Hard Exploration Robotic Task in an Uneven Terrain via Reinforcement Learning

Kieren Samarakoon
Department of Mechanical and Aerospace Engineering
West Virginia University
Morgantown, USA
kys0002@mix.wvu.edu

***Abstract*—As the application of mobile robots in dangerous, dull, and difficult scenarios become more viable, approaching these complex problems with a deep reinforcement learning (DRL) method seems compelling and appropriate. Whether it is a search and rescue mission in a cave or a difficult exploration task, robots need to be able to to search their environment effectively, especially if they are designed to be autonomous. This paper addresses the task of training a wheeled robot end-to-end using DRL to navigate an unknown environment with uneven terrain relying only on information from an on-board camera. The robot was trained in a simulated Gazebo environment with a discrete action space provided by OffWorld Gym.**

***Index Terms*—policy, rewards, state-space, action, Q-learning, value function, optimal function**

I. Introduction

Hard Exploration tasks involving Reinforcement Learning (RL) are problems that involve sparse or deceptive rewards which require exploration to solve [1]. As exploration tasks are common for mobile robots, it is important to ensure they can carry out their missions safely. Regardless of whether the robot is semi-autonomous or fully autonomous, often these difficult tasks require the robot to navigate the environment without prior knowledge of the environment and rely solely on their on-board sensors. A traditional approach to this problem is with SLAM; using a visual/ranging sensor to map an environment and localize within it [2]. But recently, it has been found that if the robot (Agent) is trained in a simulation and exposed to an order of magnitude more experience, DRL was shown to have more robust navigation policies than SLAM [3]. The motivation of this paper was to assess the implementation of an algorithm suited for difficult exploration tasks and compare the results with a Deep Q-Network (DQN) algorithm. The algorithm that was compared was the Go-Explore algorithm [4].

Applying DRL to these complex issues addresses many flaws in the SLAM approach, especially that of lacking a humane approach [5]. Reinforcement Learning approaches the problem in terms of an Agent (robot) at a current state (e.g. location) taking actions to receive rewards [2]. Take driving a car as an example. The state is the location and speed of your car, turning the steering wheel and pressing the accelerator or brake are the actions. The reward is contingent on how fast you arrive at your destination . How does this translate to reinforcement learning? If we are able to determine what actions maximize our rewards (in this case arriving at our destination in the shortest time) then we can ultimately learn an optimal policy. Of course this is quite a simple example, but if our problems were to become more abstract the order of magnitude of actions and states become grand. Taking the same car example, if we drive in the same traffic conditions to the same destination we would eventually learn how to drive efficiently, but what if we wanted to drive to a different location? The same actions we took would not give us the same reward. This also applies to RL and is still being researched. This is where simulated environments play a crucial role.

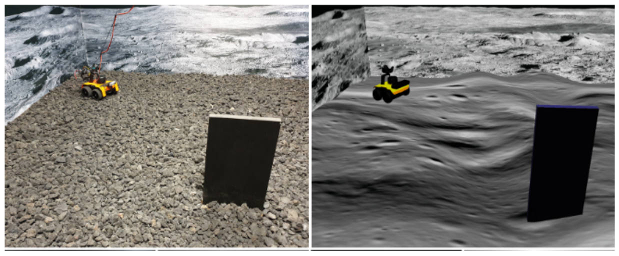


Fig. 1: Physical environment (left) and the simulated environment (right) from the of the *MonolithDiscrete* worlds from A. Kumar et al. (2020)

The simulated environment used for this project was provided by OffWorld Gym which is supposed to visually represent the surface of the moon [6]. By ensuring that the environment remains constant, we are able to evaluate the optimal policy without worrying about varying states or actions. This means all efforts can go strictly to improving the DRL algorithm. What is unique about this environment is that it there is a real-world representation of the simulated environment that can be accessed. This allows users to test their developed algorithm in a real environment which is often what the process of testing DRL algorithms lack. Since training a deep learning model requires a lot of data, it is beneficial to have a simulation and physical environment that are exact representations of each other. This way if the algorithm needs to be altered, it can be done so without being introduced to a new environment [5].

II. Related Work

Existing work for this topic is quite extensive as it is a complex problem with many different approaches. This section highlights the work that investigated tasks involving an unknown environment, unknown terrain, and visual sensors all with a RL emphasis.

Surmann et al. [6] presented a project of an indoor turtlebot that was able to avoid obstacles without a known map through the use of an RGB-D camera, laser scan and a deep reinforcement approach. They showcase the integration of a huge parallel environment simulation for mobile robots into the deep reinforcement learning process. This was done by creating a map using a SLAM package, and rewarding the Agent depending on how its location with respect to the goal.

Zhang et al. [7] investigated the use of DRL for a robot navigation in environments with unknown rough terrain. They used depth images, elevation maps and 3D orientation as inputs for their A3C Network Design to determine optimal robot navigation actions. Their results showed that the DRL approach with an end-to-end learning strategy successfully navigated an environment towards a target location.

Zhu et al. [8] used a target-drive navigation model to approach a target specified by an RGB with no known prior map or reconstruction. It used rendered images that were streamed to a deep learning framework which issues a control command based on the visual input and sends it back to the Agent. Their experiment was able to show that method generalized to new targets and scenes that were not used during training of the model.

Zhang et al. [9] demonstrated a DQN approach for a controller to provide discrete actions to an Agent to navigate a known maze-like environment relying only on its one RGB-D camera input. After being trained once, the controller was moved to a new environment on a physical robot and trained again. The results showed that the length of training is less than having to train a new network. In fact, the model was able to navigate both the simulated and real environment.

S. Josef and A. Degani [10], presented a method with reward shaping, which provides a dense reward in a goal-oriented task. Their robot was able to locally navigate and avoid binary obstacles in an unknown rough terrain with zero-range to local-range sensing using a DRL based method.

III. Methods

There were two main components to this task, first being the environment and second the algorithms.

A. Environment

The environment (Fig. 3) was provided by OffWorld Gym [6] and featured a wheeled robot on an uneven Moon-like terrain (3mx4mx2m) enclosed by four walls with a monolith at the centre. The surrounding walls are also textured to represent the Moon. The robot is equipped with one RGB-D camera with a 320 x 420 resolution to gather information. An option to choose only RGB, only depth or a combination of both is available in the offworld_gym.envs.common package. To change the image type, the code was altered depending on which type was going to be used.

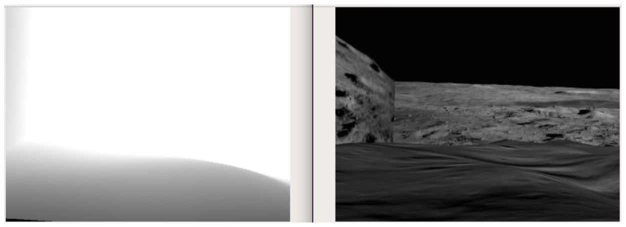


Fig. 2: Depth image (left) and the RGB image (right) from the RGB-D on-board the mobile robot

TABLE I: Parameters to choose the image type

Camera Resolution	
RGB	image(240, 320, 3)
Depth Only	image(240, 320, 1)
RGB Only	image(240, 320, 4)

Since the experiment was approached as an RL problem, it was important to model it as one. We have our Agent (the robot), state (location) and reward. The reward was predetermined by the environment chosen. In our case the *MonolithDiscreteSim* environment was used which features a discrete action space with four actions. These actions are left, right, forward and backward. When the robot reaches the monolith within a distance of 40.0 cm, it receives a sparse reward of +1.0. After completing the mission, meeting the 100-step cap, or entering the environment's boundary, the environment is reset [6]. The robot is shifted to a random location and into a random orientation after each reset [6]. The environment can be easily imported to a code by changing the name of the environment. This environment is unique because there is a real physical representation of this environment that users can connect to to test their algorithm on a real system.

B. Algorithm

To test OffWorld Gym's *MonolithDiscreteSim*, a code from OffWorld Gym was used to initialize it. This world does not integrate any deep learning. Because the environment has a vast number of states which overwhelms the capacity of contemporary computers, regular Reinforcement Learning would not be ideal to approach this problem [11]. Alternatively, DQN maps environment states to Agent actions. This allows the Agent to navigate an unstructured world and obtain knowledge, allowing them to imitate human behaviour overtime [3]. By approximating the optimal action-value function DQN works as and end-to-end RL approach only needing the pixels and game score as inputs (for Atari games) [12]. This means only minimal data is required. This is ideal for our problem as the input from our RGB-D camera (pixels) and location (score) is all the information that we have.

Because the Agent is intrinsically motivated, it will explore new behaviours for it's own benefit instead of directly solving the problem [3][13]. This comes with ”detachment” and

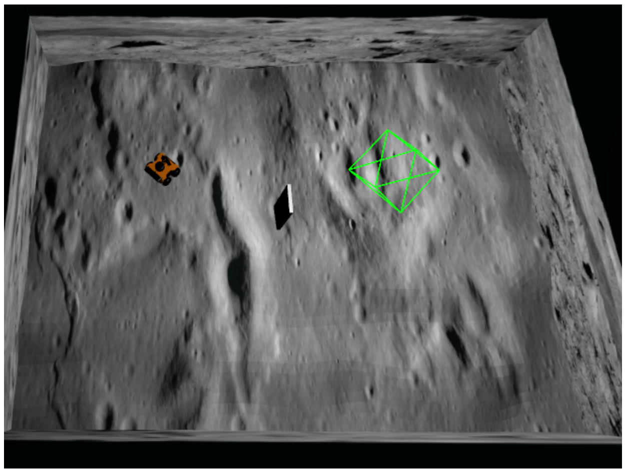


Fig. 3: Aerial view of the *MonolithDiscreteSim* environment from OffWorld Gym depicting the robot, monolith (center of world) and lunar terrain.

```
Input: the pixels and the game score
Output: Q action value function (from which we obtain policy and select action)
Initialize replay memory $D$
Initialize action-value function $Q$ with random weight $\boldsymbol{\theta}$
Initialize target action-value function $\hat{Q}$ with weights $\boldsymbol{\theta}^- = \boldsymbol{\theta}$
for episode = 1 to $M$ do
    Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequence $\phi_1 = \phi(s_1)$
    for $t$ = 1 to $T$ do
        Following $\epsilon$-greedy policy, select $a_t = \begin{cases} \text{a random action} & \text{with probability } \epsilon \\ \arg\max_a Q(\phi(s_t), a; \boldsymbol{\theta}) & \text{otherwise} \end{cases}$
        Execute action $a_i$ in emulator and observe reward $r_t$ and image $x_{t+1}$
        Set $s_{t+1} = s_t, a_t, x_{t+1}$ and preprocess $\phi_{t+1} = \phi(s_{t+1})$
        Store transition $(\phi_t, a_t, r_t, \phi_{t+1})$ in $D$
        // experience replay
        Sample random minibatch of transitions $(\phi_j, a_j, r_j, \phi_{j+1})$ from $D$
        Set $y_j = \begin{cases} r_j & \text{if episode terminates at step } j+1 \\ r_j + \gamma \max_{a'} \hat{Q}(\phi_{j+1}, a'; \boldsymbol{\theta}^-) & \text{otherwise} \end{cases}$
        Perform a gradient descent step on $(y_j - Q(\phi_j, a_j; \boldsymbol{\theta}))^2$ w.r.t. the network parameter $\boldsymbol{\theta}$
        // periodic update of target network
        Every $C$ steps reset $\hat{Q} = Q$, i.e., set $\boldsymbol{\theta}^- = \boldsymbol{\theta}$
    end
end
```

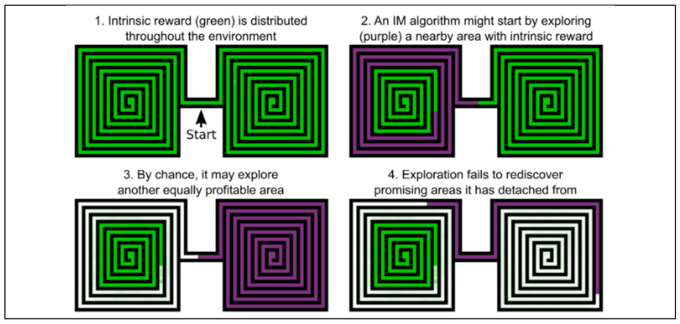
Fig. 4: Deep Q-Network(DQN),pseudo code obtained from Mnih et al. (2015)

"derailment." Detachment is when the Agent stops getting rewards after exploring a promising path, thus it forgets other paths it could have taken. A graphical explanation is shown in Figure 5. Derailment is when a promising policy is taken, and perturbed hoping to explore further. But random perturbations to the policy does not guarantee reaching that promising policy [3]. These two problems are addressed in Go-Explore's Algorithm.

The Go-Explore algorithm approaches the problem in two phases, first solve the problem, then pay extra cost to robust only the policy that we know leads to high rewards (Fig. 6)[13]. Choosing a cell to explore from is the first step in phase 1 of the algorithm [3]. A cell is a representation of the agent as well as the current state of the game. The contents of the cell are as follows:

- Agent's location.
- Actions of the Agent that led to this location.
- The reward obtained at this point.
- Attributes that determine which cell to explore from.

These attributes contribute to the cell's score, which ultimately


1. Intrinsic reward (green) is distributed throughout the environment
Start
2. An IM algorithm might start by exploring (purple) a nearby area with intrinsic reward
3. By chance, it may explore another equally profitable area
4. Exploration fails to rediscover promising areas it has detached from
Fig. 5: A hypothetical example of detachment in intrinsic motivation algorithms from Ecoffet et al. (2021)

contributes to the probability of each cell being selected. There are two scores, the cell count score and neighbor score. The cell count score is determined from:

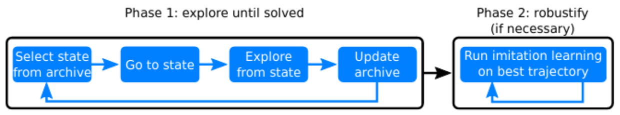
- Number of times a cell has been chosen to be explored from.
- Number of times the cell was visited during training.
- Number of times since choosing of this cell has led to the discovery of a new or better cell.

$$CntScore(c, a) = w_a * (1/v(c, a) + \varepsilon_1)^p + \varepsilon_2$$

The neighbor score increases depending on whether an undiscovered cell contains a reward and whether this cell is a vertical or horizontal neighbor [3].

$$NeighScore(c, n) = w_n * (1 - HasNeighbor(c, n))$$

The final score of a cell is the sum of these two sub scores. Upon implementing the algorithms, the intent was to book a


Phase 1: explore until solved
Phase 2: robustify (if necessary)
Select state from archive
Go to state
Explore from state
Update archive
Run imitation learning on best trajectory
Fig. 6: A high-level overview of the Go-Explore algorithm from Ecoffet et al. (2021)

time on OffWorld Gym's server to test the algorithms in a real environment. In order to do this, you had to book a time with your account, and then run the *MonolithDiscreteReal* script.

IV. Experiments

The purpose of these experiments was to test the DQN algorithm and compare the results with another algorithm suited for a hard exploration task. The initial parameters for the DQN algorithm included 1 million steps using only the depth camera. The first run took two days to run. The length of the training proved to be a major obstacle because there was only one computer to run the simulation. Because the time was too long, another test was conducted with the same parameters but the number of steps was reduced from 1 million to only ten thousand. Reducing the number steps had a significant impact on the duration of the simulation which reduced the real training time from approximately 48 hours to varying from three to seven hours. This enabled more tests of the algorithm but sacrificed learning time for the Agent. As expected the result was unfavourable.

Before comparing another algorithm, an intermediate step to compare the input image was added. Three separate tests were done here, a test with RGB, RGB-D and depth only (Fig. 9). According to an experiment done by Savva et al. [4], depth only input is superior in terms of SPL and training steps to the other two image types. This can be seen in the Figure 7.

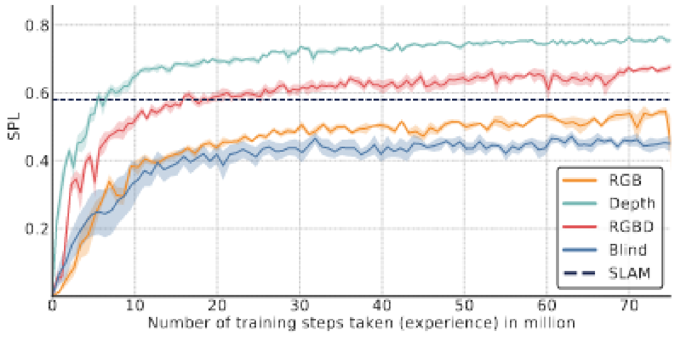


Fig. 7: Average SPL of Agents over the course of training in an experiment by Savva et al. (2019) that shows the effectiveness of RL learning in navigation methods

The goal after running the DQN algorithm was to implement the Go-Explore algorithm and compare the results. A problem with phase 1 of the Go-Explore algorithm is that it needs to store a cell for every single location that was visited. This was a difficult obstacle to overcome when trying to implement the code. As a result, the Go-Explore algorithm was never tested. There was also trouble accessing the real environment which resulted in the real-world test not being completed.

V. Results and Discussion

We wanted to answer the question, can we train a mobile robot end-to-end to solve a hard exploration task solely replying on a single camera input? Figure 8. shows that with 1 million training steps, the Agent converges close to have an average reward of approximately 0.8. This shows that the DQN is effective in teaching an agent how to navigate through an uneven terrain to reach an optimal policy. If we compare Figure 8. and Figure 9.c there is a clear correlation between training time and rewards received. Prior to reaching 10,000 steps (Fig. 9), the agents average reward were still erratic whereas in Figure 8. the average reward is within the proximity of 0.8-0.7. This means that the robot is receiving more rewards per episode, which implies that it is learning how to approach the monolith seen in Figure 3.

Figure 9 does show that using a specific image input makes a difference, but there is not enough data to determine whether a mobile robot can be trained effectively using this method. Although it is possible that with more steps the results may improve, this in contrary supports the notion that it would not be feasible in real world applications. After 1 million steps, the average reward was around 0.7. What if the training continued for another 1 million steps? Would it converge to 0.9? We do not have the knowledge to determine the length of training that is required to obtain a desirable outcome. This brings up the issue of sample efficiency which is a key challenge in applying RL into real-world scenarios. Figure 7 shows an experiment performed by Savva et al. (2019) were over 70 million steps were completed and the SPL stayed consistent after approximately 15 million steps. Although the type of experiment is a factor in determining the number of steps to converge, this demonstrates the magnitude of length it can take to train an Agent.

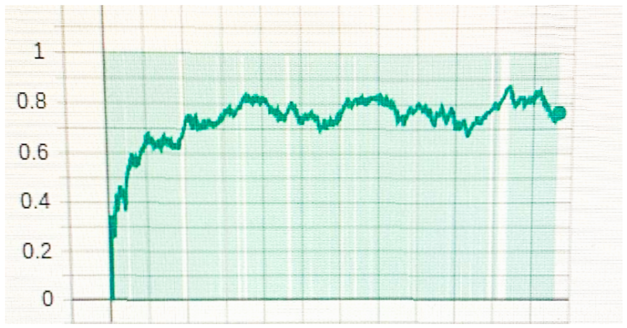


Fig. 8: Average score of the Agent over the course of training using 1 million steps and depth only input

In Figure 9 we can see that ”Depth Only” seems to provide better data than the other images. This is evident as more rewards were earned over the same course of training. That being said, it is still difficult to tell without any signs of convergence. By referring to Figure 10 we can infer that using only the depth image resulted in the Agent taking less steps per episode that the other two image types. And if it rarely reached the 100 step count, it must have either 1) collided with a boundary or 2) successfully completed the goal of approaching

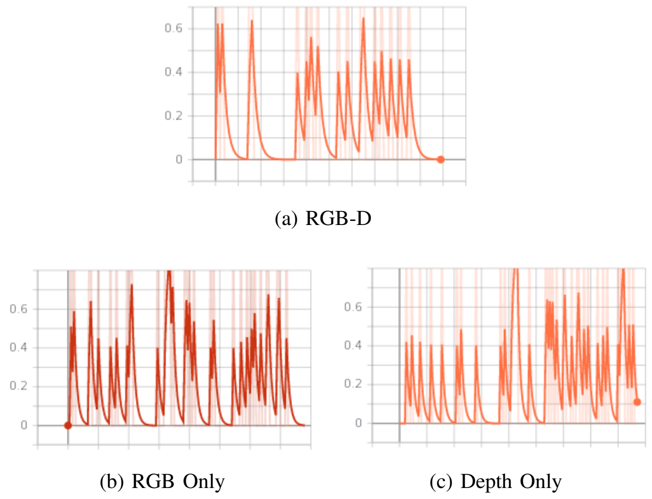
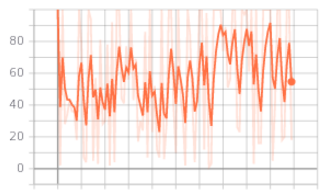
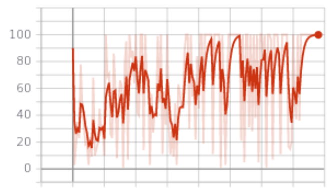


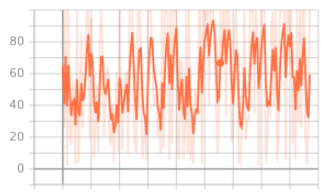
Fig. 9: Average score of the Agent over the course of training using ten thousand steps



(a) RGB-D



(b) RGB Only



(c) Depth Only

Fig. 10: Average steps of the Agent over the course of training using ten thousand steps

the monolith. This showcases another key challenge of RL which is safe explorations. In a simulation there is no real damage to an agent, it's position can be reset with minimal effort. But in a real environment, if our mobile robot was to collide with the boundary is there a guarantee that it can return to it's position without damage [13]? To improve on this algorithm, a parameter should be implemented where the robot will reset if it is within a threshold of outside boundary.

Although the exploration area was small, how can we justify the length of the training? The sparse reward of +1.0 when the robot approaches the monolith does not give any feedback on whether the actions taken were good or bad. So although it received a reward, it does not know what to do. In order to address this reward sparsity, additional feedback signals can be implemented to help the Agent learn [13]. These can be designed so that when the Agent succeeds in the task,it receives a reward and also knowledge that will be useful in succeeding the overall goal. It is a shame that we were not able to connect to the real environment even though all of the steps to do so were taken with precaution. This would have given great insight on whether the algorithm that eventually converged in the sim would do so in the physical environment.

VI. Conclusion

In this paper, the application of DRL to a hard exploration task in an uneven terrain was demonstrated. Regarding the methodology, it could be categorized into two, the environment and the algorithm. The environment provided a discrete action space to test the algorithms implemented and also had predefined states, actions and rewards. The basic DQN was tested and showed promising results with the reward of the Agent showing signs of convergence. When the same algorithm was carried out with less steps it did not showcase any promising results, even when the camera input image was altered. It did however demonstrate that using only depth provided better results than compared to RGB and depth only. The Go-Explore algorithm proved difficult to implement into the *MonolithDiscreteSim* world as all of the locations visited had to be stored in a cell. The results of the DQN proved that the problem of sparse rewards must be addressed as the Agent does not receive enough feedback on whether it is taking the right action to reach the goal. To improve this project, firstly multiple simulations should be run in order to collect as much data as possible. And paramters should be implemented to the algorithm to ensure that what the agent learns in the simulated world can be translated to the physical world.

Acknowledgment

I would like to thank Dr. Ali Baheri for his guidance on this project and also providing useful resources to access.

References

[1] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. Efros "Large-scale study of curiosity-driven learning," https://arxiv.org/pdf/1808.04355, 2018.

[2] K. Balakrishman, P. Chakravarty and S. Shrivastava "An A* Curriculum Approach to Reinforcement Learning for RGBDIndoor Robot Navigation," https://arxiv.org/pdf/2101.01774, 2021.

[3] A. Ecoffet, J. Huizinga, J. Lehman, K. Stanley, and J. Clune "Go-Explore: a New Approach for Hard-Exploration Problems," https://arxiv.org/pdf/1901.10995v4, 2021.

[4] M. Savva, A. Kadian, O. Maksymets, Y. Zhao, E. Wijmans, B. Jain, J. Straub, J. Liu, V. Koltun, J. Malik, D. Parikh, and D. Batra "Habitat: A Platform for Embodied AI Research," https://arxiv.org/pdf/1904.01201, 2019

[5] A. Kumar, T. Buckley, J. Lanier, Q. Wang, A. Kavelaars, and I. Kuzovkin "OffWorld Gym: open-access physical robotics environment for real-world reinforcement learning benchmark and research," https://arxiv.org/pdf/1910.08639, 2020.

[6] H. Surmann, C. Jestel, R. Marchel, F. Musberg, H. Elhadj, and M. Ardani "Deep Reinforcement learning for real autonomous mobile robot navigation in indoor environments," https://arxiv.org/pdf/2005.13857.pdf, 2020

[7] K. Zhang, F. Niroui, M. Ficocelli and G. Nejat, "Robot Navigation of Environments with Unknown Rough Terrain Using deep Reinforcement Learning," 2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), 2018, pp. 1-7.

[8] J. Zhang, J. T. Springenberg, J. Boedecker, W. Burgard "Deep Reinforcement Learning with Successor Features for Navigation across Similar Environments" https://arxiv.org/pdf/1612.05533, 2017

[9] L. Tai and M. Liu, "A robot exploration strategy based on Q-learning network," 2016 IEEE International Conference on Real-time Computing and Robotics (RCAR), 2016, pp. 57-62.

[10] L. Tai and M. Liu, "Towards cognitive exploration through deep reinforcement learning for mobile robots," https://arxiv.org/pdf/1610.01733, 2016.

[11] Y. Zhu et al."Target-driven visual navigation in indoor scenes using deep reinforcement learning," 2017 IEEE International Conference on Robotics and Automation (ICRA), 2017, pp. 3357-3364.

[12] S. Josef and A. Degani "Deep Reinforcement Learning for safe local planning of a ground vehicle in unknown rough terrain," in IEEE Robotics and Automation Letters, vol. 5, no. 4, pp. 6748-6755, Oct. 2020.

[13] RE.Work, Go-Explore: A New Type of Algorithm for Hard-exploration Problems, Jan. 24, 2019. Accessed on: April. 30, 2021. [Video file]. Available: https://www.youtube.com/watch?v=SWcuTgk2di8t=692s