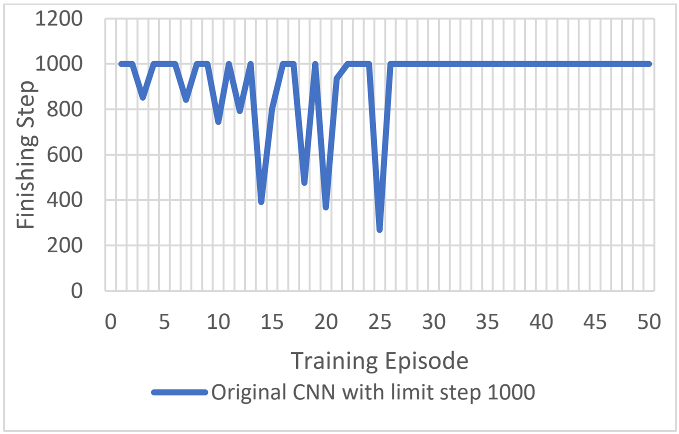


Figure 6. The finishing steps of the agent when trained using the original CNN

Using our modified CNN proposed in the methods section, the agent showed better results when evaluated in 1000 step epochs. Fig. 7 illustrates the number of steps taken by the agent to reach the victim in a single episode by using the modified CNN. If the trained agent does not reach the victim during the episode, the finishing step will be equal to the maximum number of steps used (i.e., 1000). The line shows that using the modified approach, the agent was able to reach the victim at most of the episodes, while also reaching the victim in less number of finishing steps on average after the episode 25. This, in turn, clearly demonstrates that the agent is able to reach the victim more efficiently relying only on the trained model using the modified CNN.

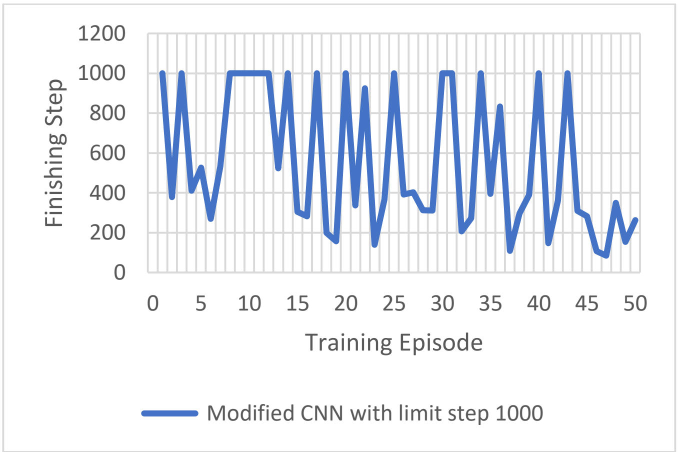


Figure 7. The finishing steps of the agent when trained using the modified CNN

The finishing step is going smaller along with the training. Since the ε is decaying linearly, this shows that the agent is performing better when it predicts the actions based on the trained model. Table 1 shows the training results of the agent during the different parts of the training.

TABLE 1. THE TRAINING RESULTS OF THE AGENT DURING THE DIFFERENT PARTS OF THE TRAINING

Training Part (Episodes)	Success Rate	Minimum Steps
0 - 50	24%	42
51 - 100	40%	38
101 - 150	46%	65
151 - 200	52%	45
201 - 250	60%	50
251 - 300	50%	68
301 - 350	64%	65
351 - 400	46%	59

Note: The success rate shows the percentage of the successful episodes, in which the agent successfully reaches the victim, whereas the minimum steps show the minimum step number at which the agent reached the victim.

TABLE 2. COMPARING THE VALIDATION RESULTS OF THE AGENTS WITH PREVIOUS WORK

Study	Agent	Average Steps	Success Rate	Minimum Steps
The results of [15]'s approach	Random Agent	22.35	5%	-
	Trained Agent	258.54	8%	-
The results of this study	Random Agent	400 (the limit)	0%	400 (the limit)
	Trained Agent (400 episodes)	242.44	56%	21

Using the methods proposed above, the agent was trained on an Intel Core (TM) i5-3230M CPU. Training 400 episodes lasted appr. 21 hours and 7 minutes. The training episodes were executed in 8 parts to ensure the stability of the simulation physics engine (i.e., after training for more than 50 epochs, the physics engine starts to produce some delay and the simulation info file stored in the RAM reaches its limits).

After the training, the agent's performance was evaluated using the same evaluation method used in [15]. The trained agent's success rate and the number of steps it took to finish the simulation were calculated and compared with a fully random-action taken agent. Table 2 shows a comparison between the evaluation results of this work and the previous study.

While the random agent was not able to reach the victim even once, after 400 episodes of training, the agent was able to reach a 56% success rate in 40 evaluation episodes. That means that the e-puck robot was able to reach the victim in 56% of the evaluation episodes. Moreover, it was able to reach the victim in 21 steps at least in one episode, which is a great result giving the complexity of the environment. The poor performance of the random agent in our study shows the complexity of our simulated scenario when compared with the simulation scenario in [15]. Hence, this proves how well the trained agent performed.

Compared with the results proposed in [15], this work has reached significant improvements in terms of the difference between the results of the random agent and the trained one. This is due to the use of a simpler CNN that learns to navigate through obstacles contained in similar settings, instead of the use of the original CNN that generalize the learning over many Atari games. Another factor is the used version of the ε-greedy policy; that is partitioning the training into parts with an ε that starts at each part from 1 and linearly decays until it reaches a 0.1. With this method, the agent has been able to explore the changing state of the environment across episodes more efficiently.

VI. Conclusion and Future Work

This paper presented a SAR simulation scenario, in which an autonomous e-puck robot's main goal is to explore an area full of randomly generated obstacles and yet to learn to navigate through them to reach a certain victim without any use of environment-related prior knowledge. The scenario and the environment, in general, can be used to test the suitability and the efficiency of different deep reinforcement learning methods to solve autonomous SAR problems. A modified DQN algorithm was implemented using some new approaches. These approaches include (a) the use of modified CNN instead of the original one, a distance-dependent reward function, and agent's low-resolution camera images to train the agent; (b) feeding one frame to the CNN as an observation instead of a stack of four images; and finally (c) training the agent in 8 parts; at each part, the ε-greedy policy's ε is decaying linearly from one until it reaches 0.1 at the middle of the training part. Using these methods, our modified DQN has shown great results compared to previous work when evaluated using the same evaluation methods and metrics, reaching a success rate of 56% at an average step of 242.

Despite the work's success, a variety of future works can be conducted to increase efficiency and real-world interpretability. The scenario can be further developed to become more realistic; e.g., by placing a variety of obstacles, increasing the number of obstacles and victims. A more complex CNN can be designed to train the agent to navigate in the developed complex scenario. The reward function can be optimized to depend on other factors, such as the distance between the agent and the nearest obstacle. Finally, as the number of victims increases together with the complexity of the environment, using a multi-agent SAR team will be a relevant strategy. Agents should learn to cooperate as a team to approach a common goal.

REFERENCES

[1] C. Kenny, *Why Do People Die In Earthquakes? The Costs, Benefits, And Institutions of Disaster Risk Reduction In Developing Countries*. Policy Research Working Paper 4823. The World Bank, 2009.

[2] "NFPA 1670, Standard on Operations and Training for Technical Rescue Incidents - National Fire Protection Association". https://www.nfpa.org/codes-and-standards/all-codes-and-standards/-list-of-codes-and-standards/detail?code=1670&tab=nextedition (accessed Aug. 03, 2020).

[3] "Rescue: Technical Rescue Program Development Manual", U.S. Fire Administration and Federal EmergencymManagement Agency, ISBN: 1482709600.

[4] R. R. Murphy *et al.*, "Search and Rescue Robotics," in *Springer Handbook of Robotics*, 2008, pp. 1151–1173.

[5] T. Bräunl, "Localization and Navigation," in *Embedded Robotics*, Springer Berlin Heidelberg, 2008, pp. 241–269.

[6] K. N. McGuire, G. C. H. E. de Croon, and K. Tuyls, "A comparative study of bug algorithms for robot navigation," *Rob. Auton. Syst.*, vol. 121, p. 103261, Nov. 2019.

[7] S. Waharte and N. Trigoni, "Supporting search and rescue operations with UAVs," in *Proceedings - EST 2010 - 2010 International Conference on Emerging Security Technologies, ROBOSEC 2010 - Robots and Security, LAB-RS 2010 - Learning and Adaptive Behavior in Robotic Systems*, 2010, pp. 142–147.

[8] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[9] D. Silver *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016.

[10] J. Xiao, A. Owens, and A. Torralba, "SUN3D: A database of big spaces reconstructed using SfM and object labels," in *Proceedings of the IEEE International Conference on Computer Vision*, 2013, pp. 1625–1632.

[11] I. Toschi, P. Rodríguez-Gonzálvez, F. Remondino, S. Minto, S. Orlandini, and A. Fuller, "Accuracy evaluation of a mobile mapping system with advanced statistical methods," in *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences - ISPRS Archives*, 2015, vol. 40, no. 5W4, pp. 245–253.

[12] P. Kim, J. Park, Y. K. Cho, and J. Kang, "UAV-assisted autonomous mobile robot navigation for as-is 3D data collection and registration in cluttered environments," *Autom. Constr.*, vol. 106, p. 102918, Oct. 2019.

[13] "ICARUS Project.", European Union's Horizon 2020 research and innovation programme, https://icarus2020.eu/ (accessed Aug. 03, 2020).

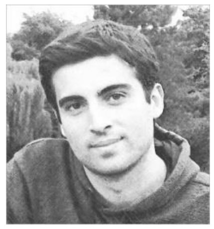
[14] L. Marconi *et al.*, "The SHERPA project: smart collaboration between humans and ground-aerial robots for improving rescuing activities in alpine environments," 2012.

[15] J. G. C. Zuluaga, J. P. Leidig, C. Trefftz, and G. Wolffe, "Deep Reinforcement Learning for Autonomous Search and Rescue," in *Proceedings of the IEEE National Aerospace Electronics Conference, NAECON*, Dec. 2018, vol. 2018-July, pp. 521–524.

[16] O. Michel, "Cyberbotics Ltd. Webots TM : Professional Mobile Robot Simulation," 2004. Accessed: Aug. 03, 2020. [Online]. Available: http://www.cyberbotics.com.

BIOGRAPHIES

Mohammad ABDEH received B.Sc. degree in Computer Engineering from Çukurova University, Turkey, in 2020. He had been awarded an undergraduate scholarship from Turkey Scholarships in 2017. He is currently a M.Sc. student and a Teaching Assistant at the Department of Computer Science and Engineering at Sabanci University, Istanbul, Turkey. His research interests include artificial intelligence, reinforcement learning, and autonomous search and rescue robotics.



Fatih ABUT received B.Sc. and M.Sc. degrees in Computer Science from Bonn-Rhein-Sieg University of Applied Sciences, Germany, in 2008 and 2010, respectively; and a Ph.D. degree in Computer Engineering in the year 2017 from Çukurova University, Adana, Turkey. He is currently an Assistant Professor in the Department of Computer Engineering at Çukurova University. His research interests include machine learning and artificial intelligence, computer and wireless communications, and IP network measurement and characterization.

Mehmet Fatih AKAY received B.Eng. and M.Eng. degrees in Electrical and Electronics Engineering from Çukurova University, Turkey, in 1999 and 2001, respectively; and a Ph.D. degree in Electrical and Computer Engineering in the year 2005 from Drexel University, Philadelphia, USA. He is currently a Professor in the Department of Computer Engineering at Çukurova University. His research interests include machine learning and artificial intelligence, simulation, and modeling of multiprocessors.